TU WIEN Informatics

Algorithmic Problem Solving in Unplugged Computer Science Outreach Activities

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Martina Landman, MEd
Matrikelnummer 01226269

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Prof. Dr. Tobias Kohn

Diese Dissertation haben begutachtet:

Tim Bell	Arnold Pears

Wien, 9. Februar 2026

Martina Landman

TU WIEN Informatics

Algorithmic Problem Solving in Unplugged Computer Science Outreach Activities

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Martina Landman, MEd
Registration Number 01226269

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Tobias Kohn

The dissertation has been reviewed by:

Tim Bell	Arnold Pears

Vienna, February 9, 2026

Martina Landman

Declaration of Authorship

Martina Landman, MEd

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.
I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Vienna, February 9, 2026

Martina Landman

Acknowledgements

Working on a doctoral degree was hard, challenging, and at times even frustrating. There were moments when I questioned myself and my expertise. I cannot even express how grateful I am to have had so many people supporting me in difficult times and cheering me on in the great moments. Over the years of working on my research, I grew personally and intellectually. There were my family, my colleagues, my friends, and the whole Computing Education Community.

First of all, I would like to thank my husband Andreas. Without him, I would not have managed to finish this *thing*. Thank you for your unconditional support and love, no matter how late I came home or how long I stayed up to write a paper for an AOE deadline. Thank you for feeding me when I forgot to eat, thank you for reminding me to take a break, thank you for your shoulder when I needed to cry, and thank you for always telling me how proud you were of me. You truly knew how to keep me going, and without you I would not be where I am now. I am incredibly grateful for your support and patience.

Next, I want to thank my supervisor, Tobias Kohn. Despite the long-distance supervision relationship, we managed to do quite well. You always responded to my emails so quickly, and your feedback was consistently rich and detailed. Thanks to your support, I had opportunities to grow and to discover places in the world that I would never have experienced otherwise, as well as to meet people I would never have met otherwise. I think everyone deserves a supervisor like you, who takes time to really read your work, answers all the questions, and listens in difficult times.

Further, a very big thank you goes to my eduLAB team! You were my work family for four years of my life, and I had many great times with all of you. All activities and events we organised together were amazing experiences. A special thank you goes to René Röpke, current leader of the team. You came at a time when I was already close to the end of my journey, and you gave me the room and the resources to be able to finish my thesis in time. In this sense, I also want to thank Lukas Lehner, who took on many of my obligations and responsibilities in the final sprint of my dissertation. I know how stressful that was, and I am infinitely grateful to you for covering my back. I wish you all the best in the eduLAB and in your work!

But there were more people than the professor and other phd-student in the lab: all the great minds and students who worked with me in the past years. Special thanks go to

those who supported me with the video analysis: Alina and Felix, you both helped a lot in carrying out my research with the cameras and the analysis. You played a big part in this research and the new findings. Patrick and Lorenz also supported me with the analysis in the later stages – thank you for your time and your thorough work, especially for your willingness to dive into something new. Many thanks also to Jessica, who always knew when I was stressed, cheered me up, and organised everything so that I could work smoothly. But even more importantly: THANK YOU for putting so much work into the card design. They turned out beautiful! I will never forget my eduLAB team! Thank you for those amazing four years we spent together.

I also want to thank Gerald Futschek, who introduced me to an amazing community with our first project and let me shape eduLAB. I really enjoyed our conversations when you visited us in the lab, as well as our adventures during numerous international project meetings.

Another big thank you goes to Laura Kovács. You adopted me into your research group through our projects, and it really helped to have your advice on so many topics.

I also want to thank Annika, who played a big part in this thesis by creating the 3D-printed weights and the balance scale. You helped me a lot with brainstorming and were also a nice and funny office mate during my visits at your department at KIT. Thank you for being my doctoral sister. Another thank you goes to Tobias A. and his helping hand in Figure 6.2.

Last but definitely not least, I want to thank my family. I am so thankful to have had my supportive parents, Karin and Werner. You always stood behind me and my decisions in life and in my career path. You both supported me throughout my whole life. Even though you sometimes did not fully understand what I was working on, I could always feel how proud you were of me. Sharing every little achievement with you made me happy, and you celebrated with me wholeheartedly, even if the details of what we were celebrating were not always clear. Thank you also to my sister Birgit. You always listened to me and celebrated my achievements with me. I am happy to have a sister like you! My final thank you is for my grandmother Hilda, who I know could not be prouder on her granddaughter reaching a doctoral degree.

Kurzfassung

Heutzutage kommen Kinder schon früh mit Technologie und Themen aus der Informatik in Berührung, weshalb Informatikinhalte in vielen Lehrplänen bereits früh eingeführt wird. Umso wichtiger ist es, dass junge Lernende früh Problemlösungsstrategien entwickeln und ermutigt werden, sich mit Informatikproblemen auseinanderzusetzen. Insbesondere in der unteren Sekundarstufe, wo Informatikthemen in der Regel erstmals in den Lehrplan aufgenommen werden, lohnt es sich, zu untersuchen, wie Kinder algorithmische Probleme intuitiv lösen und wie diese Strategien als Grundlage für spätere formale Konzepte dienen können.

Das Ziel dieser Dissertation ist es, intuitive Strategien junger Lernender im Alter zwischen 10 und 14 bei der Lösung eines algorithmischen Poblems zu identifizieren, zu untersuchen, wie sie ihre Erkenntnisse auf ähnliche Probleme übertragen können, und zu diskutieren, was dies für die Informatikausbildung bedeutet.

Diese Arbeit konzentriert sich auf drei verschiedene Aspekte der intuitiven algorithmischen Problemlösung: (1) die Wahrnehmung des Begriffs *Algorithmus*, (2) die intuitiven Problemlösungsstrategien bei einer kollaborativen Sortieraufgabe und (3) die Übertragung von Strategien auf ähnliche Sortierprobleme.

Die erste Studie verwendet eine qualitative Inhaltsanalyse der Antworten von Kindern auf die Frage, was sie unter einem *Algorithmus* verstehen. Die zweite und dritte Studie verwenden eine qualitative Analyse von Videoaufnahmen von Lernenden im Alter von 10 bis 14 Jahren, die ohne vorherige formale Einführung eine gemeinsame Sortieraufgabe lösen. Dabei beobachten wir ihre algorithmischen Strategien und verbinden sie mit bekannten algorithmischen Konzepten, insbesondere Sortieralgorithmen. Die Ergebnisse zeigen, dass alle Kinder einen dreistufigen Ansatz verfolgten, um das Sortierproblem intuitiv zu lösen: (1) Vorbereitung der Eingabe, (2) schrittweises Sortieren der Elemente und (3) Erstellen einer sichtbaren sortierten Ausgabe. Ihre Strategien ähnelten oft bekannten Sortieralgorithmen wie dem Auswahlsortieren.

Darüber hinaus zeigt unsere Analyse, dass intuitive Strategien als Einstieg in das Thema genutzt werden können und dass eine gezielte CS-Unplugged-Intervention den Lernenden helfen kann, ihre neu erworbenen Sortierfähigkeiten auf ein ähnliches Problem zu übertragen. Die Kinder waren in der Lage, weniger intuitive Sortieralgorithmen wie Mergesort und Bucketsort anzuwenden und dabei ihre bisherigen intuitiven Strategien anzupassen.

Diese Dissertation trägt zu einem besseren Verständnis der frühen Entwicklung des algorithmischen Denkens im Kontext der Informatikausbildung mithilfe des CS-Unplugged-Ansatzes bei. Sie bietet eine Grundlage und Orientierung für die Gestaltung eines altersgerechten, lernendenzentrierten Unterrichts in der zukünftigen Informatikausbildung und zeigt, wie die ersten Intuitionen der Schüler für komplexere Konzepte genutzt werden können.

Abstract

Today's children come into contact with topics related to technology and computer science from early childhood. Therefore, computer science content is covered in many school curricula from the beginning. It is crucial to develop young learners' problem-solving strategies early and to encourage them to address computing problems. Especially in lower secondary school, where computer science topics usually start finding their way into the curriculum, it is worth investigating how children intuitively solve algorithmic problems and how these strategies can serve as a basis for later formal concepts.

The goal of this dissertation is to identify intuitive strategies of young learners aged between 10 and 14 when solving a computational problem, to examine how they can transfer what they have learned to similar problems, and to explore what this means for computer science education.

This thesis focuses on three different aspects of intuitive algorithmic problem solving: (1) the perception of the term algorithm, (2) intuitive problem-solving strategies during a collaborative sorting task, and (3) the transfer of strategies to similar sorting problems.

The first study uses a qualitative content analysis of children's answers to the question of what they think an algorithm is. The second and third studies use qualitative analysis of video recordings of students aged 10–14 solving a collaborative sorting task without prior formal instruction. In these, we observe their algorithmic strategies and connect them to known algorithmic concepts and sorting algorithms. The results show that all children followed a three-step approach to intuitively solve the sorting problem: preparing the input, stepwise sorting of the elements, and creating a visible sorted output. Their strategies often resembled well-known sorting algorithms such as selection sort.

Furthermore, our analysis shows that intuitive strategies can be used as an entry point to the topic, and that a targeted CS unplugged intervention can help learners transfer their newly acquired sorting skills to similar problems. The children were able to apply more non-intuitive sorting algorithms such as merge sort and bucket sort, and thereby adapt their previous intuitive strategies.

This doctoral thesis contributes to a better understanding of the early development of algorithmic thinking in the context of computer science education. It offers a basis and orientation for creating age-appropriate, student-centred learning in future computing

education and demonstrates how students' first intuitions can be leveraged for more complex concepts.

Contents

CHAPTER 1

Introduction

Algorithms are at the heart of Computer Science (CS), yet for many students their first encounter with algorithmic concepts and the term *algorithm* itself happens not in classrooms, but in everyday experiences, metaphors, and playful activities. The term is often mentioned, for example in connection with social media, which influences and limits the idea children get of the broader concept of algorithms and their role in CS.

In addition, one challenge we face as researchers and educators, is teaching core CS concepts because they are often highly abstract and difficult to connect to children's everyday experiences[5]. For learners, it can therefore be difficult to see why they are relevant and how they apply to problems they actually encounter. This includes fostering a solid and fundamental understanding of what algorithms are and how they work. Very early in CS education, the idea of explaining the basic concept of an algorithm with a recipe occurs [40]. This analogy is only suited to illustrate some algorithmic properties, such as the concept of single steps and a sequential process. Algorithms described as simple step-by-step instructions to solve a problem [31, 59] are used in many teaching books, both for university and K–12 education [19, 35, 37]. The recipe analogy illustrates the concept of a sequence of steps well, but also limits the perception of other elements such as loops and branches, as well as other properties, such as determinism and effectiveness.

Looking at the developments in K-12 computing education in recent years, the call in our community to teach CS to a younger audience is urgent [50, 76] as the young generation grows up in a digital world where algorithms and AI systems shape everyday experiences, and countries that do not invest in computing education risk increasing social inequality and facing economic disadvantages [91]. The intention to foster ideas and abilities needed in CS is by far not new: fostering Computational Thinking (CT) [94] and teaching CS concepts in an explorative and creative way [7, 66] to a young audience have been part of CS education for decades now. Approaches such as CS Unplugged [7] have shown that

playful hands-on activities can make complex CS concepts accessible without computers and thus provide a promising setting for studying intuitive strategies for children.

What has changed since the early implementations of computing curricula in the 2000s? We have to take into account that our world now is digitally driven and fast changing. The exposure of children to social media starts earlier than ever, and this may also shape their perception of the concepts of algorithms. Recent work highlights that computing education is deeply value-laden and shaped by different rationales and traditions, and that there is no unified way of talking about computing and algorithms [74]. The *recipe* analogy never captured the richness of all algorithms' properties. It is even more difficult to connect it to today's children's experiences, as they encounter the term *algorithm* far more frequently than children did at the beginning of computing education in schools. They encounter the term in contexts such as YouTube or TikTok recommendations, where the concept of algorithms might appear as a powerful system rather than a step-by-step-procedure. Further considering that we do not have a formal definition of the term algorithm that is both precise and accessible for younger children, this leaves a tension for educators: on the one hand, the recipe analogy offers a low-threshold entry point, but on the other hand it risks oversimplifying the concept.

The expansion of computing curricula since the 2000s, the early exposure of children to algorithmic systems in social media and the term itself, and the ongoing debates about Artificial Intelligence (AI) highlight the need to investigate how children actually perceive algorithms. We do not know what the learning transfer of CS concepts for today's children, with their very different experiences in life, looks like. We know very well that children learn through the connection of new experiences with existing ones [68], but we do not know how transfer works with their daily encounters with algorithms on social media, games, digital devices, and, of course, AI systems.

The learning of algorithmic concepts is strongly related to problem solving, in particular also Computational Thinking (CT), as CT is widely considered as a set of problem solving abilities. CT has become a key educational objective in CS education [39, 75], fostering skills and abilities for problem solving, such as abstraction, pattern recognition, evaluation, optimisation, and algorithmic thinking applied to a computational context. With the advent of the CT movement, these skills were integrated into many school curricula worldwide, especially in Europe. This trend reflects a global push to provide all students with the foundational skills necessary to thrive in the 21st century [91].

Yet despite the efforts in computing education and research since the rising popularity of CT, a significant gap remains: we know very little about how children employ algorithmic and problem-solving strategies intuitively. As future CS education increasingly targets a younger audience, it is essential for educators to know more about which strategies naturally occur while solving computational problems with their own intuition.

Addressing this gap is the core aim of this dissertation. It sets the groundwork for the design of future learning activities and teaching materials, and also opens new pathways of future research that directly relates to children's experiences and their

natural intuition. In this way, learning transfer can become smoother, making sure that the topics taught in CS are age-appropriate and aligned with a modern understanding of CT and algorithmic thinking.

1.1 Research Questions

To address this research gap and gain more insight into intuitive problem-solving strategies, this thesis focuses on the algorithmic concept of sorting. Sorting is a core element of computing and essential part in computing education. Furthermore, it has inspired a wide range of educational interventions, such as CS Unplugged. Sorting can easily be demonstrated with physical objects, which allows us to observe how children approach this specific algorithmic problem and to make their thought processes visible through their sorting actions, but also apply to a large range of everyday actions.

In order to fully understand students' approaches, it is necessary to consider both their conceptual and their procedural knowledge. Conceptual knowledge refers here to how students perceive and describe the notion of an algorithm (RQ1 and RQ2). In contrast, procedural knowledge becomes visible in their actions when solving sorting problems (RQ3–RQ5). Starting from the conceptual level helps us to situate students' everyday understandings of algorithms before we examine the intuitive strategies they apply in practice.

The Research Questions (RQ) are guided by the motivation outlined in the introduction. The aim is to gain a deeper understanding of students' intuitive problem-solving strategies when encountering algorithmic sorting tasks without prior formal instruction. Furthermore, we explore whether students can be prompted or engaged to refine their initial approaches and to develop more efficient, abstract, or generalised solutions through a targeted CS Unplugged activity. Therefore, this thesis is guided by the question: *How do students perceive and apply algorithmic ideas when engaging with unplugged sorting tasks?*

This question is interesting and relevant for future CS education, as its answer can provide us with connection points for creating smooth learning transfer. However, considering all of its facets would go beyond the scope of one dissertation. Therefore, this thesis focuses on the specific domain of learning about algorithmic concepts and sorting problems and investigates the following three guiding research questions.

RQ1: How do upper elementary school students describe and explain what an algorithm is?

This preliminary question explores the perception of one of the most foundational terms in CS, the term algorithm. Answering this question provides insights into which everyday experiences children draw on, and how they describe the term 'algorithm'. It allows us to see both the notions they hold and the potential misconceptions that arise when learning about algorithms. It is especially important to know how children perceive the term, as

they encounter it in their everyday lives already before they receive any formal instruction in school, and how they relate such formal explanations to their own experiences.

RQ2: In what aspects do students' conceptions of what an algorithm is differ from established definitions?

Closely connected to the first research question, this second question focuses on the aspects and properties that algorithms must have according to the children's explanations and how these differ from established definitions. The CS community itself does not agree on a unified definition of the concept of *algorithm* [90]. This ambiguity is also reflected in education, where students and teachers disagree on which procedures should count as algorithms [11]. Answering this question provides further insight into the aspects of everyday life and the properties children associate with the term 'algorithm'.

While RQ1 and RQ2 focus on the perception of the term *algorithm*,RQ3 to RQ5 address the students' actual behaviour and intuitive approaches in practice while solving sorting problems. They investigate the intuitive approaches children employ during collaborative educational sorting interventions and transfer identical and similar CS unplugged tasks.

RQ3: What intuitive algorithmic strategies and sorting behaviours do K-12 students display when engaging in a collaborative unplugged task?

This question addresses students' actions and intuitive approaches when solving an unplugged sorting problem in practice. By answering this question we gain insights into the strategies students employ naturally without having received prior formal instruction in problem-solving or in the underlying CS concepts connected to sorting algorithms. This provides a baseline for the subsequent research questions, as it captures the strategies students display before an intervention.

RQ4: How do students revise their intuitive sorting strategies based on collaboration with peers if they encounter the same sorting problem again?

Complementing RQ3, this question examines how students revise their intuitive strategies through collaboration with peers while solving the same sorting problem again. It focuses on the transfer that happens during collaborative work without formal instruction, asking how much children can adapt and refine their intuitive strategies through interaction alone. Understanding this process is important for designing future learning interventions that build on students' own intuition and peer collaboration.

RQ5: How do students revise their initial strategies when encountering a similar unplugged sorting task?

With this last RQ we investigate whether and how students adapt or revise their initial intuitive strategies when they encounter a similar, but not identical, unplugged sorting problem. We examine the impact of a short intervention with a new instance of the

sorting task, asking whether these experiences encourage students to adapt their prior strategies and to develop more efficient, abstract, or collaborative solutions.

The five questions structure the overall research and are revisited in the concluding discussion. Each empirical study addresses them in more depth.

By addressing these five research questions, this thesis makes the following contributions to the research domain of K–12 CS education, focussing on the research field of algorithmic problem-solving in school contexts. Specific contributions include:

- An analysis of how pupils aged 11–12 perceive the term 'algorithm', and which concepts or misconceptions they show in their notion of this term.
- An explorative study of intuitive problem-solving strategies of pupils aged 10–13 shows which strategies pupils naturally employ while solving algorithmic problems without prior instruction. A description of these strategies and their links to CS concepts, is provided.
- An analysis of how these strategies we uncovered change when students are exposed to a similar problem in another context. A description of the changes is provided, as well as reasoning about the factors that influence these adaptations.
- Methodological contributions through the use of qualitative video analysis to examine group tasks in CS education.
- Insights to the social component of collaborativesorting problem, showing how students negotiate strategies with peers, use their own language to describe their actions in the sorting process, and adapt their approaches through interaction with each other.

To provide a clear structure for the reader, the following gives an outline of the thesis and its composition. It briefly introduces each chapter and its purpose.

1.2 Thesis Outline

This section is intended to guide the reader through this thesis and the different studies carried out throughout the dissertation project. Each chapter is briefly described, and its value to the overall thesis is mentioned. Previous publications are clearly marked. All included previously published work of the author is peer-reviewed.

Chapter 2: Conceptual Background. Provides the theoretical background and definitions used in the thesis, including algorithms, CT, problem-solving, and CS Unplugged, and how the author interprets them throughout the thesis. A summary of different views of these concepts is guided by relevant related work and positions the study within the broader field. It also describes the relevant knowledge of the outreach workshop setting in which the thesis studies were conducted at TU Wien Informatics eduLAB.

Chapter 3: Research Design. Describes the overall research design, which connects the three big studies conducted between 2022 and 2025 and how they intertwine with each other. The section offers an overview of the general participant selection and the methods used, especially qualitative video analysis, across the studies.

Chapter 4: The Perception of What Algorithms Are. This chapter represents the first of the three large studies we conducted during the research of this thesis. It presents a first perception of what sixth-grade pupils aged 11 to 12 years old think an algorithm is. Pupils highlighted different properties of algorithms, such as a step-by-step procedure, but also *repetition* as a concept. Data was qualitatively analysed on the basis of Mayring's qualitative content analysis [56]. The results show that pupils are able to formulate what an algorithm is in a manner strongly connected to their everyday life. This is the first study to show how pupils describe an algorithm in a classroom setting, which aligns with our preliminary goal of knowing how learners perceive the term, and further describing the concept of an algorithm. We believe that our study contributes to a wider investigation of the relationship between conceptual and procedural knowledge of algorithms and when and how the concept is best taught. It is imperative that we not only teach a thorough conceptual understanding of (specific) algorithms, but also discuss the general concept of algorithms.

Chapter 4 is based on the article: *"Something that Happens Each Day" - Students' Explanations of What Algorithms Are* [48].

Chapter 5: Intuitive Problem Solving Strategies. This study dives into pupils' intuitive strategies in solving a collaborative CS Unplugged sorting task. We highlight the algorithmic strategies that emerge during a group task and their revision through repeated attempts. The chapter addresses RQ3 and RQ4, and using video analysis of six groups of pupils, we identified patterns of action that resemble formal algorithmic approaches. We discuss the implications for teaching algorithmic concepts through unplugged activities, arguing that our findings can offer insights into how to bridge the gap between intuitive and formal reasoning of learners.

The results of this chapter exist in a shortened version and are planned to be published.

Chapter 6: From Intuition to Algorithm: Revision of Intuitive Sorting Strategies. This chapter addresses RQ5 by examining how pupils adapt their strategies after a short CS Unplugged sorting intervention between the two attempts of the task described in Chapter 5. The data consist of an additional five workshop recordings, similar to the data collection in Chapter 5. We compared strategies before and after the intervention, highlighting both the continuities and the transformations that took place. The findings will contribute in two different ways: first, they allow us to observe whether and how previously intuitive strategies become more structured or algorithmically grounded after a short CS Unplugged intervention, targeting the same concept but in a different application. Second, they help us understand to what extent children can adopt,

adapt, or critically evaluate newly introduced sorting algorithms, and how these ideas manifest in their individual and group strategies. These strategies offer new possibilities in future K–12 classrooms, like introducing data structures, especially arrays.

The results of this paper are original research of this thesis. We plan to publish a summary of these results in the future.

Chapter 7: Overall Discussion. The 'Overall Discussion' chapter integrates the findings from all studies, revisits the guiding research questions, and reflects on implications for CS education and research methodology.

Chapter 8: Conclusion and Outlook. Summarises the main contributions of the thesis, highlights its relevance for research and practice, and provides suggestions for future work.

1.3 Contributions

This thesis includes a number of peer-reviewed publications. The following article is directly included as part of the main research for the RQ1 and RQ2:

[48] Landman, Martina and Tobias Kohn. "Something that Happens Each Day" – Students' Explanations of What Algorithms Are. In: *Proc. of the 2024 on Innovation and Technology in Computer Science Education V. 1* (ITiCSE '24). ACM, 2024, pp. 199–205.

Further, while working on this thesis, the author was also involved in closely related research and practical projects in K–12 CS education. Some of these have already been published as independent contributions, all of them peer reviewed. These works were published independently, yet they are strongly connected to the overall research agenda of this thesis. Their thematic overlap and shared empirical basis increased the experience and background knowledge of the author of this work. They both inspired the design of the studies reported here and are, in turn, referenced in various sections of this thesis to provide broader context and validation.

[50] Landman, M.; Rain, S.; Kovács, L.; Futschek, G. 2023. Reshaping Unplugged Computer Science Workshops for Primary School Education. In: *Informatics in Schools. Beyond Bits and Bytes: Nurturing Informatics Intelligence in Education (ISSEP 2023)*, LNCS 14296. Springer, Cham, pp. 139–151.

[92] Vielsack, A.; Landman, M. 2026. Finding the Right Balance: Facilitating the Exploration of Sorting Strategies using 3D-Printable Weights and Scales. In: *28th Australasian Computing Education Conference (ACE 2026)*, ACM, 2026.

[51] Lehner, L.; Landman, M. 2024. Unplugged Decision Tree Learning — A Learning Activity for Machine Learning Education in K–12. In: *Creative Mathematical Sciences Communication*, LNCS 15229. Springer Nature, Cham, pp. 50–65.

[52] Lenke, M.; Lehner, L.; Landman, M. 2025. "I'm actually more interested in AI than in Computer Science" — 12-year-olds describing their first encounter with AI. In: *2025 IEEE Global Engineering Education Conference (EDUCON).*

[81] Steinert, F.; Kummer, J.; Landman, M.; Lehner, L. 2024. From Concept to Code: A Two-Day Workshop for Secondary Students on Computational Thinking and Programming. *Olympiads in Informatics* 18, pp. 89–100.

[47] Landman, M.; Futschek, G.; Unkovic, S.; Voboril, F. 2022. Initial Learning of Textual Programming at School: Evolution of Outreach Activities. *Olympiads in Informatics* 16, pp. 43–53.

[89] Unkovic, S.; Landman, M. 2023. Supporting Non-CS Teachers with Programming Lessons. In: *16th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 2023), Local Proceedings*, pp. 61–74. Zenodo.

Additionally, the thesis and its by-products, especially the designed learning interventions and materials, were recognised by the CS education community through the following invited articles, presentations, and workshops:

[46] Landman, M. 2024. Probleme algorithmisch lösen lernen. *OCG Journal* 49(1), pp. 32–33.

[49] Landman, M.; Lehner, L. 2024. Vom Schlagwort zur Praxis: Computational Thinking in der Theorie und im eduLAB. *OCG Journal* 49(4), pp. 24–26.

- Invited workshop: *Unplugged Group Activities for Solving Algorithmic Problems*, IT-õpetajate ja õppejõudude suvekool, Estonia, August 2025.
- Invited workshop: *Unplugged Power: Solving Computing Problems Without Computers*, Indian Institute of Technology Gandhinagar, India, April 2025.
- Invited colloquium talk: *Algorithmen unplugged: Wie Schüler:innen sortieren, entscheiden und Informatik entdecken*, University of Cologne, January 2025.
- Invited workshop: *Algorithmen unplugged: Wie Schüler:innen sortieren, entscheiden und Informatik entdecken*, Informatiktag STIU, ETH Zürich, June 2023.
- Workshop at conference: *Computational Thinking in Teacher Education*, World Conference on Computers in Education (WCCE), Hiroshima, Japan, August 2022.

The work on this dissertation was supported and further acknowledged through the following scholarships and awards:

- **Scholarship:** ‘netidee’, Austria’s Internet Foundation — Grant to support the research in this doctoral project (11/2024).
- **Award:** ‘Outstanding Student Presentation Award’ - for presenting “Reshaping Unplugged Computer Science Workshops for Primary School Education” [50] — at the International Conference on Informatics in Schools (ISSEP 2023), 10/2023.

CHAPTER 2

Conceptual Background

This chapter briefly introduces the most important terms, concepts and models, related work, and background information on the setting of this research. Firstly, we will explain what is meant by the term *algorithm*, how Computational Thinking (CT) is used and understood, some ideas from problem-solving and learning theories, and finally why CS Unplugged plays a role here. We will explore the background of these terms and how we understand and use them throughout this thesis. These overviews are not full reviews. They provide enough context to understand the later analysis.

Secondly, we will give insights into the TU Wien Informatics eduLAB, the outreach programme that was the setting in which the main studies from Chapters 4–6 were carried out.

2.1 Key Concepts

Our work uses specific CS Unplugged-inspired learning activities, focused on sorting and further on the general understanding of the concept *algorithm*. To fully understand the outcome of our research, we first give an overview of the concepts used and the didactical approach behind them. We also address the general problem of what an algorithm actually is as there is no consensus among experts. In particular, we discuss three key concepts that are central for this thesis: how algorithms are defined and understood, how CT is conceptualised, and why CS Unplugged plays a role in this research.

2.1.1 Algorithms

We come into contact with algorithms every day; it is a term that students usually encounter in their everyday lives even before a formal introduction to school. As in Computer Science (CS) education, one goal is to teach *algorithmic concepts*, we first need to know what we as researchers and educators understand by this term. Interestingly,

there is no consensus on how to define the the term algorithm with all its properties in CS. It is an ongoing discussion in our field [90], as as intuitive notions of algorithms are often used in practice. Working in the field of CS does not require a standardised definition, as a working definition or notion is usually enough in practical applications and work in CS.

What has become increasingly important in recent years, and suggests building a proper definition of the term, is the growing field of Artificial Intelligence (AI). The urgency of a precise definition arises from the current prevalence in the media, which makes it increasingly important to define the term algorithm [54]. Policymakers are currently working to develop standards for the evaluation and audit of algorithms in the field of AI, but there is likely to be disagreement about what constitutes an algorithm [54].

Source	**Definition**
Hetland	"An algorithm is a procedure, consisting of a finite set of steps, possibly including loops and conditionals, that solves a given problem. A Turing machine is a formal description of exactly what problem an algorithm solves [...]." [35]
Hromkovic	"An algorithm for solving a task is a description of a procedure that leads to the solution of the task. The description consists of a sequence of instructions that can be executed by anyone[...]." [translated] [37]
Grover & Pea	"Algorithms are precise step-by-step plans or procedures to meet an end goal or to solve a problem; algorithmic thinking is the skill involved in developing an algorithm." [31]
Cormen	"Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship." [19]
Merriam-Webster Dictionary	"a step-by-step procedure for solving a problem or accomplishing some end" [59]
Herold	"An algorithm can be understood as a set of instructions, which can be executed by a mechanical or electronical working device, or also by a human." [translated] [34]
Gumm	"An algorithm is a detailed and explicit instruction to a stepwise solution of a problem." [translated] [32]

Table 2.1: Selection of different algorithm definitions from common teaching books, some translated to English.

This disagreement is also reflected in the definitions used in textbooks and research (Table 2.1). While some definitions highlight formal properties such as finiteness, definiteness, and input–output relations [19, 35, 37], others describe algorithms more generally

as step-by-step instructions or everyday processes [31, 32, 34, 59].

