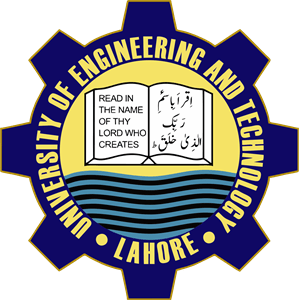
Final Project Report

**Data Structures and Algorithms Visualizer**



**Session 2023-2027**

**Submitted By:**

Ume Kalsoom 2023-CS-182

Aleena Zafar 2023-CS-183

Areeba Shahid 2023-CS-186

**Submitted To:**

Prof. Nazeef ul Haq

**CSC-200 Data Structure and Algorithms**

**University of Engineering and Technology Lahore Pakistan**

Contents

1.1 Project Overview 5

1.1.1 Summary……………………………………………………………………………………...5

1.1.2 Motivation…………………………………………………………………………………….5

1.1.3 Objectives……………………………………………………………………………………..6

1.1.4 Target Audience………………………………………………………………………………6

1.1.5 Operational Details…………………………………………………………………………...6

**1.2 Use Cases…………………………………………………………………………………………….7**

1.2.1 Visualize Algorithms………………………………………………………………………….7

1.2.2 Visualize Data Structures……………………………………………………………………..7

1.2.3 Input Custom Data…………………………………………………………………………….7

1.2.4 Generate Random Data………………………………………………………………………..8

1.2.5 Highlight Step-by-Step Execution…………………………………………………………….8

1.2.6 Provide Educational Insights………………………………………………………………….8

**1.3 Project Details……………………………………………………………………………………….8**

1.3.1 Sorting Algorithms……………………………………………………………………………8

* Bubble Sort………………………………………………………………………………8
* Selection Sort…………………………………………………………………………….8
* Insertion Sort……………………………………………………………………………..8
* Merge Sort……………………………………………………………………………….9
* Quick Sort………………………………………………………………………………..9
* Counting Sort…………………………………………………………………………….9
* Radix Sort………………………………………………………………………………..9
* Bucket Sort………………………………………………………………………………9

1.3.2 Data Structures………………………………………………………………………………..9

* Stack……………………………………………………………………………………..9
* Queue……………………………………………………………………………………9
* Link List…………………………………………………………………………………9
* Deque……………………………………………………………………………………9
* Array…………………………………………………………………………………….9
* Hashing………………………………………………………………………………...10
* Tree…………………………………………………………………………………….10
* Graph…………………………………………………………………………………..10

1.3.3 Technology Stack…………………………………………………………………………….11

**1.4 Project Wireframes………………………………………………………………………………12**

**1.4.1 Sorting Algorithms………………………………………………………………………..12**

* Main Page……………………………………………………………………………12
* Input Page……………………………………………………………………………12
* Bubble Sort…………………………………………………………………………..13
* Insertion Sort………………………………………………………………………...13
* Selection Sort………………………………………………………………………..14
* Merge Sort…………………………………………………………………………..14
* Quick Sort…………………………………………………………………………..15
* Radix Sort…………………………………………………………………………..15
* Counting Sort……………………………………………………………………….16
* Bucket Sort…………………………………………………………………………16

**1.4.2 Data Structures……………………………………………………………………………17**

* Stack………………………………………………………………………………….17
* Queue………………………………………………………………………………...17
* Link List……………………………………………………………………………..18
* Deque………………………………………………………………………………..18
* Tree………………………………………………………………………………….19
* Binary Search Tree………………………………………………………….19
* AVL Tree………..………………………………………………………….19
* Red-Black Tree…..…………………………………………………………20
* Graph…………………………………………………………………………………20
* Graph Transversal through BFS and DFS………………………………….20
* Hashing………………………………………………………………………………21
* Linear Hashing………………………………………………………………21
* Quadratic Hashing…………………………………………………………..21
* Double Hashing……………………………………………………………..22
* Array…………………………………………………………………………………22

**1.5 Challenges Faced…………………………………………………………………………………23**

**1.6 Future Improvements……………………………………………………………………………23**

**1.7 Conclusion………………………………………………………………………………………...23**

List of Figures

1. Main Page 12

2. Input Page 12

[3. Bubble Sort 13](#_Toc185196284)

[4. Insertion Sort 13](#_Toc185196284)

[5. Selection Sort 14](#_Toc185196284)

[6. Merge Sort 14](#_Toc185196284)

[7. Quick Sort 15](#_Toc185196284)

[8. Radix Sort 15](#_Toc185196284)

