SPL-1 Project Report,2023

Data Structure Simulator

SE: 305

Submitted by

Habibur Rahman Mahin

BSSE Roll no: 1422

BSSE Session:2021-2022

Supervised by

Dr. Ahmedul Kabir

Designation: Associate Professor

Institute of Information Technology

Supervisor's Approval [illegible]

(signature)

Institute of Information Technology

University of Dhaka

[17-12-2023]

Table of Contents

1.Introduction...3
2.Background of the Project...3
2.1 Data Structures and Algorithms: A Brief Overview...3

3.Description of the project...5
3.1 Objectives..5
3.2 Features..5
3.3 Workflow...6

4.Implementation and Testing...8
4.1 Technologies Used...8
4.2 Simulation Logic..8
4.3 Implementation...9
4.4 Testing..15

5.User Interface..15
6.Challenges Faced..16
7.Conclusion...16
7.1 Lessons Learned...16
7.2 Future Extensions...17
7.3 Final Reflection..17

Reference..18

1. Introduction

The Data Structure Simulator is an interactive learning tool designed to provide a dynamic and engaging platform for users to explore and comprehend various data structures and sorting algorithms .It is designed to provide a visual and interactive learning experience for users to understand various data structures and sorting algorithms.The simulator aims to help beginners and students grasp the underlying concepts of fundemental data structures like arrays, stacks ,queues, trees ,graphs and sorting algorithm techniques ,such as bubble sort ,insertion sort,merge sort etc. This report delves into the background, description, implementation, and testing of the project, shedding light on the user interface, challenges faced during development, and concluding with key takeaways.

2. Background of the Project

Understanding data structures is a fundamental aspect of computer science education. However, learners often find these abstract concepts challenging to grasp without visual aids and practical applications. The Data Structure Simulator aims to bridge this gap by offering an intuitive and interactive environment where users can visualize and interact with different data structures and sorting algorithms.

2.1 Data Structures and Algorithms: A Brief Overview

2.1.1 Data Structures

Data structures form the backbone of computer science, providing a systematic way to organize and store data for efficient retrieval and manipulation. These structures define the relationship between data elements, facilitating the execution of algorithms. They are broadly categorized into linear and non-linear data structures.

Linear Data Structures

Arrays: Arrays are collections of elements stored in contiguous memory locations. Elements are accessed using indices, offering quick retrieval but limited flexibility in size modification.

Linked Lists: Linked lists consist of nodes where each node points to the next one, forming a chain-like structure. They allow dynamic memory allocation and efficient insertion and deletion.

Stacks: Stacks follow the Last In, First Out (LIFO) principle. Elements are added and removed from the same end, simulating a stack of items.

Queues: Queues adhere to the First In, First Out (FIFO) principle. Elements are added at the rear and removed from the front, mimicking a real-world queue.

Non-Linear Data Structures

Trees: Trees are hierarchical structures with a root node and branches leading to various nodes. Binary trees, AVL trees, and B-trees are common variants.

Graphs: Graphs represent a collection of nodes and edges connecting them. They can be directed or undirected and may have weighted edges.

2.1.2 Algorithms

Algorithms are step-by-step procedures or sets of rules designed to solve specific problems or perform tasks. They operate on data structures and are crucial for efficient data processing. Algorithms can be classified based on their design paradigm:

Sorting Algorithms

Bubble Sort: Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

Merge Sort: Divides the unsorted list into n sub-lists, each with one element, and repeatedly merges sub-lists to produce new sorted sub-lists.

Quick Sort: Selects a 'pivot' element and partitions the other elements into two sub-arrays according to whether they are less than or greater than the pivot.

Searching Algorithms

Binary Search: Searches for a target value within a sorted array, repeatedly dividing the search range in half.

Depth-First Search (DFS): Explores as far as possible along each branch before backtracking, commonly used for traversing graphs.

Breadth-First Search (BFS): Explores all neighbor nodes at the current depth before moving on to nodes at the next depth level.

Understanding these fundamental concepts forms the basis for creating the interactive simulator for data structures and algorithms. The project aims to provide an engaging platform for users to visualize and comprehend the functioning of these structures and algorithms.

3. Description of the Project

3.1 Objectives

The primary objectives of the Data Structure Simulator include:

- Providing an interactive platform for learning data structures.
- Simulating various data structures, including trees, graphs, linked lists, and arrays.
- Visualizing sorting algorithms to enhance understanding.

3.2 Features

Key features of the project include:

- Interactive Visualization: Users can see a graphical representation of different data structures, such as trees, enabling a better understanding of hierarchical relationships.

- Sorting Algorithm Simulation: The simulator allows users to observe the step-by-step execution of sorting algorithms, including Bubble Sort, Merge Sort, Quick Sort, Insertion Sort, and Selection Sort.

- User-Friendly Interface: The project incorporates an intuitive user interface, making it accessible to learners at different levels of expertise.

-Dynamic Data Input: Users have the option to input their data elements to most of the data structures and sorting algorithms. This flexibility allows them to see firsthand how the structures handle different data sets and how sorting algorithms arrange them in real-time.

3.3 Workflow

3.3.1 Learning Phase

The journey of developing the Data Structure Simulator commenced with an extensive learning phase. This phase involved a deep dive into data structures, sorting techniques, and graphics programming in C++. Various resources were explored, including traditional programming books on C and C++, specialized books on data structures, and numerous video tutorials available on platforms like YouTube.

3.3.2 Skill Acquisition

To implement the simulator effectively, acquiring a solid understanding of C++ programming was crucial. This involved honing skills in memory management, data manipulation, and algorithmic thinking. Special emphasis was placed on mastering the graphics.h library, as the visual aspect of the project was fundamental to its purpose.

3.3.3 Design and Planning

Once the foundational knowledge was in place, the next step involved designing the project. This phase included outlining the structure of the simulator, deciding on the key features, and planning the user interface. Considerable time was spent on determining how different data structures would be visually represented and how sorting algorithms would be simulated step by step.

