Journal of Information Systems Education

Volume 35
Issue 2
Spring 2024

Cognitive Aspects in Problem Solving: The Case of a Data Structures Course for IS Students

Orly Barzilai, Sofia Sherman, Moshe Leiba, and Hadar Spiegel

Recommended Citation: Barzilai, O., Sherman, S., Leiba, M., & Spiegel, H. (2024). Cognitive Aspects in Problem Solving: The Case of a Data Structures Course for IS Students. *Journal of Information Systems Education*, 35(2), 175-188. https://doi.org/10.62273/JJUB4136

Article Link: https://jise.org/Volume35/n2/JISE2024v35n2pp175-188.html

Received:	March 13, 2023
First Decision:	May 5, 2023
Accepted:	September 18, 2023
Published:	June 15, 2024

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information Systems Education, 35(2), 175-188, Spring 2024
https://doi.org/10.62273/JJUB4136

Cognitive Aspects in Problem Solving: The Case of a Data Structures Course for IS Students

Orly Barzilai
Sofia Sherman
Moshe Leiba
Hadar Spiegel
School of Information Systems
The Academic College of Tel Aviv Yafo
Tel Aviv Yafo, Israel
orlyba@mta.ac.il, sofiash@mta.ac.il, mosheli@mta.ac.il, hadarsp@mta.ac.il

ABSTRACT

Data Structures and Algorithms (DS) is a basic computer science course that is a prerequisite for taking advanced information systems (IS) curriculum courses. The course aims to teach students how to analyze a problem, design a solution, and implement it using pseudocode to construct knowledge and develop the necessary skills for algorithmic problem solving and abstract thinking. While the literature acknowledges the difficulty of this course, few references were found that examine the process students undergo while solving DS algorithmic problems. The study's objective is to explore and describe IS students' problem-solving processes and challenges requiring a high level of abstract thinking in a "black box" approach. During the study, 13 students were observed while solving a complex problem, using "think aloud" (TA) techniques. Each observation was recorded, transcribed, and iteratively analyzed using principles of provisional coding in qualitative data analysis. The findings suggest that the quality and correctness of the solutions depend on three main factors: abstract thinking, flexibility applied during the solution process, and an absence of misconceptions related to concepts and the basic understanding of the problem. The students' levels of abstract thinking also influenced the quality of visualization used while trying to solve the problem. This study's findings may raise the awareness of DS course designers and instructors regarding the importance of the role of abstract thinking, possible misconceptions, and strategies used in problem solving as factors influencing students' ability to solve complex problems.

Keywords: Data structures course, Problem-solving process, Complex algorithmic problems, Abstract thinking, Qualitative study

1. INTRODUCTION

The "Data Structures and Algorithms" (DS) course consists of two main aspects: data structures and algorithms. Wang (2012) defines the two aspects as complementary and inseparable for the design of such a program, where "data structure" refers to the problem of information presentation, and "algorithm" refers to the problem of information processing.

The *Overview Report of Computing Curricula* (Leidig & Salmela, 2020) identifies the DS course as a required course in the information systems (IS) discipline, with emphasis on finding solutions to programming problems, developing proof-of-concept programs, and determining whether faster solutions are feasible. The knowledge and skills this course offers to students are often a prerequisite for taking advanced courses in the IS curriculum (Databases, Data Retrieval, etc.) and may be very useful to graduates in their career development (Kramer, 2007; Nazir et al., 2019; Wall & Knapp, 2014).

Problem solving is generally regarded as one of the most important cognitive activities in everyday and professional contexts. Most people are required to solve problems and are rewarded for it (Jonassen, 2000). The term itself has been extensively discussed during the second half of the 20th century in general terms (e.g., Newell& Simon, 1972), mainly within the field of mathematics (e.g., Pólya, 1945; Schoenfeld, 1992), and later applied to the field of algorithmic problems in computer science. For example, the later work of Çakıroğlu and Mumcu (2020) examined problem-solving steps in block-based programming environments.

In computer science problem solving, abstraction has been recognized as a fundamental and essential principle (Haberman, 2004). According to Aharoni (2000b), during the DS course, students are exposed to different levels of abstract thinking. The abstract nature of the concepts taught in this course can often be difficult for students in general to grasp (Odisho et al., 2016), and for IS students in particular (Wall & Knapp, 2014). Several studies identified the reasons for these difficulties as (1) low motivation of students, especially for those who do not perceive computational skills as being important (Meisalo et al., 1997; Wang, 2012); (2) weak programming skills, which hinders the implementation of the possible solution (Wang, 2012); and (3) perceived difficulty of the course topics, which prevents students from dealing with the tasks (Wall & Knapp, 2014).

Recognizing DS as a difficult field to teach and learn, DS instructors have proposed various methods and techniques to help students and teachers deal with different aspects of this unit (e.g., Biernat, 1993; Hakulinen, 2011; Odisho et al, 2016; Wang, 2012).

To that end, the literature refers to the challenges of problem solving and abstract thinking from different points of view, such as students' solution evaluation (Ginat & Blau, 2017), prediction of abstract thinking level (Perrenet, 2010), and the development of teaching methods (Ginat & Blau, 2017). An early study investigated the thinking process of students when dealing with simple data structures (Aharoni, 2000b) and defined different abstraction levels.

In the current study, we focus on the students' problem-solving processes and challenges requiring a high level of abstract thinking in a "black box" approach, an approach in which the internal workings and implementations of a system are ignored, to simplify the problem and make the system's general behavior easier to understand. The results of this study may shed light on the problem-solving processes that students experience and reveal possible deficiencies in their knowledge or the orientation of their abstract thinking levels, which are needed for successfully dealing with complex algorithmic tasks. In addition, these results may serve as a basis for developing teaching methodologies to scaffold students' problem-solving processes and develop problem-solving skills.

2. LITERATURE REVIEW

2.1 Problem Solving

Problem solving can be regarded as any goal-directed sequence of cognitive operations (Anderson, 1980) and occurs in a situation where an individual responds to a problem that they do not know how to solve with routine or familiar procedures. Problem solving can be described as composed of three dimensions: the problem, the process, and the outcome (Leiba, 2010). Pólya's (1945) seminal work suggested that solving a problem involves four phases (or episodes): understanding the problem, developing a plan, carrying out the plan, and looking back.

Çakıroğlu and Mumcu (2020) compared problem-solving processes in programming environments to the framework proposed by Pólya (1945). The research concluded that various studies in computer science and mathematics addressed similar problem-solving steps. Çakıroğlu and Mumcu (2020) identified three steps performed during problem solving in block-based programming environments: the "focus step," containing the reviewing, understanding, and thinking activities; the "fight step," containing the implementation activity; and the "finalize step," containing awareness regarding the solution. Focus, fight, and finalize steps can occur in a sequential manner or, in some cases, in a cyclic transition between these steps. When students turn back from the fight to the focus step, their main purpose is rethinking. When they turn back from the finalize to the fight step, they mostly notice the mistakes.

Chinn et al. (2007) characterized the students' problem-solving process in DS by five stages: understanding the problem, developing a possible solution, looking back, students' meta-comments, and interviewer intervention. The first three stages mirror Pólya's (1945) problem-solving framework, which views the process as a progression, much like the waterfall model of software development, whereas the fourth and the fifth stages do not neatly fit into that progression. They found that time spent, stage and step transition rates, and whether the student solved the problem were not able to predict performance in the course.

