APPLICATION OF DECOMPOSITION IN THE COMPUTATIONAL THINKING CONCEPT THROUGH THE MERGE SORT ALGORITHM

Nusratova Shakhzodakhon Bakhtiyorovna
Assistant , Department of Software and
Technical Support of Computer Systems,
Faculty of Digital Technologies and Artificial Intelligence,
Karshi State Technical University
nusratovashahzoda1010@gmail.com

Abstract. This article is dedicated to the course **"Introduction to Computational Thinking and Programming"**, which aims to develop fundamental skills in computational thinking, algorithmic reasoning, problem analysis, and systematic problem-solving. The study examines the core concepts of computational thinking and their role in addressing real-world challenges through structured and logical approaches. Particular attention is given to the four key pillars of computational thinking: **decomposition, pattern recognition, abstraction, and algorithm design**. These concepts are explained through practical and accessible examples that demonstrate how complex problems can be broken down into manageable components and transformed into efficient computational solutions.

In addition, the article introduces the essential foundations of the **Python programming language** as a practical and beginner-friendly tool for implementing computational ideas. Python is presented as an effective environment for translating theoretical concepts into executable programs due to its simplicity, readability, and widespread application across various technological domains. The paper also highlights the growing importance of computational thinking in modern education and workforce development, emphasizing its contribution to critical thinking, creativity, and digital literacy. The findings suggest that integrating computational

thinking principles with Python programming enhances learners' analytical abilities, supports algorithmic problem-solving, and provides a strong foundation for further studies in computer science, software development, and emerging digital technologies.

Keywords: Computational Thinking, Programming, Python, Algorithm Design, Decomposition, Abstraction, Pattern Recognition, Problem Solving, Computer Science Education, Digital Literacy.

Introduction. Computational thinking is a set of interconnected skills and practices used to solve complex problems, a method for learning and understanding concepts across various disciplines, and an essential requirement for full participation in the modern computational world. When discussing computational thinking and programming, a wide range of terms and concepts are commonly used. Computing encompasses the skills and practices associated with both computer science and computational thinking, providing the foundation for problem-solving, algorithmic reasoning, and the effective use of technology in various domains. Although Computer Science is an independent academic discipline, Computational Thinking is a problem-solving approach that can be integrated into a wide range of activities and fields. Programming, on the other hand, is the practice of designing and creating a set of instructions that a computer can understand and execute. It also involves debugging, organizing, testing, and applying code within appropriate problem-solving contexts to develop effective and efficient solutions. The skills and practices associated with Computational Thinking are broad in scope and draw upon concepts and techniques from Computer Science. These skills can be applied across core academic disciplines, such as arts, language studies, mathematics, science, and social sciences, as well as in everyday problem-solving situations. Computational Thinking enables individuals to analyze problems systematically, develop logical solutions, and transfer problem-solving strategies across diverse contexts. For educators who integrate Computational Thinking into their classrooms, it is best

understood as a collection of interconnected skills and competencies that work together to support effective problem-solving, critical thinking, and learning across various subject areas.

Methods. Problems Solved through Algorithms.

An algorithm for solving a problem, expressed as a sequence of instructions in the language of a computing machine (machine code), is called a **computer program**. Such a program provides the computer with a precise set of operations to be executed in a specific order in order to obtain the desired solution to a given problem.

A machine program instruction, or machine instruction, is an elementary machine-level command that is executed automatically by the computer without requiring any additional explanations, interpretations, or user intervention. It represents the most basic operation that the processor can directly understand and perform.

Programming is the theoretical and practical activity associated with the design, development, implementation, and maintenance of computer programs. It involves creating algorithms, writing code, testing, debugging, and optimizing software solutions to address specific problems or perform desired tasks.

The process of converting an algorithm into machine language is called translation. The first step toward making machine language more accessible to humans was the development of programs that translated symbolic names into machine code. Later, translators capable of converting arithmetic expressions were introduced. Finally, in 1958, the FORTRAN translator, one of the first widely used programming language translators, was developed. This milestone significantly advanced software development and led to the creation of numerous programming languages that are widely used today.

Computational Thinking is an approach to problem-solving and system understanding that involves the application of mathematical and logical reasoning.