3.3.4 Implementation

The implementation phase saw the transformation of design into code. The simulator was developed in C++, leveraging the graphics.h library for visual representation. Each data structure, whether it be a tree or an array, was meticulously implemented, considering factors like node creation, deletion, and traversal.

For sorting algorithms, stepwise visualization was achieved by breaking down the logic into executable segments. The project's flexibility was enhanced by allowing users to input their data for simulation, providing a hands-on learning experience.

3.3.5 Iterative Development

The development process was iterative, with continuous testing and refinement. As new features were added, such as different sorting algorithms, the simulator underwent multiple iterations to ensure smooth functionality. User feedback from initial testing rounds was incorporated to enhance the user interface and overall user experience.

3.3.6 Resource Utilization

Throughout the workflow, the knowledge gained from books and online tutorials played a vital role. Understanding the intricacies of C++ and graphics.h library functions, as well as gaining insights into data structure implementation, was facilitated by referring to these resources. The ability to translate theoretical concepts into practical code was a direct result of this resource utilization.

3.3.7 Challenges Overcome

Challenges encountered during development, such as compatibility issues with the graphics.h library and the complexity of implementing step-by-step visualizations for sorting algorithms, were addressed through a combination of research, trial and error, and collaboration with the programming community.

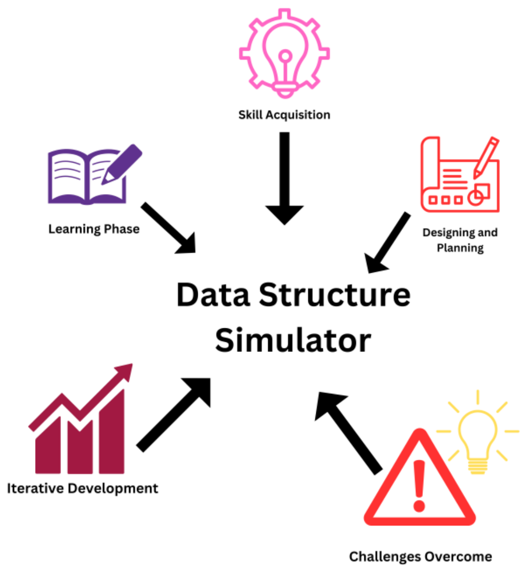


Fig 1 - Actions that led to the final project

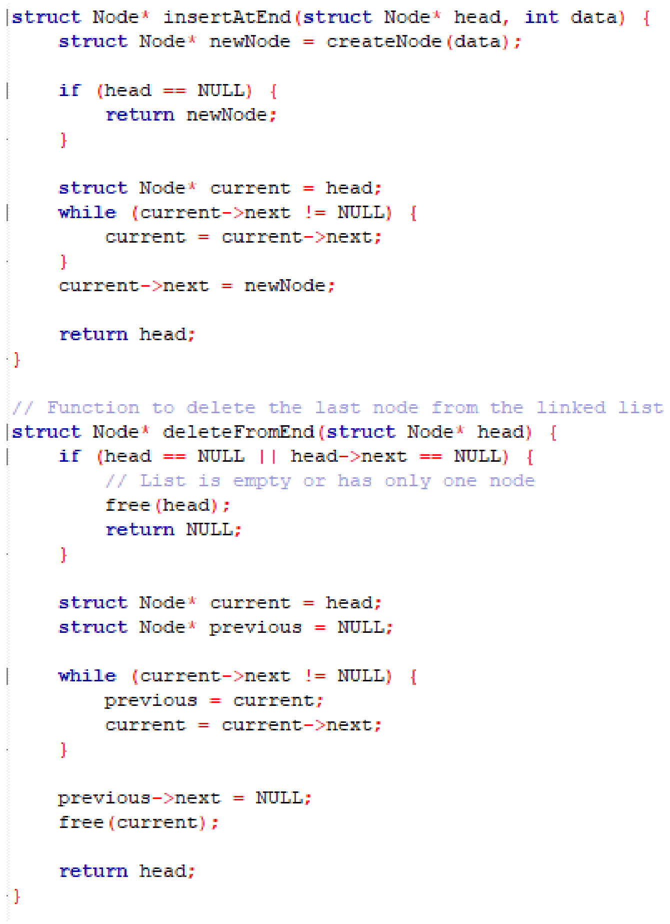
4. Implementation and Testing

4.1 Technologies Used

The Data Structure Simulator is implemented using C++ for the backend logic and the graphics.h library for graphical representation. This combination provides a robust foundation for simulating data structures and algorithms.

4.2 Simulation Logic

The simulation logic involves the dynamic creation and manipulation of data structures based on user input. For instance, when simulating a linked list, nodes are created, and the user can perform operations like insertion and deletion on that linked list.



```
struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);

    if (head == NULL) {
        return newNode;
    }

    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;

    return head;
}

// Function to delete the last node from the linked list
struct Node* deleteFromEnd(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        // List is empty or has only one node
        free(head);
        return NULL;
    }

    struct Node* current = head;
    struct Node* previous = NULL;

    while (current->next != NULL) {
        previous = current;
        current = current->next;
    }

    previous->next = NULL;
    free(current);

    return head;
}
```