Parham et al. (2009) used Bloom's taxonomy for cognitive process types (understand, apply, analyze, evaluate, create) to describe the thinking processes of computer science students when solving a complex algorithmic problem. Their study results showed that the successful problem solver seems to move from one type of cognitive process to another more frequently than the unsuccessful problem solver. In further research, Parham et al. (2010) provided more detailed insights regarding the types of metacognitive processes that occurred while solving complex data structure problems.

2.2 Abstract Thinking

Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction (Wing, 2006). Abstraction can be defined as an activity of reorganizing familiar structures used to solve known problems into new structures and relations adapted for the solution of a new problem (Armoni et al., 2006). The structures can be treated using a black box approach, where a black box is a system with a particular behavior and an unknown internal structure. The user can impact the whole system via the black box inputs and observe its reactions via its outputs (Cápay, 2014).

Recognizing familiar structures and patterns is called a reduction strategy. It offers the thinker a more global view of a problem by ignoring the details and enhances the possibility of strategic planning and an intuitive feel for the problem (Armoni et al., 2006). Hazzan (2002) showed a tendency to reduce a problem by adhering to familiar mathematical concepts from previous learning when solving problems in computability theory. The tendency to use reduction is influenced by the topic with which the problem deals, the way solvers conceive the legitimacy of reduction as a problem-solving heuristic, and the extent to which they consider that abstraction contributes to their problem solving. In addition, reduction seems illegitimate, like cheating, when using black boxes without knowing the implementation details (Armoni et al., 2006). Obstacles encountered during the transfer of prior knowledge to a new problem were related to "blurred" and vague pictures that solvers had about relevant data in the tasks to be solved, about proper utilization of basic algorithmic patterns, and about interconnections between these patterns. This may explain why less than 30% of a senior students' class used higher abstraction level solutions characterized by treating the problem as an object with properties (Ginat & Blau, 2017).

In the course described in this paper, we used a black box approach to teach students new simple or complex data structures. Each data structure was presented twice: first, as a black box with its defined operations – an abstract data type (ADT); and second, as its implementation. A more detailed description of this approach within the course may be found in the next section.

Some studies exist where the majority of the discussion revolves around abstract thinking and problem-solving aspects of computer science. For example, Haberman (2004) studied abstract thinking and concentrated on understanding the concepts of an algorithm. Perrenet (2010) proposed an instrument for the measurement of students' abstract thinking levels for the concept of an algorithm. Ginat and Blau (2017) used algorithm riddles having several solutions differing in their levels of abstraction to analyze computer science (CS) students' thinking abstraction levels.

Aharoni (2000a) investigated the thinking process experienced by CS students when solving data structure problems in a DS course. He presented three different levels of abstract thinking observed among students explaining different data structures: programming-language-oriented thinking, programming-oriented thinking, and programming-free thinking.

Programming-language-oriented thinking is a low level of abstract thinking, where the student uses programming language implementations to describe data structures. In other words, students who solved the problems at this level ignored the given structure and instead opted to create new ones while dealing with the implementation details by using a specific programming language.

Programming-oriented thinking is a middle level of abstract thinking, where the student thinks of new data structures in terms of their implementation by some program, without referring to a specific programming language.

Programming-free thinking is a high level of abstract thinking, where the solution is not related to the implementation of a data structure but to its concept as an object. In other words, students who solved the problems at this level used black box techniques, using familiar or given structures to create new ones without implementing the former.

Programming-free thinking may be invoked only if the concept of the data structure at hand has already been developed to its object stage, the only stage that enables thinking about abstract data structures. If the concept is still in its process stage, we are witnessing programming-language-oriented thinking and programming-oriented thinking, where the learner must still think about the data structure as being implemented within some program (specific or not). Aharoni (2000a) also proposed that there are two abstract thinking levels: the high abstract thinking level, as described in programming-free thinking, and the low abstract thinking level, which can be further subdivided into programming-language-oriented thinking and programming-oriented thinking.

There are several empirical studies that research undergraduate students' processes of solving complex data structure problems, as discussed above. We found no empirical studies that examine the problem-solving processes of IS students in the field of data structures. IS students differ from CS students in the basic mathematical courses they are given as part of their curricula. IS students are less exposed to higher-level mathematical courses and thus are less likely to encounter and practice abstract thinking. This study focuses on IS students' problem-solving processes and challenges requiring a high level of abstract thinking and the use of a "black box" approach.

3. METHODOLOGY AND CONTEXT

The Data Structures (DS) course is a first year, basic, mandatory course for IS students studying for the bachelor's degree (BSc.) in the Information Systems school. This course is designed to demonstrate and study various data structures and how they are used to effectively solve computational problems. The curriculum of IS students involves fewer mathematical courses than that of CS students. As a result, they have less experience in solving complex tasks that require advanced algorithmic and abstract thinking.

To achieve the study's goal, students were given a task that consolidated the "black box" thinking approach. As they solved it, their problem-solving and thought processes were documented and later analyzed to capture the different cognitive aspects and challenges they had.

3.1 The Context (Course)

The study was conducted during a DS course. Students taking this course learn complex data structures, such as stack, queue, and tree, as well as how to calculate and compare different algorithms in terms of their time and space complexities. The course is based on the Cormen et al. (2022) textbook. During the course, the following teaching techniques were used: a black box approach to develop abstract thinking skills, visualization techniques to support the black box concepts, and algorithm development and execution. All techniques were practiced in lectures, tutorials, and home assignments.

To help students develop abstract thinking skills, each data structure (starting from simple ones and progressing to intricate ones) was presented by using the black box approach. Figure 1 presents the three concepts used to present the black box approach. Each black box was presented twice: first by its capabilities (see (a)) and then by its internal structure (implementation) (see (b)). Each new black box presented in class was constructed from previously learned black boxes (see (c)).

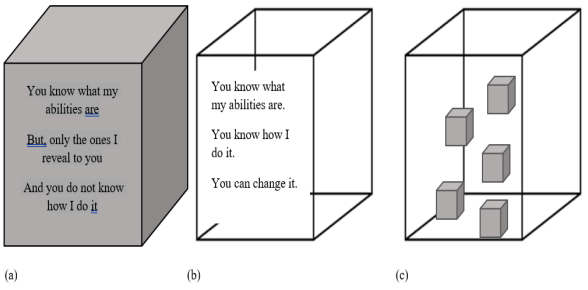


Figure 1. Different Modes Used to Present the Black Box Approach

The use of visualization and simulation was presented and demonstrated in class as a strategy for understanding the problems, testing ideas for algorithms, and verifying those ideas (Aharoni, 2000b). We used visualization techniques such as drawing the structures on the board and referring to them as objects, and simulation tools (automatic and manual) to run examples and mimic the black box capabilities.

Due to the COVID-19 pandemic, the course was conducted remotely, using Zoom video conferencing software. During these weeks, students continuously practiced abstract data structures using the black box approach in tutorials and home assignments. Each student attended classes remotely, which contributed to a suitable environment for independent work on the assignments. Questions were sent to the lecturer via a private chat.

3.2 Research Methods

To examine the processes students experienced while solving problems that require a high level of abstract thinking, and to gain a better understanding of their thinking characteristics, we

employed a qualitative case study methodology. The methodology, according to Stake (1995), is a "study of the particularity and complexity of a single case, coming to understand its activity within important circumstances" (p. xi). The characteristics of a case study are holistic (considering the interrelationship between the phenomenon and its contexts), empirical (basing the study on their observations in the field), interpretive (resting upon their intuition and seeing research basically as a researcher–subject interaction), and emphatic (reflecting the vicarious experiences of the subjects in an emic perspective) (Yazan, 2015). Yin (2014) offers a more detailed and technical definition of a case study as an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not evident. For this exploratory study, we conducted a singular case study using TA protocol analysis.