This concept is important not only in computer science but also in a wide range of disciplines, where it helps individuals analyze complex problems, develop structured solutions, and make informed decisions. Computational Thinking provides a systematic framework for breaking down problems, identifying patterns, abstracting essential information, and designing effective algorithms to solve real-world challenges.

The key stages of Computational Thinking include:

1. **Problem Identification** – Clearly defining and understanding the problem or task that needs to be solved. A well-defined problem serves as the foundation for developing an effective solution.
2. **Decomposition** – Breaking a complex problem into smaller, manageable components and analyzing each part separately. This process makes complex tasks easier to understand and solve.
3. **Pattern Recognition** – Identifying similarities, trends, or recurring structures among problems. Recognizing patterns enables the reuse of previous solutions and improves problem-solving efficiency.
4. **Abstraction** – Selecting the most relevant details of a problem or system while filtering out unnecessary information. This helps focus on the essential aspects required for developing a solution.
5. **Algorithm Design** – Developing a clear and systematic sequence of instructions or steps to solve a given problem or accomplish a specific task. Effective algorithms ensure accuracy, efficiency, and consistency in problem-solving.

The decomposition process can be described through the following steps:

- **Problem Definition:** The problem must first be clearly identified and understood, including determining the main objective and desired outcome.
- **Breaking the Problem into Parts:** Complex tasks or challenges are divided into several smaller and more manageable subproblems. For example, the task of

developing a website can be divided into components such as **design**, **coding**, and **testing**.

- **Solving Each Part Individually:** Each subproblem is analyzed separately, and an appropriate solution is developed for that specific component.
- **Combining the Results:** Once all subproblems have been solved, their solutions are integrated to produce a complete and effective solution to the original problem.

This structured approach simplifies complex tasks, improves efficiency, and supports systematic problem-solving within the framework of computational thinking.

This process is widely used not only in computing but also in various other fields, such as engineering, scientific research, business, and project management. Through decomposition, the process of solving complex problems becomes more structured, manageable, and efficient. By dividing a large problem into smaller components, individuals can focus on each part separately, leading to improved understanding, accuracy, and productivity.

Decomposition provides several important advantages, including:

- **Focused Attention:** It allows individuals to concentrate on a single small task at a time rather than dealing with multiple complex problems simultaneously, improving clarity and efficiency.
- **Easier Error Detection:** If a system does not function correctly, it is much faster to identify which specific component contains the error instead of analyzing the entire system at once.
- **Resource Allocation:** Different parts of a large project can be distributed among specialists or processed in parallel across multiple computing units, improving productivity and reducing completion time.

Overall, these benefits make decomposition a powerful strategy for managing complexity in both computational and real-world problem-solving contexts.

If you are assigned the task of "creating a personal website," it can be decomposed as follows:

1. Design: Selecting a color palette and creating the page layout structure.

2. Content: Writing the textual materials and preparing images and other media resources.

3. Development (Programming): Writing HTML and CSS code and adding necessary functionality to implement the website structure and interactivity.

Each decomposed subcomponent can further be divided into even smaller steps, which ensures a systematic and structured approach to achieving the final outcome.

Real-Life Examples (Extended Version)

- **In Programming:** The "Merge Sort" (divide-and-conquer sorting) algorithm recursively divides an array into smaller parts until each element stands alone, and then merges them back in a sorted order.
- **In Daily Life:** When organizing a school event, the process is decomposed into separate tasks such as booking a venue, sending invitations, and purchasing necessary materials.
- **In Biology:** The breakdown of complex organic substances into simpler components, such as water and carbon dioxide, can also be considered a natural form of decomposition.

Results. As an example, we consider the application of the decomposition approach in the Merge Sort algorithm. Merge Sort is one of the optimal sorting algorithms with a time complexity of **O(N log N)**. It operates with **O(N log N)** asymptotic performance in both the best-case and worst-case scenarios, making it highly efficient for large datasets. In addition, Merge Sort is a **stable sorting algorithm**, meaning that it preserves the relative order of equal elements in the input sequence. This property is particularly important in applications where maintaining the original ordering of equivalent items is required. For example, when elements

contain not only values but also associated keys, stability becomes important. If two elements have the same value, such as the number **2**, but different keys (e.g., one with key **3** and another with key **5**), then after sorting with Merge Sort, the element with key **3** will still appear before the element with key **5**. This is because Merge Sort is a **stable sorting algorithm**, meaning it preserves the original relative order of equal elements.