Fig 2 - Linked list’s insertion and deletion operation

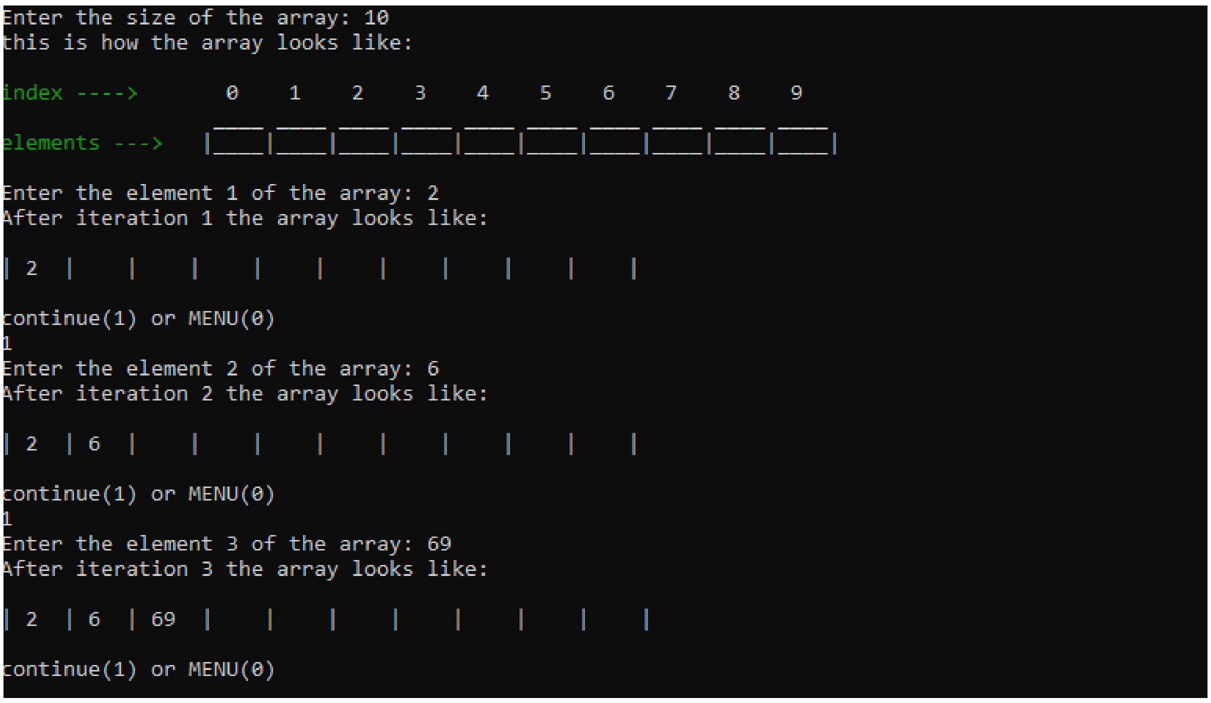
4.3 Implementation

The heart of the project lies in its simulation logic. This section delves into how different data structures and sorting algorithms were implemented and visualized.

4.3.1 Data Structure Simulation

Arrays

The array simulation involved dynamic memory allocation to accommodate varying user inputs. Elements were added, removed, and modified, showcasing the fundamental operations performed on arrays. The visual representation allowed users to comprehend the dynamic nature of array operations.

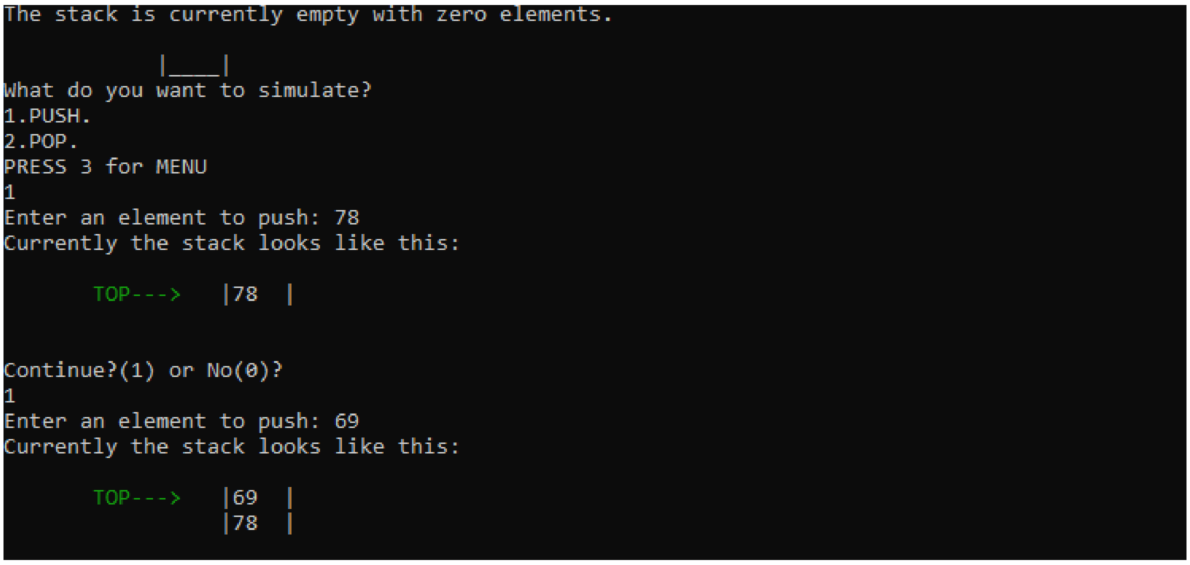


```
Enter the size of the array: 10
this is how the array looks like:

index ---->       0    1    2    3    4    5    6    7    8    9
                ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
elements --->  |____|____|____|____|____|____|____|____|____|____|

Enter the element 1 of the array: 2
After iteration 1 the array looks like:

| 2  |    |    |    |    |    |    |    |    |    |

continue(1) or MENU(0)
1
Enter the element 2 of the array: 6
After iteration 2 the array looks like:

| 2  | 6  |    |    |    |    |    |    |    |    |

continue(1) or MENU(0)
1
Enter the element 3 of the array: 69
After iteration 3 the array looks like:

| 2  | 6  | 69  |    |    |    |    |    |    |    |

continue(1) or MENU(0)
```

Fig 3 - Array simulation

Stacks

For stack simulation, the push and pop operations were implemented, visually reflecting the LIFO (Last-In-First-Out) behavior. Each iteration displayed the real-time state of the stack, aiding users in grasping the stack's dynamic nature.