TA protocol analysis (Ericsson & Simon, 1980; 1993) requires participants to actively engage in the process of verbalizing their experiences, thoughts, actions, and feelings while interacting with a task. TA investigation seeks to place the participant in their most natural state of design thinking during the protocol sessions (van Someren et al., 1994). In addition, the TA method requires participants to use their own language and to approach the assigned task as they would naturally solve it. According to Ericsson and Simon (1984), there are traditionally two basic types of TA methods: the concurrent TA (CTA) method, in which participants "think aloud" at the same time as they are carrying out the experimental tasks; and the retrospective TA (RTA) method, in which participants verbalize their thoughts after they have completed the experimental tasks. TA has three distinct levels of verbalization (Ericsson & Simon, 1980; 1993), with each being representative of the amount of cognitive processing required. Level one verbalization requires vocalization of task-relevant thoughts only. Level two verbalization requires participants to recode visual stimuli not regularly verbalized prior to providing verbalization on the task. Verbalizations should reflect stimuli affecting the focus of the participant through the task; for example, a participant might provide vocalization of stimuli within a task, including sight, sound, and smell. Eccles (2012) indicated that level one and level two verbalizations are the result of conscious thought processing in the short-term memory (STM) during the execution of a task, providing concurrent verbalization during or immediately after a task has been completed. Verbalizations occur most often in environments where participants are provided with undirected prompts to think aloud naturally during the execution of a task (Ericsson & Simon, 1980). Lastly, level three verbalization requires participants to provide explanations, justification, and reasoning for cognitive thoughts throughout the task. TA enables us to profile students based on their actual behavior as observed during the sessions. In this study, we performed a CTA with level one and level two verbalizations. While solving the problem, during the TA process, the students raised several ideas. The researcher did not intervene in this process but only observed it.

All TA sessions were recorded, transcribed, and analyzed. The quotes in this study are translations of the original TA protocols and are as close as possible to the original expressions and idioms. To secure anonymity and confidentiality, we anonymized our respondents using a code number.

3.3 Case Description and Data Collection

The DS course is thirteen (13) weeks long, and the research task was administered during the 9th week of the course. The research task (as described in Section 3.3.1) is built upon three prior assignments that were administered throughout the course (Table 1) and serve as its baseline.

Week	Activity	Details
Week 6	Assignment 1	The queue and stack structures were practiced using the black box model. Students were asked to implement a queue using two stacks. Afterward, the solution was presented and discussed by the tutors. The solution was demonstrated using a visualization of the data structure, and then through pseudocode.
Week 7	Assignment 2	The queue and stack structures were practiced using the black box approach. Students were asked to implement a stack using two queues. Afterward, the solution was presented and discussed by the tutors. The solution was demonstrated using a visualization of the data structure, and then through pseudocode.
Week 8	Assignment 3 (starter assignment)	This assignment's purpose was to prepare the students for the research assignment. Students were asked to solve it independently during the lecture class. Afterward, the solution was presented and discussed by the tutors. One of the main goals of this assignment was to present and familiarize the students with the new data structures.

Table 1. Preparation for the Study as Part of the Course Schedule

3.3.1 The Research Task. The principle that guided us throughout the task's construction was to create a compound yet complex data structure that embodies the "black box" approach's characteristics – an object assembled from other "black box" objects. This approach enables the students to propose and devise new structures with minimal regard to their particular implementation details. However, only high levels of abstract thinking will allow students to achieve the required solution.

For that purpose, we constructed unfamiliar compound data structures (double-ended queue and central queue, described in Appendix A) and asked the students to create a central queue using two double-ended queues. Since the purpose was to learn about abstract thinking levels rather than test students' ability to grasp a new data structure, in Assignment 3 (starter assignment), students were asked to solve the same assignment without the space and time complexity requirements. This made it possible for us to investigate the students' abstraction

thinking levels, as they were using familiar compound data structures. The research task is described in Appendices B and C. Table 2 describes the main features of the research task.

Description	Solution Principles	Abstract Thinking Construction
Construct a central queue, (required, compound, familiar data structure) using two double-ended queues (required, compound, familiar data structure). Required time complexity – O(1).	At any given moment the data is divided between two given data structures. No data transference is needed here except for a single item transference to balance the data between two given data structures. This allowed the extraction and insertion of data in the required complexity.	The purpose of this assignment was to test the students' ability to use high levels of abstraction when developing a new algorithm, using familiar data structures learned in Assignment 3.

Table 2. Main Features of the Research Task

3.3.2 Participants and Data Collection. This study focuses on an algorithmic problem given to students in a DS course taught at the School of Information Systems, in a higher education institution during the 2020–2021 academic year.

The course participants were 136 undergraduate students studying in their first year toward a BSc. in IS. Thirteen students studying this course, aged 22 to 27, 6 males and 7 females, volunteered to participate in this study. Each student was assigned the abbreviation ST (student), which was accompanied by a unique number for the purpose of identification (ST01 to ST13). The size of this study population corresponds with the other studies in this field. For example, Aharoni (2000a) studied students' thinking processes while dealing with simple data structures. In this qualitative research, nine students studying a CS course were interviewed, using semi-structured observational interviews. Aharoni (2000a) and Çakiroğlu and Mumcu (2020) explored problem-solving steps using qualitative method tools on 15 students studying an IT course.

All participants had the following prerequisites: a programming introduction course (course taught in the C++ language), and mathematical introduction courses (including proofs using induction). In parallel to this course, the students participated in a Python programming language course. All the students participated in lectures and tutorials and submitted homework assignments.

This study is based on the TA protocols of students' solutions to the algorithmic problem given. The quotes in this study are translations of the original TA protocols. We remained as close as possible to the original expressions and idioms. To secure anonymity and confidentiality, we anonymized our respondents with code numbers.

3.4 Data Analysis

Our data analysis was based on principles of provisional coding (Miles et al., 2014), according to which: "Prior theorizing and empirical research are, of course, important inputs. It helps to lay out your own orienting frame and then map onto it the variables and relationships from the literature available, to see where the overlaps, contradictions, qualifications, and refinements are" (p. 41). Each step of the coding method was first performed by each of the authors of the paper separately and then discussed in a joint coding session. This resulted in the construction of categories and, ultimately, a conceptual framework. In this analysis approach, categories emerge from the data and are then validated and refined throughout the analysis process. Unanimous agreement on the emergent categories was achieved by all authors in this joint coding session. This was done to maintain a continuous dialogue between researchers and consistency of the coding (Walther et al., 2013) and to establish reliability to ensure the trustworthiness of the study (Miles et al., 2014).

Our aim was to identify problem-solving processes and their characteristics while solving complex data structure problems. Thus, the categories materializing from the analysis referred to process-related phenomena. These categories are presented in the results section. We utilized Aharoni's (2000a) framework as a base for the conceptual framework, which consists of two levels of abstraction.

4. FINDINGS

The findings of this study point to three cognitive components that affect the quality and correctness of the problem's solutions: abstract thinking, knowledge toolbox, and solution approach. In Section 4.1 these cognitive components will be detailed, and Section 4.2 will focus on the students' problem-solving processes.

4.1 Cognitive Components

4.1.1 Abstract Thinking. The abstract thinking ability affected the problem-solving process. Students demonstrated different levels of abstract thinking, which we grouped into two categories: programming-free thinking and program-context thinking.