[9. Counting Sort 16](#_Toc185196284)

[10. Bucket Sort !6](#_Toc185196284)

11. Stack 17

[12. Queue 17](#_Toc185196284)

[13. Link List 18](#_Toc185196284)

[14. Dequeue 18](#_Toc185196284)

[15. Binary Search Tree(BST) !9](#_Toc185196284)

[16. AVL Tree 19](#_Toc185196284)

[17. Red-Black Tree 19](#_Toc185196284)

[18. Graph Transversal through BFS and DFS 20](#_Toc185196284)

19. Linear Hashing 21

[20. Quadratic Hashing 21](#_Toc185196284)

[21. Double Hashing 22](#_Toc185196284)

[22. Array 22](#_Toc185196284)

## **1 Project Overview**

* + 1. **Summary**

In the final year of an undergraduate program, students are required to undertake a comprehensive project that integrates knowledge from across their degree, enabling them to apply theoretical concepts, practice technical skills, and develop project management expertise. The Data Structure and Algorithm Visualizer embodies this spirit by serving as an interactive tool designed to provide a comprehensive and intuitive understanding of fundamental data structures and algorithms through visual representation. The visualizer supports eight core algorithms, allowing users to input their own data or generate random data for demonstrations. This project bridges the gap between theoretical knowledge and practical understanding by illustrating how data structures operate and algorithms execute in real time. It includes a user-friendly interface for exploring concepts like sorting, searching, graph traversal, and more, fostering deeper comprehension and engagement.

* + 1. **Motivation**

Understanding data structures and algorithms is crucial for programming and problem-solving, yet these concepts often appear abstract and challenging for many learners. Traditional learning methods, such as textbooks or static diagrams, lack the dynamic representation necessary to grasp the internal workings of these processes. This visualizer draws inspiration from **VisuAlgo**, a widely recognized tool for algorithm visualization, and seeks to build upon its foundation by offering enhanced interactivity, expanded functionality, and a more personalized user experience. This visualizer also aims to address these challenges by providing interactive and animated explanations. By allowing users to observe step-by-step execution, modify inputs, and experiment with algorithms, the project motivates learners to explore and internalize complex concepts more effectively.

* + 1. **Objectives**

The primary objectives of this project are:

* **Interactive Learning**: Provide a platform for users to learn data structures and algorithms through real-time visualizations.
* **Algorithm Coverage**: Support at least eight essential algorithms, including sorting (e.g., Quick Sort, Merge Sort), searching (e.g., Binary Search), and graph algorithms (e.g., BFS, DFS).
* **Data Customization**: Allow users to input their own data or generate random data for simulations, ensuring flexibility in experimentation.
* **Real-Time Feedback**: Demonstrate algorithm execution step-by-step, highlighting key operations and decisions made during the process.
* **Educational Clarity**: Use color coding, annotations, and indices to clarify actions like comparisons, swaps, and traversals.
* **Engagement and Accessibility**: Create an intuitive interface that caters to learners at all levels, from beginners to advanced students.
  + 1. **Target Audience**

The Data Structure and Algorithm Visualizer is designed for students, educators, job aspirants, developers, and programming enthusiasts who seek an interactive tool to learn, teach, and experiment with data structures and algorithms. It caters to all levels, from beginners to professionals, offering engaging and intuitive visualizations for enhanced understanding.

* + 1. **Operational Details**

The visualizer will function through a combination of animated graphics and interactive controls. Key operational features include:

* **Algorithm Selection**: Users can choose from a list of supported algorithms through a dropdown menu or button interface.
* **Data Input**: Users can enter custom data values via an input field or request random data generation.
* **Visualization Execution**: Animations will demonstrate the algorithm’s process step-by-step, such as comparisons in sorting or path selection in graph traversal.
* **Control Features**: Buttons for starting, pausing, restarting, and adjusting the speed of animations ensure a tailored learning experience.
  1. **Use Cases**

Following are the use cases implemented in the Data Structure and Algorithm Visualizer system:

* + 1. **Visualize Algorithms**

The system allows users to visualize eight core algorithms, such as sorting algorithms (e.g., Quick Sort, Merge Sort, Bubble Sort), searching algorithms (e.g., Binary Search), and graph traversal algorithms (e.g., BFS, DFS). Users can select an algorithm, and the system will display an animated execution step-by-step, highlighting each operation such as comparisons, swaps, or path selections.