When we refer to the term algorithm throughout this thesis, we generally follow the definition by Knuth [41]. Knuth managed to define the term in a very precise way, while still allowing a more practical description such as a step-by-step procedure or even the recipe analogy (Table 2.2). He formulates five clear *features* that must be fulfilled in order to decide whether something can be called an algorithm: (1) finiteness, (2) definiteness, (3) input, (4) output, and (5) effectiveness, with their respective details explained in Table 2.2.

Feature	**Description**
Finiteness	The algorithm must always terminate after a finite number of steps.
Definiteness	Each step of an algorithm must be precisely and unambiguously defined.
Input	An algorithm has zero or more inputs provided before or during execution.
Output	An algorithm has one or more outputs that are related to the input in a specific way.
Effectiveness	Each operation is basic enough to be carried out exactly and in a finite time, by a human or machine.

Table 2.2: Knuth's five defining features of an algorithm [41]

In fact, from a historical and etymological perspective, the term algorithm derives from the name of the Persian mathematician al-Khwārizmī [24, 41, 20], whose name appears in mediaeval Latin sources as Algorizmi. Al-Khwārizmī was named, as was typical at the time, after his place of origin, Khwarazm, an oasis region located between today's Uzbekistan and Turkmenistan. Al-Khwārizmī did not invent the concept of algorithms, but his way of systematically writing algebraic calculations as stepwise instructions made his work influential and comparatively easy to learn and therefore widely accessible in Europe. Over time, his name was Latinised and later generalised to refer to a rule-based technique of calculation [96].

Considering the history of the term and the procedure behind it shows that algorithms have existed for a much longer time than computers. This contributes to a modern interpretation of the term algorithm in which it is not necessarily tied to execution on a computer or to a computer program. Instead, it refers more generally to the idea of the unambiguous interpretability of a procedure.

2.1.2 Computational Thinking and Problem-Solving

CT is a widely used but also highly debated term [39, 87, 74]. The roots of its idea can be seen in Seymour Papert's *Mindstorms* [66], while Jeannette Wing's article in 2006 [94] made the term popular and highly visible. Both stressed that CS education is not just

about programming, but about fundamental ways of solving problems where a special way of thinking is needed. Wing describes this as "Thinking like a computer scientist", which includes more than just being able to programme [94].

Since then, many definitions have been suggested, and there is still no single agreement (e.g. [75, 30, 65]). Selby and Woollard [75] compared definitions and listed five cognitive processes that are usually mentioned: abstraction, decomposition, algorithmic thinking, evaluation, and generalisation. These highlight that CT is often described as a set of problem-solving skills.

Kafai et al. [39] remind us that CT can be framed differently: as a set of cognitive skills, as situated practice in communities, or as a critical perspective on society and power structures. Our short interventions mainly address the cognitive frame, although we keep in mind that CT also has social and cultural dimensions. The Hybrid Interaction System (HIS) model by Schulte and Budde [73] highlights these social dimensions: The model aims to conceptualise the interaction between humans and technological aspects. Guzdial et al. [33] even argue that for children "computational thinking is just thinking."

Other authors raise a more methodological concern: Nelson and Ko [63] suggest that we need domain-specific theories to avoid overloading our research with vague theoretical frameworks. This is also important for our work, as we focus on algorithmic problem-solving in a very concrete setting.

The different views about CT can be located in a spectrum of narrow and broad applicability as it has been discussed by Curzon et al. [20]. At the narrow end CT aligns closely with learning how to program, therefore focussing on developing solutions intended to be executed by a computer or a machine. At the broad end of the spectrum CT is considered to be a general problem-solving skill, useful for all disciplines and not limited to CS. Curzon et al. suggest that it is best to stay in the middle of the spectrum beside personal preferences. This aligns with our work in this thesis as the pupils act as computational agents executing a computational problem applied to the real world with our unplugged card sorting task.

A further challenge is the assessment of CT. Attempts such as Bebras tasks or smaller tools [21, 53] exist, but there is no consensus on how to measure progress in CT skills. This makes it difficult to judge the success of interventions.

For this thesis, we therefore view CT as a broad idea, mainly as a set of different problem-solving skills solving computational problems. The ability to develop, adapt, and reflect on strategies to solve a problem is central and is exactly what we investigate in our studies.

2.1.3 CS Unplugged

The CS Unplugged approach can be defined quite simply as learning about CS without using computers [10]. It is a collection of activities that explain concepts such as algorithms, data representation, or sorting without relying on digital devices, nor needing

prior programming education. The idea of a low-threshold entry into CS is not entirely new, but has gained much attention with the rise of the CT movement. One example is the interactive *Abenteuer Informatik* exhibition, which uses a wide range of haptic materials to make core CS ideas tangible [28]. This constructivist approach emphasises that students should explore the principles themselves rather than only having them explained by an instructor [8].

(1) Explorative and open-ended tasks in context of CS Unplugged can enhance CT skills, such as abstraction, decomposition and generalisation[18]. In particular, they create situations where problem-solving becomes central. (2) Further, learning through actions with physical objects, such as CS unplugged, relates to the enactive layer and therefore provides the foundation for the first step of learning [14]. Research in mathematics education shows that manipulatives, which are hands-on objects and tools to make abstract mathematical concepts tangible and easier to understand, can enhance abstract thinking [16]. (3) At the same time, these benefits require careful design, because if tasks are too open-ended they may risk causing cognitive overload and hinder learning [82]. The need to find the right amount of instruction when guiding CS Unplugged activity is crucial [16]. Therefore, our activities are designed to always include at least a supportive tutor, who can provide hints and guidance. They are structured to remain short-cycled in order to avoid long phases of frustration.

(4) Another important aspect of unplugged activities is that they reduce barriers to entry. In traditional programming courses, prior knowledge of a programming language can be a major obstacle [8]. In contrast, unplugged tasks start with everyday actions, stories, and physical materials, making them accessible to a much wider audience. This low entry point also reduces the risk of mental overload, which, according to the principles of Cognitive Load Theory (CLT), should be an important goal in educational design [83].

Taking the four described aspects above together, they show that CS Unplugged activities (1) foster central CT skills such as abstraction and decomposition, (2) provide a hands-on entry that connects to fundamental learning processes, (3) can balance openness with the need for guidance to avoid overload, and (4) lower barriers to entry by building on everyday experiences. For these reasons, they form a promising context for investigating how students approach algorithmic sorting problems.

2.2 Teaching Sorting in Practice

This section gives an overview on how CS Unplugged is realised in practical applications. We will introduce existing unplugged sorting activities, and how they are applied in learning settings, such as the TU Wien Informatics eduLAB.

We used the setting of TU Wien Informatics eduLAB to serve as the base of our research and learning interventions. The eduLAB follows common third-mission traditions that can also be found in the international CS community, aiming to foster early interest, broaden participation, and address challenges such as gender imbalances and inclusion [57, 71, 64].

It also offers a space for experimentation and for studying how learners engage with computing concepts outside formal classrooms [48], making it suitable for research with learners aged 7 to 19.

Sorting depending on the setting of the activity and the age group can mean both classifying and ordering. In this thesis when we refer to sorting we mean ordering a set in an ascending or descending order.

2.2.1 Sorting in CS Unplugged

Sorting is a central algorithmic problem and therefore a common theme in CS Unplugged activities. Various approaches, such as balance scales, sorting networks and sorting cards have been designed to make the concept of sorting tangible. Each approach has its own strengths, depending on which aspect of sorting the intervention focuses on. In the following sections we explain three well-known CS Unplugged sorting approaches and how they are implemented in practice.

Sorting Networks

Sorting networks are a great opportunity to make sorting visible. They are well-known by CS Unplugged enthusiasts and in the official material collection [10], as well as in the Abenteuer Informatik book and exhibition [28]. The official CS Unplugged website [1] offers a variety of prepared teaching lessons and printables, targeting age groups from 4 to 14 year-old learners using sorting networks. The idea of a sorting network is to visualise all comparison operations for sorting a range of sortable items. The eduLAB hosts the *Abenteuer Informatik* exhibition in their entrance hall and therefore has a sorting network printed on the floor, big enough to sort people (Figure 2.1). Our example focuses on a sorting network for the size of six sorting items, as this is the size used in *Abenteuer Informatik*, but the idea is adaptable to any size [85].

The sorting process with the sorting network presented in Figure 2.1 starts with six sortable items placed in the orange squares of the network. In each step, the elements follow a line into a black ellipse. The ellipse stands for a binary comparison between the two items in them – the *larger* item follows e.g. the blue line, the *smaller* item the other one. When all items are in a green square at the end, the algorithm ends and the items are sorted.

This can be done with pen and paper, objects on a printed sorting networks, or learners experiencing sorting through acting as sortable items by walking through a big sorting network [50]. For this purpose, the learners step into the orange squares (Figure 2.1).

Balance Scales

While a parallel sorting network visualises parallelisation and algorithmic step-by-step procedures which the learners have to follow showing the concept of pairwise comparisons

[1] CS Unplugged: `https://www.csunplugged.org/en/resources/sorting-network/`

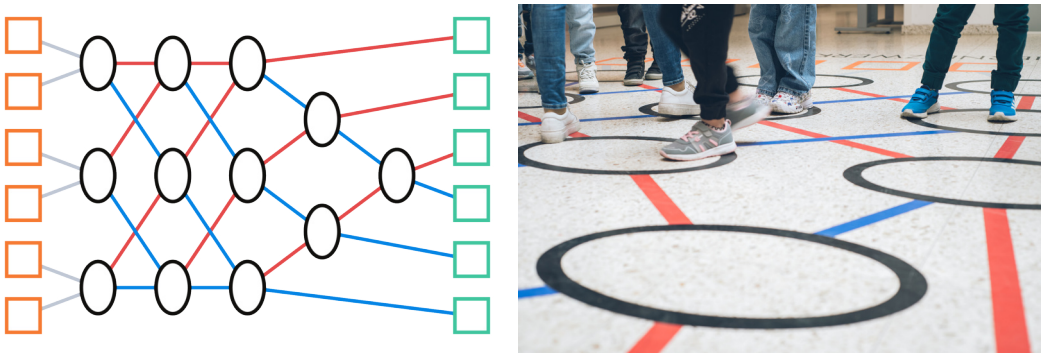

Figure 2.1: Diagram of a sorting network as it is used in *Abenteuer Informatik* (left) and the implementation of it on the floor in the entrance hall of TU Wien Informatics during an eduLAB school workshop (right, ©Amélie Chapalain).

like a computer would compare, they are not meant to teach or conceptualise classical sorting algorithms. For this purpose the CS Unplugged approach of using a balance scale to compare weights can be beneficial [10].

The balance scale fulfils the role of the comparator being able to only compare two items. The weights are the items to be sorted. The key idea behind the weights is that learners do not know which element is the smallest/lightest or largest/heaviest right away by looking at it. The different weights should be of the same size and differ only marginally in mass so the learners cannot *feel* the difference easily. This results in the need of the scale, and also the need of an algorithmic concept on how to find the lightest or heaviest element.

The scale can be used in an enactive and explorative way to find the lightest or heaviest element [10], but also in a demonstrative way of visualising each step of a specific sorting algorithm [28].

Cards

Using cards to sort is an easy way to make the sortable items tangible and moveable. For example, cards can be easily arranged on a table and arranged in different structures as stacks and arrays, e.g. by laying the cards out on the table in a list or as a card stack. While using cards with sortable entities written or drawn on them and arranging them in a sorted list or stack are closest to how sortable items are organised in a computer. Therefore, they offer a good visualisation of the data to be sorted.

They are also very low-cost and easy self-made through printing or crafting. The items that are displayed on the cards can be varied easily and adapt to the learners prior knowledge, e.g. using animals in different sizes instead of numbers for very young learners who are not comfortable yet with numbers [50], or using playing cards, e.g. from ace to king [79].

Figure 2.2: A group of children using the balance scale during an eduLAB intervention, using the adapted materials from the *Abenteuer Informatik* exhibition.

Connections Between Approaches

These three approaches illustrate how sorting can be addressed from different perspectives in CS Unplugged. Sorting networks highlight step-by-step procedures and make parallelisation visible, while balance scales highlight the binary nature of comparisons and the necessity of algorithmic strategies to identify minima or maxima. Cards, in contrast, provide a highly flexible medium that can be adapted to different age groups, different kinds of sortable attributes, and allow the modelling of classical sorting algorithms in practice.

What these approaches have in common is that they make the limited operations of a computer tangible. Humans can compare and arrange several items at once while sorting and often use multiple attributes such as colour, size or value simultaneously. In contrast, a computer can only perform pairwise comparisons and needs repeated step-by-step procedures to address additional attributes. Humans swap items seamlessly, while a computer requires a temporary placeholder, which adds an extra step to the algorithm [79]. The challenge in unplugged sorting activities is therefore to encourage learners to adopt a computer's perspective rather than relying on their own intuitive and natural strategies.

Carefully designed materials, such as balance scales, sorting networks, and cards, enable learners to experience different aspects of sorting in the way that a computer sorts data items. They offer opportunities for discovery learning as well as guided learning sessions, in action, and to connect their experiences with the algorithmic strategies computers

apply when sorting data.

2.2.2 Computer Science Education in the eduLAB

To better understand how CS Unplugged can be applied in practice, and how studies can be carried out in the eduLAB, we provide a short overview of how the eduLAB is organised and what it has to offer.

All people affiliated with the eduLAB are members of TU Wien Informatics. All activities are organised by CS education researchers, including PhD students. Together with interested undergraduate CS students (i.e. tutors), we organise school workshops and coding camps, participate in events like exhibitions, and organise special school events and cooperations, including creating new special curricula in schools. We mainly target school pupils aged 7–18, but also offer teacher training and workshops, as well as academic talks and workshops.

Most of the audience we reach are school pupils. They visit eduLAB workshops during their school hours, organised by their school teachers in the facilities of our faculty. In the past four years, the eduLAB has grown fast, and we can look back on the attendance over 3 400 pupils in 2024 and more than 100 adults, including teachers and parents.

Designing Learning Activities

The heart of the eduLAB are our self-created CS workshops, heavily inspired by CS Unplugged activities [7] and the *Abenteuer Informatik* (engl: Adventure Informatics) exhibition, an interactive exhibition using many CS Unplugged ideas [28]. The exhibition is permanently hosted in our faculty's building and is accessible to the public. Due to the demand for guided tours through the exhibition, we started to run workshops based on specific exhibits. These workshops were initially only aimed at secondary school students and were funded by the faculty. As the programme became more popular, we expanded it to primary schools, funded by "Let us empower Austria" (LEA).

The workshops we host show the possible practical applications of the CS Unplugged approach mentioned in Section 2.1.3: engaging workshops for all ages, teaching CS concepts with a low entry threshold.

Topics in regularly hosted eduLAB workshops are:

- **Algorithms.** The eduLAB offers the *Algorithm* workshop for three different age groups: primary school (aged 7–10), middle school (aged 10–14), and high school (aged 14–18). It explores what an algorithm is, with interactive CS Unplugged activities using different algorithmic concepts, starting from simple step-by-step solutions for open-ended creative tasks to find, e.g., a path out of a maze.
- **Codes.** This workshop explores what encoding and decoding information means. The idea of binary codes is explored via a pantomime game, and also with learning activities using 3D-printed sticks representing one bit.

Figure 2.3: Two CS Unplugged activities created in the eduLAB using 3D-printed binary sticks in two ways: using them to code binary information in the coding workshop (left) and building a decision tree using them as paths to make binary decisions in our AI workshop (right). Picture: ©Anja Rott | TU Wien Informatics eduLAB

.

- **Decision Tree Learning.** This workshop explores the mathematics behind machine learning, completely unplugged. We evaluated the change in students' self-assessment about AI and machine learning. We found that there was a positive effect directly after students participated in the unplugged workshop [51].

- **Programming.** Once a year the eduLAB organises a summer coding camp for young students aged 10–14, using block- and text-based programming languages. We also supported teachers in online programming lessons with no or little programming experience to supplement their CS lessons [89, 47].

Besides school workshops, we also offer teacher training [89] and coding camps, as well as special coding activities and courses [47]. Further, we have several larger cooperations with schools with yearly events. One example is a two-day workshop intervention with approximately 150 pupils of grade 7 (aged 11–12). Cooperations like this are ideal as a research setting, as we have access to many participants at once. Therefore, this event is always used to test newly created activities, evaluate their significance [81], or gain more insights into certain topics about CS [48, 52]. For example, one of the big studies of this thesis, exploring pupils' perception of the term algorithm, was conducted during one of these cooperations [48]. New topics like AI were also explored. We explored pupils' first encounter with AI [52].

The Algorithm Workshop

The Algorithm Workshop is the main setting of this thesis research. Since its first inception, the Workshop undergoes regular improvements and adaptions as part of the daily work in the eduLAB. This process generally follows a Design-Based Research (DBR)-inspired approach, although without an explicit aim of scientific methodology or

validity. The experience of conducting the workshop leads to ideas and insights on how to improve the various activities.

DBR as an established methodology typically consists of iterative cyclic steps, in this case: (1) (re)design, (2) implement and (3) analyse [58]. We (1) redesign existing eduLAB interventions and materials, (2) implement them through the eduLAB workshops, and (3) analyse the pros and cons of the interventions and materials through cyclic reflection rounds of tutors or student feedback, as well as observations. In contrast to full DBR our improvements lack the necessary documentation or grounding in theory[6].

In addition to their outreach function, the workshops also provide undergraduate tutors with an opportunity to gain real-life teaching experience. They reflect on their observation and try out new approaches with as low a threshold as possible, leading to a less-scientific and more teaching-oriented variant of DBR.

While improving the workshop itself serves as a training ground for teaching undergraduate students, this thesis focuses on a different approach of scientific analysis. For this purpose, we separated out a specific activity, namely that of sorting cards. We saw the potential of making thought processes visible during a sorting intervention. We excluded it from the usual cycles of other activities and refined it to be suitable as a learning intervention but also as a research instrument.

In general the workshop consists of several shorter CS Unplugged activities using an explorative approach, with a focus on the following algorithms and concepts:

- sorting, parallelisation and optimisation
- scheduling algorithms
- algorithmic elements (such as single steps, sequences, decisions and loops)
- shortest path

The workshop is usually 90 minutes long and is aimed at students aged 10–14. Besides this dissertation project, we further developed a smilar workshop together with the workshop tutors targeting the age group 14–18, and also for primary school aged 7–10 [50].

The sorting card activity serves as the entry activity in the Algorithm Workshop and therefore is not affected by the other activities of the workshop. It also acts as the research instrument of this thesis: an activity to learn about the concept of Divide-and-conquer while sorting cards. The original idea was to use this activity as a connection for a further discussion about the divide-and-conquer concept and parallelisation. The task explanation was simple for a group of 4–5 students: sort the entire deck of cards as quickly as possible.

Beginning in 2022, we saw the potential of using this activity as an instrument to observe problem-solving because of the open-ended task, its exploratory character, and sorting.

Figure 2.4: The original *Ligretto* card game (left) and the newly designed sorting cards with their four different backs and fronts (right).

The original activity used cards from an existing card game called 'Ligretto'[2]. The rules of the game are not important for the idea of the sorting task. To explain the structure of the card deck in the game, it is important to know that it is intended for four players. Each player plays with a deck, each with a different back colour, that contains 40 cards. The fronts also have four different colours, each colour with sequential numbers from one to ten (Figure 2.4, left).

The first time we redesigned the task to address a younger audience was in 2022, when we created our first primary school workshop [50]. We successfully reduced the mental load and the needed prior knowledge so children from age seven onwards are able to experience this workshop. We reduced the tasks complexity through only using one back colour deck in the first round, only adding another back colour deck in the second round. We also increased the training time through a third round to give the children more time to get used to the material.

However, in our pilot video studies the computational problems this task addressed were limited because it was impossible to observe applied sorting algorithms using only numbers 1–10, especially with students aged 10+. Specific sorting operations, such as comparisons and how to find and choose the next element, were not visible.

As a result, we designed and produced set of cards inspired by the original card game. We kept the colour system the same, four different back colours, each with four different front colours, but increased the number of cards for each front colour sub-deck. The reasons were as follows.

- Instead of sequential numbers we used random numbers, a specific set of 20 cards out of the numbers 10 to 99. This should ensure that the students do not automatically know which number is the lowest and the highest, and which number is the next one. They have to find a strategy.

[2] https://www.schmidtspiele-shop.de/kartenspiele/ligretto

- The specific set of 20 random numbers is unique for each front colour sub-deck. This should ensure that the students do not remember the specific numbers in repeated sorting attempts. This results in having 16 different sets of random numbers.
- Increasing the number of cards for each front colour sub-deck from 10 cards to 20 cards should trigger a card organisation strategy. It also increases the chance to observe a searching strategy and how they compare numbers with each other.

The final version of the task is described in more detail in Section 5.3.

In general, students enjoy eduLAB workshops and we get good feedback, including a rise in interest in CS [81, 50].

2.3 Learning Principles

In the literature, we find several learning theories that support our eduLAB hands-on approach as well as our explorative and open-ended learning activities. For example, when redesigning workshop tasks for younger or older students [50], we kept the underlying objectives but adapted the level of complexity.

This idea is strongly related to Spiral Curriculum [15], suggesting that central concepts can be revisited at different ages and with increasing levels of complexity.

Collaborative open-ended tasks are also linked to Zone of Proximal Development (ZPD), defined as the distance between what a learner can achieve independently and what can be reached with guidance or support from others [93]. In our workshops, and especially in the card-sorting activity, this concept is visible when hints from peers or minimal tutor support help students to take the next step in their problem-solving process.

Furthermore, experiential learning [43] is addressed through our exploratory setting, which builds on existing experiences as a source for learning new concepts. In this context, CS Unplugged reflects Dewey's pedagogical principle of learning by doing [22], as it connects new knowledge to hands-on practical experiences.

The underlying explorative approach also aligns with constructivism, where new knowledge is actively created by the learners based on their prior experiences [68]. In our CS Unplugged activities, this is reflected in how learners test and develop their own solution ideas.

CHAPTER 3

Research Design

We designed the research of this doctoral thesis in three main studies, all of which were carried out between September 2022 and June 2025. The first study covers RQ1 and RQ2, the second study RQ3 and RQ4, and the third study covers RQ5. While we discuss the studies together and compare their results, each chapter can stand alone and therefore has an own conclusion section.

In this chapter, we elaborate on the details about how the three main studies are connected, who was involved, and how the methodology of the studies fits into the big picture of this thesis. The detailed methodology for each study is described in their respective chapters (Chapters 4-6).

3.1 Overview of the Three Studies

The three main studies are structured in a way that they focus on different levels of research: first, exploring the starting point of students' perceptions, second, observing their intuitive strategies in action, and finally, examining how these strategies can change through learning transfer. In this sense, the sequence moves from what learners bring with them to what they actually do and to how they adapt when guided by interventions.

The Basis. The first study lays the ground for the following two studies, since we focus on the perception of the term *algorithm*, which mainly reflects students' conceptual knowledge. To be able to answer the main research question, we first need to find out how children perceive the term. We let students write down their definition of the term algorithm in their own words. Using a qualitative content analysis [56] of short written explanations, we gained insight into everyday interpretations and associations. This was important for RQ1, as it shows the background knowledge and mental models that learners bring with them into our workshops, and also leads to insights about misconceptions.

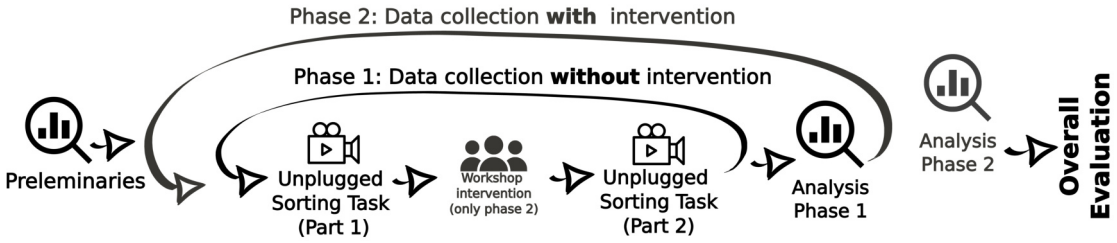

Figure 3.1: Overview of the process to conduct the 3 main studies guiding this thesis.

The Strategies. The second study moved from perceptions to actions. Here, we analysed how students intuitively solved the unplugged card-sorting task in small groups, which gave us insights into their procedural knowledge. The activity was open-ended and students had to develop their own strategies without intervention. Using qualitative video analysis, we observed individual approaches and collaborative processes, helping us to answer RQ3 and RQ4. These observations provided a detailed picture of the strategies and difficulties that naturally emerge.

The Transfer. The third study extended this perspective by adding an intervention between the two sorting rounds of the unplugged card-sorting task from the previous study. After a short intervention phase in which sorting algorithms were introduced, the students then continued with the second round of the sorting task. This allowed us to compare their strategies before and after this intervention, and to study how this influenced their approaches. The focus lies on RQ5.

3.2 Overall Methodological Approach

3.2.1 Preliminaries and Preparation

In 2022, the first pilot workshops and test recordings were carried out. These early trials showed that the original card game we wanted to use was not suitable for our purpose: with only sequential numbers from 1–10, the sorting process was almost trivial and strategies were hardly observable. Based on these insights, we developed a new card deck, increasing complexity and randomness so that students needed to make actual choices and comparisons. In the same way, the technical setup was tested and refined. Recording from a bird's-eye view turned out to be the best compromise between anonymity and observation of actions. These pilots shaped the design in a cyclical way: after every round, material and setup were adjusted and then tested again. This iterative preparation was important to create the ideal recording setup. The final recording setup included the camera on a stand attached to the table, from a bird's-eye perspective, and separating the groups with a mobile felt poster board to reduce background noise during the recordings (Figure 3.2).

Figure 3.2: The camera setup for the video recordings showing the camera stand and the bird's-eye perspective, both before and during the recordings. Picture: own photo, TU Wien Informatics eduLAB.

3.2.2 Qualitative Approaches

The overall project followed a qualitative orientation. If the goal is to understand how students develop algorithmic strategies, measuring speed or accuracy alone would never be enough. Interviews or questionnaires after the workshop could only capture what students remembered or decided to tell. We were interested in their actual actions: the moments when they hesitated, when they invented a plan, when a mistake led to a discussion. For this reason, qualitative methods, and especially video analysis, were the only way to capture the processes in sufficient detail. In this sense, the project is exploratory: it aims to describe and interpret naturally occurring strategies in an unplugged setting, not to prove predefined hypotheses.

3.2.3 Age Group

The participants were mainly between 10 and 14 years old. This choice was guided by both practical access and theoretical reasons. Practically, the TU Wien TU Wien Informatics eduLAB workshops are very popular with school classes in this age range, so we could recruit more than 300 participants across all three studies, and even more who gave us their consent to participate. Overall, we analysed 58 students' answers in our first study, and analysed 44 participants in 11 videos in our second and third study. Moreover, this age group is particularly interesting: it is the time when children already bring rich everyday experiences with algorithms (from games, social media, daily routines), but

before formal algorithm teaching really begins in their curriculum. With younger pupils, many concepts would have been to be simplified so much that complex strategies were rarely visible. With older pupils, on the other hand, we would already have seen formal school knowledge rather than their first intuitive approaches.

3.2.4 Ethics, Consent and Data Protection

Working with minors required careful preparation on ethics and data protection. Since we planned to record videos of the workshops, we developed a consent procedure that was clear and transparent for both children and their parents. Participation in the study was voluntary and only children with a signed consent form were recorded. Others could still participate in the workshop, but were not filmed. The study design, including the consent forms, was reviewed and approved by the Ethics Committee of TU Wien. The camera setup was also chosen with privacy in mind: the bird's-eye perspective allowed us to observe the problem-solving processes while ensuring that faces were not visible.

Access to the recorded data was restricted to the research team and supervised student assistants directly involved in the analysis. Videos were stored on secured university servers with password protection and were not shared with teachers or external parties. According to the consent procedure, participants and their parents could withdraw their consent at any time without giving reasons, in which case all related data were deleted. The videos are retained only for the duration of the project and deleted after the completion of the dissertation and related publications, in accordance with the university's data protection guidelines.

3.2.5 Methodological Orientation

The methodological choices in this thesis can be summarised in three strands:

- **Qualitative Content Analysis (QCA).** In the first study, students' written answers about the term algorithm were analysed using qualitative content analysis [56]. This allowed us to identify recurring themes and everyday conceptions without imposing predefined categories.
- **Qualitative Video Analysis.** In the later studies, the focus was on group work during the sorting task. Here we used qualitative video analysis, inspired by the Documentary Method [12] and Thematic Analysis (TA) [13]. The analysis proceeded in cycles: first descriptive coding of visible actions, then interpretative steps to understand the strategies behind them. Coding was done collaboratively in a team to ensure consistency.
- **Cyclical Refinement.** The research instrument (i.e. the card sorting activity) was iteratively developed before the start of the data collection (see Section 2.2.2 and Chapter 5). The cards, acitivity schedule, and coding schemes for later analysis were revised in several pilot phases and first analyses. Through discussions in

reflective rounds after the pilot sessions we adjusted the activity. Additionally as a side effect, the project produced high-quality unplugged activities that are now used in outreach and teaching.

3.2.6 Connecting Research Design and Research Questions

Taken together, the overall research design aligns closely with the three main research questions. Study 1 explored how students perceive the term *algorithm* (RQ1 and RQ2). Study 2 examined how students intuitively solved the sorting task without guidance (RQ3 and RQ4). Study 3 added an intervention and asked how strategies developed when a similar problem was encountered again (RQ5).

The cyclical nature of the preparation and the combination of different qualitative methods ensured that each study built on the previous one. From initial conceptions, to observed actions, to strategy change under guidance, the design of this project was meant to provide a coherent picture of how children deal with algorithmic problems.