4.1.1.1 Programming-Free Thinking. Programming-free thinking is a high level of abstract thinking where the solution is unrelated to the implementation of a data structure but regards each component as a black box. In this study, six students were observed acting at a high level of abstract thinking while solving the given problem. They used a double-ended queue ADT, without referring to its implementation or other programming-oriented details. For example, ST08 said: "*I understood that a central queue is based on two double-ended queue structures, and therefore there is no need to use other structures such as a linked list.*" Although all six students presented a programming-free thinking level, only three of them successfully solved the given problem.

4.1.1.2 Programming-Context Thinking. The programming-context thinking level combines low and middle levels of abstract thinking (Aharoni, 2000b). We decided not to differentiate between these two abstraction levels since we did

not find evidence of a specific language being used while solving the problem. Although the students had successfully completed the CS1 course, taught in the C++ language, and had studied the Python language in a parallel course, we decided to use pseudocode when describing the data structures, without any reference to a specific language.

Five students presented programming-context thinking. For example, ST03 conceptualized the central queue as an array: "*I can describe the central queue as an array where the first index is the head, and the last index is the tail.*" ST13 internalized the central queue as a linked list receiving its input from the double-ended queues: "*I am thinking of how to connect the two double-ended queues to a linked list which is a central queue."* ST04 was not sure how to address the functions of the double-ended queues: "*I am not sure whether to implement the double-ended queue functions or just use them.*"

Two other students shifted between the programming-context thinking level and the programming-free thinking level during the problem-solving process. The first student, ST06, presented the programming-free thinking level during the problem-understanding phase: "*The two double-ended queues behave like a central queue.*" However, at the solution phase, ST06 suddenly started to use programming-context thinking: "*In order to count the number of items in the double-ended queue I should subtract the head from the tail.*" (the terms *head* and *tail* referred to the indices). The second student, ST09, presented the opposite behavior, starting from the programming-context thinking level during the problem comprehension phase, using the concepts of arrays and indices: "*In order to insert an item into the central queue tail, I should define some pointer to the end. I should define it as an array, this double-ended queue.*" Later, when ST09 was asked to write pseudocode, she re-read the problem description and started to show the programming-free thinking level: "*I should use the following double-ended queue functions: extraction from tail and insertion to head.*" This phenomenon may happen when the formation of the data structure type has started but is still a weak mental structure that does not take control unless there is no other option (Aharoni, 2000b).

4.1.2 Knowledge Toolbox. The students' control of the knowledge toolbox affected the problem-solving process. Students are expected to use previous knowledge and experience they had acquired and accumulated, which we refer to as the knowledge toolbox. This toolbox is built from layers of knowledge, including programming concepts and languages acquired in CS courses, ADTs acquired in this course, knowledge of compound data structures acquired in the solution of preliminary assignments 1 to 3 (see Table 1), and the internalization of the "black box" approach.

A lack of knowledge fostered misconceptions, which ultimately affected the solution to the task. For example, ST01 said: "*We will move to another structure which is an array, which is a stack.*" The student confused a data structure, "stack," with its implementation, "array." Another example was ST05, who said: "*I should insert items to both sides of a new stack […] Items should be inserted into a new array or queue.*" In this case, we can see the former misconception along with a misconception regarding the "stack" data structure's capabilities. Some students mixed basic data structures with the complex data structures described in the given assignment. An example is ST03, who said: "*The central queue is a data structure of a queue, which is similar to a stack or an array.*" In addition, this student said: "*What I can think of is a linked list. This is what we learned in a CS course. I programmed a linked list in this course.*" We can learn from this statement that this student chose to rely on his most "accessible" knowledge, although it contradicted the black box approach.

4.1.3 The Approach to Solution. The manner in which students approached the task and its solution affected their problem-solving process. The research task (see Table 2) included a list of operations that had to be implemented. However, the order in which they were presented is not necessarily the optimal order in which they should be implemented (see Appendix A). Although the first operation has a simple solution per se, in combination with the two subsequent operations, its solution needs to be reconstructed and refined. In this regard, we observed two solution processes: the linear process – where students solved the problem based on the presentation order of the assignment operations – and the flexible process – where students demonstrated flexibility in changing earlier proposed algorithms or changing the order of the operation's execution based on an overview of the entire assignment.

Six students followed the linear process approach. We found that none of them were successful in solving the given problem. In this category, we identified students from both abstraction levels (programming-free thinking level and programming-context thinking level). Consequently, we may conclude that the linear approach hinders the ability to conceive and promote successful problem-solving algorithms that involve data structures. Students who used this approach reported that they felt stuck and did not know how to proceed. For example, ST12 said: "*From what I understand, to insert and extract an item (referring to the edges) is not a problem. The only issue is what to do with the middle. I should do something different with it. I feel stuck.*"

Seven students demonstrated the flexible process approach. A single student (ST11), who exhibited the programming-context thinking level, solved the assignment's operations in the order of their occurrence and then retraced back to adjust the algorithms of the previous operations. During the solution-planning step, when he recognized that the algorithm he had initially proposed failed, he returned to his prior steps in order to review their solutions: "*The length of the central queue is the sum of the double-ended queue lengths. This requires me to change the insertion [meaning the insertion to the tail]*." He successfully solved the problem. The other six students began by overviewing the operations in an attempt to identify the main requirement of the assignment, and they then used this as the starting point for the development of an algorithm. Three of the students (ST02, ST07, ST08) had a programming-free thinking level. Of these, ST08 and ST02 successfully solved the given problem. The other three students (ST03, ST05, ST13), who were at the programming-context thinking level, failed to solve the given problem.

4.2 Problem-Solving Process

We identified three main steps of problem solving, which are aligned with the problem-solving steps defined in the literature by Pólya (1945) in the field of mathematics and adjusted by Çakiroğlu and Mamcu (2020) for CS. The steps we identified

in the current study are: (1) understanding the problem, (2) planning the solution, and (3) implementation.

4.2.1 Step 1 – Understanding the Problem. In the TA sessions, all (13) students demonstrated an attempt to understand the given problem. Following the research task description in Table 2, at this stage, the students had already solved Assignment 3 (which was identical to the research task except for the required complexity). All aspects of Assignment 3 were explained in class, including the given abstract data structures, the required complexity, and the solution. It was expected that after all these explanations, students would adequately understand the task's data structures and requirements.

Misconceptions were observed regarding the research task requirements. Students were expected to refer to the central queue as two double-ended queues that act together as a central queue. Instead, some of them perceived the central queue and the two double-ended queues as entirely separate structures. Students with the programming-context thinking level did not understand the ADTs defined in the assignment with their operations and relationships, in addition to their black box approach misconceptions. These students applied the same solution given to Assignment 3 to the research task, ignoring the required O(1) complexity. For example, ST13 said: "*I need to extract items from the double-ended queues and merge them to a linked list … I am thinking how to connect H and T [the double-ended queues] to a linked list, a structure called the central queue.*" Another example is ST01: "*I am using an array which is a queue, which is a double-ended queue. I will move the data to another structure which is an array which is a stack.*"

Students exhibiting the programming-free thinking level showed misconceptions related to the research task as well yet showed no black-box approach misconceptions. It appears these students did not fully internalize the knowledge related to the assignment's data structures, acquired in Assignment 3.

For example, ST07 said: "*I am not sure whether to insert items to H and T [the double-ended queues] separately or whether I should refer to them together as a central queue.*" One of the students, ST11, managed to reach a solution once he overcame this misconception and identified that the central queue is constructed from two double-ended queues: "*If the central queue is composed of two double-ended queues, I can say that the central queue is expressed by both of them. I can insert an item at the head of one of them and say that I inserted the item in the middle. The length of the central queue is the length of both double-ended queues.*"

4.2.2 Step 2 – Planning the Solution. To understand the problem and devise a solution, students were aided by various utility tools:

4.2.2.1 Visualization **–** A tool that helps lower the abstraction level. Throughout the Data Structures course discussed in this paper, visualization was constantly used to describe new material and to solve problems. Visualizations were used for both illustrations of the data structures and for simulating algorithms.