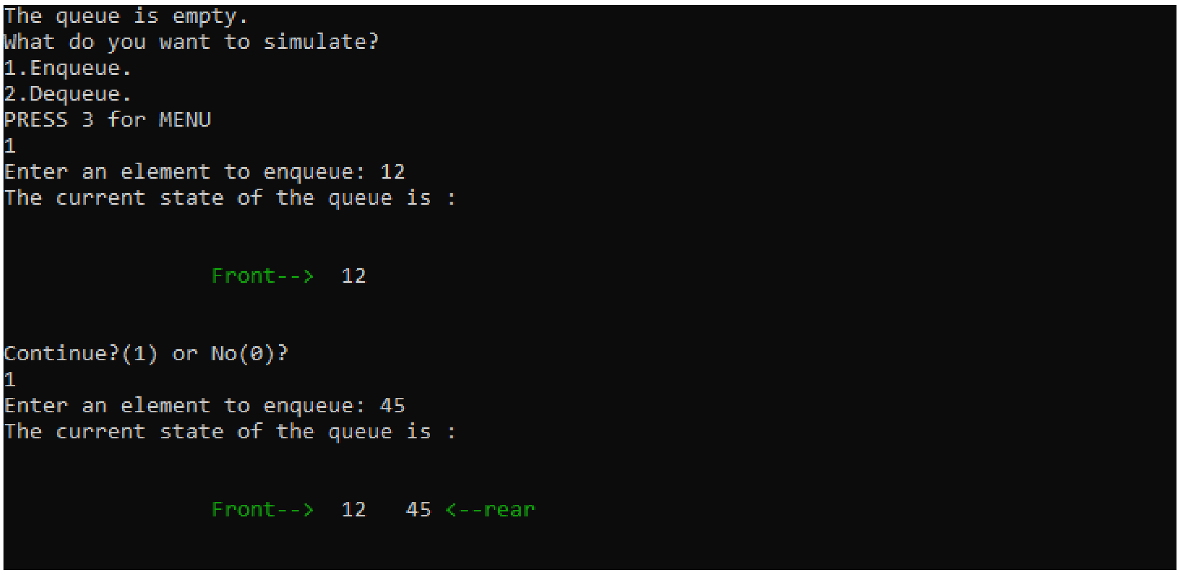


```
The stack is currently empty with zero elements.

            |____|
What do you want to simulate?
1.PUSH.
2.POP.
PRESS 3 for MENU
1
Enter an element to push: 78
Currently the stack looks like this:

       TOP--->   |78  |


Continue?(1) or No(0)?
1
Enter an element to push: 69
Currently the stack looks like this:

       TOP--->   |69  |
                 |78  |
```

Fig 4 - Stack simulation

Queues

Queue simulation showcased the enqueue and dequeue operations. The visual representation demonstrated the FIFO (First-In-First-Out) concept, emphasizing the orderly processing of elements in a queue.

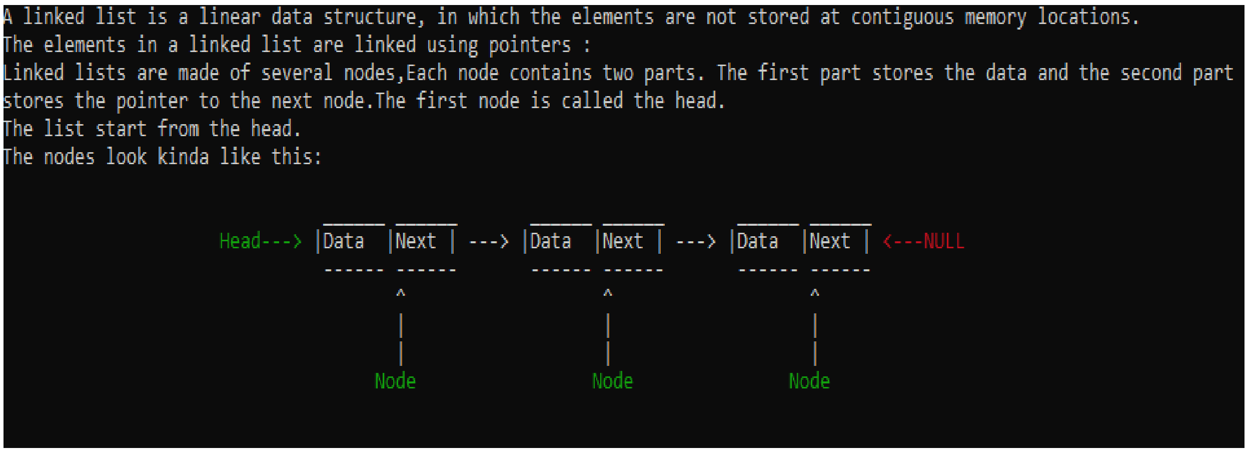


```
The queue is empty.
What do you want to simulate?
1.Enqueue.
2.Dequeue.
PRESS 3 for MENU
1
Enter an element to enqueue: 12
The current state of the queue is :


                Front-->  12


Continue?(1) or No(0)?
1
Enter an element to enqueue: 45
The current state of the queue is :


                Front-->  12   45 <--rear
```