CHAPTER 4

The Perception of What Algorithms Are

This chapter is based on article [48]:

M. Landman and T. Kohn: "Something That happens each day" - students' explanations of what algorithms are, in Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (M. Monga, V. Lonati, E. Barendsen, J. Sheard, and J. Paterson, eds.), ACM Digital Library, (Milan, Italy), pp. 199–205, Association for Computing Machinery, 2024.

4.1 Introduction

The concept of *algorithms* is at the very core of computer science [41]. Yet, good definitions or explanations remain elusive to the extent where many courses introduce the concept by example and leave it at preliminary, often somewhat vague, working definitions. In the public discourse the term 'algorithm' also finds increasing popularity but with a somewhat different meaning [60]. This is reflected to a certain degree in our observations during outreach activities that many pupils answer that they know the term because of the '⟨insert-random-social-media-platform⟩ algorithm'.

Our K-12 outreach activities are based on CS Unplugged [7] and run with the aim of increasing the pupils' understanding of computational thinking and algorithmics. However, although various algorithms are at the core of the activities, the concept is usually not formally defined or introduced. We often observe that some of our tutors choose to discuss the concept with the pupils. Particularly given the highly abstract nature of the algorithm-concept, it is not clear what notion of 'algorithm' pupils actually have in their minds.

During a two-day outreach activity we therefore ran a 'competition' asking the pupils to give their best shot at concisely explaining what an algorithm is. The students were allowed to both write or draw a picture on a small card. We then analysed these 58 explanations and definitions and present our findings in this paper. The study presents a first snapshot of what 6^{th} grade pupils (age 11–12 years) from a single school think what an 'algorithm' actually is. To the best of our knowledge it is one of the first studies to explicitly look at K-12 students' understanding of the algorithm-concept itself.

Our study was guided by the following two research questions:

RQ1 How do upper elementary school students describe and explain what an algorithm is?

RQ2 In what aspects do students' conceptions of what an algorithm is differ from established definitions?

Mostly in line with the canonical analogue of a 'cooking recipe' we found that students highlighted step-by-step execution as well as the fixed or predefined nature of an algorithm. Additionally, most students mentioned repetition as a key concept and showed some uncertainty concerning who would execute an algorithm. Even though many students also mentioned the problem-solving aspect of algorithms, few considered finiteness.

Given the prominent role of algorithms in computer science, we believe that our study contributes to a wider investigation of the relationship between conceptual and procedural knowledge of algorithms (i.e. being able to formulate what an algorithm is versus working with algorithms), as well as when and how the concept is best taught. After all, the recent rise of AI has led to a public awareness of and discourse about algorithms that clearly requires a thorough conceptual understanding of what an algorithm is, its possibilities and its limitations.

4.2 Background

4.2.1 Related Work

The concepts of computation and algorithms clearly lie at the heart of computer science [41, 61]. In computer science education, there has recently been an increasingly strong emphasis on 'computational thinking' (CT) [94]—a term that has replaced the earlier 'algorithmic thinking' [27, 55] and is based on a broad view of computation that includes 'algorithms' as one of its key concepts [55, 75].

There has also been a growing awareness that the underlying model of computation is of paramount importance for a proper definition and conception of computation and algorithms [1, 42]. In the context of programming, this has been discussed for some time with the idea of the 'notional machine' [23, 25], although this concept has not been widely adopted or applied to computation and algorithms in general so far.

Correctly expressing or representing algorithms is a non-trivial task. Students generally seem to lack an understanding of algorithms [27]. With the difficulty of learning to program to begin with, it usually requires a lot of time and training before students are capable of implementing algorithms as computer programs. The CS Unplugged initiative therefore offers an alternative route without programming and where algorithms and computational thinking are at the centre of attention [7, 9, 86]. Unplugged activities allow students to act as computational agents, supporting them in developing mental models to build a notional machine [62]. Alternatively, algorithm visualisation uses technological means to provide dynamic representations of algorithms so as to foster a better understanding [38, 77]. In contrast to these approaches, our study focuses on the abstract and general concept of algorithm rather than specific algorithms.

A study conducted with undergraduate computer science students in the Netherlands looked into the students' understanding of the concept of algorithm [67]. Their focus was on levels of abstraction used by the students with four proposed levels: at the *execution level* an algorithm is essentially the execution of a program, at the *program level* the algorithm is the process described by the program, at the *object level* the algorithm can be viewed as an object in its own right and at the *problem level* different algorithms can be used to describe problems and their intrinsic complexity. The study found an increase in the abstraction levels with years of study, but noted that typical answers given settled around the program and object levels. During their study, the undergraduate students were also asked to give their definition of 'algorithm'. However, the study does not go into details of the students' answers.

The previously mentioned paper on students' understanding of algorithms [67] also notes that many students have difficulties clearly expressing their thoughts, limiting the reliability of answers collected from students. On the other hand, we also have to be aware of the limitations of explanations given by instructors and what students understand thereof. Explanations are often based on metaphors and analogies—even more so for pupils in elementary/secondary schools—which provide a powerful means of teaching, but may also easily lead to misconceptions [26]. In the answers students gave in our study we can very clearly recognise misconceptions that arise from the examples and metaphors provided by the instructors.

From a cognitive point of view we are more interested in the students' *conceptual* (and *factual*) *knowledge* than their *procedural knowledge* [4, 69]. Procedural knowledge is explicitly defined as "how to do something [...] and criteria for using skills, algorithms, techniques and methods" [4] whereas conceptual knowledge is about "the interrelationships among the basic elements within a larger structure that enable them to function together" and "knowledge of principles, generalizations, theories, models, and structures" [4]. Our focus on the conceptual understanding is therefore in contrast to most studies that look into specific algorithms and the students' abilities to apply them.

4.2.2 The Term 'Algorithm'

According to the Merriam-Webster dictionary an *algorithm* is "a step-by-step procedure for solving a problem or accomplishing some end" [59]. This description is quite general and can be interpreted and understood differently. In computer science, the notion of 'computation' plays an important role even in informal definitions, such as, e.g., an algorithm is "a series of elementary computation steps which, if carried out, will produce the desired output" [61] or "an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output" [19]. The difference between the public perception and more mathematical definitions has been pointed out before [36, 60]. Given that the term is increasingly picked up by the media, particularly in the context of AI and marketing, it would not be surprising to find that students' notion of algorithms are strongly informed by media and public discourse rather than education.

For our study we follow Knuth's definition of what an algorithm is, since he managed to give a rather precise definition without referring to computational models such as Turing machines [41]. Paraphrasing Knuth, an algorithm is a sequence of operations for solving a specific type of problem that can be compared to terms like 'recipe' or 'process', but needs to exhibit five major 'features' [41]:

- *Finiteness.* The algorithm must always terminate after a finite number of steps.
- *Definiteness.* Each step of an algorithm must be precisely and unambiguously defined.
- *Input.* Algorithms have zero or more inputs provided before or during execution.
- *Output.* An Algorithm has one or more outputs that are related to the input in a specific way.
- *Effectiveness.* An algorithm's operations should be basic enough for manual execution in a finite time.

The point of 'effectiveness' is that each step in an algorithm must either be an algorithm itself or a simple computational step. The two items 'definiteness' and 'effectiveness' put hard limits to what an individual step of an algorithm might entail, which, of course, boils down the model of computation or notional machine.

4.3 Methodology

4.3.1 Data Collection

The faculty of computer science at our institution runs an outreach programme aimed at local schools. This includes a two-day event that is attended by all 6^{th} grade students of an elementary school (i.e. approx. 150 pupils aged 11–12 years). The two days include

various ‘unplugged’ activities on, e.g., artificial intelligence, sorting algorithms and data encoding. The activities are mostly run by undergraduate students who have received training in working with pupils through the materials, but are given the freedom of teaching according to their own style and preferences. Graduate students additionally oversee these workshops. The first author of this paper is responsible for the training of the tutors as well as the organisation of the workshops.

According to the national school curriculum, the pupils participating in this study should have a basic understanding of the concept of algorithms. This means they should be able to comprehend, execute and formulate independently clear instructions. This is taught once a week as part of a subject that focuses mainly on digital literacy, as a CS subject is not yet provided at this school level.

During the two days we run a competition among the participating pupils, asking for their definition or explanation of what an ‘algorithm’ is. Each entry to the competition was to be written on a paper card (about half a page). The pupils were permitted to write or draw pictures, but were asked to come up with their own explanations. The involved graduate students then chose the ‘best’ answer(s) and handed out a prize to the respective students.

We received an anonymous copy of the students’ answers after the winners had been chosen. That is, the authors of this paper set up the competition and received the compiled list of answers, but were not involved in running the competition itself. Our institution’s review board approved the study.

There were a total of five drawings, which all illustrated parts of the written text (e.g., one drawing showed a number of playing cards where the text mentioned that algorithms are about sorting cards). We therefore ignored the drawings for our analysis.

Of the 70 answers we received, three were empty, one was obviously copied from the internet and eight were verbatim copies of other answers. Removing these duplicates and empty answers left us with 58 distinct answers (see table 4.1).

4.3.2 Analysis

The collected data was analysed qualitatively based on Mayring’s inductive category formation [56]. In this method, the codes and categories emerge during the sifting and analysis of the material.

The two authors of the published paper of this chapter reviewed the material individually over several rounds and discussed any differences until the two authors reached agreement on the coding categories and then classification/tagging of the data. These agreements included re-coding with each other’s categories and finding matches and overlaps. Redundant codes were removed, highly overlapping codes were combined and (sub)categories were created. This also included a translation of the students’ answers to English, followed by a fresh coding of the English translations and a comparison whether

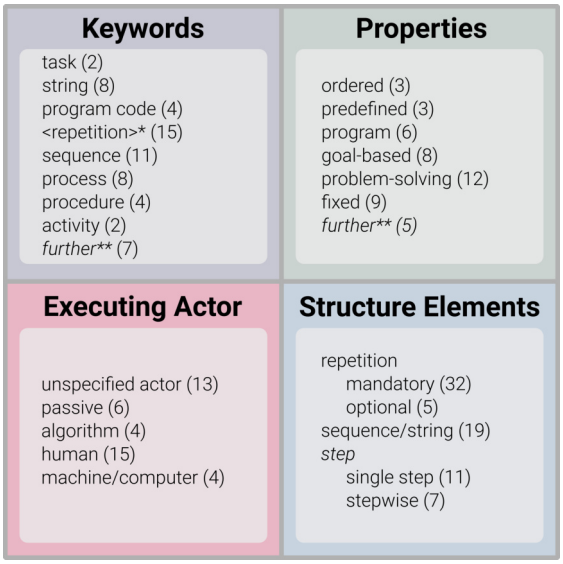

Figure 4.1: Overview of all code occurrences, clustered in four main categories.
* paraphrased / ** codes that occurred only once

the coding remained stable. The final coding system is clustered into four groups, each with several sub-codes and/or subcategories (fig. 4.1):

- *Keywords.* Characterisations that use synonym-like descriptions or mention specific examples to explain the term algorithm, such as, e.g., 'a sequence of instructions'.
- *Properties.* Properties attributed to the algorithm, such as, e.g., 'fixed/predefined' or referring to a 'program'.
- *Executing Actor.* The entity that is executing the algorithm or acting it out, respectively, usually inferred, such as, e.g., 'an activity *you* do' tagged as 'human actor'.
- *Structure elements.* Description of structure such as, e.g., 'mandatory repetition' or 'stepwise execution'.

Note that some codes may appear in more than one category (e.g., 'sequence'), since they both act as a keyword and a structural description, say.

After coding the data material in a descriptive way, we evaluated and interpreted the data by counting occurrences, comparing potential semantic dependencies and looked at keywords.

$\mathcal{A}_1$	Something is called an algorithm if it is a step by step sequence to, e.g., bake something, to solve a problem...
$\mathcal{A}_2$	An algorithm is a sequence of several steps and instructions, which are commonly aimed at a specific problem so as to solve it. Some commands may be repeated.
$\mathcal{A}_3$	An activity that you do time and again each day
$\mathcal{A}_8$	If a certain process regularly repeats itself
$\mathcal{A}_9$	An algorithm is a procedure to solve a problem. Ex.: sorting cards
$\mathcal{A}_{10}$	Executing commands (can be repeated)
$\mathcal{A}_{11}$	An algorithm is also used to solve problems and an algorithm is something that continuously repeats.
$\mathcal{A}_{14}$	A way to, e.g., solve problems, such as shuffling cards by colour or value
$\mathcal{A}_{15}$	A frequently recurring process
$\mathcal{A}_{17}$	Something recurring that repeats time and again. You always do it the same way or a procedure, respectively
$\mathcal{A}_{18}$	A sequence of instructions that are repeated
$\mathcal{A}_{21}$	Predefined procedure that solves problems or tasks one step at a time. Program code that continuously repeats itself
$\mathcal{A}_{22}$	An algorithm is a task that repeats itself
$\mathcal{A}_{26}$	An algorithm explains to a program what it should be doing and how it can do it.
$\mathcal{A}_{27}$	An algorithm is a procedure that solves problems or tasks in a stepwise manner.
$\mathcal{A}_{28}$	An algorithm is a fixed programmed system that is capable of solving specific problems and tasks.
$\mathcal{A}_{29}$	Something that is repeated time and again.
$\mathcal{A}_{30}$	Algorithm is a string of instructions that are executed to solve a task/problem.
$\mathcal{A}_{31}$	If something repeats, it will slowly turn into an algorithm. For instance traffic lights. Another great example is the lift in a tower block. For instance the lift waits at 7am on the ground floor close to the entrance (each day from Monday to Friday) and because this is happening an algorithm will form automatically.
$\mathcal{A}_{33}$	When something repeats at a specific time of the day. For instance if a lift is always used at 7am on the 4th floor, it will learn to wait at 7am on the 4th floor. A specific order, time and again.
$\mathcal{A}_{34}$	A sequence of commands and planned things.
$\mathcal{A}_{35}$	An algorithm is understood to be something that is programmed, a program that repeats
$\mathcal{A}_{36}$	For instance you go to bed every day, this is an algorithm (a repetition). An algorithm solves problems
$\mathcal{A}_{37}$	A recurrence of things in life
$\mathcal{A}_{39}$	A (repeated) sequence of instructions, e.g. go there, do that
$\mathcal{A}_{40}$	Recurring things that we do time and again such as, before going to bed, brushing our teeth and changing clothes. Also when sorting cards you have to figure out your own algorithm to sort the cards as fast as possible
$\mathcal{A}_{42}$	An exactly defined sequence
$\mathcal{A}_{45}$	A continuously recurring event
$\mathcal{A}_{47}$	If on a computer something repeats time and again
$\mathcal{A}_{50}$	A string of different commands that are repeated time and again. Clearly defined tasks or a specific process of tasks, respectively. For instance eating or recipe for cooking
$\mathcal{A}_{53}$	An algorithm is a continously recurring process in daily life, e.g., getting ready, brushing teeth, putting on pyjamas, going to bed
$\mathcal{A}_{55}$	A specific procedure to solve a problem/task
$\mathcal{A}_{56}$	a string of subsequent actions such as in a recipe for cooking. The actions are commonly done time and again and are never changed
$\mathcal{A}_{58}$	Algorithm is if something has an exact order that needs to be followed

Table 4.1: A selection of the answers given by the students

4.4 Results

We provide an overview of selected answers in table 4.1, which are representative of the 58 different answers we received in total. Additionally, we highlight some of the results according to themes that emerged from the coding process in this section.

4.4.1 Descriptions and Properties of Algorithms

Most answers specify the algorithm through a categorising keyword such as, e.g., $\mathcal{A}_2$ "An algorithm is a *sequence of several steps and instructions*, [...]" or $\mathcal{A}_{42}$ "an exactly defined *sequence*" (highlights added). As shown in fig. 4.1, we found 15 answers characterising algorithms as some repeating entity such as in, e.g., $\mathcal{A}_{11}$ "an algorithm [...] is *something that repeats*" (similarly $\mathcal{A}_{29}$), $\mathcal{A}_{36}$ "*a repetition*" and $\mathcal{A}_{37}$ "a *recurrence* of things in life". Another 11 answers characterised algorithms as sequences and eight as strings of something. Some characterisations, however, were more surprising, such as $\mathcal{A}_{22}$ "a *task* that repeats itself" or $\mathcal{A}_{45}$ "a recurring *event*".

Repetition Repetition is perceived as a key characteristic of algorithms with almost two thirds of students explicitly mentioning repetition (37 out of 58). A majority of 32 students said that some form of repetition is a mandatory part of an algorithm whereas five students saw it as an optional possibility.

While the theme of repetition occurs in a majority of the answers given, there is less agreement on what exactly is repeated and in what manner. Compare, for instance, $\mathcal{A}_3$ "an activity that you do time and again each day", $\mathcal{A}_{18}$ "a sequence of instructions that are repeated", and $\mathcal{A}_{37}$ "a recurrence of things in life". In $\mathcal{A}_3$ we find a nested repetition, i.e. an activity that is repeated both during a single day and each day, bringing together the idea of *repetition* (as in $\mathcal{A}_{18}$) and *recurrence* (as in $\mathcal{A}_{37}$). Overall, three answers mention both as in $\mathcal{A}_3$, 23 answers mention repetition and 11 answers mention a recurrence.

With regards to *what* is repeated we find ten answers saying that "something" is repeated, six answers saying that "a process" is repeated and five answers each for "program code" and "commands" or "instructions", respectively. Other answers are even less specific such as, e.g., $\mathcal{A}_{37}$ speaking of "things". An answer that stands out is $\mathcal{A}_2$, which mentions "[...] Some commands may be repeated" as the only example where the repetition does not encompass the whole process or activity. In other words, repetition refers to a *loop* in only a single answer, although two other answers might also have loops in mind ($\mathcal{A}_{10}$ and $\mathcal{A}_{39}$).

The Executing Actor As an 'executing actor' we refer to an instance that executes the algorithm's steps or instructions, if mentioned by the participants. Three quarters of the answers imply that there is an executing entity (42 out of the 56). Of these 42 responses that provide some evidence that the algorithm is 'executed' in some form or performed by something/someone, there were 15 answers that imply human actors, e.g.,

$\mathcal{A}_3$, $\mathcal{A}_{17}$ and $\mathcal{A}_{40}$. Four answers refer to machines or computers as actors of the algorithms (e.g., $\mathcal{A}_{47}$) and six indicate through passive voice that an algorithm is executed but do not indicate any actor (e.g., $\mathcal{A}_{30}$).

Four answers mention the algorithm or procedure itself as an executing actor (e.g., $\mathcal{A}_{21}$, $\mathcal{A}_{27}$ and $\mathcal{A}_{28}$). Answer $\mathcal{A}_{26}$ stands out in that it attributes a remarkable 'cognitive' ability to an algorithm as an entity that "explains to a program what it should be doing and how it can do it".

Finally, 13 answers imply that an algorithm requires some execution but do not mention this explicitly. $\mathcal{A}_2$ is an example for a required action (mentioning 'instructions' and 'problem-solving'), but not explicitly mentioning an actor.

Algorithmic Attributes With regards to RQ2 we especially looked out for attributes that are also found in formal definitions of algorithms, such as finiteness, stepwise execution, etc. Since none of the answers explicitly mentioned finiteness (or determinism), we used the tag 'goal-oriented' as a proxy to indicate that the algorithm would terminate and come to a conclusion.

Seven answers refer to a *stepwise* behaviour of the algorithm.

Eight answers were tagged as *goal-oriented*, as shown for example in answer $\mathcal{A}_{55}$ by the clear determination that a problem must be solved. In contrast, $\mathcal{A}_{14}$ mentions problem solving, but does not presuppose it as a mandatory criterion, since it only speaks of "e.g., solve problems". We therefore did not tag it as goal-oriented, but considered this problem-solving aspect more of an option than the purpose of the algorithm.

Seven answers mentioned the *fixed* nature of the algorithm. For instance, $\mathcal{A}_{55}$ talks of a "specific procedure" whereas $\mathcal{A}_{56}$ says that actions "are never changed". In contrast, $\mathcal{A}_7$ states "an algorithm is a program code than can be repeated forever. They continuously improve", explicitly contradicting the idea of a fixed procedure.

4.4.2 Origins and Purpose of Algorithms

Relatively few answers gave hints as to where algorithms come from or their specific purpose. However, we consider the views expressed by the pupils interesting enough to point them out in this section.

The Aspect of Solving Problems and Tasks Twelve answers specify the purpose of algorithms as problem-solving. One answer $\mathcal{A}_2$ says that an algorithm is commonly used to solve a "specific problem", whereas six more answers say that an algorithm solves "a problem" and five say that an algorithm solves "problems". For instance, $\mathcal{A}_{55}$ says "a specific procedure to solve a problem/task". In contrast, $\mathcal{A}_{11}$ says "an algorithm is used to solve problems [...]" and $\mathcal{A}_{27}$ says "an algorithm is a procedure that solves problems or tasks in a stepwise manner".

Of those five answers referring to algorithms as solving "problems", three also mentioned the requirement for repetition. Answer $\mathcal{A}_2$ mentions that some instructions might be repeated, whereas all other answers that mention the problem-solving aspect do not mention repetition at all.

Formation of an Algorithm $\mathcal{A}_{31}$ is remarkable in that this answer describes the genesis of how an algorithm forms: "if something repeats, it will slowly turn into an algorithm. [...] an algorithm will form automatically." A similar notion is expressed by eight other answers, including $\mathcal{A}_8$, $\mathcal{A}_{47}$ and $\mathcal{A}_{20}$ "if a specific process repeats itself regularly" or $\mathcal{A}_{15}$ "a frequently recurring process". An algorithm is therefore not something that is actively designed, but either forms like a tradition because of repetition or is a synonym for a repeated process, event or tradition. Another possible explanation could be recommendation algorithms, which students encounter on a daily basis, like in online shops or streaming platforms.

Compare this with $\mathcal{A}_{21}$ and $\mathcal{A}_{42}$ whose mentioning of '(pre) defined' indicate an intentional design. $\mathcal{A}_{50}$ seems at first to also lean towards a formation by repetition, but then speaks of "clearly defined tasks". Given the two examples "eating" and "recipe" it is not clear whether this answer can be clearly coded as either one or the other; in fact, we assume that the student accepts both possibilities as legitimate process of how algorithms form.

4.5 Discussion

4.5.1 The Role of Repetition

The prominence of repetition in the students' answers is remarkable. Even more so considering that, except for one case, all answers referred to a (necessary) repetition or recurrence of the algorithm itself rather than a looping structure within the algorithm. So, where does this come from?

We hypothesise that the instructors' original messages were emphasising that algorithms are fixed and static entities. That is, each time you execute an algorithm, you follow the same instructions in the same order. $\mathcal{A}_{50}$, for instance, starts with the concept of repetition, followed by a clarification "clearly defined tasks or a specific process of tasks" and the canonical example of a recipe for cooking. Likewise, $\mathcal{A}_{33}$ says "a specific order, time and again" and $\mathcal{A}_{17}$ clarifies the repetition by "you always do it the same way".

It is interesting to observe that the 13 answers directly mentioning the fixed or predefined nature of algorithms are dwarfed in number by the 37 answers referring to repetition. We could probably classify at least 31 of these answers as misconceptions in that they focused on the wrong aspect of the explanation, story or examples given. The explanations given by the tutors, and the analogues and metaphors employed in particular, clearly need to be revised as they seem unfit to solicit a correct understanding. The tutors mainly used the same recipe analogies, but were free to add more details or additional explanations in their teaching.

4.5.2 The Executing Actor

Concerning the 'executing actor' of the algorithm, there is some disagreement and insecurity among the students. For instance, $\mathcal{A}_1$ refers to problem-solving and baking, but the answer does not provide any information about *who* actually does the problem solving or the baking and executes all the steps mentioned. In contrast, $\mathcal{A}_{40}$ describes a human actor using the term "we" and $\mathcal{A}_{56}$ uses passive voice for explaining an execution without mentioning an actor.

Strongly related to the question of the actor is the nature of the algorithm itself. $\mathcal{A}_{21}$, $\mathcal{A}_{27}$ and $\mathcal{A}_{28}$ speak of the algorithm as an entity that (actively) solves problems. $\mathcal{A}_{26}$ even goes so far to state that an algorithm "explains to a program what it should be doing and how it can do it", very clearly expressing not only actorship, but also higher cognitive functioning. In stark contrast, $\mathcal{A}_{58}$ sees an algorithm much more as a recipe to be followed, i.e. where the algorithm itself has no active role at all. The bulk of answers, however, are much more vague in whether algorithms are seen as entities that themselves perform actions or as passive instructions to be followed and executed by a distinct actor.

Interestingly, merely four answers explicitly spoke of a computer or machine being directly involved (e.g., $\mathcal{A}_{47}$) and six indicated a 'program' or 'program code' (e.g., $\mathcal{A}_{21}$). This is somewhat surprising but hints at some success in teaching computational thinking not as necessarily machine-based.

4.5.3 Defining Properties of Algorithms

Comparing with formal definitions of algorithms, we find a mixed bag. While a considerable proportion of the pupils described algorithms as sequences, processes, procedures or program code of some kind, other aspects such as finiteness seem to have been entirely neglected. Almost a third of the students explicitly mentioned the stepwise nature of algorithms with others implicitly indicating it through examples or keywords such as "a *string* of commands". Hence, the notion of a step-by-step procedure seems to have been fairly well understood by a majority.

Finiteness Nothing about 'finiteness' can be found in the given answers of the students. In none of the answers is it ever explicitly mentioned that an algorithm has to terminate.

What we can find are eight students who attribute the goal of solving a problem to algorithms. One interpretation is that the algorithm has completed its task—and therefore terminates—after this goal has been reached, i.e. the problem has been solved. The mentioning of endless repetition, as for example in $\mathcal{A}_{29}$ "something that is repeated time and again" speaks rather of an opposite understanding. Here the repetition was put so strongly into the foreground that the most important criterion, the termination of the algorithm, got completely lost.

One possible explanation could be a confusion between computation and algorithm with the distinction between the two lying in the termination of the latter. While it is unlikely

that the pupils are aware of these two concepts, we would argue that most modern computer applications have a never-ending character with a continuous query-reply-cycle. This is particularly true for applications such as chat and social media applications as well as internet search engines. At the same time, the term 'algorithm' is widely used in the media and public discourse.

Definiteness and Effectiveness Some answers explicitly mention definiteness, such as $\mathcal{A}_{42}$ and $\mathcal{A}_{50}$. Much more common, however, was the theme of a fixed order or fixed sequence.

The notion of a (computational) step that is *not* precisely defined might be rather alien to pupils of that age. If we assume that the individual steps of an algorithm are 'obviously clear' then any uncertainty as to the execution of the algorithm would necessarily come from a change of the sequence of steps (either a rearrangement or the addition/removal of certain steps). The pupils' emphasis that the sequence itself is fixed could either indicate that an algorithm as such cannot be modified, or it means that no steps can be skipped or executed out-of-order. We argue that the latter interpretation would be closely related to definiteness.

As mentioned above, definiteness and effectiveness strongly relate to the idea of a computational model, which is hardly ever taught explicitly in the context of algorithms. However, as elaborated in our discussion of the executing actor above, there are some vague and implicit ideas about an underlying computational model expressed through the executing actor. Hence, even though no answer actually speaks of effectiveness as such, we would argue that some of it is still encoded in the answers given by the pupils.

Input and Output Nothing about I/O is mentioned by any of the answers. Moreover, there is virtually no indication of algorithms as entities that process or work on data. Algorithms seem to either be processes embedded in (and interacting with) 'daily life' or procedures to solve a problem with no explicit interaction mentioned.

4.6 Research Questions

4.6.1 RQ1: How do upper elementary school students describe and explain what an algorithm is?

The student responses are dominated by somewhat unspecific and nebulous 'salient properties', but also very concrete examples. There are a number of variations on the theme of repetition as a defining property, but whether that repetition pertains to a continuous loop, the scheduled execution at specific times or just means that an algorithm can be 'reused' is not quite as clear.

We perceive a potential threat to the students' comprehension stemming from the examples mentioned by the students. While 'brushing your teeth' is surely meant to illustrate the idea of following a specific routine, the pupils seem to rather pick up the

idea of doing something over and over again. In other cases, the pupils even deduced that algorithms (automatically) emerge out of a specific habit or recurrence. We would therefore caution against leaning too heavily on 'real-life' examples when teaching the concept.

When considering responses from novices, we have to be mindful that they might lack the vocabulary and understanding needed to even formulate precise questions. However, the answers we collected contain a number of keywords and properties which suggest that the students were able to express their ideas well enough to take the responses as actual reflections of their comprehension.

4.6.2 RQ2: In what aspects do students' conceptions of what an algorithm is differ from established definitions?

In general, students seem to have understood that algorithms are step-by-step procedures to be executed or followed. With some indication of a computational model as expressed through the executing actor, the explanations partly match the definiteness/effectiveness aspect of algorithms. However, the students' answers do not mention the need for an unambiguous and precise language or instructions, although the notion of a fixed sequence carries some of that characteristic. Moreover, instead of a guaranteed termination (aspect of finiteness), we often find infinite repetition to be brought forward as a key feature.

To arrive at better explanations or definitions we see two main attributes missing that should be more emphasised: the purpose of an algorithm as computing a 'result' (and thus necessarily terminating) as well as the idea of an underlying computational model.

4.7 Limitations and Future Work

Our study clearly has an explorative character with a relatively small sample size. Moreover, since all pupils attended the same school, there is a high interdependence of the collected answers, further exacerbated by the shared tutors, some of whom might have discussed the idea of what an algorithm is in some detail and with various examples. We should also expect that students had some discussions among each other, further putting the independence of the collected answers into question. The provided results are therefore not necessarily representative of a larger student population.