* + 1. **Visualize Data Structures**

The system enables users to explore the internal operations of data structures such as stacks, queues, linked lists, Hashing, binary search trees (BST), and graphs. Users can perform operations like push, pop, enqueue, dequeue, traversal, or insertion and see these actions represented visually.

* + 1. **Input Custom Data**

Users can input their own data sets for the algorithms and data structure operations, ensuring they can explore specific scenarios or test cases. This feature allows personalization and deeper understanding.

* + 1. **Generate Random Data**

Users who do not wish to input custom data can opt to generate random data sets. The system will provide appropriate data values, ensuring diversity in the scenarios presented.

* + 1. **Highlight Step-by-Step Execution**

The system breaks down each algorithm or data structure operation into clear, step-by-step visualizations. Each step is annotated with details such as current operations, decisions, and actions, ensuring clarity and comprehension.

* + 1. **Provide Educational Insights**

The system provides additional educational information, such as explanations of the algorithms, their use cases, and the significance of specific operations. It aims to enhance learning by providing theoretical context alongside practical demonstrations.

* 1. **Project Details**
     1. **Sorting Algorithms**

The project includes visualizations for eight common sorting algorithms, showcasing their step-by-step execution and key operations like comparisons and swaps:

1. **Bubble Sort:** A simple sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order.
2. **Selection Sort:** Finds the smallest element from the unsorted part and swaps it with the first unsorted element.
3. **Insertion Sort:** Builds the sorted array one element at a time by placing each element in its correct position.
4. **Merge Sort:** A divide-and-conquer algorithm that divides the array into halves, sorts each half, and then merges them back.
5. **Quick Sort:** Another divide-and-conquer algorithm that selects a pivot element and partitions the array around it.
6. **Counting Sort:** A non-comparison-based sorting algorithm that works by counting the frequency of each element.
7. **Radix Sort:** Sorts numbers by processing their digits one at a time, starting from the least significant to the most significant digit.
8. **Bucket Sort:** Divides the array into buckets and sorts each bucket individually before combining them.
   * 1. **Data Structures**

The visualizer supports the following data structures and provides animated representations of their operations:

1. **Stack:**  
   Implements the LIFO principle, where elements are added and removed from the top. Operations visualized: push, pop, and peek.
2. **Queue:**  
   Implements the FIFO principle, where elements are added at the rear and removed from the front. Operations visualized: enqueue, dequeue, and peek.
3. **Linked List:**  
   Displays singly linked list operations such as insertion, deletion, and searching, with clear visualizations of node references.
4. **Deque (Double-Ended Queue):**

Demonstrates insertion and deletion of elements from both ends (front and rear).

1. **Array:**

The visualizer demonstrates array operations like insertion, deletion, and linear search. It shows how elements are shifted for insertion or deletion and highlights each step of sequentially checking elements during a linear search.

1. **Hashing Function:**

The visualizer includes animations for understanding and implementing hashing techniques, focusing on efficient collision resolution strategies. It supports:

* **Linear Hashing:**

Demonstrates how collisions are resolved by sequentially searching for the next available slot in the hash table. Users can observe the process of inserting keys, handling collisions, and rehashing as the load factor increases.

* **Quadratic Hashing**

Visualizes the resolution of collisions by searching for the next available slot using a progressively increasing gap between attempts, reducing clustering compared to linear probing. The process is illustrated step-by-step to enhance understanding.

* **Double Hashing**

Showcases a more advanced collision resolution technique where a second hash function is used to compute the step size for resolving collisions. Users can input their custom hash functions and observe how the combination ensures better distribution and fewer collisions.

1. **Tree:**  
   Visualizes hierarchical structures, supporting operations like insertion, deletion, traversal, and searching. Includes:

* **Binary Search Tree (BST):** Nodes follow left < root < right property.
* **AVL Tree:** Self-balancing tree with height differences of at most one.
* **Red-Black Tree:** Self-balancing binary search tree with additional balancing rules.

1. **Graph:**  
   Graph visualizations include the following functionalities:

* **Breadth-First Search (BFS):** Explores the graph level by level.
* **Depth-First Search (DFS):** Explores as far along a branch as possible before backtracking.
* **Prim's Algorithm:** A minimum spanning tree algorithm that builds a tree from the graph by adding the shortest edge at each step.
* **Kruskal's Algorithm:** Another minimum spanning tree algorithm that sorts edges by weight and adds the smallest edge without forming a cycle.
* **Dijkstra's Algorithm:** Finds the shortest path from a source node to all other nodes in a graph with non-negative edge weights.
  + 1. **Technology Stack**

The project is implemented using:

* **Python:** For logic implementation and handling data structure and algorithm processes.
* **PyGame:** To create the graphical visualizations and animations for data structures and algorithms on a desktop platform.
  1. **Project Wireframes**
     1. **Sorting Algorithms**

1. **Main Page**

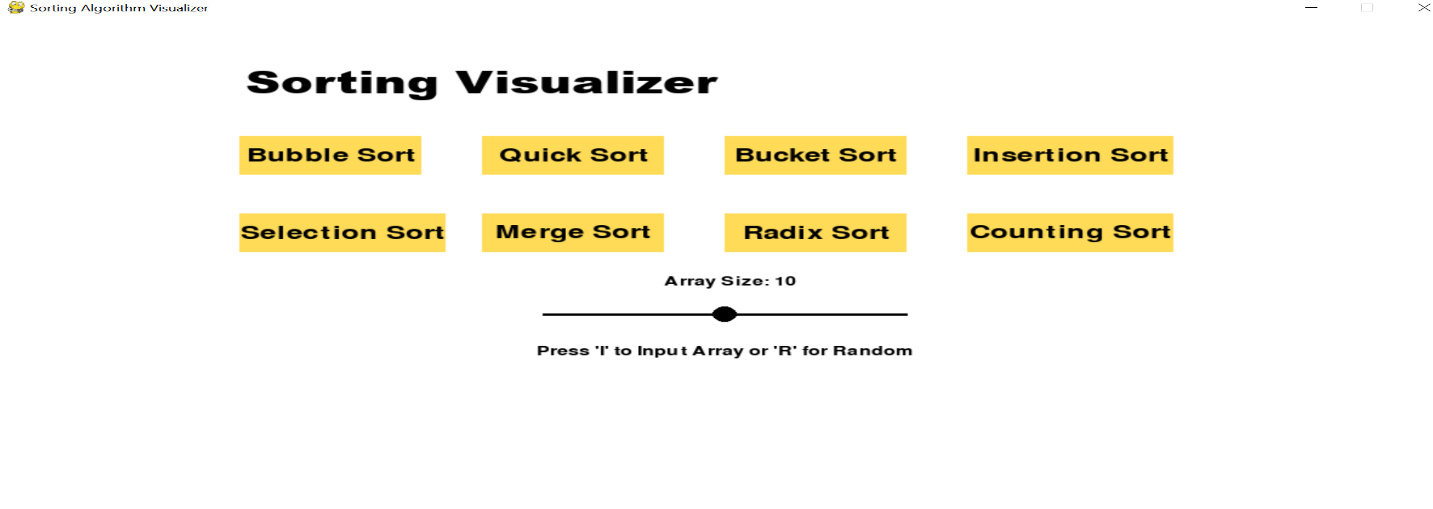
****

Fig:1 Main Sorting Page

1. **Input Page**

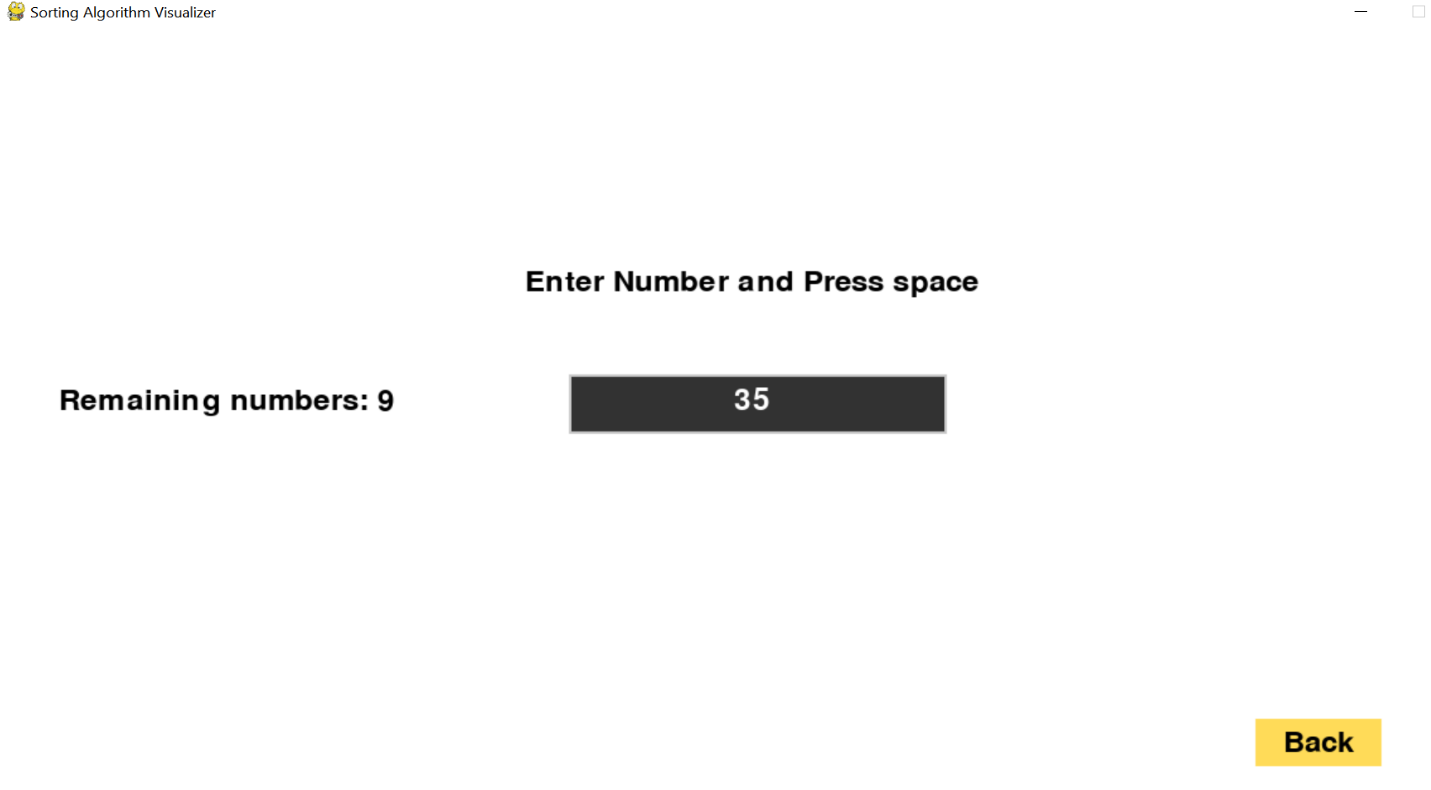
****

Fig:2 User Input Data Page