Fig 5 - Queue simulation

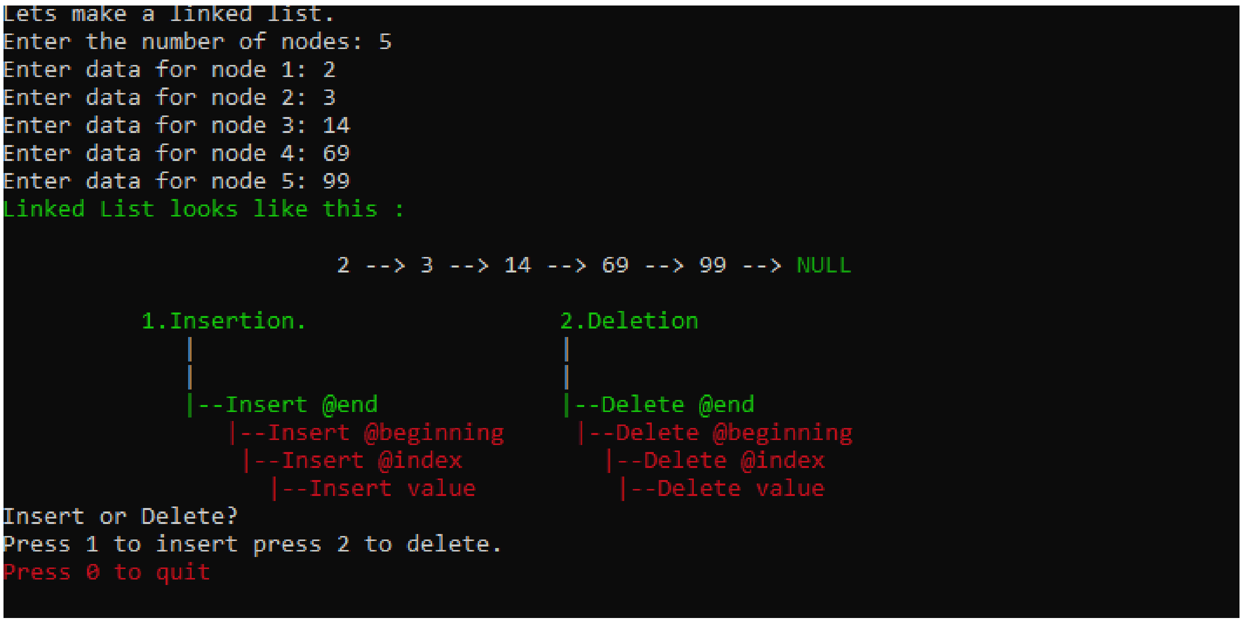
Linked Lists

The linked list simulation involved the creation of nodes, insertion, and deletion operations. Users could witness how linked lists handle dynamic data and appreciate the ease with which elements can be inserted or removed.



```
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
The elements in a linked list are linked using pointers :
Linked lists are made of several nodes,Each node contains two parts. The first part stores the data and the second part
stores the pointer to the next node.The first node is called the head.
The list start from the head.
The nodes look kinda like this:

                     ______ ______        ______ ______        ______ ______
          Head---> |Data  |Next |  ---> |Data  |Next |  ---> |Data  |Next |  <---NULL
                     ------ ------        ------ ------        ------ ------
                          ^                    ^                    ^
                          |                    |                    |
                          |                    |                    |
                         Node                 Node                 Node
```

Fig 6 - Linked list visualized



```
Lets make a linked list.
Enter the number of nodes: 5
Enter data for node 1: 2
Enter data for node 2: 3
Enter data for node 3: 14
Enter data for node 4: 69
Enter data for node 5: 99
Linked List looks like this :

                        2 --> 3 --> 14 --> 69 --> 99 --> NULL

          1.Insertion.                    2.Deletion
             |                              |
             |                              |
             |--Insert @end                 |--Delete @end
                |--Insert @beginning         |--Delete @beginning
                 |--Insert @index              |--Delete @index
                   |--Insert value              |--Delete value
Insert or Delete?
Press 1 to insert press 2 to delete.
Press 0 to quit
```

Fig 7 - Operations perform-able on a linked list

Binary Trees

The simulation extended to binary trees, illustrating the insertion and deletion of nodes. The visual representation showcased the hierarchical structure of binary trees, emphasizing how nodes are added or removed while maintaining the tree's integrity.

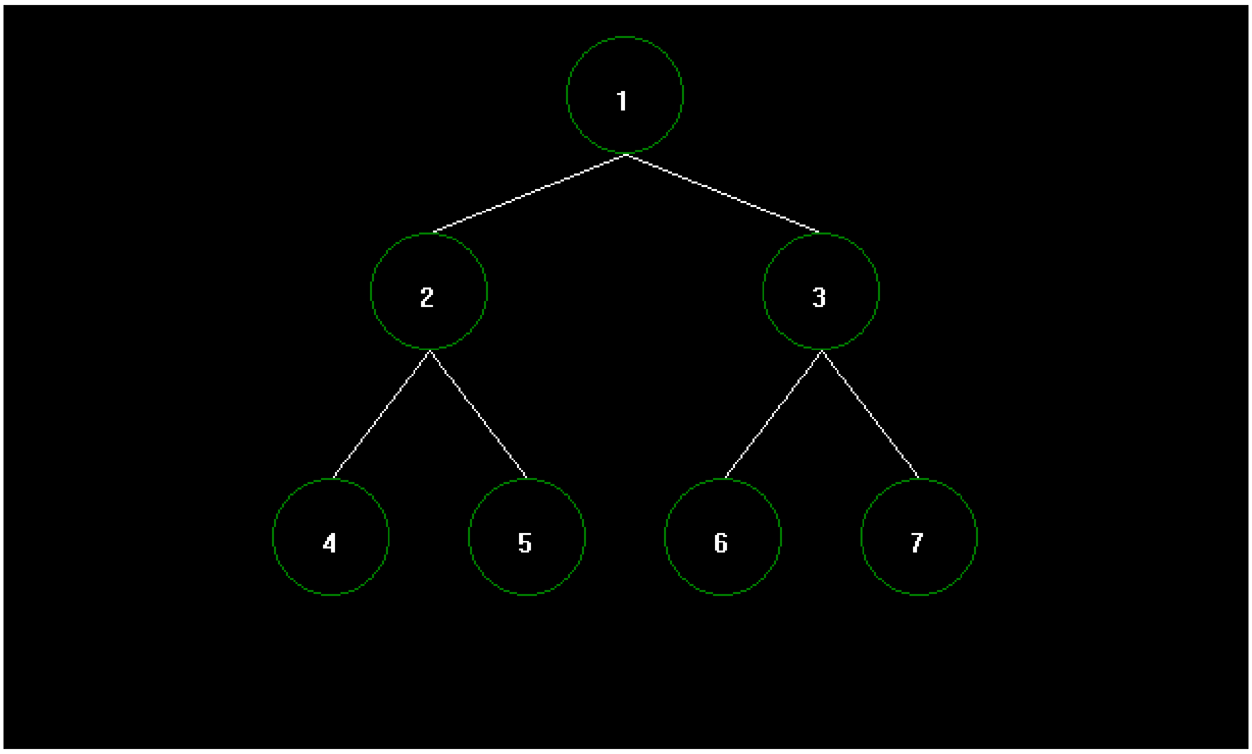
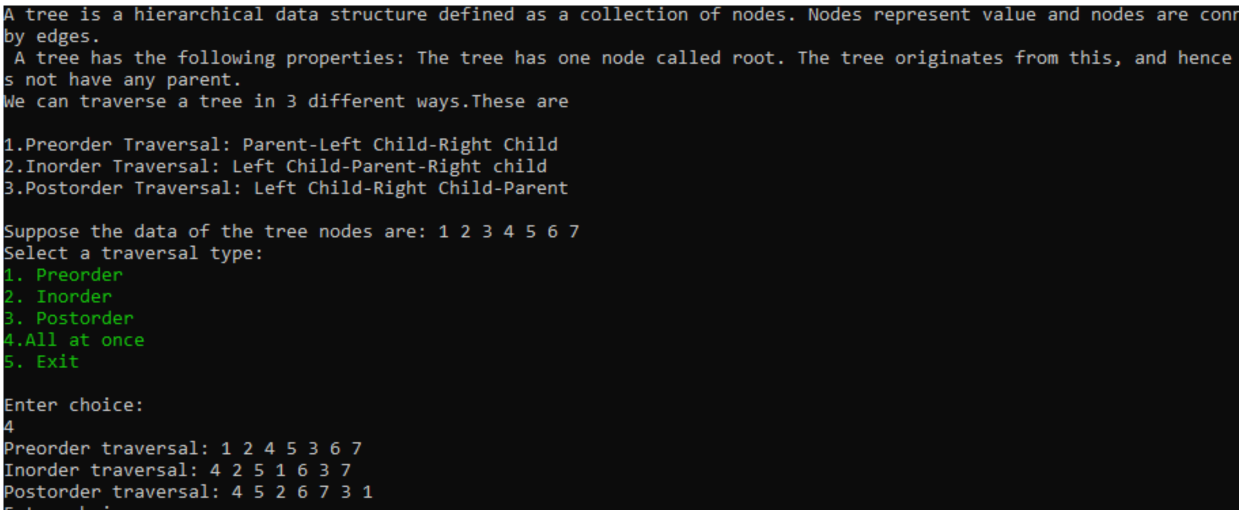