Due to the voluntary and competitive nature of the survey, students who already had some initial idea of what an algorithm is likely participated with greater enthusiasm. Thus, the data set is probably missing answers from those pupils who had no working notion about the term algorithm at all. Receiving responses from all children holds potential for a future follow-up study. However, considering the limitations of children in expressing their comprehension of the concept of the term algorithm in a short written statement, a test or questionnaire to quantify their occurrence of the identified misconception could give us even more insights. Exploring and comparing algorithmic understanding across different educational levels is also a topic for future work.

Probably the greatest issue is that the pupils may not have the language to correctly express salient features of the algorithm-concept. As indicated above, the 'fixed sequence'-property of algorithms might actually refer to the idea of determiniteness. Follow-up studies will have to seek to better differentiate what the pupils mean.

4.8 Conclusion

It is imperative that we not only teach procedural knowledge and an intuitive understanding of (specific) algorithms, but also discuss the general concept of algorithms, their possibilities and limitations. As a first step towards establishing such a discussion of algorithms in general education, we have looked at how 6^{th} grade pupils describe the concept of an algorithm.

Our data shows a somewhat hazy notion that seems to be primarily based on the concept of repetition. However, we also found frequent mentioning of properties such as step-wise execution or the aim of solving problems. This indicates that the pupils have indeed developed a notion of the concept of algorithms, but that we need to improve the instruction and teaching towards working out the key properties more clearly.

CHAPTER 5

Intuitive Problem-Solving Strategies

A shortened version of this chapter is planned to be published.

5.1 Introduction

When students work on computing problems, they often rely on ideas and strategies based on intuition rather than a formal algorithm [95]. These initial strategies are shaped by experience, trial and error, and sometimes by observation. Especially in early Computer Science (CS) learning, such strategies emerge spontaneously while exploring e.g. CS Unplugged activities. They may be taken for granted or overlooked, yet they are important *thought processes* that offer valuable insights into students' intuitive understanding of computational concepts.

Understanding these intuitive strategies matters for educators, as they can connect new ideas that need to be taught with these intuitive strategies and existing experiences. Empirical studies show that novice students often bypass formal problem-solving processes in favour of spontaneous, experience-driven approaches when faced with complex coding tasks [95]. Recognising such behaviour is crucial for designing instructional strategies that shape rather than override learners' intuitive approaches. It allows educators to build on what learners already know and do, and further helps them connect informal behaviours, everyday logic, and their experiences with the core concepts of CS. These strategies reflect students' prior knowledge and emergent problem-solving structures. From another perspective, building on these learner-generated strategies supports meaningful engagement with algorithmic thinking [68, 66].

A learning intervention employing the CS unplugged method, utilising hands-on activities that do not involve computers, provides a promising opportunity to observe students'

problem-solving strategies. When students work collaboratively on specific problems, their approaches become observable and can be discussed. Although CS-Unplugged is widely used in outreach and early education, our understanding of the algorithmic thinking that students demonstrate when independently engaging with these tasks remains limited.

This study dives into students' intuitive approaches to solving a collaborative CS-Unplugged sorting task. We highlight the algorithmic strategies that emerge during a collaborative group task and their revision through repeated attempts. The complexity of the collaborative sorting task is designed to spark conversation, delegate roles, and navigate challenges such as limited memory and structural constraints. Based on video analysis of six groups of students (aged 10-13) working on the sorting task, we identified patterns of action that resemble formal algorithmic approaches. These include selection-, insertion-, or merge-like behaviours, similar to often taught sorting algorithms in CS, despite the absence of any prior instruction on sorting algorithms. We also observed more general but also CS-related concepts such as trial-and-error, optimisation, and parallelisation.

This chapter is guided by two research questions:

- **RQ3:** What intuitive algorithmic strategies and sorting behaviours do K-12 students display when engaging in a collaborative unplugged task?
- **RQ4:** How do students revise or refine these strategies based on interaction and group feedback?

5.2 Background and Conceptual Framing

In this study, we use three interrelated terms to characterise how students tackle a sorting task:

- **Problem-Solving strategies** refer to the observable methods employed by students while seeking a solution, such as engaging in trial-and-error experimentation, devising plans, optimising, etc.
- **Algorithmic strategies** involve systematic and consistent problem-solving methods based on procedural logic, such as persistently selecting the smallest item or merging sorted lists. Despite resembling known algorithms (e.g., selection sort), they emerge naturally without prior instruction.
- **Computer science concepts** encompass formal ideas such as sequencing, conditionals, iteration, data structures, and decomposition. While these aren't directly taught to our students, we assess how their intuitive strategies inherently integrate these principles.

This vocabulary enables us to describe learners' thought processes without imposing formal labels, facilitating an exploration of how intuitive strategies are linked to foundational CS education ideas.

Before presenting our analysis, this section outlines the theoretical framework and background that guide our work. We start by looking at how computational thinking (CT) and problem-solving are conceptualised in the literature, then focus on the educational relevance of sorting tasks, and finally discuss the role of unplugged approaches in this context.

Problem-Solving in CS Education. A key aspect of an algorithm is that it solves a problem [40]. Children are aware of this [48], although they often struggle to articulate this concept formally. Computer Science (CS) fundamentally focusses on systematic problem-solving, whether through classic algorithms or AI advancements, driving its knowledge, goals, and importance in education. This core concept is evident in the Computational Thinking (CT) paradigm, a key concept in CS education. CT encompasses skills for solving problems methodologically. Selby and Woollard [75] highlight skills like abstraction, decomposition, generalisation, evaluation, and algorithmic thinking. Many researchers also emphasise problem-solving as key to CT [31, 39, 66, 94]. These skills are important not only in Computer Science (CS) but also for everyday tasks, as well as our card sorting activity in this study. Sorting, comparing, and optimising are everyday-live activities, closely related to formal CS concepts. Students still find it challenging to apply them strategically. A previous study has shown that students already bring initiated ideas to sorting problems [80]. Simon et al. analysed students' explanations of how to sort a set of numbers before they took the CS1 course. Their research shows that most of them are able to explain an algorithm that works, but they will not explain the concept of pairwise comparisons. They also found that students see the numbers as strings and not digits and therefore sort a set of numbers by their length rather than their numeric value. Our research idea is close to their research, but we focus on K-12 students and their emerging intuitive strategies, not their explanations how they would sort. Furthermore, the study by Simon et al. was already published in 2006. Back then it was assumed that students had no prior experience in CS, especially sorting, which might be different today. Therefore we need to look at younger students.

Cognitive Load and Instructional Design. Learning computer science encompasses both developing problem-solving skills and understanding formal concepts like programming, requiring conceptual understanding and efficient cognitive load management. Sweller's Cognitive Load Theory explains that learning becomes more challenging when processing multiple elements together. When cognitive load exceeds a learner's capacity, performance and retention decline. Yet, excessive cognitive load alone does not fully explain students' challenges [82, 83]. Similar to our approach, in programming education many students start with enthusiasm but soon lose motivation when faced by complex syntax and abstract logic. Programming courses often lead to frustration and anxiety when learners struggle with the language's structure [17]. This shows that

excessive abstraction can quickly create barriers to understanding. In both contexts, such difficulties can be mitigated by providing structured guidance focused on computational thinking (CT) and by building on what learners already know. Prior experiences, for instance, influence how students deal with cognitive challenges: those with earlier exposure to related concepts often show higher self-efficacy, while newcomers are more likely to disengage [2]. Fear or frustration may therefore rise early in the learning process, especially when tasks are unnecessarily complex [88].

In our study, we avoid such complexity by focusing on the algorithmic aspects of problem-solving without involving syntax or programming. This allows us to observe how intuitive strategies emerge and develop in a sorting task. Understanding these intuitive approaches helps design learning environments that manage the cognitive load and connect new ideas to existing problem-solving strategies of students.

Sorting as a Context for Algorithmic Reasoning. The sorting task provides an opportunity to examine how students develop and adjust problem-solving strategies. Sorting is a fundamental algorithmic problem in CS, with well-known approaches such as selection sort, insertion sort, and merge sort that are commonly taught both in university contexts [44] and in K-12 computing education [10].

At the same time, everyday experiences, such as arranging objects by size or grouping similar items, play a role in how students intuitively structure these tasks.

Therefore, several unplugged activities around sorting exist to explore the sorting concept, most prominently sorting networks or step-based games as in CS Unplugged [10]. Silapachote et al. proposed a structured card game where learners deliberately practice pairwise comparisons and algorithmic steps until they rediscover known sorting algorithms [79]. In contrast, our study does not provide learners with rules but observes the strategies they spontaneously develop when facing the task.

Observations from classroom settings indicate that students often associate algorithms with repetitive everyday routines, rather than structured problem-solving strategies Chapter 4. These observations support the assumption that many students hold an intuitive understanding of algorithmic processes, even if they struggle to express it verbally. Sorting tasks provide a concrete setting to observe such thinking in action and to explore how learners might be supported in developing more structured problem-solving strategies.

Using a card sorting activity as an instrument to measure learners' understanding has recently been explored in teacher education. Allsop et al. [3] introduced the "Match it!" card sorting activity to evaluate future teachers' knowledge of CT such as algorithmic thinking, decomposition, or abstraction. Their findings showed that while the activity was engaging and useful for reflection, it primarily measured the recognition of concepts rather than their application, and some of the cards were perceived as ambiguous.

In contrast, in our setting we treat the sorting task as a tool to make problem-solving and algorithmic strategies visible, and further observable for our research. It becomes

an activity that enables students to use comparison, control flow, memory management, iteration, parallelisation, and optimisation. Through the analysis of these emergent strategies, our aim is to connect intuitive problem-solving strategies, and further algorithmic elements, to core CS concepts. In this context, CS unplugged activities [10], which are tasks designed to teach computing concepts through hands-on activities without computers offer a promising way to externalise learners' reasoning and reduce extraneous load.

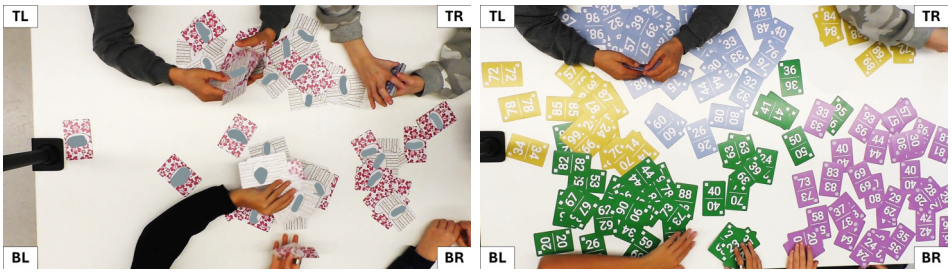

Figure 5.1: Display of the card game. Study setup: four participants (abbreviated as their positions on the video from a birds' eye perspective.: top left (TL), top right (TR), bottom left (BL), bottom right (BR). Backs of the cards visible (left) and frontsides with numbers (right).

Taken together, these perspectives frame our study as an attempt to make learners' intuitive algorithmic reasoning visible and with that observable and interpretable. Rather than measuring performance, we seek to understand students' intuitive strategies, which could inform the design of learning activities that connect intuition, abstraction, and Computational Thinking (CT). In this sense, our study contributes to a broader vision of CS education that values exploration as an essential component of algorithmic understanding.

5.3 Developing the Card-Sorting Activity

This unplugged card-sorting task serves as both a learning setting and a research tool, designed to make students' algorithmic strategies visible without prior instruction. The main task for the students in this activity is to restore a shuffled deck of cards to a predefined order. The students are instructed to do so collaboratively in small groups of up to four students. The intention is to bring together different computational aspects that can be addressed in a discussion afterwards or to build on in future lessons. These concepts include sorting, parallelisation (task stealing, bottlenecks, etc.), optimisation, and data structures such as arrays or stacks. Each card in the game has a specific colour on the back and a colour with a number on the front. In the sorted deck, the cards should be sorted as follows (see Figure 2.4):

1. The cards are first grouped by the colour of their back.

2. Within each group of a back colour, the cards are further grouped by their front colour.

3. The numbers within each front-colour group are arranged in ascending order.

Originally, this task was carried out using cards from an existing card game, consisting of four colours on the back, each building a stack containing four colours on the front, and numbers ranging from 1 to 10 for a total of 160 unique cards. The four stacks of cards, grouped by back colour, are shown in Figure 2.4 on the left. Three stacks are displayed with the back colour facing upwards, while one stack is folded to also reveal the corresponding front colours. However, sorting was almost trivial due to the low, consecutive numbers, strategies were not visible. To increase complexity and also to be able to observe their sorting strategies, we designed a custom card deck (Figure 5.1). This custom deck consists of 160 cards with two different colours on the back. Each back colour has four frontside colours and each contains 20 random numbers between 10 and 99. Although no number appears twice within a single front-back colour combination, duplicates can exist across different colour groups. The selection of numbers does not follow a regular pattern.

This design aimed to encourage participants to develop more complex sorting strategies, as the larger number of cards per colour and irregular number gaps could make simple search-based strategies less feasible. Furthermore, managing 20 cards requires considering table space and memory strategies, as cards need to be placed strategically to allow quick access.

The card-sorting task is part of a workshop on algorithms performed at our institution's outreach programme. Participants are divided into small groups of approximately four people and seated around a table.

The workshop session begins with a short plenum discussion as an intoducation. The students are asked what they know about CS and the concept of algorithms, which is described as step-by-step instructions. Next, the card deck is introduced, and the task is motivated by demonstrating that finding a specific card is much easier in a sorted deck than in an unsorted one.

Participants are handed the card deck and asked to shuffle it. They are then tasked with sorting the cards and restoring the original order as described earlier. Once all groups have completed the task, we record the completion time for each individual group, and each group is asked to discuss how they could improve their approaches. Afterwards, a second sorting round follows, where participants can apply improvements to their strategies. Finally, different approaches are discussed in the plenary session.

We recorded the participants of six groups on video during these card-sorting activities, to subsequently analyse their strategies, discussions and interactions. Throughout the task, instructors are available to answer questions but are tasked not to intervene unless participants completely misunderstand the task.

Our study employs a qualitative research approach using an iterative-reflective and inductive coding process to identify intuitive problem-solving strategies from video data. This approach allows us to naturally observe students' actions during the card-sorting task without interrupting them. Identifying intuitive actions can be challenging with traditional qualitative methods, such as interviews, which are usually conducted after the intervention. Using video and audio analysis, we can precisely examine what the participants did at each stage of the task and how their interactions unfolded within the group.

The nature of our video data differs from typical video analyses in psychology, as our focus is not on the children's facial expressions or gestures but rather on their actions and computational processes while solving the task. To best capture these aspects, we recorded from a bird's eye perspective above the table where the children were working on the task (Figure 5.1). This perspective provides an optimal view of the problem-solving process, allowing us to closely examine the computational strategies employed by the participants. We also recorded audio tracks for each group.

To structure our analysis, we combined *thematic analysis (TA)* [13], *qualitative content analysis (QCA)* [56] and the *documentary method (DM)* [12], leveraging the strengths of each method for the qualitative video analysis. TA provides the general framework for identifying patterns in students' problem-solving strategies, allowing us to examine emerging approaches without predefined categories. However, TA alone does not offer a way to ensure structured comparability between cases. To address this, we incorporated descriptive coding elements from QCA, which helped structure the material while maintaining an inductive approach. Additionally, to ensure coding consistency, we conducted intercoder agreement checks, aligning our coding approach among multiple researchers. We further applied DM to guide the interpretation of the coded sequences. This approach distinguishes between descriptive coding, which systematically categorises actions (e.g., participant question, tutor interaction), and interpretative analysis, which focusses on identifying underlying problem-solving patterns and meanings.

In the first coding round, we initially focused on the overarching categories, ensuring that key elements of student interactions were systematically captured. The subsequent thematic differentiation emerged through an iterative process, refining codes and recognising patterns in the data. Although the coding process primarily aimed at structuring the material, the interpretation of how these codes relate to problem-solving strategies is discussed in the results and discussion sections.

This methodological approach ensured a balance between structured coding for comparability and flexibility to capture emerging strategies, allowing for a nuanced analysis of students' problem-solving and collaborative interactions.

5.3.1 Participants and Data Collection

Before participation, all children and their legal guardians were informed about the study and provided their written consent. The study and the consent form were reviewed and

approved by the ethics committee of our institution. With the bird's-eye perspective of the video recordings (see Figure 5.1), we also ensured the privacy and anonymity of all participating children.

The study was carried out with students aged 10 to 13, who participated with their classes in our workshop. Recruitment was carried out by publishing study information and registration options on our website and using an email newsletter. Any school and class in this age group was eligible to register for the workshops as long as they had not participated in one of our algorithm-themed workshops before. Participating in the study was voluntary and while we only recorded the students who provided the consent form, everyone else was equally able to participate in the workshop. For this purpose, we divided the class into two smaller groups and held the same workshop in two separate rooms: one was recorded on video, the other was not.

As part of this study, we conducted six workshops with one school class each. Class sizes ranged from 22 to 26 students; however, we recorded three groups of four students each per participating class due to the separation into two large groups. In total, we recorded 18 videos, three videos in each school class participating in the study, each recording a group of four students, with a total of 72 participants recorded. We chose six recordings for an in-depth analysis, one from each of the six school classes visiting to have a diverse sample. We previewed the videos and chose one video from each school class that was the most different from the other videos recorded covering a wide range of exploratory intuitive strategies (Table 5.1).

Video ID	School Year	Recording Duration	Sorting Time 1	Sorting Time 2	Improvement(%)	Gender (F/M)
1	5	50:24,5	15:51,4	06:54,3	56.4%	1/3
2	5	41:51,4	14:46,1	05:15,0	64.4%	0/4
3	5	40:55,7	13:53,6	07:21,3	47.1%	0/4
4	8	26:57,6	06:11,0	04:29,2	27.5%	2/2
5	8	27:10,5	06:56,4	04:17,2	38.2%	1/3
6	7	24:03,3	08:56,8	04:10,0	53.3%	1/3
Total		**3h 31m**	**1h 6m**	**32m**	**Overall: 51.0%**	

Table 5.1: Overview of recorded video data, including the students' school year, which indicates the school year of the participating group, total duration and improvement, which represents the relative reduction in completion time from the first to the second sorting round. The recorded duration includes the discussions between the two sorting rounds.

5.3.2 Video Analysis

Three independent researchers contributed to the coding process. After each round, the findings were discussed to refine the structure codes. Afterwards, we identified

the emerging themes and strategies analysing code relations and patterns. We created descriptive paraphrases for each participant's individual sorting strategy during the sorting rounds and discussion sequences. For this purpose we used abbreviations *TL (Top Left)*, *BL (Bottom Left)*, *TR (Top Right)* and *BR (Bottom Right)* to identify the four individual students in our comments and paraphrases. These abbreviations made it easier to distinguish between each student, using their position around the table in the video (Figure 5.1). Additionally, we add a number to this abbreviation to indicate the group they were in, e.g. TL4. The analysis proceeded as follows:

1. **Data Preparation:** Since our video data consists of six video recordings, each capturing a group of four participants (Figure 5.1) solving the card sorting task described in Section 5.3, we split each video into the six task stages "Instruction 1", "Sorting Round 1", "Instruction 2", "Discussion", "Sorting Round 2" and "Closing".
2. **Iterative Coding Process to create descriptive codes:**
   This process was carried out in six rounds over eight weeks.
   a) **Individual Coding:** Each of the three researchers independently analysed four of the six videos so every video was coded by two researchers independently.
   b) **Reflection:** Joint discussion and reflection to refine codes, modify codes, and discuss the results so far. After each round, each researcher reworked/recoded the assigned videos again, including the changes they agreed on.
3. **Interpretation:** Identification, discussion, merging and categorisation of themes and topics related to the students' strategy development.
4. **Produce a list of strategies:** Compilation of identified strategies, their development pathways, and illustrative examples.

5.3.3 Coding Scheme and Intercoder Reliability

To answer our research questions, we actively searched for code patterns that signal uncertainty (e.g. "participants' questions") or learning moments (e.g. "strategy revision"), as well as visible or verbalised computational processes (e.g. "computational process visible").

We intentionally used neutral codes, to be able to look at the different code patterns afterwards and to draw connections to the emerged themes and strategies used. Over eight weeks, we developed a coding scheme and grouped them into five supercodes 5.3, consisting of 19 subcodes.

The intercoder-reliability was assessed by comparing each coded segment and calculating the percentage of the temporal overlap. Each video was coded by two of the three researchers. The pairings were carried out in such a way that each researcher coded the video with another researcher the same number of times ($2 \times AB$, $2 \times BC$, $2 \times AC$). The 50% threshold was chosen based on preliminary trials that showed smaller overlaps

Video	Coder Pair	Brennan & Prediger's κ
1	A - B	0.70
2	A - C	0.71
3	B - C	0.78
4	C - A	0.69
5	B - A	0.71
6	C - B	0.68

Table 5.2: Intercoder reliability per video measured using Brennan & Prediger's κ.

(e.g., 20–30%) often reflected different events, while larger thresholds (e.g., 70–80%) underestimated agreement due to minimal temporal shifts. A 50% overlap therefore represented a reasonable balance for identifying the same situation across coders. This threshold ensures that coders recognised the same situation, even if the exact timing differed slightly. For example, if one researcher marked a participant question a fraction of a second earlier or later than the other, both still identified the same episode, although their segments would not fully overlap (Table 5.2). In some cases the codes were extremely short and even the 50% threshold was not enough, even though the two researchers noticed the same situation. Therefore, we aim for a Brennan & Prediger's $\kappa > 0.65$, which is a good value considering the challenges of extremely short segments.

5.4 Results

In this section, we present the findings of our video analysis, focussing on the intuitive algorithmic strategies the students used (RQ1) and how these strategies evolved throughout the sorting task (RQ2). The results are structured in two main dimensions: first, we describe the initial group and individual strategies observed during the first round of sorting. Then, we analyse changes in approach and revised strategies that appeared during or after discussion phases. We conclude by highlighting selected scenarios that illustrate key aspects of the students' CT.

To support our analysis, we grouped our coded segments into thematic categories that reflect relevant dimensions of the students' problem-solving processes that we observed. Rather than focussing on frequency or isolated segments, we used these categories to find patterns and development across both individual and collaborative behaviour to be able to browse the video material fast and efficiently, to answer our research questions. Table 5.3 gives an overview of the main categories used in our interpretation.

The scheme includes descriptive codes grouped into five thematic supercodes: (1) tutor interventions, (2) error, mistakes, and error manipulation, (3) individual cognitive strategies, (4) group collaboration, and (5) visible sorting behaviours. Although code frequencies are not the focus of this paper, they helped identify key segments and patterns for qualitative interpretation to find the emergent intuitive strategies.

Code Category	Purpose and Description in the Analysis
Tutor Interventions	All tutor actions during the task. Includes procedural reminders, reactions to student input, and subtle hints. Used to understand how external guidance influenced group behaviour and the emerged strategies.
Error, Mistakes and Error Management	Incorrect or unintended actions by students and how they handled these mistakes, individually or as a group. Helped identify moments of confusion or challenges in understanding the task and indicates awareness, control mechanisms, or learning from errors.
Participants' thoughts and computational processes	Moments revealing inner reasoning, such as questions, visible steps (e.g. laying out cards using a structured system), or self-talk. Signals emerging strategies or reflections. Also planning or adjusting strategies, either in discussion or during sorting. Central to analysing how approaches evolved (RQ2).
Social and Group Collaboration	Task organisation within the group. Includes assignments, helping behaviour, and minor conflicts. Used to trace group dynamics.
Sorting Behaviour	Describes observable sorting patterns similar to known algorithms (e.g., selection, insertion), and data structures used (e.g. stacks or arrays). Supports interpretation of intuitive algorithmic thinking.

Table 5.3: Super-code categories used to browse and interpret the video data to answer the research questions during the sorting task.

5.4.1 Initial Strategies

All six groups engaged in the sorting task by applying a divide-and-conquer strategy. The card set design, which includes two different back colours and four frontside colours per back, lends itself to task splitting and role distribution. While this general principle appeared across all groups, the ways in which the students implemented it varied from a dynamical approach during the sorting rounds to a fully prepared approach, where they assigned tasks before starting the sorting.

Task Distribution and Role Assignment

Each group started by informally dividing the task into subtasks. In four of the six groups, a brief discussion occurred before sorting round one. These discussions were usually initiated by a participant asking what to do, or by someone making a proposal (e.g., "I'll take blue," or "Everyone takes a stack and sorts it"). In three cases, the tutor

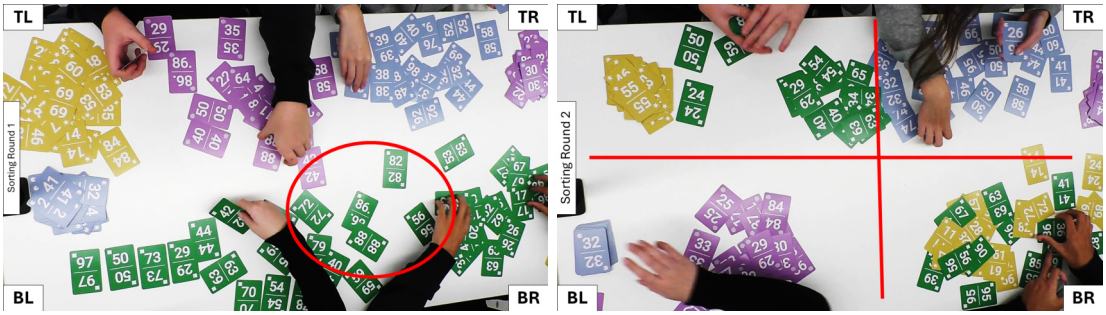

Figure 5.2: Group 3: Comparison of sorting round 1 (left) vs sorting round 2 (right). The red circle highlights the area where an error occured. In the second round this group adapted their strategy to avoid this mistake by using visible separation of the cards on the table.

explicitly reminded the group to think about a strategy and discuss how to organise themselves.

The most common form of task distribution was colour-based: either each participant took responsibility for a particular front colour, or groups divided the deck by the back colour and further split subtasks among pairs. For example, in Group 1, participants first split the deck into four piles according to the number of participants, two stacks for each back colour. Then each student took a pile and further divided it into front-side colours. From there, they individually worked to sort numbers within their assigned colours. Some groups did this tacitly, while others explicitly coordinated the responsibilities.

In Group 3, the *divide and conquer* approach was also guided by seating position: diagonally opposite students formed sub-teams to handle one back colour each (Figure 5.2. Within these subgroups, cards were laid out visibly between partners, and verbal coordination was minimal but effective. This spontaneous team formation shows how spatial positioning influenced collaboration.

Interestingly, even when no explicit strategy was discussed before the first round, participants quickly began to act in ways that indicated shared assumptions about how the task should be handled. For example, two groups (1 and 2) exhibited a pattern in which the first student began laying out cards by front-side colour, and the others imitated this behaviour.

Although most of the groups opted for individual parallel work within a divided structure, the degree of interdependence varied. In some cases, participants occasionally reached over to help a peer, handed over a card, or verified a pile. In others, they stayed focused entirely on their own subset, even when they were visibly idle.

Observed Sorting Strategies

Across all six groups, a range of intuitive sorting strategies was observed. These varied not only between groups, but also within groups and across the two sorting rounds.

Despite no prior instruction on sorting algorithms, many participants employed intuitive algorithmic strategies that resembled established algorithmic patterns, such as selection or insertion sort (all groups), but also elements of bucket sort (Group 2 and 3) and merge sort (Group 3 and 6).

An approach commonly observed was a version of *selection sort*, where participants placed all cards of a certain frontside colour on the table and repeatedly picked the smallest/largest remaining card to place it on a stack. Before doing so, they often prepared the unsorted cards into structures resembling arrays or loose piles, either in-hand or on the table. This prestructuring facilitated the subsequent search and placement process. It reflects an intuitive separation of "working memory" and "output structure", conceptually close to the use of temporary storage in formal sorting algorithms. This was typically done in silence or accompanied by verbalised self-instructions (e.g., "Where is 23?").

Other participants used behaviours reminiscent of *insertion sort*. They held a stack in their hands and inserted the incoming cards into the correct position. This was especially common when fewer cards were involved or when participants were focused on a single colour. The hand-based stack was often sorted "on the fly" indicating a high level of familiarity or confidence. In some cases, students laid out cards one-by-one in a horizontal row, adjusting positions during the process , a behaviour that aligns with list-based or array-based insertions.

Merge sort-like behaviour was less common but observable in later stages (Group 3 and 6), particularly when participants combined their individually sorted stacks. In one case, this merge was not planned but became necessary due to a sorting mistake, leading the students to align two already-sorted sequences and interleave them systematically. This indicates not only error handling, but also the emergent application of comparison-based merging.

Participants frequently switched between sorting strategies, even within the same task. For example, one student (TL, Group 5) began with insertion into a hand-held stack, switched to laying cards out on the table, and eventually grouped cards before applying a selection-like approach. These intuitive strategies emerged as students explored the task without formal instruction, shaped by immediate feedback and the structure of the materials.

Importantly, we found that nearly all participants organised their individual sorting task into three intuitive phases: (1) preparation of the unsorted input (e.g., by grouping, laying out or stacking cards), (2) card selection or insertion into a sorted structure, and (3) transfer to a final, visibly sorted output. Across students, this process aligned with algorithmic thinking, even if not named or formalised, and reflected consistent algorithmic strategies. In particular, the separation of input preparation, rule-based processing, and structured output demonstrates an emergent application of procedural decomposition and data organisation, which are two key aspects of CS education.

Importantly, none of the participants verbalised algorithm names or showed signs of having been taught these procedures formally. The strategies emerged naturally from the

task structure and were shaped by their environment such as visibility, table space, and peer behaviour. It is also notable that the strategy discussions in the group, between the two sorting rounds, never included optimisation of individual sorting behaviour and mainly focused on the distribution and parallelisation aspect of the task. Students sorted algorithmically on their own, but did not use terms to label or explain these procedures. The gap between visible processes and verbal strategies underscores the importance of recognising intuitive algorithmic thinking through actions rather than words.