Twelve (12) students visualized data structures in the first two steps of problem solving (understanding the given problem and planning the solution). Only one student, ST03, did not draw any structure while planning his solution. Table 4 shows the first use of a visualization in the process of solving the assignment per student, divided between the two abstract thinking levels observed.

Programming-Free Thinking			Programming-Context Thinking		
Step 1	Step 2	Step 1: Another attempt	Step 1	Step 2	Step 1: Another attempt
ST12 ST10 ST07	ST08 ST02	ST11		ST09 ST04 ST05 ST06 ST01	ST13

Table 3. Students Mapped by Abstraction Level and Problem-Solving Steps Related to the First Use of Visualization

Half of the students located at the programming-free thinking level (ST07, ST10, and ST12) began using visualization during the "understanding the given problem" step (step 1). They drew data structures while reading the assignment. Later, they were aided by these visualizations while designing algorithms for the solution. Two more students, ST02 and ST08, began using visualizations only after reading the entire assignment, during the solution-planning step (step 2), as they claimed they already understood the given data structures from solving Assignment 3. A single student, ST11, after several attempts to plan an algorithm that would solve the assignment without using any form of visualization, returned to re-reading the exercise and only thereafter started to draw data structures. Students from this group drew central queues and double-ended queues in an abstract manner without adding specific details. For example, ST11, when starting to draw, said: "*I am drawing a queue that looks like an array. I am not writing indices because it is not an array.*"

None of the students with a programming-context thinking level used visualization tools during step 1 (understanding the given problem). After a while, most of them began sketching during the solution-planning phase, and one of them, ST13, did so after returning for further reading in an attempt to understand the problem. As mentioned before, all of them failed to solve the given problem.

Students from the abovementioned group drew visual representations with implementation details, which led to some misconceptions. For example, ST05, when starting to draw, said: "*I will draw again and see in the new array how I should do it*"; ST06 said: "*I will try to draw. I have two arrays. I should find the middle. I am wondering whether the head or the tail marks some sort of an index?*"

4.2.2.2 Pseudocode **–** A tool that can be used to understand a problem and for considering algorithms. Using pseudocode to formulate a solution corresponds to a level of abstraction that is neither too low (it is not a programming language) nor too high (a verbal description). Pseudocode can thus play an important intermediate role in the decomposition of a problem (Copus & Copus, 2018). We observed the use of pseudocode as a solution-planning tool in four students, ST11, ST09, ST08, and ST04. For example, ST11 (programming-free thinking): "*I am*

Proposed algorithms scale	Abstraction level		Misconceptions		Flexible process	
	Programming -free thinking	Programming -context thinking	Yes	No	Yes	No
1 (The correct solution)	ST02 ST08 ST11			ST02 ST08 ST11	ST02 ST08 ST11	
2	ST12 ST07 ST10		ST07 ST12	ST10	ST07	ST10 ST12
3		ST01 ST05 ST13	ST01 ST05 ST13		ST05 ST13	ST01
4		ST04 ST06 ST09	ST04 ST06 ST09			ST04 ST06 ST09
5 (No solution)		ST03		ST03	ST03	

Table 4. Distribution of Students by the Proposed Algorithms Scale and by Cognitive Attributes of the Problem Solver

writing pseudocode in order to make it clear." Another example is ST09, who acted at the programming-context level, and the use of pseudocode made her contemplate the way she understood the problem: "*I am not sure whether I can use the function (the double-ended queue's function) or if I should implement it.*"

4.2.2.3 The Quality of Algorithm Ideas. In general, when solving programming assignments, an algorithm is first planned, and then it is implemented in a computer program environment. In data structure assignments, the culmination of the solution occurs before the implementation phase. The student must develop several propositions for algorithms that aim to solve the problem at hand and test them using visual simulations. For these reasons, the phases of planning and execution that take place while solving data structure exercises occur intermittently and are mixed with one another. Each student considered several algorithms, from which, during the analysis stage, they selected the highest quality algorithm. By highest quality algorithm, we refer to the algorithm that is closest to the solution. The following list includes five algorithm categories, scaled from 1 to 5, where 1 represents the highest quality algorithm.

1. Divide the items between the double-ended queues during insertion while re-balancing the double-ended queues. This is the correct solution (selected by three students).
2. Divide the items between double-ended queues during insertion without re-balancing (selected by three students).
3. Moving items to another structure, to support operations performed at the center of the queue. This algorithm is a repetition of the starter assignment solution (selected by three students).
4. Finding the item in the middle by using array indices (selected by three students).
5. No solution. One student was not able to provide an algorithm for a solution.

Table 4 summarizes the distribution of the students in accordance with the above scale, vs. the previously described cognitive attributes of the problem solver, detailed in Section 4.1. Six students who acquired the programming-free thinking level suggested high-quality algorithms (categories 1 and 2 from the above scale). Of the three students who successfully solved the problem (proposed the algorithm from category 1), all displayed programming-free thinking, did not exhibit any misconceptions, and solved the problem using the flexible process approach. The other three students who displayed programming-free thinking but exhibited misconceptions or solved the problem in a linear approach came up with the second-best solution. Misconceptions were apparent in all the students who displayed a programming-context thinking level. These students offered low-quality algorithms (categories 3–4) or could not suggest one (category 5).

4.2.3 Step 3 – Implementation. The last step in problem-solving activities is writing the solution and verifying the correctness of the algorithm. This is achieved by running the algorithm on all the required ADT operations using pseudocode (as a solution to the assignment). In our study, we observed four cases of writing pseudocode. However, only one student (ST08) used pseudocode exclusively during the implementation step, while the others used pseudocode as a supporting tool for their thinking process (described in "planning the solution step").

5. DISCUSSION

This study aimed to explore and describe IS students' problem-solving processes involving a high level of abstract thinking. To achieve this aim, students' problem-solving processes were analyzed during a "black box" approach-based task in an effort to capture different cognitive aspects and challenges.

This study mapped three main requirements that impact the success of a problem-solving process: obtain program-free thinking level, no misconceptions (which stems from a solid "knowledge toolbox"), and exhibit flexibility during the solution process. Only students who mastered these three components could solve the research task (see Figure 2). Students who lacked one or more components failed to solve the research task.

The study shows that only 6 out of 13 students (46%) were able to apply the programming-free abstract thinking level when solving the research task. This finding corresponds with that of Aharoni (2000a), who concludes that the phenomenon of low abstraction levels is common in DS courses and stems from the fact that students were not sufficiently exposed to abstract thinking practices. Although in our study, students practiced problems involving different levels of complexity, it

can be argued that at this stage of the course, students had not yet acquired the required expertise.

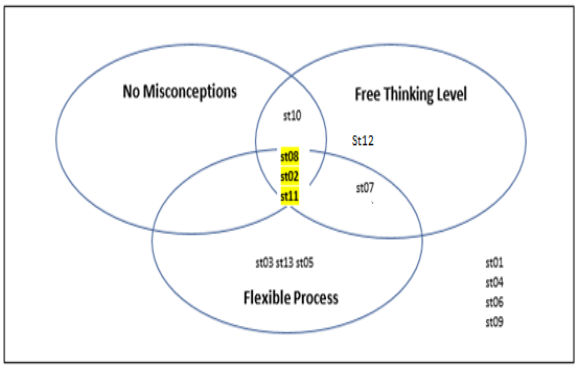


Figure 2. Mapping of Students Based on Abstract Thinking, Flexibility, and Misconceptions

We observed several types of misconceptions related to the acquired knowledge in different data structures and to the black box approach, which prevented students from solving the complex problem. This finding is consistent with the claim of Smith et al. (1994) that students' prior knowledge is a major source of their misconceptions. Following the findings presented by this study earlier, all students who had misconceptions failed to solve the assignment.