Fig 8 - Hierarchical structure of a tree data structure



```
A tree is a hierarchical data structure defined as a collection of nodes. Nodes represent value and nodes are conn
by edges.
 A tree has the following properties: The tree has one node called root. The tree originates from this, and hence
s not have any parent.
We can traverse a tree in 3 different ways.These are

1.Preorder Traversal: Parent-Left Child-Right Child
2.Inorder Traversal: Left Child-Parent-Right child
3.Postorder Traversal: Left Child-Right Child-Parent

Suppose the data of the tree nodes are: 1 2 3 4 5 6 7
Select a traversal type:
1. Preorder
2. Inorder
3. Postorder
4.All at once
5. Exit

Enter choice:
4
Preorder traversal: 1 2 4 5 3 6 7
Inorder traversal: 4 2 5 1 6 3 7
Postorder traversal: 4 5 2 6 7 3 1
```

Fig 9 - Preorder,in-order and postorder traversal of the tree nodes

Graphs

Graph simulation introduced users to the world of vertices and edges. The project allowed users to input data for graph nodes, establish connections, and witness the dynamic creation of complex structures. The adjacency matrix or list representation was visually represented, aiding users in comprehending the relationships between graph entities.

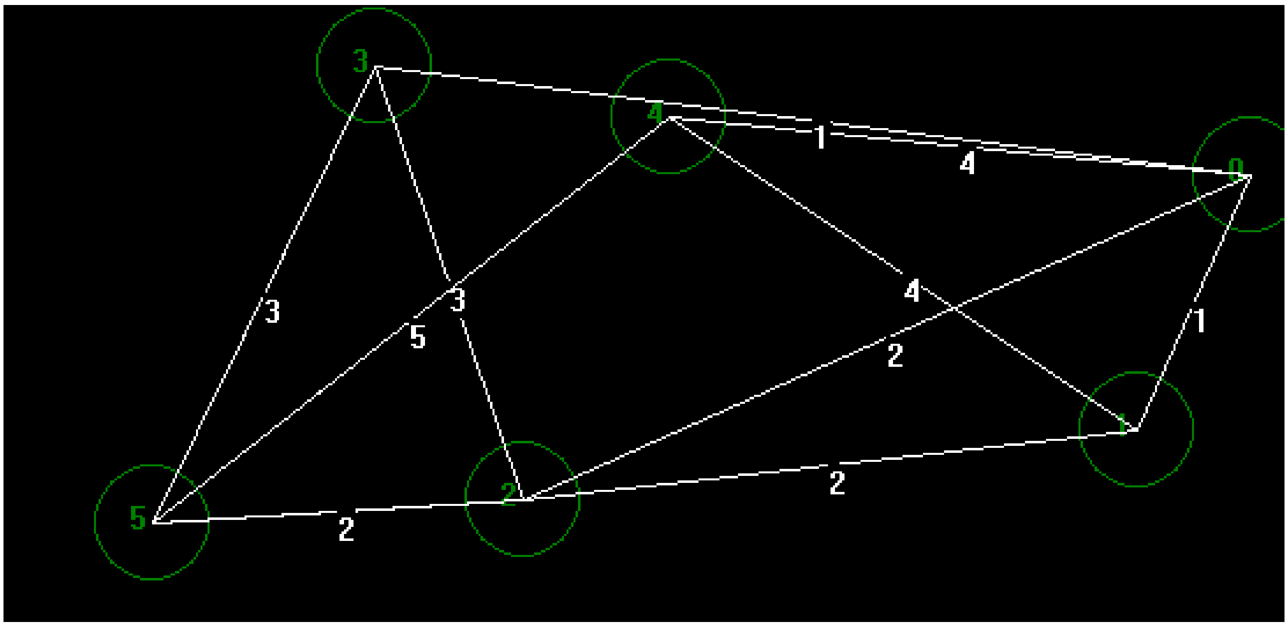


Fig 10 - Visualization of a graph

4.3.2 Sorting Algorithm Simulation

Bubble Sort

Bubble Sort, a simple yet instructive sorting algorithm, was broken down into individual iterations. Users could observe the comparison and swapping of elements in real-time, reinforcing the algorithm's logic.