Scenario 5.1 illustrates the range and shift of observed behaviours: It exemplifies how students initially acted based on everyday logic ("one person per colour") and adjusted their approach after encountering an inconsistency. It demonstrates a spontaneous but revisable problem-solving strategy, shaped through joint action and minimal verbal coordination.

Before Round 1:
After the general instruction of the task, the group appears unsure about the procedure. After receiving the sorted stack and being advised to get familiar with it before shuffling, one student begins turning cards around and recognising a pattern. Another student suggests, "We are four, and there are four colours, everyone takes one." The others agree.

Round 1:
When the sorting round starts, one student (BR) begins to separate by frontside colour. Shortly after, all other students started doing the same. They then start individually sorting the front colours (not taking the back colours into account).

Before Round 2:
The group changes the task distribution and assigns different colours to not confuse the back colours any more.

Scenario 5.1: Observed strategy shift after initial trial-and-error approach of one group.

5.4.2 Strategy Revisions and Changes in Round 2

Although each group started applying a divide-and-conquer strategy in the first round, the way these strategies evolved in round two varied considerably. Across the six groups, we observed explicit or implicit revisions of the strategy, often triggered by previously encountered challenges or errors in either computational process or miscommunication. In most groups, the post-round-1 discussion served as a moment of reflection and negotiation of changes, although the degree of elaboration and clarity of these discussions differed.

A common pattern among groups who had experienced mistakes or coordination issues in round 1 was adding control mechanisms in round 2. For instance, in Group 2, the participants dynamically started to turn cards around randomly during sorting to double check the back colour after previously mixing colours by accident. Similarly, Group 6

added a spatial separation system by placing stacks with similar front colours far away from each other on the table to avoid confusion.

In three groups (2, 3, and 6), the strategy was adapted to include dynamic task assignment, i.e., participants did not stick to one specific colour or subtask but took over new ones as they became available. This behaviour was not discussed beforehand but emerged organically from within the process, especially in moments when one group member finished early.

In contrast, two groups (1 and 4) stuck to their previously developed strategy, often with only minor adjustments. In these cases, their revised strategy was already implicitly stabilised toward the end of round 1, and the discussion before the second round mainly served to confirm or reinforce what had proven effective.

In nearly all groups, strategy revision occurred without explicit reference to formal algorithms. Instead, students described their changes in everyday terms (e.g. “We’ll do it like last time, but better” or “Let’s help if someone is slow”). Scenario 5.2 illustrates a case where a group revises their strategy both structurally and in terms of task awareness. One group did not revise their strategy.

Round 1: The group starts by separating the backs, then assigning one frontside colour per person. Each participant then sorts the numbers. After encountering errors and mixed-up cards due to visually similar colours, they discuss improvements.

Round 2: The group decides to skip merging all back-colour stacks and instead assigns each subteam one colour to work on separately. They also physically move stacks with the same colour far apart to avoid accidental mixing. This subtle spatial optimisation emerges directly from reflecting on their earlier mistake.

Scenario 5.2: Strategy refinement through spatial reorganisation and error reflection of one group.

While all groups improved their performance in the second round, the ways in which they did so ranged from minor refinements to substantial structural changes. The revisions reflected a growing awareness of task complexity and showed clear signs of adaptive problem-solving behaviour.

5.4.3 Group Collaboration and Emerging Roles

The collaboration patterns in the groups were shaped both by the task design and by the preferences and social dynamics of the participants. While the overall goal was to sort the designed card deck meaningfully, the concrete ways of organising and coordinating varied across groups and evolved during the task, also according to their social behaviour.

In some groups, leadership emerged early on. Participants such as BR (Group 2), TL (Group 5), and TR (Group 6) frequently initiated actions, gave instructions, or

(re)assigned tasks to others. These roles were not formally established, but emerged through action, often because these participants acted first, spoke more frequently, or responded to uncertainty. In most cases, the others followed their lead without resistance, suggesting a form of situational authority.

At the same time, help was often offered spontaneously and without explicit request. Particularly in Group 3 and Group 4, we observed passive helping behaviour: students supported slower group members or quietly corrected small mistakes. This indicates a high degree of mutual awareness, even in the absence of verbal coordination. The role of idle participants was also interesting: in several groups, students who had finished their part started helping others, sometimes by taking over new tasks, sometimes by physically rearranging cards to make the task easier for others.

In particular, task assignment took different forms. In some groups (e.g., Group 4), the roles were distributed at the beginning and remained largely stable. In others, tasks were dynamically negotiated throughout the sorting process. Group 3 even developed a unique vocabulary to distinguish the colours of the card backs ("snow" and "palm"), illustrating how collaboration also involved shared language development to resolve ambiguity. One example of coordinated collaboration and emergent leadership can be observed in Group 5. Their internal structure, task division, and use of space developed progressively, supported by the verbal guidance of a single participant. This is illustrated in Scenario 5.3.

Round 1: Right after the instruction phase, the group is visibly unsure and asks the tutor for clarification about the task. TL takes a central role: they suggest how to divide the task and guide the others through the first steps. The group divides into two subteams, each sorting cards of one back colour. TL repeatedly verbalises what to do next ("First we separate by colour, then by number") and asks others if they need help.

During sorting, the group builds a shared understanding of their table space: the right side is used for completed stacks, the left for unsorted piles. The participants support each other, both verbally and non-verbally, whenever someone falls behind. Despite some mistakes in the first round, the group adapts well, remains calm, and shows strong internal cohesion.

Scenario 5.3: Emerging leadership and structured collaboration of one group.

Although most groups avoided overt social conflict, small moments of miscommunication or coordination breakdown occurred, often triggered by unclear task distribution or parallel actions on shared card piles. However, in all cases, these conflicts were resolved internally without a tutor intervention needed.

Taken together, the observed group dynamics highlight how collaborative problem-solving involves more than dividing the task: it includes negotiation of responsibilities,

management of idle time, spontaneous support, and shared sense-making, all of which contributed to the evolving group strategies.

5.4.4 Strategy Patterns on the Individual and Group Level

To summarise the patterns observed across all six groups, we derived a set of recurring strategies and behaviours at both the individual and group level. These were not predefined, but emerged inductively from our video analysis and reflect the diversity of ways in which students intuitively approached the task, collaborated, and adapted their strategies. Table 5.4 provides an overview of these observed phenomena, structured along three main dimensions: strategy development and change, collaboration and role dynamics, and visible sorting behaviour.

These findings show that students developed systematic, algorithmic strategies (RQ1) and collaboratively adapted them to address challenges during peer interactions (RQ2).

5.5 Discussion

This section reflects on the main findings of our study and discusses their implications in light of the existing literature. We focus on the intuitive strategies identified (RQ1), the ways in which students revised and improved their approaches (RQ2), and the broader implications for teaching algorithmic thinking through unplugged activities.

5.5.1 Intuitive Strategies and Algorithmic Thinking (RQ1)

In all six groups, diverse intuitive strategies emerged naturally during the card-sorting task, as summarised in Table 5.4. Although participants were not familiar with formal sorting algorithms, they still applied intuitive algorithmic strategies similar to selection, insertion, and sometimes merge sort. These untrained responses align with earlier findings on young learners' algorithmic intuition [48, 90].

What stood out, however, were the group-level differences in how these strategies emerged and stabilised. Some groups tended to show more visible coordination early on: they asked each other clarifying questions, verbalised their sorting actions, or explicitly reflected on what was working. These groups achieved notably shorter completion times in the second sorting round, indicating more coordination and strategy revision. In contrast, participants in other groups, especially those that initially lacked structure, also addressed short questions to the tutor, either to confirm their understanding or to reorient the group. While the tutor responses were often brief and focused on the instruction rather than conceptual hints, they occasionally helped resolve confusion or refocus attention.

These observations suggest that intuitive problem-solving is not purely individual, but can be supported by social and instructional cues. Short verbal interactions, whether peer- or tutor-driven, seemed to facilitate the articulation and refinement of strategies.

Theme	Individual Level	Group Level
1. Strategy Development and Change		
Trial-and-Error	Trying out sorting without a clear structure or plan	Group revises overall approach after noticing mistakes or inefficiencies
Self-Correction	Realising a mistake and adjusting it individually	Mistakes are discussed or corrected collaboratively
Strategy Revision	Adapting one's own approach between rounds	Group reflects on what worked and agrees on a refined procedure
Questioning	Asking oneself or others about the next step	Expressing uncertainty and negotiating how to proceed
Control Mechanisms	Turning cards to check correctness, double-checking piles	Introduction of joint checking strategies to prevent errors
2. Collaboration and Role Dynamics		
Task Focus	Sorting one's assigned subset without distraction	Responsibilities are distributed explicitly or implicitly
Helping Behaviour	Supporting others without being asked	Idle members join in or assist actively
Leadership	Verbally structuring the process or explaining one's method	One or two members take on a guiding role during the task
Dynamic Allocation	Picking up new tasks once finished with own stack	Subtasks are reassigned flexibly as needed
Shared Language	Using self-made terms to refer to colours or actions	Group creates and adopts terminology to coordinate better
3. Sorting Behaviour and Structures		
Selection-Based Sorting	Picking smallest/largest next card into hand or pile	Use of shared stacks or areas for stepwise selection
Insertion and Rearrangement	Re-sorting by inserting cards into correct positions	Rearranging table layouts collaboratively
Use of Arrays and Stacks	Structured use of table space (arrays) or in-hand stacks	Regions on the table represent subtasks or data structures
Merge and Combine	Combining own sorted piles with others	Team-based merging strategies emerge during second round
Space as Structure	Using table space to separate sorted from unsorted	Shared understanding of visual boundaries and placement

Table 5.4: Observed intuitive problem-solving and algorithmic strategies at individual and group levels.

This dynamic interplay between intuitive behaviour and external scaffolding opens new questions about how such support can be used intentionally in future interventions.

Interestingly, while algorithm-like behaviours emerged, they were highly fluid and adaptive. Participants frequently switched between approaches, sometimes combining them within a single sequence. This flexibility supports the view that algorithmic thinking is less

about rigid procedures and more about structuring action in a purposeful way. However, when it comes to machine execution, algorithms still rely on precisely defined and rigid procedures.

Rather than following fixed rules, the students developed strategies through observation, trial and error, and imitation. These patterns reinforce previous research that emphasises the role of low-threshold exploratory activities in developing CT skills in K-12 education [18, 70].

5.5.2 Strategy Revision and Learning (RQ2)

The second research question focused on how participants revised their strategies during the task. While some groups refined their approach only slightly, others restructured their entire workflow after encountering difficulties in the first round. These adjustments reflect meaningful learning processes, in which students became aware of inefficiencies or sources of error and took action to avoid them.

A central finding was that strategy revisions rarely occurred in abstract or general terms. Instead, they were triggered by concrete experiences, errors, slow progress, or coordination breakdowns. The groups then introduced changes such as reassigning roles, reorganising task sequences, or introducing control mechanisms such as spatial separation of card stacks or peer checking. This kind of situated learning through reflection-in-action is a common feature of problem-solving in collaborative settings [72].

These observations also highlight the importance of cognitive load in shaping strategy revision. In particular, in the first round, some participants seemed overwhelmed by the complexity of the sorting task and failed to coordinate effectively. Afterward, their revised strategies aimed to reduce cognitive effort, for example, by simplifying the distribution of subtasks or avoiding confusing colour combinations. This aligns with findings from cognitive load theory [82, 83] and more recent studies in programming education showing that cognitive relief fosters better performance and motivation [17, 88].

Moreover, the tendency to build on what had worked in round one suggests a shift from unstructured experimentation to increasingly systematic thinking. This was not always verbalised by participants; in fact, many struggled to articulate their strategies when asked, but their actions became more deliberate. The gap between performance and explanation also echoes previous findings that procedural understanding often precedes the ability to express abstract concepts.

In this sense, the unplugged nature of the task supported not just CT but also metacognitive growth. As students externalised their process through sorting, discussing, and revising together, they made their own thinking visible, both to peers and to themselves.

5.5.3 Collaboration and Group Dynamics

Beyond individual approaches, our findings highlight the essential role of collaboration in the development of algorithmic problem-solving. While theoretically feasible through

parallel task division, the real group interactions were more intricate and frequently crucial to success.

In many groups, one or two participants assumed the guiding role, either by giving verbal instructions, suggesting how to distribute tasks, or supporting others during idle times. These roles emerged informally, without being explicitly assigned, and were often accepted by the rest of the group without resistance. The presence of such roles helped structure the process and facilitated coordination, especially in moments of uncertainty or error.

Help was also frequently offered spontaneously, particularly when someone was visibly slower or made a mistake. In some cases, group members began dynamically reassigning subtasks or physically rearranging cards to support others, behaviours that were not discussed in advance but emerged naturally from the situation. These observations echo patterns found in other studies of collaborative learning, where short interaction sequences, turntaking, and implicit negotiation of roles shape how joint strategies develop [78].

Interestingly, participants developed shared systems and strategies not only for sorting, but also for managing complexity, such as using spatial separation or inventing their own vocabulary to avoid confusion between similar colours. These forms of implicit coordination reflect a high level of procedural engagement, even when explicit communication was minimal.

Our findings suggest that collaborative problem-solving in CS tasks includes more than just dividing the work. It also involves ongoing negotiation of responsibilities, shared awareness, and the readiness to support others in moments of difficulty, all of which contributed to the evolving group strategies.

5.5.4 Computer Science Concepts in Intuitive Sorting Strategies

One of the most striking results of our analysis is the degree to which CS concepts emerged, even in the absence of any formal instruction or explicit terminology. Across all groups, we observed behaviours that corresponded to fundamental algorithmic principles: selecting minimal elements, maintaining sorted structures, distributing data by attributes, and merging intermediate results. These are not trivial actions. They reflect the core of what we understand as algorithmic thinking [29, 94].

What makes these findings particularly valuable is that they were not the result of prior formal programming education or rehearsed strategies. While individual prior experiences cannot be fully ruled out, the observed strategies appeared naturally through interaction with the task and through collaboration with others. This shows that algorithmic thinking is not tied to syntax, code or tools but is rooted in the structure of problems and how learners make sense of them.

The way in which the students adapted their strategies illustrates a deep intuitive grasp of core CS concepts, such as control flow, data organisation, and procedural decomposition. These patterns confirm the potential of unplugged activities not only to teach concepts,

but also to reveal how children already think computationally, before they are even aware of it.

In our view, this is a key insight for the future of computing education. If we want to teach algorithms meaningfully, we need to start by understanding how learners already approach algorithmic problems: on their own terms, with their own strategies. Supporting and expanding these intuitions may be more effective than replacing them with formal procedures from the start.

This perspective also opens new research directions: How do learners develop these intuitions on different types of tasks? What kinds of instructional settings foster productive and transferable strategies? And how can we make these emerging processes visible not just to researchers, but to learners themselves?

5.5.5 Implications for CS Education

Our findings suggest several implications for the design of learning environments in CS education, particularly in the context of early algorithmic thinking.

- The spontaneous emergence of structured problem-solving behaviours in sorting tasks highlights the importance of connecting new content with existing learners' intuitions. Rather than starting with formal definitions or abstract procedures, it may be more effective to start with tasks that make implicit strategies of learners visible. Recognising and naming what students already do, such as grouping, sorting, checking, and merging, we can reduce entry barriers, cognitive overload and increase engagement. This approach may also support self-efficacy, especially for those with little or no prior exposure to CS.
- The instructional design can explicitly build on the themes of intuitive strategies observed in this study. Many participants employed strategies that resembled well-known algorithmic structures, even without prior instruction. Teachers can use these moments as a bridge to introduce formal concepts, for example, by showing how a student's approach mirrors the sort of selection or how the merging strategies of two participants resemble recursive thinking. Making these connections explicit could foster a deeper understanding of algorithms as flexible, purposeful tools rather than rigid sequences.

Furthermore, our study underlines the value of hands-on collaborative activities, especially those that allow visible manipulation and shared reasoning. The unplugged setting allowed students to externalise their thinking, observe each other, and refine strategies while learning from each other.

5.5.6 Limitations and Future Work

This study offers detailed information on how intuitive algorithmic strategies emerge in collaborative sorting tasks. However, several limitations must be acknowledged. Due to

the outreach setting, participant selection was limited by logistical factors, particularly the requirement of parental consent for video recording. As a result, the sample was neither intentionally diverse nor homogeneous. Furthermore, our analysis was based exclusively on video data from a bird's eye perspective, which limited access to student reasoning beyond what was verbalised or made visible. Tutor interventions, though intended to be minimal, also varied slightly between groups.

These limitations open up directions for future work. An important question is how intuitive strategies can be made more visible and discussable within groups, whether through prompts, guided reflection, or peer explanation. Future research could also explore how such strategies develop in other types of problem, beyond sorting, such as search, decision-making, or control structures. It is notable that participants had no prior exposure to sorting algorithms. Introducing formal training on these algorithms, followed by a repeated experiment, could enhance their strategy articulation.

5.6 Conclusion

The card-sorting task provided a window into how students approach algorithmic problems when given the freedom to act without predefined methods. Many of the observed strategies reflected key ideas from CS, such as ordering, grouping, and dividing work, but these patterns emerged naturally, shaped by the task and the group setting.

We found that the students not only developed intuitive approaches but also revised them when they proved ineffective. These revisions were often subtle: changes in distribution, clearer role assignment, or the introduction of simple control mechanisms. In several cases, tutor interventions or peer questions played a role in triggering these improvements.

What stood out most was how much of this thinking remained unspoken. The learners acted with intention, but rarely gave names to their strategies. This gap between performance and explanation points to an opportunity in CS education: Rather than replacing intuitive behaviour with formalism too early, we argue that CS interventions should deliberately build on these existing strategies. Tasks that first elicit the learners' own approaches and make them visible within a group can provide a powerful foundation for later introducing formal concepts and give feedback to the teacher in the students' learning process. We should focus more on helping students recognise and refine the strategies they already use, taking them as a starting point for future discussions, e.g. formalising sorting algorithms to programming languages.

By observing how algorithmic thinking emerges in collaborative settings, we gained a better understanding of how to support it, not just through instruction but by creating an environment for learners to develop their own solutions first.

CHAPTER 6

From Intuition to Algorithm: Revision of Intuitive Sorting Strategies

So far, we explored how students perceive the term *Algorithm* and found out that they have a string disposition to connect it to repetitions, but in two different ways: Most commonly (1) repeating the algorithmic procedure itself, or (2) a repetition as an algorithmic construct inside an algorithmic sequence to repeat a specific command or a set of commands. In addition to that, they also build heavily on their personal experience and daily life (Chapter 4).

This outcome is important because we now know that the term algorithm truly is an abstract construct for children between the ages of 11-12. Building on this, our second study (Chapter 5) went further, trying to find out more about these intuitive and naturally occurring algorithmic perceptions in the form of their intuitive strategies while solving an algorithmic task. We successfully displayed the problem-solving strategies that naturally occur in children of this age group, such as divide-and-conquer, trial-and-error, and parallelisation.

In this chapter, we go even further and ask the question:

RQ1: How do students' individual and collaborative sorting strategies change after a targeted algorithmic intervention?.

To answer this question, we conducted another study using video analysis, but with a curcial difference. We used a learning intervention between the two sorting rounds of the card-sorting activity of Section 5.3.

The findings will contribute in two different ways: first, they allow us to observe whether and how previously intuitive strategies become more structured or algorithmically

grounded after a short CS Unplugged intervention, targeting the same concept but in a different application. Second, they help us understand to what extent children can adopt, adapt, or critically evaluate newly introduced sorting algorithms, and how these ideas manifest in their individual and group strategies.

We can trace both the continuity of intuitive approaches and the emergence of more formalised Computational Thinking (CT), and in particular problem-solving. The analysis bridges the gap between implicit problem-solving strategies and explicit algorithmic reasoning in the context of a collaborative unplugged sorting task.

6.1 Methodology

This study builds on the previous investigation of intuitive strategies in unplugged sorting tasks (Chapter 5, [45]) and explores how students refine their strategies after a structured algorithmic intervention.

The data consists of five workshop recordings, each involving four lower secondary school students working collaboratively on the card-sorting task (Section 5.3) in two rounds, with an intervention between.

Participants and Setting.

The workshops were held in the eduLAB, a dedicated outreach space for computer science education at TU Wien (see 2.2.2). All participating students were between 10 and 12 years old and had no prior formal instruction in sorting algorithms or algorithms in general. Each group consisted of four students, seated around a shared workspace. The workshop was led by university student assistants who introduced the task and facilitated the sessions without providing strategic guidance during the sorting phases.

The core task remained identical to the one used in Chapter 5: the students were asked to sort a shuffled deck of cards as quickly as possible. The cards featured two different back colours (two decks) and four front colours, with numbers ranging from 10 to 99. The goal was to sort the cards in such a way that the colours were grouped together and that the numbers were ordered within each front colour. This structure allowed for multiple sorting strategies and made collaborative work necessary.

CS Unplugged intervention Between the Sorting Rounds.

Between the two sorting rounds, a targeted algorithmic intervention was introduced. Each student temporarily left their group to participate in a short expert session focused on a specific sorting algorithm. One student learnt bubble sort, one merge sort, one quick sort, and one bucket sort (Figure 6.1).

This intervention involved a 3D printed sorting balance bar (Figure 6.2) that helped demonstrate the respective algorithmic processes. The intervention was inspired by the CS Unplugged activity “Lightest and Heaviest – Sorting algorithms” [10] . In this

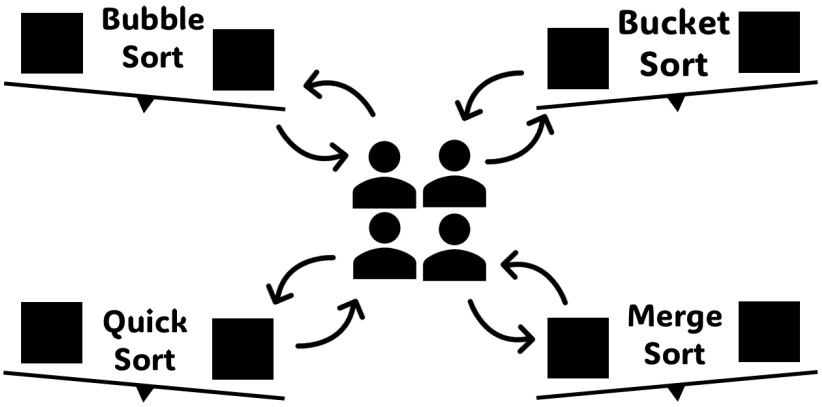

Figure 6.1: Overview of how the sorting intervention grouping happened: Each group member learns about a different sorting concept with a 3D-printed balance scale and comes back afterwards in their group again.

activity, the balance scale serves to compare two values. The weights of the cubes differ by only 2 grams, which is crucial to prevent children from intuitively perceiving the lighter or heavier cube by hand alone. Thus, they are *forced* to use the balance scale for determination. This approach not only demonstrates the sorting algorithm but also emphasises the method of binary comparisons, which is how a computer program works.

The 3D printed models we used were developed by Annika Vielsack [92] . The idea was to create a small version of a classical balance scale as a light and inexpensive classroom version, so it is easy to produce a scale for every child.

The group separation in this intervention happened randomly. Each expert group of one sorting algorithm was supported by one workshop tutor. The tutors demonstrated the concept of the algorithm with the balance scale, and the children had a few minutes to practice the algorithm with it. The scales' main purpose is to show that a computer compares can only compare two digits. For bucket sort the tutors used an additional unicoloured small range of numbered cards. All children also had the opportunity to transfer the concept to number cards at the end of the intervention session. We intentionally introduced them to a new sorting concept with a different tool than previously used to be able to observe their ability to transfer the concepts to the card-sorting problem.

After learning and trying out the algorithm individually with the sorting scale, the students returned to their groups and engaged in a collaborative discussion. They were asked to agree on a strategy for the second sorting round, ideally integrating the new insights from their expert session. The tutors did not guide this discussion. The second round then followed under the same conditions as the first, but the students were expected to optimise and execute a refined sorting process.

Figure 6.2: The CS Unplugged intervention to demonstrate different sorting algorithms during the intervention. A balance scale with differently weighed cubes to sort.

Video Analysis and Data Segmentation.

Each group session was recorded from a bird's-eye perspective. The analysis focused on both group-level dynamics and individual strategies throughout the two rounds. Each video was segmented into four key phases:

Pre: Instructional Phase – Student Assistants explain the structure of the card set and the task.

1. Preparation Phase - Initial discussion before the first sorting attempt.
2. Sorting Phase 1 – Sorting without prior formal introduction to problem-solving strategies and sorting algorithms.
3. Intervention Phase – Each group of four splits up into an expert group to learn one specific sorting algorithm.
4. Discussion Phase – The original group of four comes together again and discusses the demonstrated algorithms. Their task is now to find a strategy to beat their own previous attempt and be faster in a second attempt.
5. Sorting Phase 2 – Sorting with the groups' revised strategy.

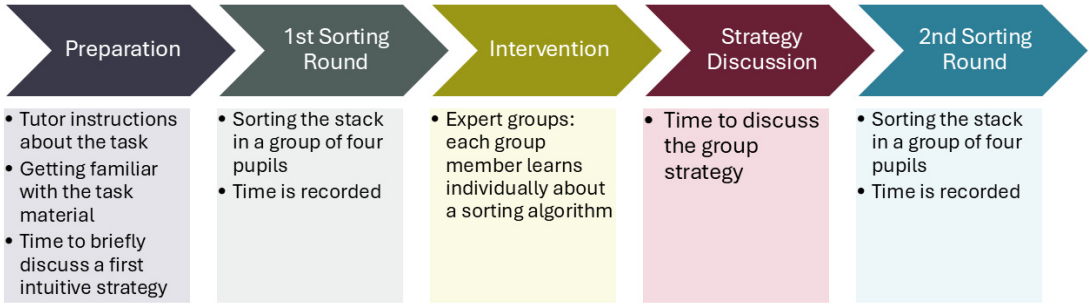

Figure 6.3: Overview of the task process: from the first instruction through the tutors to the end of the second sorting round.

Analytical Approach.

The analysis of the videos in this study is based on detailed paraphrases of the observable sorting strategies of the students in two rounds. Instead of transcribing spoken language word for word, the video sequences were paraphrased in a structured way, aiming to describe what was done and how the sorting process unfolded, rather than what was said or their emotional reaction.

This methodological approach is influenced by several qualitative traditions. First, it follows the idea of qualitative content analysis as described by Mayring [56], where the complex material is paraphrased and reduced without losing its essential structure. We also know this purely *describing* aspect of the research material from the documentary method [12]. In that sense, the sorting activity of each child was condensed into a clear narrative of the sorting process.

At the same time, principles from Thematic Analysis (TA) [13] were applied. Particularly, we identified recurring elements such as selection-sort-based, bucket-like or unstructured sorting strategies. These patterns were not coded in a technical sense like it is typical for Qualitative Content Analysis (QCA), but they were recognised and categorised through repeated comparison and occurence. We also compare these emerging themes with the students' first and second sorting attempt to identify patterns.

From the results of [48] and Chapter 5 we already know that children have difficulty describing and articulating their skills. Pseudocode-like reconstructions were then derived from these definitions to represent the observed logic in an abstracted form. These pseudocode reconstructions accompanying each case are not meant as formal models but rather as analytical sketches that deliver additional a condensed overview of the used *algorithm*.

For each task-phase we analysed group-level strategies in terms of coordination, division of tasks, and consistency of strategy to be compared with the outcomes of Chapter 5. In addition, each student's individual sorting actions were paraphrased in detail. These

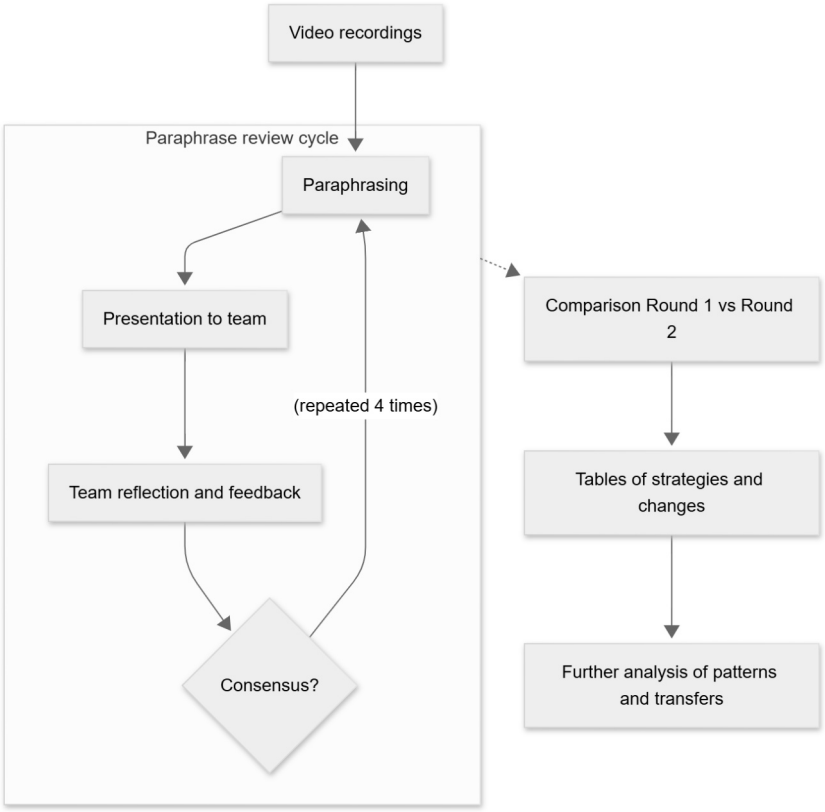

Figure 6.4: Video Analysis Process to create the paraphrases.

paraphrases were used to explore the shift of in their sorting strategies. We then compared sorting round one and sorting round two per individual to assess changes, such as data preparation (e.g. how unsorted cards were laid out), sorting structure (e.g. pile formation, buckets, arrays), selection logic (e.g. selection criteria of the next card) or adapting their strategies (e.g. shift to another sorting strategy).