Two approaches were applied when solving the research task: a linear approach and a flexible approach (non-linear). These approaches are mentioned in several studies. Pólya (1945) described four stages that must be followed in a linear sequence when solving a problem. Mason et al. (1982) and Schoenfeld (1985) suggested that the process of problem-solving is not always strictly linear and that it can include forward and backward steps between analyzing, planning, and exploring a problem. Wilson et al. (1993) proposed a dynamic and cyclic interpretation of Pólya's stages, which allows for forward and backward movement between all phases of problem solving, even after the looking-back phase. Similarly, Yimer and Ellerton (2010) proposed a model that includes transitions between all phases of problem solving, emphasizing the non-linear nature of the process. Our interpretation of the flexible approach that was observed in our study is that students apply a process of monitoring and self-regulation, which is referred to by Schoenfeld (1992) as the "control" factor (ways in which students monitor their own problem-solving process, use their observations of partial results to guide future problem-solving actions and decide how and when to use available resources and heuristics). Following the findings presented earlier in this study, all the students who used the linear approach failed to solve the assignment.

We used visualizations throughout the course (mentioned in Section 3.1) as a primary tool for explaining algorithms and simulating their execution. Visual representation of a data structure is used to reduce the abstraction level by making the data structure more concrete and familiar (Aharoni, 2000b). The use of visualization as a teaching tool is supported by Akram and Fang (2015), who found that using visualization prototype applications during lessons in a DS course engaged students' attention toward cognitive learning. They concluded that graphic representations, such as pictures, graphs, charts, and diagrams, help people grasp the meaning and attain an understanding of information more easily and quickly.

In our study, visualization tools were used by most students at various stages of solving the research assignment, but these tools did not necessarily help. We found that the quality of the use of the visual tools depended on the student's level of abstract thinking. This phenomenon can be explained by students operating at an inadequate level of abstraction, which caused difficulties in problem comprehension and an inability to simplify the problem by drawing. Some of the students' visualizations were not effective, because the drawings did not fit the required data structures presented in the assignment. For example, one student drew an array instead of a double-ended queue. The ineffectiveness of using visualization is explained by Aharoni (2000a) as being caused by misconceptions of some of that data structure's properties. When visualizing a data structure, its picture should not include the implementation details but only its properties and organization (Aharoni, 2000b). In our study, the drawings of the students located at the programming-context thinking level contained implementation details, which contributed to their misconceptions.

6. CONCLUSIONS AND RECOMMENDATIONS

The purpose of this study was to characterize what is required from IS students to solve complex problems. Results showed that to solve a complex problem successfully, an IS student should have a programming-free abstract thinking level, no misconceptions regarding concept comprehension and problem understanding, and be able to apply a flexible problem-solving process. In our study, the assignments were designed with increasing difficulty levels to prepare the students for the research task (see Table 1)**.** However**,** 10 out of 13 students failed to solve the assignment. Most of these students (7 out of 10) did not exhibit the required level of abstract thinking. For these students, using visualization as a supporting tool for solving the problem contributed to their misconceptions.

Based on these results, recommendations can be made for two groups of stakeholders. (a) Faculty members can consider highlighting common misconceptions while teaching this course (and similar ones); better explaining what an appropriate visualization is vs. a lacking one; exploring more examples that require a high level of abstraction during the class; and aiming to develop in students a flexible approach to solving complex problems by applying appropriate assignments, such as class examples demonstrating the flexible approach to solving problems and presenting common mistakes created by a linear approach. In addition, faculty members should evaluate the student's level of abstraction and variations in abstract thinking while the course is being taught and adjust their teaching methodologies and assignments accordingly (formative evaluation). (b) Policymakers can consider developing new abstract thinking development tools in basic courses and postponing this specific course (DS) to an advanced level of the IS curriculum, where the students' abstract thinking is likely to be more developed.

7. LIMITATIONS AND FUTURE RESEARCH

This study was conducted as part of an introductory course where students had not yet established and refined their problem-solving skills. The study serves as a first step in

investigating thinking processes and solving complex problems as performed by IS students. We suggest expanding this study to more advanced courses, such as the Information Retrieval course given to students in their last year, where complicated algorithms using different data structures are presented.

Due to the COVID-19 pandemic, the course was conducted through video conference meetings, and it is not entirely clear how this might have affected teaching, learning, and data acquisition. Out of 136 students in the course, only 13 volunteered for this study, which may limit the generalization of the difficulties we observed. Quantitative tools should be developed and implemented at a larger scale to overcome this limitation.

8. REFERENCES

Aharoni, D. (2000a). Cogito, Ergo Sum! Cognitive Processes of Students Dealing with Data Structures. *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education* (pp. 26-30). https://doi.org/10.1145/331795.331804

Aharoni, D. (2000b). What You See Is What You Get: The Influence of Visualization on the Perception of Data Structures. *Document Resume*, 11(4), 10-17.

Akram, J., & Fang, L. (2015, April). Cognitive Effects of Visualization on Learning Data Structure and Algorithms. *The Third International Conference on Digital Enterprise and Information Systems* (p. 70).

Anderson, J. R. (1980). *Cognitive Psychology and Its Implication*. Worth Publishers.

Armoni, M., Gal-Ezer, J., & Hazzan, O. (2006). Reductive Thinking in Computer Science. *Computer Science Education*, 16(4), 281-301. https://doi.org/10.1080/08993400600937845

Biernat, M. J. (1993). Teaching Tools for Data Structures and Algorithms. *ACM SIGCSE Bulletin*, 25(4), 9-12. https://doi.org/10.1145/164205.164211

Çakıroğlu, Ü., & Mumcu, S. (2020). Focus-Fight-Finalize (3F): Problem-Solving Steps Extracted From Behavioral Patterns in Block Based Programming. *Journal of Educational Computing Research*, 58(7), 1279-1310. https://doi.org/10.1177/0735633120930673

Cápay, M. (2014, December). Algorithmic Thinking Observation: How Students of Applied Informatics Break the Mystery of Black Box Applications. *2014 International Conference on Interactive Collaborative Learning* (pp. 535-540). https://doi.org/10.1109/ICL.2014.7017829

Chinn, D., Spencer, C., & Martin, K. (2007, June). Problem Solving and Student Performance in Data Structures and Algorithms. *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 241-245). https://doi.org/10.1145/1268784.1268854

Copus, B., & Copus Jr, W. P. (2018). Pseudocode Quality Correlations With Ultimate Answer Quality in CS1. *Journal of Computing Sciences in Colleges*, 33(5), 145-150.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms*. MIT Press.

Eccles, D. (2012). Verbal Reports of Cognitive Processes. In G. Tenenbaum, R. C. Eklund, & A. Kamata (Eds.), *Handbook of Measurement in Sport and Exercise Psychology* (pp. 103-117). Champaign, IL: Human Kinetics. https://doi.org/10.5040/9781492596332.ch-011

Ericsson, K. A., & Simon, H. A. (1980). Verbal Reports as Data. *Psychological Review*, 87(3), 215-251. https://doi.org/10.1037/0033-295X.87.3.215

Ericsson, K. A., & Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data*. MIT Press.

Ericsson, K. A., & Simon, H. A. (1993). *Verbal Reports as Data*. MIT Press.