The working principle of the algorithm is as follows: the array is divided into two equal halves. The right half is sorted separately, and the left half is also sorted independently. Then, the two sorted halves are merged in **O(n)** time, meaning using *n* operations, to obtain a fully sorted array. It is precisely this merging operation that gives the algorithm its name, **Merge Sort**.

To sort the left and right halves, the same process is repeated recursively: each sub-array is again divided into two parts, and the same operations are applied. This recursive division continues until each sub-array contains only two elements. At that point, the elements are easily sorted and merged step by step back into larger sorted sequences, ultimately producing the final sorted array.

Discussion. Let us assume we have a function Merge(a: Array of Integer, Left, Mid, Right: Integer), where the subarray a[Left..Mid] is already sorted and the subarray a[Mid + 1..Right] is also sorted. The purpose of this function is to merge these two sorted subarrays into a single sorted segment within the range a[Left..Right].

Here, Mid represents the middle index that divides the interval [Left, Right] into two parts. The merging process compares elements from both subarrays and places them in correct order into a temporary array, ensuring that the resulting segment a[Left..Right] becomes fully sorted. After completing the merge operation, the sorted result is copied back into the original array.

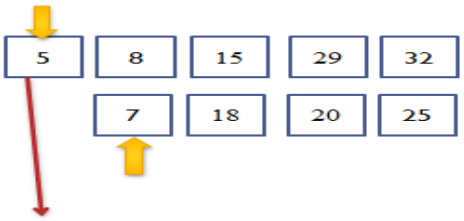


Figure 1.

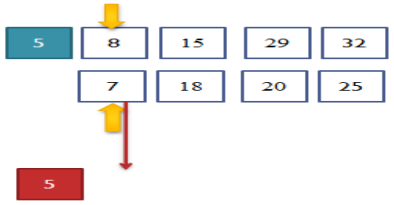


Figure 2.

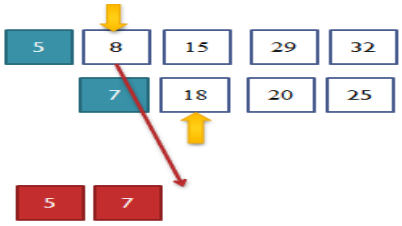


Figure 3.

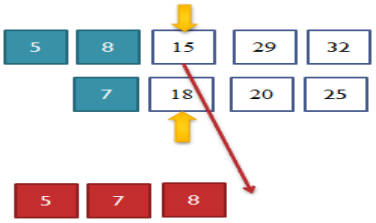


Figure 4.

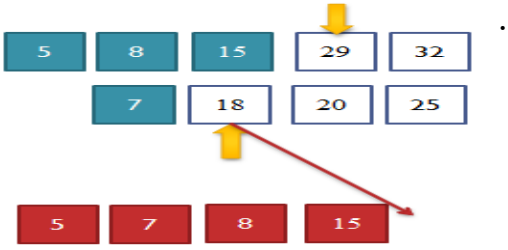


Figure 5.

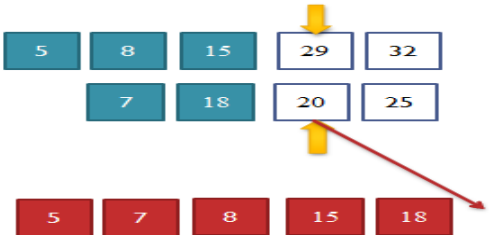
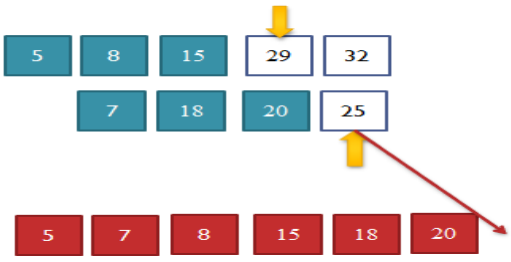


Figure 6.