The paraphrases were developed in an iterative and collaborative research process that involved the author and four undergraduate research assistants who are also experienced in conducting eduLAB workshops. Each video (= each group) was paraphrased in detail by one person and then presented to the other researchers. In regular team meetings, the sequences were reviewed and discussed collectively with the goal of reaching agreement on the descriptions and collecting the interpretation of the sorting actions. Since the paraphrasing was deliberately focused on visible behaviour due to the research setup (describing what could be seen rather than interpreting), disagreements were infrequent and usually minor. All cases went through four rounds of reflection and feedback, and each paraphrase was seen and confirmed (or commented on) by all of the other team members(Figure 6.4).

Based on these results, pseudocode reconstructions were created to represent the core logic

of each student's sorting behaviour. These reconstructions did not aim for perfect formal correctness, but focused on capturing the algorithmic structure behind the observed actions and labelling them with a well-known sorting algorithm, e.g. selection sort. Small errors or inconsistencies were ignored if the underlying pattern remained stable. In this sense, the pseudocode sketches served as abstractions that helped to identify and compare procedural strategies between students and between rounds to better be able and observe the changes from round one to two.

All the data and the observed strategy changes were collected in a table and the basis for further analysis to look at different cases of sorting strategies used (e.g. transfer from selection sort to merge sort). One table specifically looked at each student individually and another was used to observe the overall group strategies. The analytical aspects which include both the analytical parts and the interpretive parts are summarised and described in Table 6.1.

In contrast to the previous study (Chapter 5), where open coding was applied to identify emergent problem-solving strategies, this analysis did not repeat that process. In an initial browsing through the video material, we found no new key types of general problem-solving strategies beside those already identified in the previous study (see Table 5.4). Instead, the focus in this study lies on how previously observed sorting strategies were refined or restructured after the newly added intervention between the two sorting rounds (Figure 6.3). This supports our overall goal to develop an environment that offers transfer from intuitive algorithmic strategies and everyday problem-solving logic to new and more complex concepts. For example, being able to talk about specific sorting strategies and use more complex ones.

6.2 Observations and Findings

We will give an overview of each student's individual sorting strategy and a summary of the general group strategy, focussing on the shift between the first sorting round and the second sorting round. This section is based on the outcomes of the researchers' group discussions during the analysis process and the paraphrases, including some important direct quotes from the participating children.

To improve readability, we have used randomly use male and female pronouns 50 % for anonymised participants. As we need to distinguish between the group and the individuals, it can get confusing using the gender-neutral pronouns 'they' and 'them' while talking about individuals and groups.

6.2.1 Descriptive Overview of Strategy Changes

In general, all five groups showed improvements in the second sorting round compared to the first sorting round in the discussion part of their strategies. This means that they actively talked about the algorithms demonstrated during the intervention time. We

also provide a condensed table version as an overview of the findings before starting to interpret and discuss them in Section 6.3.

Analytical Focus	**Description**
Individual Round 1 – Paraphrased Description	A detailed and narrative description of the student's individual sorting process during the first round. Focus lies on how data is prepared and structured, how cards are selected, and whether the process remains stable or shifts mid-way. Based entirely on video observations.
Individual Round 1 – Pseudocode Sketch	An interpretive sketch of the sorting logic behind the observed individual sorting strategy, written in pseudocode-like form. Minor inconsistencies or interruptions were ignored if the underlying structure was recognisable and coherent.
Round 1 – Interpretation	An informal commentary on how clear, systematic or algorithmic the observed strategy appeared. These interpretations were always part of the reflective discussions of the researchers in the analysis process.
Individual Round 2 – Paraphrased Description	A detailed narrative description of the student's sorting behaviour during the second round.
Round 2 – Pseudocode Sketch	A revised sketch of the sorting logic in pseudocode-like form, adapted to reflect the second round behaviour. Again, small deviations are ignored if the general structure holds.
Individual Strategy Changes	A reflection on what changed between both rounds. Includes adjustments in card preparation, search method, structure, or coordination. Comments highlight any other observed shifts.
Individual Round 2 – Interpretation	A brief interpretive comment on the student's strategy in round two, with regard to adaptability and possible transfer of new ideas from the intervention.
Group Preparation	A brief account of what the group did before the sorting began. Notes if there was any form of planning, proposed strategy, division of labour or visible hesitation. Often paraphrased from short utterances or observed actions.
Group Sorting Strategy (Round 1)	A compact paraphrase of how the group sorted the cards during the first round. Focus is on the organisation of work (e.g. divide-and-conquer), role distribution and observable coordination patterns.
Group Intervention Participation	A short summary of how the group approached the sorting in the second round. Notes whether and how the coordination or task distribution changed.
Group Intervention Discussion	A paraphrased account of what was said during the group discussion after the first round. May include short transcript excerpts or rephrasings of key contributions. Focus is on whether the group revised or negotiated their strategy.

Table 6.1: Analytical foci used in individual and group-level video interpretation

6.2.2 Strategy Development Across Two Sorting Rounds in Detail

This section gives a condensed overview of the paraphrased videos from the perspective of each group and each individual. Afterwards, we will summarise the different types of changes we observed, e.g. changing from a selection sort to a bucket sort strategy.

Group 1

This group showed greater initiative compared to other groups in looking at the structure of the card deck in the beginning. The distribution of cards was discussed quite thoroughly at the beginning, including where to place the sub-decks and how to divide the different tasks between the four children. They decided to sort first by back colour, then by front colour, and finally by number. This group explicitly discussed an initial division of the table into two sides: one for bamboo backs and one for flower backs. In general, this strategy was used by most of the groups and videos, also in Chapter 5. This group was special because they explicitly brought up the topic in the discussion before even starting.

Each child received approximately one-quarter of the full deck. After sorting their allocated cards by back colour, the group split into two teams, one for each back colour, on each side of the table. From there, the sorting continued collaboratively, first by front colour (with four stacks, one for each colour), and finally by numerical order. In particular, the group decided to use bucket sort in the second round, but not all children adapted their original approach consistently.

The intervention discussion featured explanations of merge sort, quick sort, and bubble sort. Some children criticised quick sort for being 'too random' and dismissed bubble sort as too slow.

In the second round, the structural division of work remained the same. However, there were differences in how rigorously individual children implemented bucket sort or stuck to previously used methods. Two showed more confidence and speed, while the other two continued with their previous approach, regardless of the group agreement.

Group1/Top Left: TL began the task by dividing his assigned quarter sub-stack by separating the cards by their back colour into two piles. He helped TR to further separate the backs allocated to TR, and also helped to separate them by front colour. He then helped to combine back-colour cards to create two back-colour sub-stacks. For number sorting, he used a selection-sort-like approach. The cards were first laid out on the table unstructured, and then he looked for the smallest remaining card, which he picked and added to a hand-held stack. Later, he introduced one single pivot-based split in the beginning (e.g. under and over 40), and then combining bucket and selection sort strategies.

In the second round, he repeated a similar pattern but changed the pivot from 40 to 50. The card handling was similar, but now he created smaller buckets and used them for selection-like sorting. His strategy remained somewhat hybrid and unstructured, even

Student	Round 1 Strategy	Round 2 Strategy	Observed Change
Top Left	Selection-sort strategy with unstructured card layout; later used pivot to split cards into two halves	Continued selection with bucket-like split using adjusted pivot; layout remained hybrid	Slightly more structured and confident; faster execution but strategy still inconsistent
Top Right	Insertion strategy using array-like layout on the table; shifted cards to insert new ones	Repeated same strategy with identical structure and handling	No observable change; consistent execution despite group agreement on bucket sort
Bottom Left	Started with confusing double-sort logic, then reorganised to front- and back-colour-first sorting; used array-like layout for numbers	Adopted bucket sort with consistent sorting process and layout; helped others after finishing	Clear strategic shift; faster, structured and more collaborative handling
Bottom Right	Insertion strategy with spatial layout; grouped cards by magnitude before sorting	Used bucket sort with spatial sorting and partner assistance	Strategy became more confident and structured; increased use of space and collaboration

Table 6.2: Summary of individual strategy development – Group 1

though the group had agreed on a clean bucket sort approach. However, the structure with respect to organising the cards and his confidence in the sorting process appeared to improve slightly because he was faster and better prepared.

Group1/Top Right: TR was responsible for distributing the cards at the beginning and then separated his own quarter stack by back colours. He worked together with TL separating one back-colour sub-deck by front colours to build four front-colour sub-decks. He used an insertion sorting strategy that placed cards in a sorted sequence (Figure 6.5): cards were placed in a linear sequence on the table in roughly the right order, with high numbers to the right and low numbers to the left. If a new card needed to go between others, he shifted the cards manually to make space to insert the new number.

This card-organising structure was visually similar to working with an array, although gaps and overlaps made the layout more flexible. Once he had placed all the cards, he collected them one after another in order to create a sorted stack. This was done for two front-colour sub-decks, each time using the same array-like 'data' structure. His execution was quite careful, and he also rearranged cards after some mistakes.

In the second round, he followed exactly the same approach. No changes were made, despite the fact that the group had agreed on using bucket sort. He neither challenged the agreement nor explained his own method to the group. His contribution to the group discussion was minimal, and his sorting strategy, as well as card organisation, showed no influence from the intervention. However, his execution remained stable.

Figure 6.5: Organising the card deck by using the table space to separate the two back-colour sub-decks and laying the cards out in a roughly array-like sequence on the table

Group1/Bottom Left: BL attempted to sort his assigned cards by both back and front colour simultaneously. After sorting about half the stack, he abandoned this dual-strategy approach and focused first on separating the two different back colours. After he finished this task, he took over a pile of already back-colour-separated cards and continued sorting them by front colour into separate sub-decks. While sorting, he gave verbal instructions to BR, indicating a sense of structure and group coordination.

As the sorting by numbers started, he took one front-colour sub-deck and laid them out on the table, placing each card approximately where it belonged in comparison to those already laid out. If needed, he shifted cards to make space. The limited table space led him to start a second row beneath the first one (Figure 6.5). Once finished with this task, he hesitated whether to start with the smallest or largest value but ultimately chose the smallest and built a sorted stack in his hands. He then repeated the same process with another front-colour sub-deck.

In the second round, he switched to a clearly visible bucket sort approach, consistently applying it across the two sub-decks he was responsible for. Although one error remained unnoticed, his sorting was systematic and faster than in sorting round one. After sorting, he helped BR with a stack and played a key role in merging the final sorted sub-decks. He had been highly active during the intervention discussion, interrogating others about their algorithms and proposing quick sort before agreeing that bucket sort would be fastest. He also specified how the sorted stacks should be laid out, although later he broke his own rule by starting a new stack to avoid spatial interference with BR, which also shows a high level of adaptability.

Group1/Bottom Right: BR began by separating the cards from his assigned quarter stack by back colour while keeping all the cards on the table and not in his hands. He put all the same back-coloured cards on his side, and put the other back-colour cards to TL. Some mistakes occurred during this process, such as placing flower-back cards on his own side, but these were quickly corrected by himself, which shows awareness of the process. He then merged the cards on his side with those of BL with the same back colour, which had already been separated by front colour, and started number sorting the blue front-colour sub-deck.

He used an insertion sort-like method: He laid the cards out on the table in a row, placing each card approximately in the correct position. BR seemed to aim for visibility and a rough order first. After all cards were laid out, he picked them up starting from the smallest and paid attention to the exact sequencing order. He then reversed the sorted stack so that the largest card was on top. He did not sort a second stack, but instead helped BL with the green front cards and had a passive role while the other group members merged the final stack. His second stack remained unsorted, and no one noticed.

In the second round, BR applied bucket sort as discussed in the group. Although he was initially confused, which was observable through his hesitant movements, he gradually became more confident. It seems that by observing the other children in his group he adapted the strategy and became more confident. His buckets were spatially laid on the table and sorted through a mixture of dragging, repositioning, and picking up cards, rather than doing everything with a stack in his hands. He also assisted BL in completing the sorting, using a hand-over-hand strategy in which each child focused on a part of the range. This spatial approach and the reliance on physical layout were already apparent in the first round, suggesting a consistent preference for working with visual structures on the table rather than holding cards in hand as many other children in this study did.

Group 2

This group started with a fairly coordinated distribution of cards by back colour. Two group members were together responsible for one of the two back-colour sub-decks, resulting in each child being responsible for sorting two different front-colour sub-decks. They briefly discussed the structure of laying cards out, but did not continue to shape this idea in more detail. During the first round of classification, many of the intuitive strategies discovered in Chapter 5 were used, e.g. *shared language, self-correction, trial-and-error* and *dynamic allocation*.

In the discussion phase, the group could not agree on one sorting strategy, but they focused on new sorting algorithms: bucket sort and merge sort received the most attention.

In the second sorting round there was visibly more structure, e.g. they organised the cards in a stack and/or an array structure to be faster to browse through the cards than just laying them out unorganised on the table. The overall strategy in this group regarding the sorting algorithms changed for every group member, as they did not initially use the

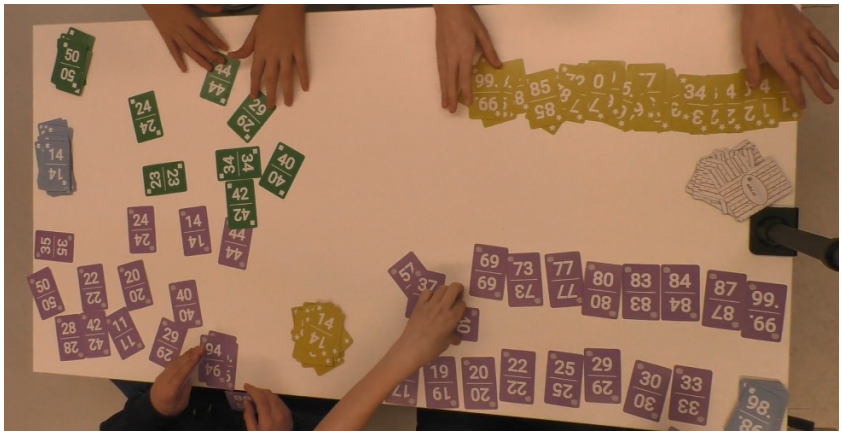

Figure 6.6: Overview of different card-structuring strategies, representing arrays (BR, TR) though laying out cards in an array-like structure, stacks (blue fornt colour sub deck on the left side) and piles (unstructured yellow front coloured card pile between BR and BL).

algorithms they used in the second round, but they were more successful and showed signs of more confidence through faster movements and fewer mistakes, e.g. always putting the smallest number on the stack.

Student	**Round 1 Strategy**	**Round 2 Strategy**	**Observed Change**
Top Left	Selection-sort-like strategy with some trial-and-error, limited algorithmic clarity	Adopted bucket sort, improved structure, clearer layout	More structured card handling, slightly more systematic but still hesitant
Top Right	Insertion sort, some confusion and hesitations, slowest in group	Adopted bucket sort, structured execution, confident handling	Clear improvement, checked for errors, higher speed
Bottom Left	Partially structured strategy with attempts of bucket-like grouping, shifting between methods	Adopted Merge Sort, very structured and fastest in group	High clarity, improved algorithmic expression, leadership behavior visible
Bottom Right	Insertion sort with two-line structure, active reflection while sorting	Applied Merge Sort thoroughly and correctly	Very consistent execution, self-corrected one mistake, showed confidence

Table 6.3: Summary of individual strategy development – Group 2

Group2/Top Left: TL initially used a selection-sort strategy, but later shifted to a trial-and-error approach, assuming that larger numbers were the biggest, necessitating card insertions in the final stack. During the intervention, TL explained bubble sort to

the group, but they doubted its efficiency. Although TL began exploring BR's idea of bucket sort as a faster strategy, she could not hear the complete explanation. Despite the lack of practice time, TL employed bucket sort in the second round, indicating the ability to grasp and implement the basic concept, even while playing a passive role in the discussion process.

Group2/Top Right: TR started using an improvised mixed selection-insertion-like sorting strategy. Although understanding the group strategy, he struggled with sorting the cards, which was indicated by many mistakes and switching cards on the table. The approach involved placing cards in an array-like structure on the table and inserting new cards in between, but the execution lacked consistency and efficiency.

During the discussion, TR opted for bubble sort, but was overruled by the others quickly and adopted the newly learnt bucket sort in the second attempt. TR showed a structured shift from the first-round insertion-like strategy to adopting a structured bucket sort algorithm while sorting the cards. A small mistake happened, and TR showed confidence and understanding by quickly reacting to the mistake, exactly knowing what to do to correct it.

Group2/Bottom Left: BL used a semi-unstructured, selection-sort-based approach to sort his individually assigned stack in the first round and shifted his strategy after the intervention to merge sort. He worked with multiple smaller stacks and sorted them one by one before merging them. While a few cards were forgotten at the beginning, the rest of the process was much more structured. BL finished quickly and again helped others. The change in sorting logic is clearly visible, and so is the confidence in the chosen strategy.

There was also a recognisable shift to organise the cards in more stack-like structures than before while dynamically helping the group members after finishing his individual task. He showed signs of leadership by announcing tasks for the others in rounds one and two. Between the sorting rounds, he tried to explain his ideas to the other group members that he got from the intervention, but struggled to verbally explain why he thought merge sort was the best strategy. BL could not convince the others to use merge sort in the second attempt and refused to take a different algorithm than he wanted.

Group2/Bottom Right BR also began with an improvised selection-based strategy, placing cards into array-like rows, roughly putting big numbers in one and small numbers in the other row. There was no clear consistency or sorting routine and also small pauses before picking the next number, assuming he was thinking about the next steps.

During the discussion, BR attempted to explain the quick sort algorithm to the group. The explanation was interrupted by the group members and commented on as impractical. BR showed awareness and participation in the discussion, e.g. by openly criticising the effectiveness of bubble sort. In the end, BR preferred merge sort, although it was not explained to the end.

In the second round, BR applied a merge sort strategy, which is quite accurate considering the very brief introduction by BL. He seemed to have picked up merge sort well, despite the earlier interruption. He managed to complete the task reliably, with only one minor error incident, which was also handled with confidence.

Group 3

The children engaged in unstructured dialogue, frequently interrupting each other, yet adhered to a methodology involving the division of the entire deck into sub-stacks to manage the workload effectively. They demonstrated an understanding of algorithmic concepts by using phrases such as "first we do this, and afterwards we do that," indicating their recognition of procedural steps. This reflects an application of algorithmic thinking to a real-world context, which aligns with the findings in Chapter 4. Furthermore, they perceived the table as a means of segmenting the cards based on back colours, demonstrating their ability to organise problem-related data. Throughout the sorting process, they encountered various errors, such as incorrect card placement in sub-decks by back colours, but successfully addressed these issues during subsequent discussions. There was a progression in their communication skills through the development of a shared vocabulary, as evidenced by their evolving use of terms such as 'background colour.' They did not talk about algorithm efficiency after the intervention and continued to carry out the same overall strategy in the second round.

One social situation in this group happened during the second round: BL was distracted because TR and BR started arguing about something. TL noticed her distraction and gave her a friendly slap on her cheek to make her continue her task. BL started laughing and continued her task.

Overall, this group was outstandingly chaotic and did not apply a new strategy based on the intervention. Nevertheless, they seemed to have a lot of fun. Still, there are minor strategy changes visible after the intervention in the individual sorting processes.

Group3/Top Left: TL appeared to have major difficulties in understanding the group's initial 'divide-and-conquer' strategy to distribute the cards and separate the different back colours, so everyone received an unsorted stack or pile of the same back colours. She observed how the others were separating and then took some of the cards allocated to her. She did not look at the back colours and began separating them randomly, producing incorrectly sorted back colours. An assumption could be that this child did not understand the structure of the card deck and that there were different back and front colours. She attempted to mimic what the others were doing without understanding what was happening. After half a minute and some comments from her group members, she began separating the cards by back colours. She performed with less confusion and more confidence in her individual front colour sorting. She used a clean selection sort-based strategy and organised the unsorted cards in a stack laid down in an array-like structure beginning from the lowest number. However, she appeared confused by the fact that the numbers were not sequential, which again indicates that she did not understand

Student	Round 1 Strategy	Round 2 Strategy	Observed Change
Top Left	Misunderstood group strategy, chaotic start; used clean selection sort later	Same strategy with more control and confidence; used variable-like naming	Better understanding of structure; used more control mechanisms
Top Right	Took initiative; clean selection sort with corrections	Less structured layout, but same selection-insertion strategy	Increased confidence, but less emphasis on layout structure
Bottom Left	Slow and confused, semi-structured selection approach	Tried merge sort, switched back to selection with minor improvements	More structured behaviour, but algorithm understanding unclear
Bottom Right	Assertive, selection with insertion mix, led without supporting others	Same strategy, inconsistent sorting direction, distracted group dynamic	Dominant role without strategic change; some control elements used

Table 6.4: Summary of individual strategy development – Group 3

the overall structure of the card deck. As TL was the only one who was confused, we assumed that she could not pay the same attention as the other group members during the initial instruction and contact with the card deck.

During the discussion phase, she expressed her feelings about the poorly prepared group strategy by commenting, “we did just something”. She did not explain her intervention algorithm, nor did the others.

In the second sorting phase, she began calling the back colours ‘winter’ and ‘summer’ instead of ‘blue’ and ‘green’. She did not explain why she did this, but it is presumable that she realised the overloaded terms with the green and blue back colour, as well as the green and blue front colour cards. This already was a very algorithmic behaviour and can be compared to the uniqueness of variable names and declarations in computer science, especially algorithmic design. She worked more confidently at the beginning by separating the back colours and then used the same sorting strategy as in her first attempt, adding control mechanisms, e.g. turning cards to check if they were correct, or browsing through an already sorted stack to verify the order of the numbers.

Group3/Top Right: TR appeared to have understood the concept of the card deck from the beginning. He listened carefully to the commands he received from BR. He was the one in the group who took the initiative to ask the tutors if the numbers should be sorted in ascending or descending order. He organised his cards into stacks to separate the front colours but laid them out in front of him using the table space to find the lowest number, employing a selection-based sorting strategy to always place the next higher number into a sorted sub-deck in his hands. He used some minor insertion-based strategies to correct a missed number in his selection-based approach. He also used

control mechanisms and checked if the back colours were correctly separated in the unsorted sub-decks for the front colours. This indicates that he had learnt from the chaotic beginning and confusion while separating the back colours with TL.

During the discussion, he insisted on splitting the table space into two halves to avoid mixing already separated back colours again.

In the second sorting phase, the only thing he changed was the structure of his unsorted front colour sub-deck. While it was a clean array-like structure previously, there was no visible structure in the second attempt. This could suggest that he realised it was not too important how he laid out the cards in order to find the smallest number. Perhaps this was because, having become accustomed to the cards and the card deck structure during the second attempt, he could proceed in a somewhat unstructured manner to be faster.

Group3/Bottom Left: BL worked slowly during the initial sorting phase because he took each card into his hands sequentially. He held the cards with the front facing up, necessitating him to turn around every single card in order to separate the back colours. No optimisation strategy was evident in his approach to organising the data (cards). This was also apparent through the sorting process of the front cards. He laid them out on the table in an array-like structure, without considering the space required for the rest of the cards in the stack in his hands. He started sliding the cards upward individually when he tried to lay down a card and found no space remaining. This required a lot of time because he had to touch every single card again. His sorting strategy was mainly a selection-based approach with insertion processes with overlooked numbers.

During the discussion, BL remained quiet.

In the second sorting round, he managed to be more structured and learned from his previous attempt. At the beginning of the second sorting phase, he attempted to apply his learned algorithm during the intervention: merge sort. What we observed was that he decided shortly after starting with this algorithm to switch back to his selection-based approach.

Group3/Bottom Right: Compared to BL, BR began more efficiently by already turning the entire stack in his hands upside down to immediately see the back colours. He observed what the other group members were doing and gave short commands, drawing their attention to errors, such as a wrong back colour in the pile. Interestingly, he compared the numbers of his individual sorting task in his hands rather than laying them out on the table as the majority of the other participants did.

During the group discussion after the intervention, he assumed the leadership role. He did not explain how his intervention algorithm worked, but he described optimisation potential by using a stack in the hands and indicating in which direction the cards should face, such as back colours up when separating the back colours, but turning them around and placing them with the front colour up on the predefined location on the table. He explained that this reduced the workload afterwards and increased the speed because

they did not have to turn around all cards in a pile again. His explanations were not well articulated for the other children as he once said, "We had it, but then we forgot to think and did it wrong." From the observation in the video, it was not clear what he meant by "it," but he definitely tried to draw attention to a mistake in their general card-distribution strategy.

In the second sorting phase, he did not change major things. He continued checking the cards for errors and used a selection- and insertion-based approach in equal distribution, but no structure (such as BL3) was observable when he chose which strategy to employ. He intuitively switched between the two sorting strategies. One assumption would be that he memorised the cards already sorted and decided to insert a card he was currently browsing through in his in-hand stack rather than strictly searching for the next number.

Group 4

This group took initiative at the very beginning and became familiar with the entire card deck. They verbalised the need to be careful and not mix the different back and front colours before the first sorting round. They also identified that there would be eight sub-decks: one for each front colour. This was the only group to address this issue in advance and verbalised the exact amount of sub-tasks needed. They clearly outlined the task division: First, they would separate the different back colours to create two sub-decks. Then, they split into two pairs, each responsible for separating the front colours of one of the back colour sub-decks. After this, each person sorted two front colour sub-decks by their numbers. This group was by far the most advanced in terms of group strategy preparation according to the verbalisation of the process. They also began to refer to the back colours by their shape rather than the actual colour in which the design was printed. Another strategy they immediately adopted and also addressed in their discussion was using the table sides as a separation to prevent mixing the back colours again after turning them upside down to separate the front colours.

Even though this group considered many aspects beforehand, they still left a lot of things unsaid, e.g. how to sort the numbers or how to separate the front colours in pairs. One pair used a more efficient method by taking more cards at once in their hands and made four stacks, each for one front colour in their back colour sub-deck. The other pair always took one card after another and took, respectively, longer.

During the discussion after the intervention they explained to each other all four of the presented algorithms. They listened to every explanation and decided that bubble sort is the slowest option. The quick sort algorithm was also rejected for the same reason. They were not sure which strategy to adopt in the second round and revisited all four algorithms again. They discussed whether it would be faster to do a bucket sort approach or stick to the same strategy as before. At the end of the discussion round they decided that everyone could decide on their own whether they wanted to do the bucket sort or stick to their previous strategy.

Two of the four children decided to use the bucket sort approach in the second round and also told the other two during the sorting process that it really helps to be faster to make buckets. This is a good example of how they realised the efficiency and the difference between different sorting algorithms.

Student	Round 1 Strategy	Round 2 Strategy	Observed Change
Top Left	Focused, clean selection sort, corrected own mistake	Used buckets to group by tens digit, then sorted	Shift from flat array to bucket-based structure, more efficient
Top Right	Used selection with insertion for corrections, clear structure	Bucket sort approach, merged buckets correctly	Clear transfer from intervention; confident and fast execution
Bottom Left	Slightly hesitant selection-insertion mix, corrected mistake	Used bucket sort with less hesitation, clean sorting	Applied learned strategy independently; increased control
Bottom Right	Used selection-insertion mix, corrected forgotten cards	Same strategy with confidence, no major changes	Strategy stable; shows strong control, but no structural change

Table 6.5: Summary of individual strategy development – Group 4

Group4/Top Left: In the divide-and-conquer process at the beginning, TL took approximately 1/5 of the entire card deck with the backs up in her hands as a stack of cards. She placed card after card in the designated spot on the table with the corresponding back colour. Once she finished separating the cards in her hands, she took the initiative to take new cards from the middle. The topic of what to do if one finished their personal task was not addressed in the group discussion prior. This could be a sign of self-initiative to assist with the unseparated back colours from the middle of the table. When she started grouping the cards from one of the back colour sub-decks by front colours, she changed her strategy to be more efficient and took more than one card at a time from the stack in her hands to lay down on the table. This indicates a transfer from her earlier experience of separating the back colours, which motivated her to adapt a more efficient strategy.

While she tried to start the number sorting for one front colour sub-deck she talked to BL because she was confused that the numbers were not starting with one. The whole group needed to ask one of the tutors, but later understood that the numbers are not sequential. TL then laid out the front colour sub-deck on the table in front of her and started sorting them by putting the next higher number on a stack in her hands.

During the intervention she tried to explain merge sort but confessed that she did not understand it properly to be able to explain it to the others.

In the second sorting phase she used the same selection-based sorting strategy as before, but with more confidence, especially now knowing that the numbers are not sequential.

Group4/Top Right: The strategy of TR was nearly identical to TL. He separated the back colours as they had decided in the group strategy and began with a selection-sort-based approach, using the same card organising structures as TL. He laid out the cards visible on the table with the front colour facing up and always picked up the smallest visible into a stack in his hands. There was one mistake immediately corrected by swapping two neighbouring cards with each other in the hand.

During the group discussion, TR was more irritated than the others, showing this with more questions about merge sort and quick sort. Despite showing confusion in the discussion phase, especially around the more abstract elements of merge sort and quick sort, TR was able to transfer aspects of the discussed strategies into his own sorting approach in the second sorting phase. This indicates implicit understanding or at least the ability to adapt a strategy based on observation.

In the second sorting phase, TR implemented a bucket sort strategy instead of relying solely on the selection sort strategy. The decision to use a bucket sort approach in round two may have been influenced by peer explanations or the observable effectiveness of this strategy during the group discussion. While TR did not express full confidence in explaining the algorithms, he nonetheless adjusted his method to be more efficient.