Ginat, D., & Blau, Y. (2017, March). Multiple Levels of Abstraction in Algorithmic Problem Solving. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 237-242). https://doi.org/10.1145/3017680.3017801

Haberman, B. (2004). High-School Students' Attitudes Regarding Procedural Abstraction. *Education and Information Technologies*, 9(2), 131-145. https://doi.org/10.1023/B:EAIT.0000027926.99053.6f

Hakulinen, L. (2011, November). Card Games for Teaching Data Structures and Algorithms. *Proceedings of the 11th Koli Calling International Conference on Computing Education Research* (pp. 120-121). https://doi.org/10.1145/2094131.2094157

Hazzan, O. (2002). Reducing Abstraction Level When Learning Computability Theory Concepts. *ACM SIGCSE Bulletin*, 34(3), 156-160. https://doi.org/10.1145/637610.544461

Jonassen, D. H. (2000). Toward a Design Theory of Problem Solving. *Educational Technology Research and Development*, 48(4), 63-85. https://doi.org/10.1007/BF02300500

Kramer, J. (2007). Is Abstraction the Key to Computing? *Communications of the ACM*, 50(4), 36-42. https://doi.org/10.1145/1232743.1232745

Leiba, M. (2010). Assessing Mathematical Problem Solving Behavior in Web-Based Environments Using Data Mining. *EC-TEL Doctoral Consortium* (pp. 37-42).

Leidig, P., & Salmela, H. (2020). *IS2020 A Competency Model for Undergraduate Programs in Information Systems*. The Joint ACM/AIS IS2020 Task Force.

Mason, J., Burton, L., & Stacey, K. (1982). *Thinking Mathematically*. Addison-Wesley.

Miles, M. B., Huberman, A. M., & Saldana, J. (2014). *Qualitative Data Analysis: A Methods Sourcebook* (3rd edition). Sage publications

Meisalo, V., Sutinen, E., & Tarhio, J. (1997, June). CLAP: Teaching Data Structures in a Creative Way. *Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education* (pp. 117-119). https://doi.org/10.1145/268819.268854

Nazir, S., Naicken, S., & Paterson, J. H. (2019, November). Teaching Data Structures Through Group Based Collaborative Peer Interactions. *Proceedings of the 8th Computer Science Education Research Conference* (pp. 98-103). https://doi.org/10.1145/3375258.3375270

Newell, A., & Simon, H. A. (1972). *Human Problem Solving* (Vol. 104, No. 9). Prentice-Hall.

Odisho, O., Aziz, M., & Giacaman, N. (2016). Teaching and Learning Data Structure Concepts via Visual Kinesthetic Pseudocode With the Aid of a Constructively Aligned App. *Computer Applications in Engineering Education*, 24(6), 926-933. https://doi.org/10.1002/cae.21768

Pólya, G. (1945). How to Solve It. *Education and Information Technologies*, 15(2), 87-107.

Parham, J., Chinn, D., & Stevenson, D. E. (2009, March). Using Bloom's Taxonomy to Code Verbal Protocols of Students Solving a Data Structure Problem. *Proceedings of the 47th Annual Southeast Regional Conference* (pp. 1-6). https://doi.org/10.1145/1566445.1566499

Parham, J., Gugerty, L., & Stevenson, D. E. (2010, March). Empirical Evidence for the Existence and Uses of Metacognition in Computer Science Problem Solving. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 416-420). https://doi.org/10.1145/1734263.1734406

Perrenet, J. C. (2010). Levels of Thinking in Computer Science: Development in Bachelor Students' Conceptualization of Algorithm. *Education and Information Technologies*, 15, 87-107. https://doi.org/10.1007/s10639-009-9098-8

Schoenfeld, A. H. (1985). *Mathematical Problem Solving*. Academic Press.

Schoenfeld, A. H. (1992). Learning to Think Mathematically: Problem Solving, Metacognition, and Sense-Making in Mathematics, In D. Grouws (Ed.), *Handbook for Research on Mathematics Teaching and Learning* (pp. 334-370). Macmillan.

Smith III, J. P., DiSessa, A. A., & Roschelle, J. (1994). Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *The Journal of the Learning Sciences*, 3(2), 115-163. https://doi.org/10.1207/s15327809jls0302_1

Stake, R. E. (1995). *The Art of Case Study Research*. SAGE Publications.

van Someren, M. W., Barnard, Y. F., & Sandberg, J. A. C. (1994). *The Think Aloud Method: A Practical Guide to Modelling Cognitive Processes*. Academic Press.

Wall, J. D., & Knapp, J. (2014). Learning Computing Topics in Undergraduate Information Systems Courses: Managing Perceived Difficulty. *Journal of Information Systems Education*, 25(3), 245-259.

Walther, J., Sochacka, N. W., & Kellam, N. N. (2013). Quality in Interpretive Engineering Education Research: Reflections on an Example Study. *Journal of Engineering Education*, 102(4), 626-659. https://doi.org/10.1002/jee.20029

Wang, Z. (2012). Research on Teaching Ideas of "Data Structures and Algorithms" in Non-Computer Majors. *Advances in Computer Science and Education* (pp. 249-254). Springer. https://doi.org/10.1007/978-3-642-27945-4_39

Wilson, M. R., Lesh, R., & Pollock, E. (1993). *Foundations of Mathematics Assessment*. ERIC.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35. https://doi.org/10.1145/1118178.1118215

Yazan, B. (2015). Three Approaches to Case Study Methods in Education: Yin, Merriam, and Stake. *The Qualitative Report*, 20(2), 134-152. https://doi.org/10.46743/2160-3715/2015.2102

Yimer, A., & Ellerton, N. F. (2010). *Problem-Solving in Mathematics Education*. Sense Publishers.

Yin, R. K. (2014). *Case Study Research: Design and Methods*. SAGE Publications.

AUTHOR BIOGRAPHIES

Orly Barzilai is a lecturer at the School of Information Systems at The Academic College of Tel Aviv Yaffo. She has more than 20 years of experience in the information systems industry. She is signed on a backup algorithm patent approved by the US Patent Office. Her research interests include smart transportation, abstract thinking, and data science. Orly received her Ph.D. from the Industrial Engineering and Management department at the Technion - Israel Institute of Technology.

Sofia Sherman is a senior lecturer in the School of Information Systems at The Academic College of Tel Aviv Yaffo. Her research interests include requirements engineering, architecture processes and the role of the architect in different development methodologies, and human aspects of software engineering. Sherman received her Ph.D. from the Department of Information Systems at the University of Haifa, Israel.

Moshe Leiba is an assistant professor at the School of Information Systems at The Academic College of Tel Aviv Yaffo. He also acts as Deputy Director General of World ORT Kadima Mada. Moshe holds a BSc. in Electrical Engineering, a Teaching Certificate, a MA (Magna Cum Laude) and a Ph.D. focusing on Digital Education from Tel Aviv University. Moshe participated in several European ICT research programs (FP7, FP6, Erasmus) as a researcher and has experience as a policy maker in formal education systems at governmental and municipal levels.

Hadar Spiegel is a mathematics lecturer in the School of Information Systems, at The Academic College of Tel Aviv-Yaffo, Israel. Spiegel holds a Ph.D. in Mathematics Education from Tel Aviv University, Israel. Her fields of research include heuristic techniques for mathematical problem-solving, aspects of analyzing mathematical problems and their solutions, mathematical creativity, and aesthetics in mathematical problem-solving.

APPENDICES

Appendix A. Research Assignment Description

The central queue is a data structure that supports all the activities defined for a regular queue, in addition to insertion, extraction, and reading from the middle of the queue.