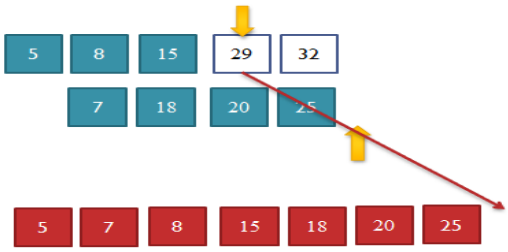


Figure 7.

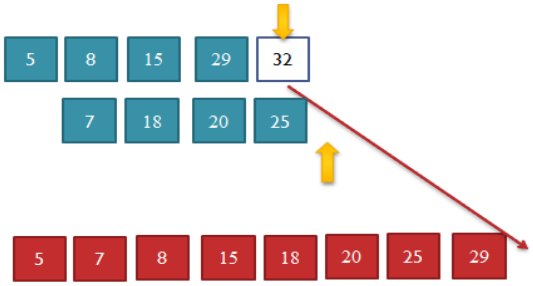


Figure 8.

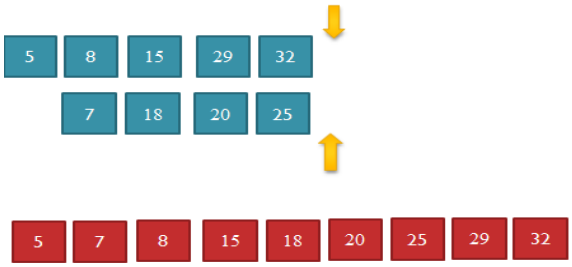


Figure 9.

Figure 10.

The interval [L, R] is divided into two subintervals using the midpoint m = (L + R) / 2, resulting in [L, m] and [m + 1, R], which are then sorted separately.

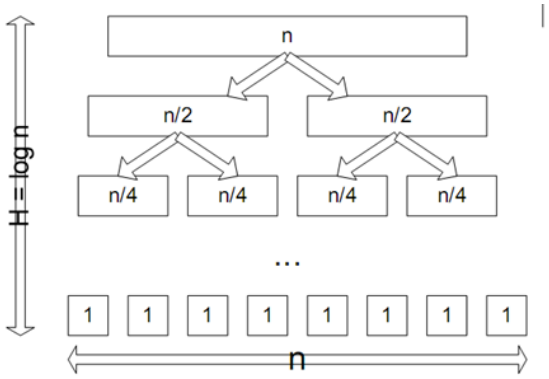


Figure 11. Recursive Decomposition Tree of the Merge Sort Algorithm.

Merge Sort is a divide-and-conquer sorting technique in which the array is repeatedly split into smaller subproblems and sorted independently. Finally, the sorted subarrays are merged to obtain the overall sorted result. This step-by-step division followed by gradual combination clearly represents the essence of the decomposition approach.

Conclusion. In conclusion, the Merge Sort algorithm is one of the most important pillars of computational thinking and serves as a perfect example of the decomposition concept in practice. The principle of solving complex and large-scale

problems by breaking them down into smaller, more manageable parts forms the core of this algorithm.

Effectiveness of decomposition: The Merge Sort algorithm relies on the "divide and conquer" strategy, recursively splitting an array into two parts until only single elements remain. This process demonstrates how effectively a problem can be simplified for both the human mind and computer memory when dealing with large-scale data.

Formation of computational thinking: Studying this topic develops not only coding skills in students and programmers, but also fundamental analytical thinking abilities such as systematically breaking down real-life problems into parts, organizing them, and optimizing solutions.

In the era of vast and rapidly growing data streams, understanding the synergy between Merge Sort and decomposition serves as a key principle for building stable and optimal systems in software engineering and algorithm design.

References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

2. Downey, A. B. (2015). *Think Python: How to think like a computer scientist* (2nd ed.). Green Tea Press. https://greenteapress.com/wp/think-python/

3. Knuth, D. E. (1998). *The art of computer programming, Volume 3: Sorting and searching* (2nd ed.). Addison-Wesley.

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

5. Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Data structures and algorithms*. Addison-Wesley.

6. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.

7. Grokking Algorithms. (2016). Bhargava, A. Y. *Grokking algorithms: An illustrated guide for programmers and other curious people*. Manning Publications.

8. Guttag, J. V. (2016). *Introduction to computation and programming using Python* (2nd ed.). MIT Press.

9. Python Software Foundation. (2024). *The Python tutorial*. https://docs.python.org/3/tutorial/