Group4/Bottom Left: BL slightly disregarded the previously agreed group strategy, as they got in each other's way while separating the back colours (Figure 6.7). She started separating them in front of her to avoid the longer distance across the table to put the back colours in their respective place and did it with one move at the end. This is an efficient way of reducing movements and being faster. This strategy was adopted by the others in the second sorting phase. For the number sorting she used a selection sort-based strategy.

She actively participated in the group discussion after the intervention. She explained very clearly how a quick sort algorithm works and also suggested a good pivot element. She also grasped the idea of bucket sort very quickly and took over in the explanation process from the group member explaining this algorithm.

In the second sorting phase, she used a bucket sort approach with cards laid out on the table in small sub-stacks with one bucket for each tens digit place. She seemed very confident and secure in applying the newly learnt algorithm from her team.

Group4/Bottom Right: BR also adopted, as did all of his teammates, a selection sort-based approach in his individual number sorting task after separating the back colours. He was initially confused because the numbers were not sequential but understood it after a short time. He made some mistakes because he tried to be fast and had to move some cards in his already sorted stack.

During the group discussion, he asked questions about quick sort and stated that he was confused about it, though he understood bubble sort and explained it. Even though he understood bubble sort more, he acknowledged that it was not the most efficient

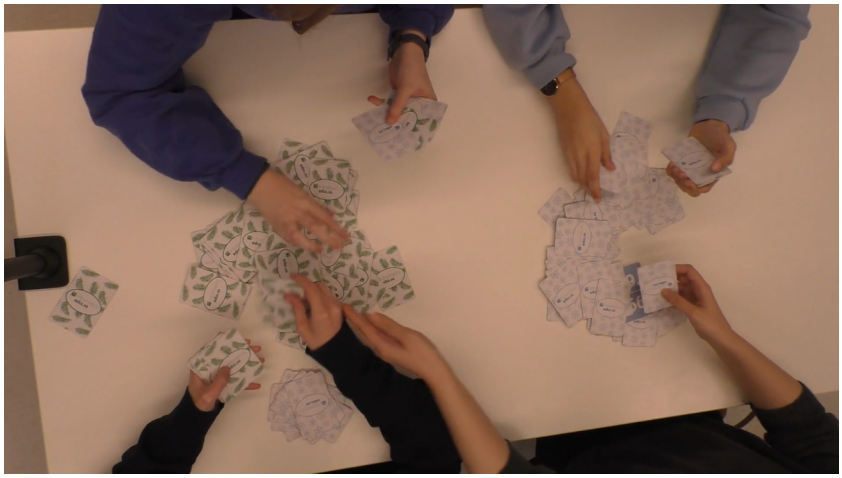

Figure 6.7: A group separating back colours on two different card piles. BL is separating the second back colour in front of her to avoid getting in each other's way.

algorithm. This could be a sign that even though he did not fully understand quick sort, he was able to follow the argumentation and also understand that the bubble sort approach is the least efficient in this card-sorting scenario.

In the second sorting phase, he applied the newly learnt bucket sort and was successfully faster than in the previous attempt. Regarding the card structure, we could not observe anything new in his strategy.

Group 5

Similar to Group 4, Group 5 began with a careful inspection of the card deck and quickly developed a well-coordinated and structured group strategy. They had already worked in a very structured way in the first sorting phase, for instance, dividing the entire card deck into four nearly equal sub-stacks for each team member to separate the back colours of these sub-stacks. They utilised the middle of the table as an invisible border to divide it into two sides to avoid mixing up the different back colours. They also discussed early on the order in which they would separate the backs, the fronts, and sort the numbers; however, they did not discuss the number sorting itself. Notably, they assisted each other more than the other groups did, particularly in the second sorting round, for example by taking over unfinished decks or helping others before being asked. This was evident through taking on additional cards and tasks without being asked for help by the group members. BL took especially many unsorted front colour sub-decks and received help at the end again when she was still sorting and the others were free. BR was the last to finish but had sorted more individual stacks than any other participant, which highlights his engagement and high contribution.

During the discussion between the two sorting phases, they discussed the algorithms presented. The conversations about these algorithms within the group were very nuanced,

involving numerous demonstrations and questions. Unfortunately, none of the newly learnt sorting algorithms was applied in the second sorting round. This might have been because the first round had already proceeded very well and was structured, and they probably feared being slower when applying a new algorithm. Therefore, they adhered to their tried-and-tested strategy: divide-and-conquer to separate backs and fronts, and then a selection-based sorting approach. As in Chapter 5, they did not explicitly address the sorting strategy.

Student	**Round 1 Strategy**	**Round 2 Strategy**	**Observed Change**
Top Left	Selection sort, laid cards on table; some hesitation visible	Same approach, faster and with more confidence	Familiarity increased performance; helped others
Top Right	Careful, asked tutors; clean selection sort with laid-out cards	Same strategy, helped more actively	Confidence increased; no algorithm change
Bottom Left	Started in-hand, then laid out cards; selection sort	Same strategy, began laying out cards earlier	Faster execution; no structural change
Bottom Right	Laid out cards and grouped small numbers first; pseudo-pivot strategy	Same as round 1; assisted others in the end	No strategic change; high productivity and engagement

Table 6.6: Summary of individual strategy development – Group 5

Group5/Top Left: TL followed the group's strategy proposed in the preparation phase. Similar to other members (e.g. BL2, TL3), TL in this group also laid the cards on the table and implemented a selection sort-based approach, consistently adding the next higher number into a sorted stack in her hands. It was not a clean execution, and there were some idle moments where she seemed to check what she had done and if it was correct.

During the discussion after the intervention, she explained the bubble sort algorithm very actively. Interestingly, she stayed very passive and quiet until the end of the discussion. As described above in the group strategy, TL did not change her strategy in the second phase. She was very fast and helped with two additional unsorted stacks of cards. Her speed could be explained by her now being less confused and more familiar with the sorting process.

Group5/Top Right: TR followed the group's strategy. In his individual number sorting task, he asked the tutors how the numbers should be sorted: ascending or descending. This showed that he really did not want to make a mistake before beginning to sort. He laid out the cards with the front colour up on the table and picked up the next highest number, exactly as TL.

During the discussion, he presented bucket sort and was enthusiastic about applying it. He announced that he would implement this algorithm in the next round. Interestingly, there was no observed bucket sort during the second sorting phase. He made improvements and participated even more in assisting others, but his selection sort-based approach did not change.

Group5/Bottom Left: Once more, BL followed the group's strategy. In her individual card-sorting task, she also employed a selection sort-based strategy, beginning by identifying the smallest number with the cards in her hands. After some time, she began laying out the cards on the table and constructing a sorted stack.

She explained the quick sort algorithm during the discussion phase and also defined what a pivot element is, but did not contribute to the discussion thereafter or offer input on what strategy she believed would be suitable.

In the second sorting round, her strategy remained the same, with the sole difference being that she immediately began to lay out the cards in her individual sorting task, saving time as a result. The strategy remained consistent, and she became faster, most likely due to training and routine from the first round.

Group5/Bottom Right: BR actively asked for more cards in the divide-and-conquer group strategy application in the beginning. He actively wanted to help and contribute to the group's success. His number sorting differed from the other nearly clean selection sort-based approaches: he took a few cards and placed them with the front colours up on the table, while already grouping smaller numbers (< 28) to one side of the table. Then he started sorting the small numbers and subsequently sorted the rest of the numbers. This was the only observed sorting strategy in the first round that involved the idea of a pivot element to split the 20 cards into two smaller subsets of numbers.

In the discussion, he tried to explain merge sort but failed. During the rest of the discussion, he remained passive but concentrated during the other algorithm explanations. There was no change in his strategy in the second sorting phase.

6.3 Discussion

Let us discuss RQ5 in light of the results obtained from these experiments: *How do students revise their initial strategies when encountering a similar unplugged sorting task?*

Our findings suggest that:

1. the structure of the intervention led to actual discussions within the groups about the different strategies;

2. the intervention triggered abstraction, evaluation and reflection;

3. the intervention led to a gradual shift of the sorting strategies from mainly selection sort to more sophisticated strategies such as bucket sort or merge sort.

Moreover, we also found indications that students' concept of *efficiency* led them to categorise *quick sort* as a rather inefficient sorting strategy.

6.3.1 Increase of Intra-Group Discussions

As a confirmation of the findings in Chapter 5, the students in this study also did not discuss strategies regarding the sorting process before the first round. The selection and insertion-based strategies occurred naturally without anyone talking about them in advance before the first sorting round. In contrast to this, they verbalised the algorithms after the intervention and also started talking about efficiency even without being asked to do so. This shows that our targeted CS Unplugged intervention supports students' verbalisation of their already existing intuitive strategies and discussion of them in a CS related context.

Another new aspect was the explanation of the algorithms. As every child in the group tried to explain the learnt algorithm, all of them took at least some part in the discussion. This is a sign that the idea of grouping the children into different algorithms in the intervention was good for engaging the discussion to involve everyone.

In summary, before the intervention, the students relied almost exclusively on intuitive selection and insertion strategies, which occurred spontaneously and without explicit discussion. The intervention led to an explicit mention and verbalisation of the algorithmic concepts.

6.3.2 Abstraction, Evaluation and Reflection

The aspect in which the intervention could have helped the children start evaluating and assessing their strategies is particularly relevant. Although they did not discuss *why* certain algorithms were executed slower than others, it is a base on which teachers can build to explain efficiency and optimisation. This shows that the intervention provided the opportunity to not just experience certain computer science concepts but also opens up a meta-view on those.

Students even started to address efficiency in their discussions, linking their strategies to ideas of being faster or using less effort. This shows that the intervention not only supported the evaluation of strategies, but also opened the door to more abstract considerations such as efficiency, which is discussed in Section 6.3.

The reason many of them stuck to their original strategy was also partly due to the new evaluation aspect: it would take more time to learn and explain something new than to continue with what they already knew. This reflects on general efficiency, though not in a desirable way of algorithm optimisation. Therefore, it adds more general problem

solving value of training optimisation and evaluation of a problem rather than directly connect it to algorithm optimisation.

Theme	Description
Bucket-Like Strategies	Separating cards into value ranges
Merge Strategies	Combining two sorted sub-stacks
Pivot/Divide-and-Conquer	Choosing a pivot card and splitting remaining cards around it
Strategy Evaluation	Reflecting on efficiency of one's chosen method
Strategy Rejection	Dropping a method if it feels inefficient
Joint Control Mechanisms	Double-checking order and correctness individually
Explicit Role-Taking	Acting as a "teacher" by explaining steps to others
Negotiation of Rules	Agreement of joint rules
Structured Use of Space	Arranging cards neatly into arrays or stacks with structural awareness

Table 6.7: Strategies observed after the intervention.

6.3.3 Shift of Sorting Strategies

Before the intervention, almost all participants relied on approaches similar to selection or insertion sort. Only one student showed something close to a bucket sort. When comparing the first and second sorting rounds, it becomes evident that bucket sort and merge sort were the only algorithms from the intervention that actually appeared in the students' strategies. In contrast, neither bubble sort nor quick sort was observed during their first, intuitive attempt, or the second round, after the intervention. This shift from initial intuitive strategies towards newly acquired concepts is shown in Figure 6.8. It is recognisable that that the intervention triggered the students to change their strategy rather than stick to their initial idea. This indicates that the targeted CS Unplugged intervention supports exploring new ideas leading the students to optimise their strategies.

Generally, the children used selection sort strategies in the first round. These were usually carried out in a clean and obvious way, easily observable. The students looked for the largest (or lowest) number. While there were differences in how they organised the cards on the table or in their hands, the overall idea of selection sort to 'find the next number' was clearly visible. In some cases, insertion sort strategies also appeared, often with the sorted output placed on the table.

Interestingly, some students mixed the two strategies in their first attempt (e.g. TR2, BL3). They applied insertion in two different ways: either by combining two strategies at once (TR2, BL3), or by using insertion for single elements when they noticed a mistake, such as a missing card (BR3). This mixing of strategies illustrates everyday logic and general problem-solving. It shows that combining approaches to reach a goal, or reflecting on mistakes, can occur even without additional training.

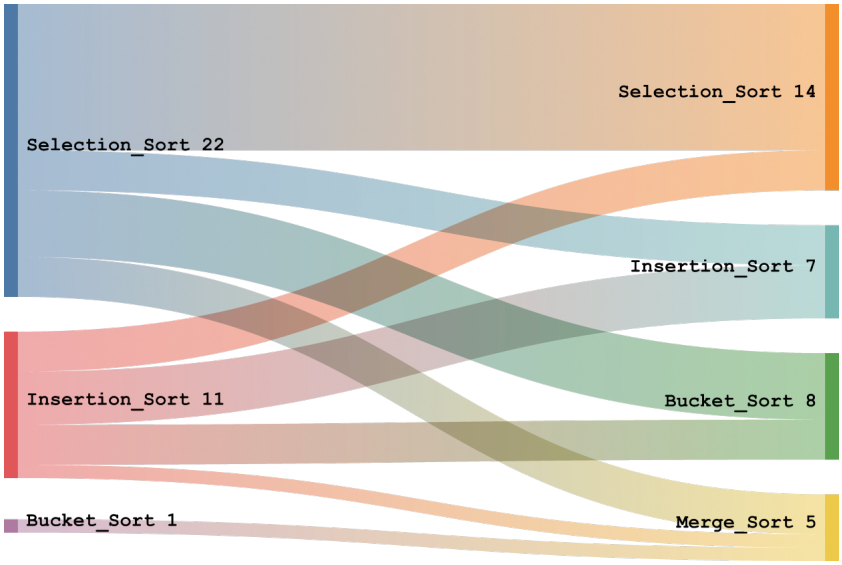

Figure 6.8: Change in pupils' sorting strategy from round 1 to round 2. Mixed strategies were counted individually.

Selection and insertion based strategies remained the most dominant, but they were increasingly adapted in the second round. Insertion was often used as a backup strategy to correct mistakes, and some students combined selection and insertion in flexible ways. This shows that intuitive routines were not replaced but refined and adjusted. Even when a pivot-like strategy appeared, it was mostly integrated into a selection-based process rather than standing alone. In one case, a student began with insertion-like handling of cards into an array structure on the table, but once the majority of cards were laid out, the strategy shifted towards a selection approach, showing adaptive switching between strategies.

Interestingly, TL1 had shown a version of a pivot element in her strategy before learning about the quick sort through her teammates. Analysing this, it is not entirely clear if this can be already counted as a quick sort-based strategy or a bucket sort strategy, or if this is a divide-and-conquer strategy on individual level. The procedure was to split the card deck into similar large half-games and to use selection sort afterward for each half. Although it is not a clean quick or bucket sort approach, this shows that the idea of reducing elements can improve efficiency in the sorting process by using smaller stacks instead. The concept of enhancing efficiency through dividing the problem into smaller problems, e.g. grouping the cards in buckets, is already present. TL1 also adjusted the pivot element in the second round from the number 40 to 50. This, again, shows that TL1 is evaluating the original strategy and found a way to optimise it.

Overall, the students did not just change their sorting strategies in the second attempt but also worked closer together. The collaboration between the students increased in the second attempt. It is not clear why exactly this happened, but we think one reason

might be that not everyone learnt about the same algorithm. This could lead to a setting where every student is needed to optimise the problem-solving.

6.3.4 The Case of Efficiency

It is customary in CS to express efficiency through the concept of big-O notation. However, this is mainly concerned with scalability and, due to omitted constant factors, might not always represent actual efficiency in concrete cases. In particular, more complex algorithms might incur an extra cost of cognitive load during their execution, leading to a higher 'constant'.

However, the children understood efficiency in a more practical way. The topic of efficiency was addressed directly by the children during the discussion, as it was integral to the competitive nature of the card sorting task. Most of them were able to (1) apply the newly learnt algorithm from the weights to the cards and (2) assess the efficiency of the algorithm. However, the children used a more practical notion: how many steps the procedure would take in their setting, how much space it needed on the table, how often they had to change the structure of the cards from stacks to arrays on the table and back, or how long it would take to explain to peers and learn it. None of the groups used quick sort and bubble sort in the second sorting round, even though they were part of the intervention (Table 6.7). This is a reminder that when children think about efficiency, they do it from their own real-world perspective. In this case, both bubble sort and quick sort ended up being seen as inefficient. In contrast, they understood that a bucket-like sort in their specific card application is the most appropriate, similar to merge sort.

It is noteworthy that our observations included some early hints at divide-and-conquer along the lines of quick sort. However, no group actually used quick sort, but all stopped just short of employing recursion. Recursion is indeed well known for being a difficult concept to master. This might have rendered the cognitive load and thus the cost of actually using quick sort for the entire pile just too high. Quick sort and bubble sort were therefore mostly put aside as impractical. In the case of bubble sort this is understandable, since it has the same run-time as selection and insertion sort but involves more steps in practice. Quick sort, however, should theoretically be faster. A possible reason why quick sort was rejected by all students is the missing transfer from the weights in the intervention to the card task. In addition, compared to a computer program, the manual execution of a quick sort with cards requires much more physical space on the table, which could also have contributed to its rejection. While quick sort is based on recursion, merge sort can be understood as iterative, which may explain why merge sort was preferred. Another reason can be found from the perspective of cognitive load. Quick sort requires learners to keep track of multiple steps at once: choosing a pivot, separating cards into subgroups, and later merging them again, which exceeds their intuitive strategies in several aspects. While these actions are easily manageable and programmable for a computer, the simultaneous coordination of them in a manual setting while learning a new algorithmic concept creates high demands on the children's working memory. According to cognitive load theory [84], it is likely that these demands exceeded the children's

cognitive resources in the task situation. It makes quick sort appear infeasible, leading them to prefer simpler, more familiar strategies such as selection or insertion.

The role of efficiency was very present during the students' discussions. This was also led by the task itself, as their goal was to be faster in the second sorting attempt. As mentioned above, the optimisation of the students' second attempt sorting is not represented in algorithm runtimes, but more in the overall aspects of practical efficiency, considering the environment and the human factors in organising cards in hands and on the table.

6.4 Conclusion

Finding these new strategies, building on intuitive ones, shows that through the intervention learners did not just receive knowledge. They connected new ideas with their existing intuitive strategies and skills, which helped them to approach more formal algorithmic concepts through more precise verbalisation as well as application in the second sorting attempt. As we were aiming to make this transfer visible through our intervention and video analysis, we see it as a success that our explorative approach can set the basis for such algorithmic ideas. The transfer happened as the children mapped newly learnt concepts to processes they already knew (e.g. intuitive selection sorting) and adapted them.

Similar connections between intuition and formal concepts could also be applied in areas of algorithms, such as search algorithms. Our results therefore hint for a more general potential that CS Unplugged intervention can serve as a bridge between everyday logic and CS concepts.

The knowledge we gained from the study may seem simple at first glance, but we have filled an important gap here: we now know that there are CS concepts that are applied intuitively – without prior explanation – and that we can successfully produce a transfer to let the children experience broader CS concepts using CS Unplugged. We know from previous work that CS Unplugged is suggested to be used as an introduction to topics and that its effectiveness depends on how the teacher picks up the ideas [9] and continues their lessons on a topic. We underscore this assumption with our findings, but we want to go further: showing the change in the students' discussions about efficiency showed that the main ideas of CS, such as optimisation, are already natural to the children.

This further underscores that the competency of teachers is crucial, as they have to be able to pick up these spontaneous ideas of children during their lesson. With more work towards the explorative approach to let children build their own experience and opinion first, we can lay the groundwork for formal CS concepts and learn more about their intuitive and natural strategies. By doing so, we might be able to bridge the gap between children's intuitive approaches and the development of formal algorithmic reasoning. This connection can be made by relating their everyday strategies of trial and error, optimisation, and revision to structured CS concepts.

The CS Unplugged intervention we added to our activity successfully triggered abstraction, evaluation, and reflection. These are not trivial outcomes, as they represent the first steps towards meta-cognitive engagement with algorithms and CT [75], even in short outreach settings. Another great opportunity for learning is the aspect of data structures occurring through the discussion of and the application of how to organise cards. Abstracting these strategies and operations, they offer new possibilities in future K-12 classrooms, like introducing data structures, especially arrays. In long term it would therefore be interesting to investigate how stable these effects are: do the newly acquired strategies stay in later learning phases, and can they be applied to new contexts?

Building on this, our findings have several implications for CS education in general:

- Unplugged activities should *not* only be used to demonstrate algorithmic concepts, but also to make them visible and connect with the intuitive strategies that children already employ. This can help to make the transfer to formal CS concepts more smoothly.
- Introducing formal CS concepts directly can risk cognitive overload. Our study shows, however, that even partial transfer occurs without prior formal instruction. Allowing learners to work with their intuitive concepts first may therefore ease the later connection to formal concepts and reduce the risk of overload.
- The observed discussions about efficiency suggest that unplugged activities can foster reflection on algorithmic properties in a way that is both tangible and engaging for students. This opens opportunities to build further connections to formal CS concepts in future teaching.

CHAPTER 7

Overall Discussion

This chapter revisits the findings and results of the overall thesis and we take a step back from the individual studies and look at them together. The three main studies of this thesis (Chapter 4, 5 and 6) explored different perspectives on how children address algorithmic problems: (1) first by exploring their perceptions of the term 'algorithm' (Chapter 4), (2) second by analysing their intuitive algorithmic problem solving strategies (Chapter 5) and (3) third by observing how their strategies change when we include an additional CS Unplugged intervention (Chapter 6).

Looking at the big picture of these studies together, they provide a more complete picture than any one individual can offer on its own. We moved from everyday concepts and the notion students bring with them with learning about algorithms, through the strategies naturally occurring while intuitively applying algorithms, and the learning transfer that happens after a short and carefully targeted intervention.

The goal of this chapter is to connect the results and findings and discuss them in a broader sense. We will also add potential practical applications in CS education and answer the research questions.

7.1 Implications for Teaching and Learning

Starting with the findings from the more practical aspect of this work, we extend the conceptual findings of this thesis to practical implications for CS education, especially CS Classrooms. We would like to highlight implications for teachers and their CS lessons and suggest which applications are best suited for using the material developed in this work. This includes the card sorting activity (Section 5.3), but also the sorting intervention (Section 6.1), connected to the newly gained insights of this work. These suggestions are not limited to K-12 classrooms, as they can also be used for CS in higher education institutions.

7.1.1 Applications of the Card-Sorting Activity

Low Entry Threshold. Children associate algorithms with their everyday life experiences, e.g. connections to traffic lights or brushing their teeth, as we found in Chapter 4. Further, we found that naturally, i.e. without prior instruction, children tend to use selection- and insertion-based sorting strategies. These findings imply that there are already existing experiences that we can directly address and name from their lives (e.g. 'traffic light'), as well as certain behaviours that cannot be addressed by a name, but can be described and connected with further information (e.g. unnamed selection-sort behaviour). These existing links and behaviours make it possible to introduce algorithmic concepts at a very low threshold as detailed below.

Scaffolding. The card sorting activity can be used by teachers to build scaffolds on these unnamed intuitive strategies and everyday experiences. By creating such experiences through the task, teachers can explicitly connect them to formal concepts. In this way, the activity supports a movement from *intuitive* to *algorithmic*, helping learners to expand and stabilise their strategies through guided steps.

Bridging Practice and Misconceptions. Teachers can also integrate unplugged interventions directly into the school curriculum. A good example is the sorting task, which in our study (Section 6.1) showed that even a very short intervention already led students to revise and improve their strategies. Such tasks can serve as a preparation for programming lessons, where the observed strategies are then connected to formal sorting algorithms in code. At the same time, these interventions open the possibility to address misconceptions. As we have seen in Section 4.4, many children strongly connect algorithms with the idea of repetition, while aspects like finiteness or input-output are hardly mentioned. In class, teachers can contrast these everyday notions with more canonical definitions (Table 2.1), and in this way make clear how formal understanding differs but also builds on what students already bring with them.

7.1.2 Implications for Teachers

As we showed in Section 5.5, open-ended CS Unplugged tasks can offer a variety of topics and concepts connected to children's intuition. This aligns with the constructivist idea of letting children explore. Teachers should resist giving too much instruction initially and let strategies emerge.

These emerging strategies give an opportunity to connect formal concepts, e.g. starting programming in school. Nevertheless, it should be considered to not use activities like the card sorting activity as learning method itself. Problem solving itself is not considered a good learning method [83], but rather an opportunity to build experiences and to further rely on them in the lessons.

Balancing the cognitive load is another important implication for teachers. As discussed in Section 5.2, Cognitive Load Theory reminds us that working memory has strict limits,

especially when several elements must be processed at the same time. If tasks are too complex right from the start, learners can become overwhelmed and lose focus on the actual learning goal. A way to handle this is to provide stepwise scaffolding, for example, by reducing the number of different concepts used in the sorting task, as we did in the primary school workshop ([50]). We reduced the number of cards so only one back colour was left. In this way, the *divide and conquer* process and the coordination of the parallelisation was reduced. Such scaffolds lower the extraneous load and make it possible for children to still engage with the core problem-solving process without being overloaded.

Teachers can also facilitate structured reflection after tasks to consolidate learning. These reflections can happen through short, targeted interventions within a learning session. For example, while programming sorting algorithms, a hands-on activity in the middle of the lesson can trigger new strategies and move students further in their thinking. We observed this effect in Section 6.3, where a short unplugged intervention encouraged groups to revise their approach. We therefore suggest that the idea of combining hands-on interventions with reflective phases can be effective in other settings and topics as well. However, further research is needed to examine under which conditions such combinations are most beneficial and how sustainable their impact is.

7.2 Addressing Research Questions In the Big Picture

7.2.1 RQ1: Perception of Algorithms

Students strongly connect algorithms with repetition Section 4.5. The concept of repetition arises in two different ways: repeating the whole procedure, or using repetition as a construct within an algorithm. This shows that children compare the term and their understanding of it primarily with concepts that are more familiar to them. Repetition seems to be something more graspable than the more abstract properties of algorithms. This suggests that, while teaching algorithms, it may be easier for children to first understand loops and repeating processes.

In addition, everyday examples dominate their perceptions. This phenomenon is also visible in Section 4.5, though there it appears more in behaviour than in articulation. Certain concepts, such as selection or insertion sort, are naturally applied and therefore might be easier to grasp for them. We need to be careful not to oversimplify concepts in our teaching methods, as we risk losing important aspects of other algorithmic ideas.

Comparing the dominance of repetition in the first study (Chapter 4), it might also be the very repetitive character of selection and insertion sort that makes these strategies more attractive to apply, rather than considering other approaches.

This observation also connects to RQ2, which focuses more directly on how students' conceptions differ from formal definitions.

7.2.2 RQ2: Conceptions Versus Established Definitions

Only few formal aspects appeared in the results of the first study. Pupils rarely mentioned termination and correctness, which may indicate that infinity and endless processes are not part of their everyday thinking. This might seem as a contrast to the algorithm property of finiteness, but students understand that algorithms consist of fixed sequences, but they often interpret them as continuous or recurring processes.

There was uncertainty about the executing actor of algorithms. Some of the students described algorithms as entities that perform actions on their own while others perceived them as instructions to be followed by a human.

What we learn from this is that teachers must choose analogies carefully. Explicitly discussing and addressing children's experiences can strengthen their understanding. Clarifying the distinction between everyday routines and formal algorithms may help students build a more conceptually accurate understanding of what defines an algorithm.

7.2.3 RQ3: Intuitive Problem-Solving Strategies

The strategies that occurred naturally without prior instruction were most commonly selection, insertion, merge-like, trial-and-error, and parallelisation strategies (Chapter 5). These strategies resembled well-known sorting algorithms, such as selection and insertion sort, without having been taught them. Other strategies were more heuristic in nature, for example, trial-and-error and random comparisons.

Group dynamics also shaped the problem solving process. We observed the delegation of roles, negotiations, conflicts, and emerging leadership. These roles appeared naturally, often with one student coordinating the parallelisation process and instructing others how to distribute the subtasks.

Many actions stemmed from everyday reasoning, such as dividing the stack and distributing cards to the group. These strategies can be mapped onto CS concepts, but they were not explicitly named by students, which was expected given the lack of expert vocabulary.

The implications for CS classrooms are that teachers can build bridges from intuitive to formal concepts by giving explicit guidance and naming these strategies and by comparing them to well-known algorithms. Group work also offers unique opportunities for externalising thought processes that would remain hidden in individual tasks. Finally, collaborative analysis of strategies in class discussions provides another opportunity to make student reasoning visible and link it to algorithmic concepts.

In addition, the observed strategies were flexible and adaptive. Participants often switched between approaches or combined them within a single sequence. This suggests that algorithmic thinking, as observed in our study, is less about rigid procedures and more about structuring purposeful actions in problem-solving. However, while flexibility is essential for human reasoning, machines need precise and rigid procedures for execution.

7.2.4 RQ4: Strategy Changes Through Repeated Tasks

The intervention led to revisions of the initial strategies. Comparing these findings to the results from RQ3, it becomes clear that intuitive strategies can be enhanced with guidance from peers and tutors, which aligns with the idea of keeping students in their zone of proximal development [93].

Moreover, the tendency to build on what had worked in the first sorting task round suggests a shift from unstructured experimentation to increasingly systematic thinking. Although students rarely verbally expressed these processes, their actions became more deliberate and coordinated in the second round.

Efficiency gains were visible in both the completion times and in smoother coordination. Some groups optimised their approaches to avoid earlier mistakes, showing signs of evaluation and optimisation as basic concepts of CT.

7.2.5 RQ5: Strategy Transfer to Similar Tasks

While RQ4 focused on repeating the same task, RQ5 addresses transfer to a similar but not identical problem.

Students also showed a tendency to transfer elements of the demonstrated algorithm, even though often only in a simplified form or as partial adaptations. This transition indicates that pupils can take up algorithmic principles already after a short exposure to a similar, but not identical, problem. It also suggests that complicated concepts can be scaffolded with short hands-on interventions. The combination of training, intervention, and repetition can lead to a transfer of new learning, even when the formal concepts were never introduced.