Merge Sort

The complexity of Merge Sort was simplified through stepwise visualization. Dividing the array into halves and merging them back together at each step facilitated a clear understanding of the divide-and-conquer strategy.

Quick Sort

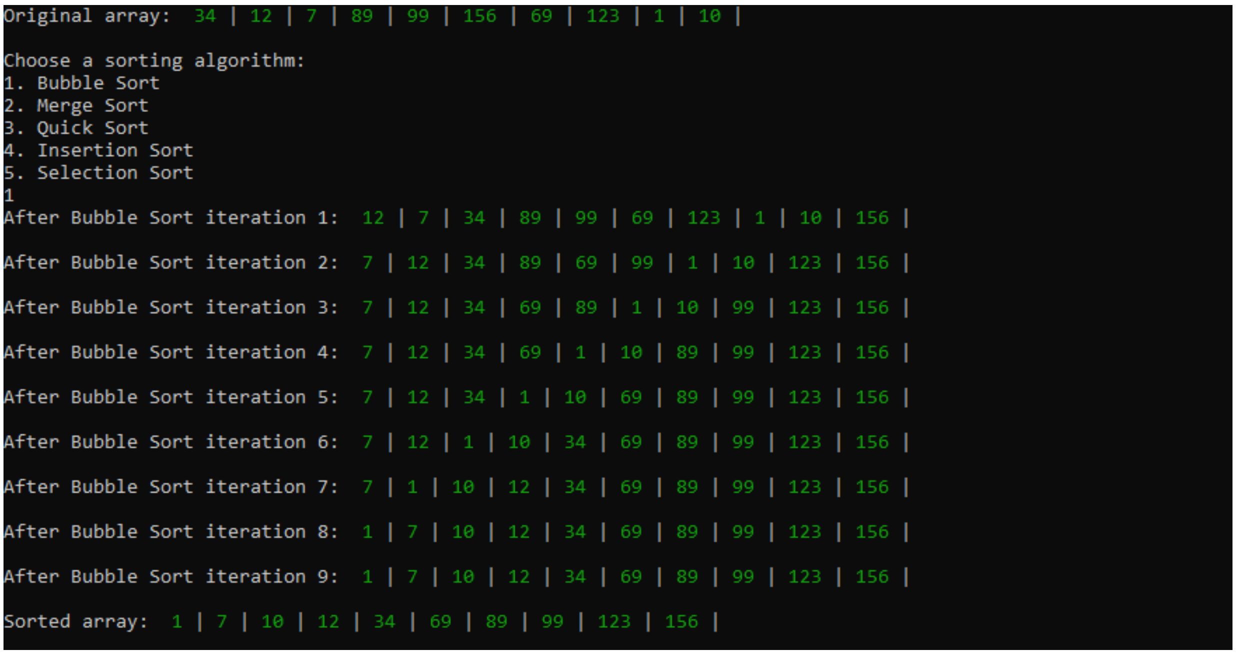
Quick Sort, known for its efficiency, was implemented to showcase the partitioning and recursive sorting steps. Users could witness the algorithm's agility in sorting elements by selecting appropriate pivots.

Insertion Sort

Insertion Sort's simplicity was highlighted through visual representation. Each iteration demonstrated how elements are selectively inserted into their correct positions, gradually achieving a sorted array.

Selection Sort

Selection Sort's stepwise simulation provided insights into the algorithm's operation. Users observed how the smallest element is repeatedly selected and placed at the beginning of the array.



```
Original array:  34 | 12 | 7 | 89 | 99 | 156 | 69 | 123 | 1 | 10 |

Choose a sorting algorithm:
1. Bubble Sort
2. Merge Sort
3. Quick Sort
4. Insertion Sort
5. Selection Sort
1
After Bubble Sort iteration 1:  12 | 7 | 34 | 89 | 99 | 69 | 123 | 1 | 10 | 156 |

After Bubble Sort iteration 2:  7 | 12 | 34 | 89 | 69 | 99 | 1 | 10 | 123 | 156 |

After Bubble Sort iteration 3:  7 | 12 | 34 | 69 | 89 | 1 | 10 | 99 | 123 | 156 |

After Bubble Sort iteration 4:  7 | 12 | 34 | 69 | 1 | 10 | 89 | 99 | 123 | 156 |

After Bubble Sort iteration 5:  7 | 12 | 34 | 1 | 10 | 69 | 89 | 99 | 123 | 156 |

After Bubble Sort iteration 6:  7 | 12 | 1 | 10 | 34 | 69 | 89 | 99 | 123 | 156 |

After Bubble Sort iteration 7:  7 | 1 | 10 | 12 | 34 | 69 | 89 | 99 | 123 | 156 |

After Bubble Sort iteration 8:  1 | 7 | 10 | 12 | 34 | 69 | 89 | 99 | 123 | 156 |

After Bubble Sort iteration 9:  1 | 7 | 10 | 12 | 34 | 69 | 89 | 99 | 123 | 156 |

Sorted array:  1 | 7 | 10 | 12 | 34 | 69 | 89 | 99 | 123 | 156 |
```

Fig 11 - Bubble sort iteration results

4.3.3 User Interaction

The user interface facilitated seamless interaction. Users could input their data, choose the data structure or sorting algorithm of interest, and witness the step-by-step simulation. The interactive nature of the simulator aimed to enhance user engagement and understanding.

The iterative development approach ensured that each simulation aspect was refined for clarity and accuracy. User feedback during testing played a pivotal role in shaping the final simulation logic.