The index of the middle of the queue is defined as the highest integer at or below n/2 in a queue that contains *n* elements; for example, if the queue contains 10 elements, then the index of the middle of the queue is the 5th element. If the queue contains 11 elements, then the index of the middle of the queue is also the 5th element.

Suggest an implementation of the central queue, based on two double-ended queue structures, H and T.

Below is the description of the structure of a double-ended queue:

Double-ended queue
Data structure on which the following actions are defined:
Insert item to the head of the structure – insert_head(Q, X)
Insert item to the tail of the structure – insert_tail(Q, X)
Read item from the head of the structure – read_head(Q)
Read item from the tail of the structure – read_tail(Q)
Extract item from the head of the structure – delete_head(Q)
Extract item from the tail of the structure – delete_tail(Q)
Is the structure empty? – is_empty(Q)
Complexity
For all activities: time complexity O(1); space complexity O(n)
*The double-ended queue structure is implemented via an array

Below is the description of the structure of a central queue:
Insert item (performed to the tail of the central queue): insert_tail(MQ, X) (complexity O(1))
Read item ((performed on the head of the central queue): read_head(MQ) (complexity O(1)).
Extract item ((performed to the head of the central queue): delete_head(MQ) (complexity O(1)).
Insert item in the middle (performed to the middle of the central queue): insert_mid(MQ,X) (complexity O(1)).

Read item from the middle ((performed on the middle of the central queue): read_mid(MQ) (complexity O(1)).
Extract item ((performed to the middle of the central queue): delete_mid(MQ) (complexity O(1)).
Is the central queue empty?: is_empty(MQ) (complexity O(1)).

a. Write a verbal description of a data structure than supports the description of the central queue.
b. Implement by a verbal description and by pseudocode the following activities which the central queue supports (all activities with a time complexity of O (1)):
    i. Insertion to the tail of the queue
    ii. Insertion in the middle of the queue
    iii. Deletion from the middle of the queue
c. Explain the complexity of each activity.

Appendix B. Research Assignment Description of Compound Data Structures and Solution

Central Queue

A central queue is a compound data structure that supports all data operations of a queue, in addition to supporting those activities performed on the middle of the queue. Figure B-1 illustrates the operations performed on a central queue.

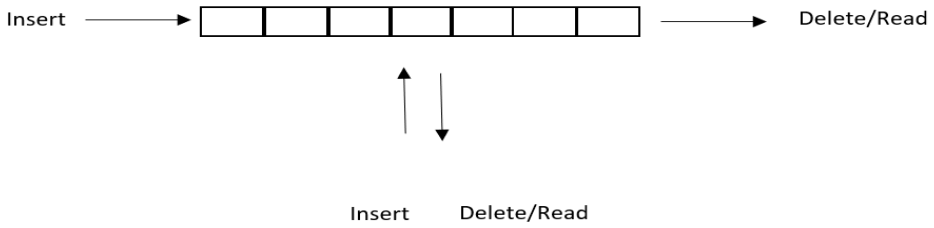


Figure B-1. Illustration of a Central Queue

Double-Ended Queues

The central queue is built from two double-ended queues. Figure B-2 illustrates the operations performed on a double-ended queue. A double-ended queue is an enhancement of a queue that supports all data operations of a queue performed from both sides (front and back).

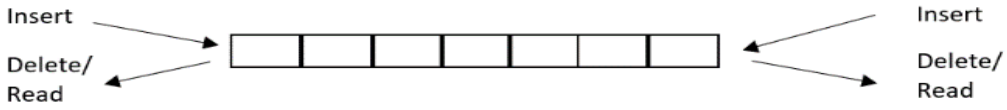


Figure B-2. Illustration of a Double-Ended Queue

Central Queue Operations

Students were asked to solve three operations using double-ended queue operations:

- Insert to tail
- Insert in middle
- Delete from middle

The "insert to tail" solution is simple when considered independently of the two other operations. The insert_tail(Q, X) operation of the double-ended queue is used (see Appendix A).

However, to support the other two operations, "insert in middle" and "delete from middle," a new algorithm should be developed, ensuring that items are divided between the two double-ended queues. Figure B-3 illustrates the "insert tail" operation using the two double-ended queues.

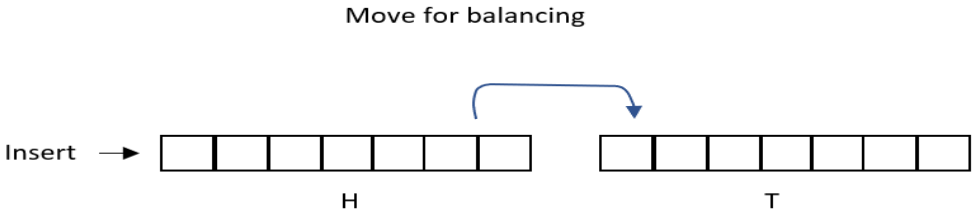


Figure B-3. Illustration of the "Insert Tail" Operation

The "insert tail" operation contains the following actions:

- Insert an item to the front of H (head of the double-ended queue). To do that, use the double-ended queue's insert_head(Q, X) operation (see Appendix A).
- Balancing H (Head) and T (Tail) double-ended queues.

The "insert in middle" operation contains the following actions:

- Insert an item to the tail of H. To do that, use the double-ended queue's insert_tail(Q, X) operation.
- Balancing H (Head) and T (Tail) double-ended queues.

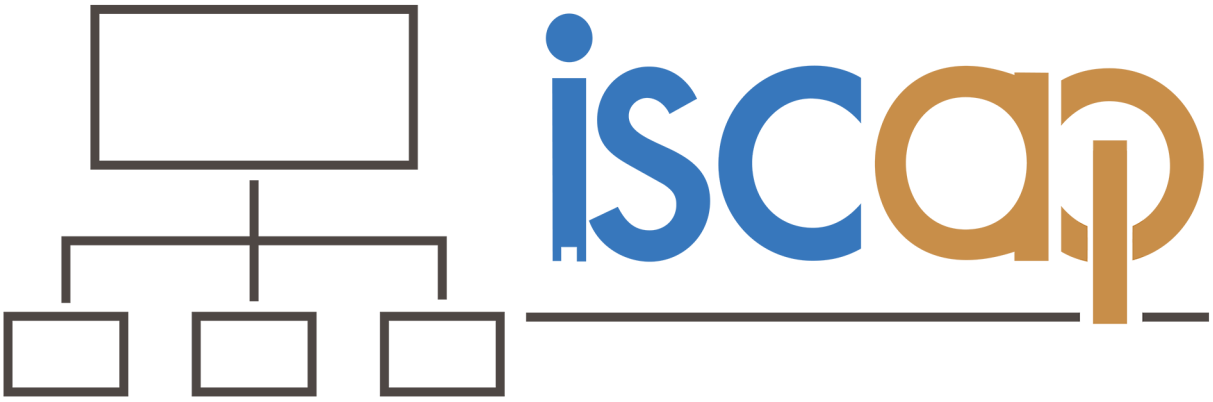
The "delete from middle" operation contains the following actions:

- Delete an item from the tail of H. To do that, use the double-ended queue's delete_tail(Q) operation.
- Balancing H (Head) and T (Tail) de-queues.

Balancing H (Head) and T (Tail) double-ended queues:

- If H contains more items than T, pass one item from H to T.

INFORMATION SYSTEMS & COMPUTING ACADEMIC PROFESSIONALS



STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the ***Journal of Information Systems Education*** have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN: 2574-3872 (Online) 1055-3096 (Print)