Our findings support the idea that short CS Unplugged interventions can spark visible improvements, but sustained change requires repeated exposure. Structured reflection and multiple rounds are crucial to developing a stable learning transfer.

7.2.6 Overall Research Question

The five research questions together address the overall research aim formulated in Chapter 1.

To address our overall research question, “How do students address algorithmic problems in computer science?”, we carried out three individual studies. Our findings show that children perceive and connect the term algorithm mainly with everyday notions such as routines, with repetition as the dominant feature. In parallel to these perceptions, we also discovered the non-articulated, intuitive application of algorithmic concepts through general problem solving strategies that resemble algorithmic patterns. When adapting and refining their strategies after a CS unplugged intervention, we observed that transfer happens and that more complex algorithmic strategies can be applied.

To answer this general question according to our findings, students have initial ideas about algorithms that are strongly connected to their everyday lives and experiences. In our second study, we confirmed that certain concepts such as selection sort or parallelisation come more naturally than others, even if they are not explicitly articulated. In the third study, we saw that children are also able to revise and adapt their strategies based on short interventions.

These findings suggest that teaching should acknowledge what children already know and should use learning activities or short interventions to make implicit and intuitive strategies explicit. In this way, learning can build on children's everyday reasoning, while also guiding them towards a more formal understanding of algorithmic concepts.

7.3 Limitations

Given the small number of children involved in this study, due to its qualitative nature, and especially since video analysis is a time-consuming method, the findings are not intended to be generalised. Instead, the studies aim to provide an in-depth exploration of students' intuitive understanding and a reliable source of intuitive strategies.

While the qualitative approach allowed us to gain rich insights into how children address algorithmic problems, it also limits the scope to a small number of groups. Furthermore, the analysis was based on a single task and setting, which may not capture all possible variations of intuitive strategies. These limitations, however, are inherent to qualitative exploratory research and were accepted in favour of capturing authentic and detailed learning processes.

The outreach setting also resulted in a group of participants that was neither intentionally diverse nor homogeneous. As participation required parental consent, it is possible that families with a stronger interest in academic activities or higher educational backgrounds were more likely to provide consent. We deliberately did not collect data on the educational background of the participating children's households, in order to keep the consent process as simple and non-intrusive as possible for parents and children.

The bird's-eye perspective of the video recordings limited the visibility of students' facial expressions. This setup was intentionally chosen to respect participants' privacy while ensuring a clear view of their actions during the task. Our focus was on their problem-solving behaviour rather than their emotional reactions, although facial expressions and social dynamics within the groups could provide additional insights in future analyses.

During the third study, we used an additional CS Unplugged activity in which each group member learned about a different sorting algorithm. Since the tutors explained these algorithms repeatedly across different workshop cycles, slight variations in their explanations may have influenced students' behaviour in the subsequent tasks.

CHAPTER 8

Conclusion and Outlook

This thesis was not only meant to reveal the perception of algorithms, intuitive strategies, and how transfer to students' application of sorting algorithms can happen, but also to open the stage for further research and classroom applications. As our findings support the idea of scaffolding using CS Unplugged activities, we also gained new insights into how students learn about algorithms. For example, by letting them first explore the concept on their own, we give them the opportunity to make their own experiences, which include also minor struggles. These struggles and initial experiences can then be addressed by a targeted intervention.

In our case, we used a different CS Unplugged activity that worked as an intervention to broaden and extend the ideas that the students already had. Using weights and a balance scale to extend the card sorting activity, we addressed a similar problem, but in a different setting. We were able to show with our qualitative analysis that it is possible to use a CS Unplugged sorting intervention to extend the sorting strategies used by learners. With this approach, we successfully managed to produce a learning transfer from one sorting problem, where only basic sorting algorithms and problem-solving strategies naturally appeared, to another similar sorting problem through a guided unplugged activity. After introducing the children to more complex sorting algorithms with the balance scale, they were able to use their newly acquired knowledge and transfer it back to the card-based task. Although long-term effects were not measured, we still found positive effects of short-term CS Unplugged interventions in an outside-classroom outreach setting.

Our findings also show that selection- and insertion-sort are the most natural and intuitive approaches during sorting, whereas other CS problem-solving concepts such as parallelisation, divide-and-conquer, data structures, optimisation, and evaluation are in more universal situation and other domains applicable, and therefore seem easier to grasp during explorative learning. We therefore should be careful when introducing more complex algorithms, such as quick-sort, and make sure to build on already intuitive approaches.

Building on this, teachers can use explorative tasks at the beginning and continue with more targeted and structured tasks addressing similar problems but from a different conceptual angle. We believe that the ideas and findings of our research can be adapted to other concepts in CS to successfully introduce new ideas into K–12 classrooms.

The work presented here offers several promising directions for future development in both research and practice.

Other CS Topics. Intuitive strategies as observed in this study can be further investigated and validated in a variety of computer science contexts beyond sorting. Exploring similar patterns in problems such as searching, decision-making, or control structures could deepen our understanding of how students intuitively apply algorithmic reasoning in different domains. Such research could also connect intuitive problem solving more directly to fostering Computational Thinking skills.

Applications in Programming Education. The connection between CS Unplugged activities and formal instruction offers many opportunities for classroom integration. Linking unplugged sorting tasks with programming lessons, for example, can help students bridge the gap between their intuitive reasoning and the abstract logic of code. Especially in teacher education, long-term studies could explore how teachers design and adapt such transitions in their classrooms.

Long-Term Effects. It would be valuable to examine how short-term unplugged learning experiences relate to long-term conceptual growth. While this thesis focused on immediate learning transfer, future research could explore how repeated or extended exposure influences students' sustained understanding of algorithmic ideas. Replication studies conducted by teachers themselves could offer authentic perspectives from real classroom practice.

Explorative Learning with CS Unplugged. The design of creative and open-ended learning materials remains a central opportunity. Building on the intuitive strategies identified in this thesis, future K–12 teaching materials can further connect exploratory unplugged activities with formal computing concepts. By starting from learners' natural reasoning processes, educators can reduce cognitive load and enable smoother transitions towards programming, data structures, or even machine learning.

Together, these directions show that there is much more to explore: how students think, act, and communicate when solving algorithmic problems offers a rich foundation for further educational innovation. Our work presented here can serve as a starting point for a broader movement towards integrating intuitive, hands-on, and reflective approaches into computer science education.

CHAPTER 9

References

Bibliography

[1] A. V. Aho. Computation and computational thinking. *The computer journal*, 55(7):832–835, 2012.

[2] E. Aivaloglou and F. Hermans. Early Programming Education and Career Orientation. In E. K. Hawthorne, editor, *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ACM Conferences, pages 679–685, New York, NY, USA, 2019. ACM.

[3] Y. Allsop, F. Kalelioglu, and M. Aslan Unlu. Using Card Sorting Activity as a Strategy for Evaluating Students' Learning of Computational Thinking Concepts. *International Journal of Computer Science Education in Schools*, 6(4), 2024.

[4] L. W. Anderson and D. Krathwohl, editors. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman, New York, complete ed. edition, 2001.

[5] C. Angeli and M. Giannakos. Computational thinking education: Issues and challenges. *Computers in Human Behavior*, 105:106185, 2020.

[6] S. Barab. *Design-Based Research: A Methodological Toolkit for Engineering Change*, page 177–195. Cambridge Handbooks in Psychology. Cambridge University Press, 2022.

[7] T. Bell, J. Alexander, I. Freeman, and M. Grimley. Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1):20–29, 2009.

[8] T. Bell, P. Curzon, Q. Cutts, V. Dagiene, and B. Haberman. Overcoming Obstacles to CS Education by Using Non-programming Outreach Programmes. In I. Kalaš and R. T. Mittermeir, editors, *Informatics in Schools. Contributing to 21st Century Education*, volume 7013 of *Lecture notes in computer science*, pages 71–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[9] T. Bell and J. Vahrenhold. CS Unplugged—How Is It Used, and Does It Work? In H.-J. Böckenhauer, D. Komm, and W. Unger, editors, *Adventures between lower bounds and higher altitudes: essays dedicated to Juraj Hromkovič on the occasion of*

his 60th birthday, volume 11011 of *Lecture notes in computer science*, pages 497–521. Springer, Cham, 2018.

[10] T. Bell, I. Witten, and M. Fellows. *CS Unplugged: An enrichment and extension programme for primary-aged students*. 2015.

[11] C. Bellettini, V. Lonati, M. Monga, and A. Morpurgo. To Be Or Not To Be . . . An Algorithm: The Notion According to Students and Teachers. In B. Stephenson, J. A. Stone, L. Battestilli, S. A. Rebelsky, and L. Shoop, editors, *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 102–108, New York, NY, USA, 2024. ACM.

[12] R. Bohnsack. *Qualitative Bild- und Videointerpretation: Die dokumentarische Methode*, volume 8407 of *UTB Erziehungswissenschaft, Sozialwissenschaften*. Budrich, Opladen, 2 edition, 2011.

[13] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006.

[14] J. Bruner. *Toward a Theory of Instruction*. Belknap Press. Harvard University Press, 1966.

[15] J. S. Bruner. *The Process of Education*. Harvard University Press, 1960.

[16] K. J. Carbonneau, S. C. Marley, and J. P. Selig. A meta-analysis of the efficacy of teaching mathematics with concrete manipulatives. *Journal of Educational Psychology*, 105(2):380–400, 2013.

[17] L.-C. Chang, H.-R. Lin, and J.-W. Lin. Learning motivation, outcomes, and anxiety in programming courses—A computational thinking–centered method. *Education and Information Technologies*, 29(1):545–569, 2024.

[18] P. Chen, D. Yang, A. H. S. Metwally, J. Lavonen, and X. Wang. Fostering computational thinking through unplugged activities: A systematic literature review and meta-analysis. *International Journal of STEM Education*, 10(1):1–25, 2023.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, Cambridge, Massachusett, fourth edition edition, 2022.

[20] P. Curzon, T. Bell, J. Waite, and M. Dorling. *Computational Thinking*, page 513–546. Cambridge Handbooks in Psychology. Cambridge University Press, 2019.

[21] V. Dagienė and G. Futschek. Bebras International Contest on Informatics and Computer Literacy: Criteria for Good Tasks. pages 19–30. Springer, Berlin, Heidelberg, 2008.

[22] J. Dewey. *Experience and Education*. Touchstone, New York, 1997. Originally published 1938.

[23] B. Du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[24] B. Editors. Al-Khwarizmi | Biography & Facts. *Encyclopedia Britannica*, 20.07.1998.

[25] S. Fincher, J. Jeuring, C. S. Miller, P. Donaldson, B. du Boulay, M. Hauswirth, A. Hellas, F. Hermans, C. Lewis, A. Mühling, J. L. Pearce, and A. Petersen. Capturing and Characterising Notional Machines. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 502–503, New York, NY, USA, 2020. Association for Computing Machinery.

[26] M. Forišek and M. Steinová. Metaphors and analogies for teaching algorithms. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 15–20, New York, NY, USA, 2012. Association for Computing Machinery.

[27] G. Futschek. Algorithmic Thinking: The Key for Understanding Computer Science. In R. T. Mittermeir, editor, *Informatics education - the bridge between using and understanding computers*, volume 4226 of *Lecture notes in computer science*, pages 159–168. Springer, Berlin and Heidelberg, 2006.

[28] J. Gallenbacher. *Abenteuer Informatik: IT zum Anfassen - von Routenplaner bis Online-Banking*. Spektrum-Sachbuch. Spektrum, Akad. Verl., Heidelberg, 2 edition, 2008.

[29] S. Grover. Learning to Code Isn't Enough. *EdSurge*, 29.05.2013.

[30] S. Grover and R. Pea. Computational Thinking in K–12. *Educational Researcher*, 42(1):38–43, 2013.

[31] S. Grover and R. Pea. Computational Thinking: A Competency Whose Time Has Come. In S. Sentance, E. Barendsen, and C. Schulte, editors, *Computer science education*. Bloomsbury Academic, London and New York, 2018.

[32] H.-P. Gumm. *Einführung in die Informatik*. Oldenbourg Verlag, München, 10., vollst. überarb. aufl. edition, 2013.

[33] M. Guzdial, A. Kay, C. Norris, and E. Soloway. Computational thinking should just be good thinking. *Communications of the ACM*, 62(11):28–30, 2019.

[34] H. Herold, B. Lurz, and J. Wohlrab. *Grundlagen der Informatik*. Pearson Higher Education, München, 2 edition, 2012.

[35] M. L. Hetland. *Python algorithms: Mastering basic algorithms in the Python language, second edition*. The expert's voice in Python. Friends of ED/Apress, [New York], 2nd ed. edition, 2014.

[36] R. K. Hill. What an algorithm is. *Philosophy & Technology*, 29:35–59, 2016.

[37] J. Hromkovič. *Berechenbarkeit: Logik, Argumentation, Rechner und Assembler, Unendlichkeit, Grenzen der Automatisierbarkeit ; Lehrbuch für Unterricht und Selbststudium*. Studium. Vieweg + Teubner, Wiesbaden, 1 edition, 2011.

[38] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.

[39] Y. Kafai, C. Proctor, and D. Lui. From Theory Bias to Theory Dialogue. In R. McCartney, A. Petersen, A. Robins, and A. Moskal, editors, *ICER'19*, pages 101–109, New York, NY, USA, 2019. The Association for Computing Machinery.

[40] D. E. Knuth. Computer programming as an art. *Communications of the ACM*, 17(12):667–673, 1974.

[41] D. E. Knuth. *The art of computer programming. Volume 1, Fundamental algorithms*. Addison Wesley, 3rd ed. edition, 1997.

[42] T. Kohn and D. Komm. Teaching Programming and Algorithmic Complexity with Tangible Machines. In S. N. Pozdniakov and V. Dagienė, editors, *Informatics in Schools. Fundamentals of Computer Science and Software Engineering*, pages 68–83, Cham, 2018. Springer International Publishing.

[43] D. A. Kolb. *Experimental learning: Experience as the source of learning and development*. Prentice-Hall, Englewood Cliffs, N.J., 1984.

[44] A. N. Kumar, R. K. Raj, S. G. Aly, M. D. Anderson, B. A. Becker, R. L. Blumenthal, E. Eaton, S. L. Epstein, M. Goldweber, P. Jalote, D. Lea, M. Oudshoorn, M. Pias, S. Reiser, C. Servin, R. Simha, T. Winters, and Q. Xiang. *Computer Science Curricula 2023*. ACM, New York, NY, USA, 2024.

[45] M. Landman. Emergent Algorithmic Strategies in an Unplugged Group Task: Insights from a Collaborative Sorting Task with Cards. (submitted and under review).

[46] M. Landman. Probleme algorithmisch lösen lernen. *OCG Journal*, 49(1):32–33, 2024.

[47] M. Landman, G. Futschek, S. Unkovic, and F. Voboril. Initial Learning of Textual Programming at School: Evolution of Outreach Activities. *Olympiads in Informatics*, pages 43–53, 2022.

[48] M. Landman and T. Kohn. "Something that Happens Each Day" - Students' Explanations of What Algorithms Are. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, page 199–205, New York, NY, USA, 2024. Association for Computing Machinery.

[49] M. Landman and L. Lehner. Vom Schlagwort zur Praxis: Computational Thinking in der Theorie und im eduLAB. *OCG Journal*, 49(4):24–26, 2024.

[50] M. Landman, S. Rain, L. Kovács, and G. Futschek. Reshaping Unplugged Computer Science Workshops for Primary School Education. In J.-P. Pellet and G. Parriaux, editors, *Informatics in Schools. Beyond Bits and Bytes: Nurturing Informatics Intelligence in Education. (ISSEP2023)*, volume 14296 of *Lecture notes in computer science*, pages 139–151. Springer, Cham, 2023.

[51] L. Lehner and M. Landman. Unplugged Decision Tree Learning – A Learning Activity for Machine Learning Education in K-12. In H. Fernau, I. Schwank, and J. Staub, editors, *Creative Mathematical Sciences Communication*, volume 15229, pages 50–65, Cham, 2024. Springer Nature.

[52] M. Lenke, L. Lehner, and M. Landman. "I'm actually more interested in AI than in Computer Science" - 12-year-olds describing their first ecnounter with AI. In *2025 IEEE Global Engineering Education Conference (EDUCON)*. 2025.

[53] Linda Werner, Jill Denner, Shannon Campe, and Damon Chizuru Kawamoto. *SIGCSE'12 Proceedings of the 43ed ACM Technical: Symposium on Computer Science Education Februray 29-March 3, 2012 Raleigh, North Carolina*. Association for Computing Machinery, Danversm MA, 2011.

[54] K. Lum and R. Chowdhury. What is an "algorithm"? It depends whom you ask. *MIT Technology Review*, 26.02.2021.

[55] L. Mannila, V. Dagiene, B. Demo, N. Grgurina, C. Mirolo, L. Rolandsson, and A. Settle. Computational Thinking in K-9 Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, ITiCSE-WGR '14, page 1–29, New York, NY, USA, 2014. Association for Computing Machinery.

[56] P. Mayring. *Qualitative content analysis: A step-by-step guide*. SAGE Publications Ltd, Thousand Oaks, 1st edition edition, 2022.

[57] M. M. McGill, A. Decker, and A. Settle. Does Outreach Impact Choices of Major for Underrepresented Undergraduate Students? In B. Dorn, editor, *Proceedings of the eleventh annual International Conference on International Computing Education Research*, ACM Digital Library, pages 71–80, New York, NY, USA, 2015. ACM.

[58] S. McKenney and T. C. Reeves. Educational Design Research. In *Handbook of Research on Educational Communications and Technology*, pages 131–140. Springer, New York, NY, USA, 2014.

[59] Merriam-Webster. Algorithm, 2023.

[60] B. D. Mittelstadt, P. Allo, M. Taddeo, S. Wachter, and L. Floridi. The ethics of algorithms: Mapping the debate. *Big Data & Society*, 3(2):1–21, 2016.

[61] C. Moore and S. Mertens. *The nature of computation*. Oxford University Press, Oxford, 2011.

[62] B. Munasinghe, T. Bell, and A. Robins. Computational Thinking and Notional Machines: The Missing Link. *ACM Trans. Comput. Educ.*, 23(4), Dec. 2023.

[63] G. L. Nelson and A. J. Ko. On Use of Theory in Computing Education Research. In L. Malmi, A. Korhonen, R. McCartney, and A. Petersen, editors, *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 31–39, New York, NY, USA, 2018. ACM.

[64] K. Nolan, R. Faherty, K. Quille, B. A. Becker, and S. Bergin. Developing an Inclusive K-12 Outreach Model. In M. Giannakos, editor, *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ACM Digital Library, pages 145–151, New York,NY,USA, 2020. Association for Computing Machinery.

[65] T. Palts and M. Pedaste. A Model for Developing Computational Thinking Skills. *Informatics in Education - An International Journal*, 19(1):113–128, 2020.

[66] S. Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York, NY, USA, 2nd edition edition, 1993.

[67] J. Perrenet, J. F. Groote, and E. Kaasenbrood. Exploring students' understanding of the concept of algorithm: levels of abstraction. *ACM SIGCSE Bulletin*, 37(3):64–68, 2005.

[68] J. Piaget. *Science of education and the psychology of the child*. Orion Press, New York, 1970.

[69] B. Rittle-Johnson. Iterative development of conceptual and procedural knowledge in mathematics learning and instruction. In J. Dunlosky and K. A. Rawson, editors, *The Cambridge handbook of cognition and education*, pages 124–147. Cambridge University Press, Cambridge, 2019.

[70] B. Rodriguez, S. Kennicutt, C. Rader, and T. Camp. Assessing Computational Thinking in CS Unplugged Activities. In M. E. Caspersen, editor, *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ACM Conferences, pages 501–506, New York,NY,USA, 2017. Association for Computing Machinery.

[71] M. Sabin, P. Snow, and M. Laturnau. Evaluation of a computing and engineering outreach program for girls in grades 8–10. *J. Comput. Sci. Coll.*, 30(6):119–126, 2015.

[72] D. A. Schön. *The Reflective Practitioner*. Routledge, London, 1 edition, 1992.

[73] C. Schulte and L. Budde. A Framework for Computing Education. In M. Joy, editor, *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, ACM Other conferences, pages 1–10, New York, NY, USA, 2018. ACM.

[74] C. Schulte, S. Sentance, S. Sparmann, R. Altin, M. Friebroon-Yesharim, M. Landman, M. T. Rücker, S. Satavlekar, A. Siegel, M. Tedre, L. Tubino, H. Vartiainen, J. Á. Velázquez-Iturbide, J. Waite, and Z. Wu. What We Talk About When We Talk About K-12 Computing Education. In M. Monga, V. Lonati, E. Barendsen, J. Sheard, and J. Paterson, editors, *2024 Working Group Reports on Innovation and Technology in Computer Science Education*, pages 226–257, New York, NY, USA, 2025. ACM.

[75] C. Selby and J. Woollard. *Computational thinking: the developing definition*. University of Southampton (E-prints), 2013.

[76] S. Sentance. Computer science education is a global challenge, 2021.

[77] C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards. Algorithm Visualization: The State of the Field. *ACM Trans. Comput. Educ.*, 10(3), aug 2010.

[78] J.-L. Shih, M. M. Chiu, and C.-H. Lin. Personalities, sequences of strategies and actions, and game attacks: A statistical discourse analysis of strategic board game play. *Computers in Human Behavior*, 133:107271, 2022.

[79] P. Silapachote, A. Srisuphab, A. Hoonlor, and T. Sunetnanta. Mastering basic Sorting Algorithms through Computational Thinking Activities for Everyone. In *2024 IEEE Global Engineering Education Conference (EDUCON)*, pages 1–5, 2024.

[80] B. Simon, T.-Y. Chen, G. Lewandowski, R. McCartney, and K. Sanders. Commonsense computing: what students know before we teach (episode 1: sorting). In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, pages 29–40, New York, NY, USA, 2006. Association for Computing Machinery.

[81] F. Steinert, J. Kummer, M. Landman, and L. Lehner. From Concept to Code: A Two-Day Workshop for Secondary Students on Computational Thinking and Programming. *Olympiads in Informatics*, (18):89–100, 2024.

[82] J. Sweller. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12(2):257–285, 1988.

[83] J. Sweller, P. Ayres, and S. Kalyuga. *Cognitive load theory*. Explorations in the learning sciences, instructional systems and performance technologies. Springer, New York and Dordrecht and Heidelberg and London, 1. ed. edition, 2011.

[84] J. Sweller, J. J. G. van Merriënboer, and F. Paas. Cognitive Architecture and Instructional Design: 20 Years Later. *Educational Psychology Review*, 31(2):261–292, 2019.

[85] S. Szeider. Large and Parallel Human Sorting Networks. In H. Fernau, I. Schwank, and J. Staub, editors, *Creative Mathematical Sciences Communication*, pages 194–204. Springer, Cham, 2025.

[86] R. Taub, M. Ben-Ari, and M. Armoni. The Effect of CS Unplugged on Middle-School Students' Views of CS. *SIGCSE Bull.*, 41(3):99–103, jul 2009.

[87] M. Tedre and P. J. Denning. The long quest for computational thinking. In J. Sheard and C. S. Montero, editors, *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pages 120–129, New York, NY, USA, 11242016. ACM.

[88] A. Unal and F. B. Topu. Effects of teaching a computer programming language via hybrid interface on anxiety, cognitive load level and achievement of high school students. *Education and information technologies*, 26(5):5291–5309, 2021.

[89] S. Unkovic and M. Landman. Supporting Non-CS Teachers with Programming Lessons. In J.-P. Pellet and G. Parriaux, editors, *16th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2023, Local Proceedings*, pages 61–74. Zenodo, 2023.

[90] M. Y. Vardi. What is an algorithm? *Communications of the ACM*, 55(3):5, 2012.

[91] E. Vegas, M. Hansen, and B. Fowler. Building skills for life: How to expand and improve computer science education around the world. *Brookings*, 2021.

[92] A. Vielsack and M. Landman. Finding the Right Balance: Facilitating the Exploration of Sorting Strategies using 3D-Printable Weights and Scales. In *Proceedings of the 28th Australasian Computing Education Conference*, New York, NY, USA, 2026. Association for Computing Machinery (ACM).

[93] L. S. Vygotsky. *Mind in society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, Massachusetts and London, England, 1978.

[94] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[95] K. Woo and G. Falloon. Problem solved, but how? An exploratory study into students' problem solving processes in creative coding tasks. *Thinking Skills and Creativity*, 46:101193, 2022.

[96] H. Zemanek. Al-khorezmi his background, his personality his work and his influence. *Algorithms in Modern Mathematics and Computer Science*, 122:1–81, 1981.

Overview of Tools Used

Overview of Generative AI Tools Used

During the writing process of this dissertation, I made limited use of AI tools in a supportive way:

- *ChatGPT-4 and ChatGPT-5*: assistance with language feedback, suggestions for structure, phrasing alternatives, and support for LaTeX code.
- *Writeful*: suggestions for synonyms, splitting and joining sentences, grammar and spelling, paraphrasing my own written text as a feature in Overleaf.

All content-related ideas, analyses, coding processes, references, and interpretations presented in this thesis are entirely my own. The AI tools did not generate research results, conduct analysis, or replace any part of the scholarly reasoning and were only used for language, structure and writing support as mentioned above to support me as a non-native English speaker and improve the readability and soundness of my work.

Text passages influenced by AI support were carefully reviewed, edited, and integrated by me to ensure accuracy, originality, and consistency with my academic writing style. The AI was never used to create independent text passages and always had my own writing as the base for all alterations.

Overview of Other Tools Used

In addition, I used several non-generative-AI tools during the research:

- *Overleaf*: spelling check and integration of Writeful as the LaTeX-editor for this thesis.
- *Citavi*: organisation and export of references to BibTeX.
- *Grammarly*: readability checks.

- *GIMP & Inkscape*: preparation of figures and images, and creating materials for learning activities.
- *Mermaid*: creation of flowcharts and diagrams.
- *DeepL*: translations between German and English.
- *MAXQDA*: qualitative data analysis and preliminary transcription of audio files.

List of Figures

List of Tables

Glossary

Abenteuer Informatik An interactive exhibition using CS Unplugged methods to explain basic ideas of computer science developed by Jens Gallenbacher. 16, 19

Bebras An international initiative offering short tasks to promote and assess informatics and computational thinking in K–12 education. 14

Cognitive Load Theory (CLT) An instructional design theory focusing on the limits of working memory and how instructional methods can reduce extraneous load to optimise learning. 15

Computational Thinking (CT) A set of problem-solving practices including abstraction, decomposition, algorithmic thinking, evaluation, and generalisation. 1, 2, 11, 49

Computer Science (CS) Computer Science (CS): The academic study of computation and computing systems, encompassing algorithms and data structures, hardware and software, and the processing of information. . 1, 11, 47

CS Unplugged A collection of hands-on activities that teach computer science concepts without using computers. 1, 11, 16–18, 45, 68, 70, 90, 91, 94, 95, 101–104, 118

Design-Based Research (DBR) An iterative research approach in education that designs, implements, and analyses educational interventions in real-world settings to refine both theory and practice. 20

Divide-and-conquer A problem-solving strategy that repeatedly divides a problem into smaller subproblems, solves them (often recursively), and combines their solutions. 21

Documentary Method An interpretive research approach focusing on reconstructing implicit knowledge and orientation frameworks from communicative actions. 28

Hybrid Interaction System (HIS) A conceptual model by Schulte and Budde integrating human and technological aspects in computing education. 14

Qualitative Content Analysis (QCA) A systematic, rule-guided method for categorising and interpreting textual data to derive themes or categories. 28

RQ1 "How do upper elementary school students describe and explain what an algorithm is?" It is mainly discussed in Chapter 4. 3, 4, 7, 25, 54, 99

RQ2 "In what aspects do students' conceptions of what an algorithm is differ from established definitions?" It is mainly discussed in Chapter 4. 3, 4, 7, 25, 54, 99, 100

RQ3 "What intuitive algorithmic strategies and sorting behaviours do K-12 students display when engaging in a collaborative unplugged task?" It is mainly discussed in Chapter 5. 3, 4, 6, 25, 26, 46, 100

RQ4 "How do students revise their intuitive sorting strategies based on collaboration with peers if they encounter the same sorting problem again?" It is mainly discussed in Chapter 5. 6, 25, 26, 46, 101

RQ5 "How do students revise their initial strategies when encountering a similar unplugged sorting task?" It is mainly discussed in Chapter 6. 3, 4, 6, 25, 26, 89

Spiral Curriculum An educational principle proposing that core ideas are revisited at increasing levels of complexity as learners develop. 23

Thematic Analysis (TA) A method for analysing qualitative data by identifying patterns or themes across the material. 28

TU Wien Informatics eduLAB The outreach programme of TU Wien Informatics, offering workshops for K–12 students using an explorative CS Unplugged approach.. 11, 15, 27

Zone of Proximal Development (ZPD) The distance between what a learner can achieve alone and what is achievable with guidance or collaboration. 23

Acronyms

AI Artificial Intelligence. 2, 12, 20, 47, 117

BL Bottom Left. 77, 78, 80, 81, 83–89, 91, 118

BR Bottom Right. 77–83, 86, 87, 89, 91, 118

CS Computer Science. 1–5, 7, 8, 12–15, 19, 20, 23, 50, 57, 90, 93–95

CT Computational Thinking. 2, 3, 5, 11, 13–15, 47, 68

DBR Design-Based Research. 21

QCA Qualitative Content Analysis. 71

RQ Research Question. 4

TA Thematic Analysis. 71

TL Top Left. 75, 76, 78–83, 85, 86, 88, 92

TR Top Right. 75, 76, 79–82, 86, 88, 91, 118