4.4 Testing

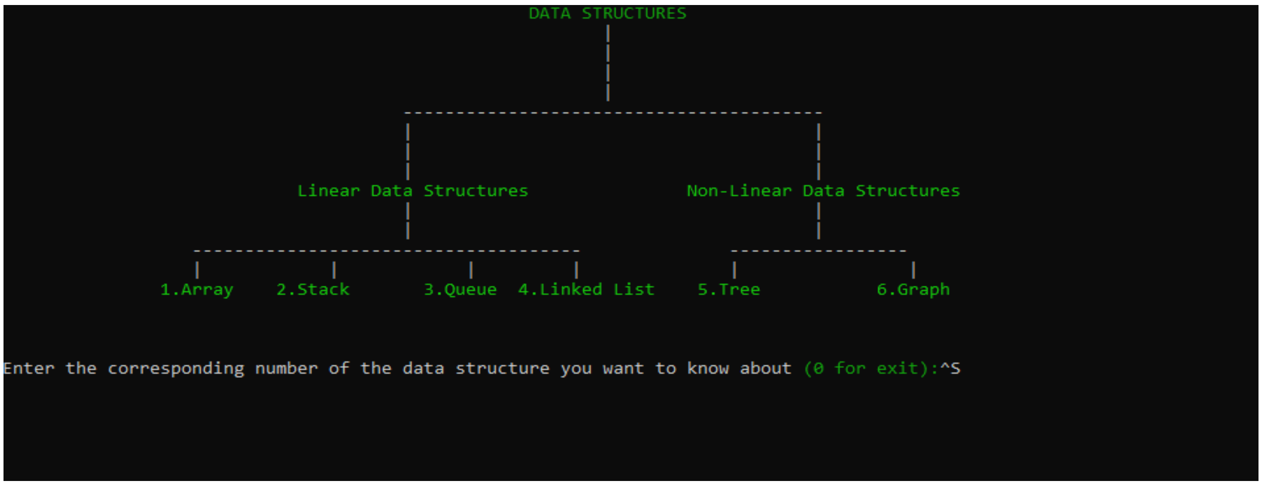
To ensure the reliability and functionality of the simulator, rigorous testing was performed. Unit tests were conducted for individual components, and system tests were carried out to evaluate the integration of different features.

5. User Interface

The user interface is a critical component of the Data Structure Simulator, aimed at providing a seamless and enjoyable learning experience. The interface includes:

-Navigation Panel: Users can easily switch between different data structures and sorting algorithms.

-Visualization Window: The main area where data structures and algorithms are displayed visually.



```
                                                  DATA STRUCTURES
                                                         |
                                                         |
                                                         |
                                                         |
                                      ----------------------------------------
                                      |                                      |
                                      |                                      |
                                      |                                      |
                            Linear Data Structures               Non-Linear Data Structures
                                      |                                      |
                                      |                                      |
                  --------------------------------------             ------------------
                  |            |            |         |              |                |
               1.Array    2.Stack       3.Queue  4.Linked List    5.Tree           6.Graph


Enter the corresponding number of the data structure you want to know about (0 for exit):^S
```

Fig 12 - User Interface

6. Challenges Faced

Developing the Data Structure Simulator posed several challenges:

- Graphics Library Compatibility: Integrating the graphics.h library with modern compilers presented compatibility issues that required careful consideration.

- Algorithm Complexity: Implementing step-by-step visualizations for sorting algorithms demanded meticulous planning to ensure accuracy and clarity.

- User Interaction: Striking a balance between simplicity and functionality in the user interface required multiple iterations to meet diverse user needs.

7. Conclusion

In conclusion, the Data Structure Simulator project has been a significant endeavor in combining theoretical knowledge with practical implementation, fostering a deeper understanding of data structures, algorithms, and graphics programming. The journey involved studying diverse data structures such as arrays, linked lists, stacks, queues, trees, and graphs, along with exploring various sorting and searching algorithms. Additionally, it required mastering the nuances of graphics programming using the graphics.h library.

7.1 Lessons Learned

7.1.1 Technical Proficiency

The project demanded a comprehensive grasp of C++ programming, data structures, and algorithms. Mastering the intricacies of the graphics.h library was crucial for creating an interactive and visually appealing user interface. The implementation of sorting algorithms added another layer of complexity, enhancing algorithmic understanding.

7.1.2 Project Management

Executing a project of this scale emphasized the importance of effective project management. Breaking down tasks, setting milestones, and ensuring systematic progression were essential. The iterative development process facilitated continuous improvement and refinement.

7.1.3 Interdisciplinary Learning

The project served as a bridge between various domains – from data structures and algorithms to graphics programming. The interdisciplinary nature of the project not only enhanced technical skills but also encouraged lateral thinking and problem-solving.

7.2 Future Extensions

7.2.1 Additional Data Structures and Algorithms

The project can be extended by incorporating more data structures and algorithms, providing a broader learning experience. Including advanced data structures such as priority queue, circular queue, circular linked list, hash tables, AVL trees, or graph algorithms like Dijkstra's or Kruskal's algorithms would enrich the simulator.

7.2.2 Enhanced User Interface

Future developments could focus on refining the user interface to make it more intuitive and user-friendly. Implementing features such as real-time code visualization, algorithm step annotations, or interactive quizzes could enhance user engagement.

7.2.3 Performance Metrics

Integrating performance metrics for algorithms, such as time complexity and space complexity analysis, would add an educational dimension. Users could analyze and compare the efficiency of different algorithms within the simulator.

7.2.4 Cloud-based Collaboration

Enabling cloud-based collaboration features could transform the simulator into a collaborative learning tool. Users could share simulations, discuss algorithms, and engage in collaborative problem-solving.

7.2.4 Exercise and Quiz options

As this is a learning tool it would have a separate module for self-tests and exercise where the program would be the judge. There would be a good number of questions sets that the user has to answer to improve their knowledge of the data structures and test their levels.

7.3 Final Reflection

Completing the Data Structure Simulator project has been an enriching experience. It has deepened the understanding of core computer science concepts, honed technical skills, and instilled confidence in tackling complex programming challenges. The project not only serves as a testament to the acquired knowledge but also lays the foundation for continuous exploration and innovation in the realm of data structures and algorithms.

References

[1] Herbert Schildt's "Teach Yourself C" 3rd Edition.

[2] Stanley B. Lippman's "C++ Primer".

[3] Ellis Horowitz and Sartaj Sahni's "Fundamentals of Data Structures".

[4] Rivest, Clifford Stein's "Introduction to Algorithms".

[5] https://www.geeksforgeeks.org/introduction-to-sorting-algorithm/

[6] https://www.programiz.com/dsa