1. **Bubble Sort**

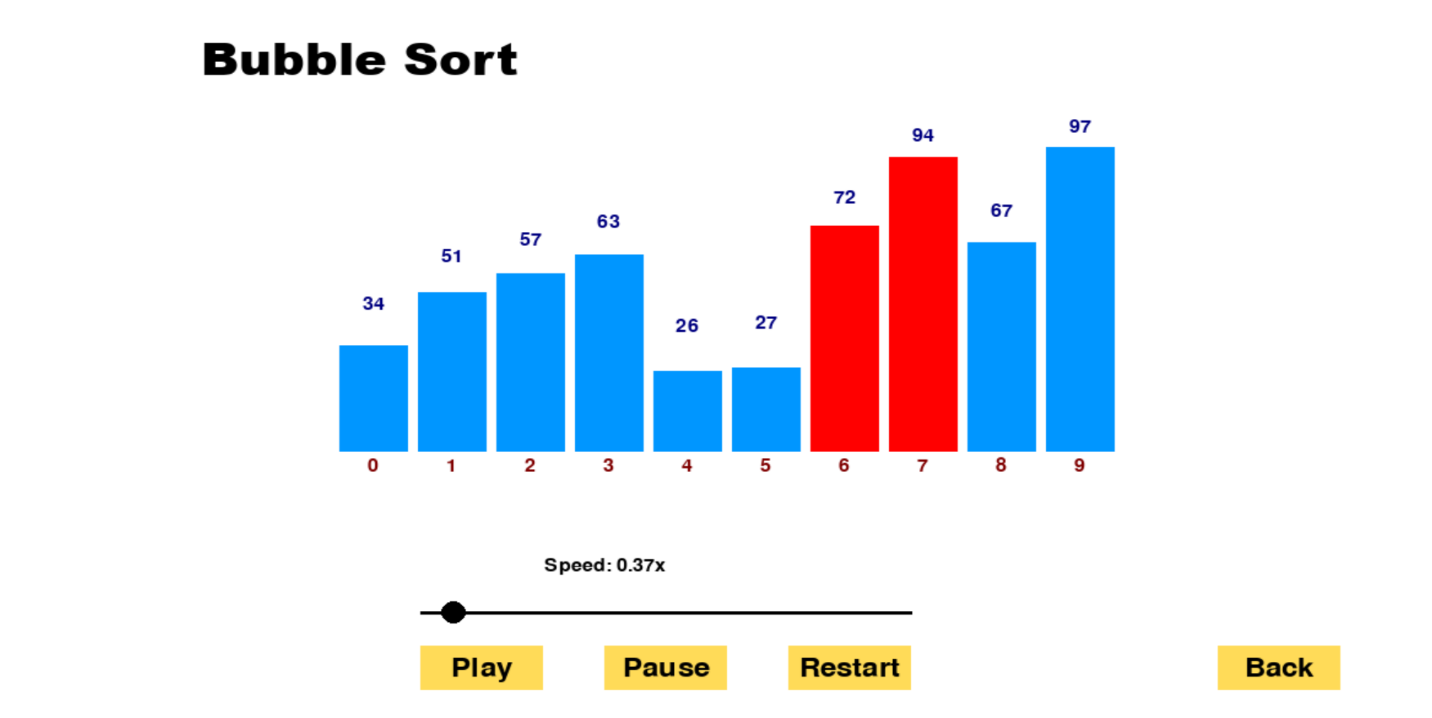
****

Fig:3 Bubble Sort Page

1. **Insertion Sort**

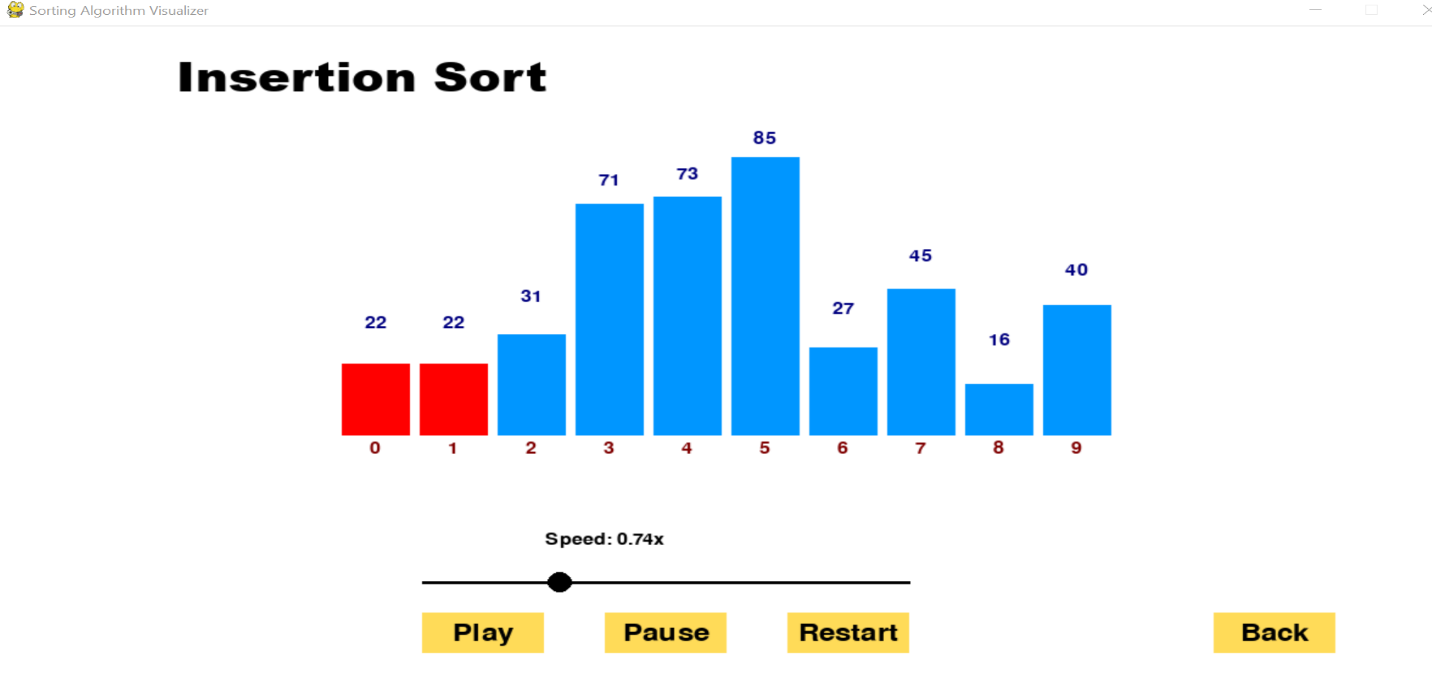
****

Fig:4 Insertion Sort Page

1. **Selection Sort**

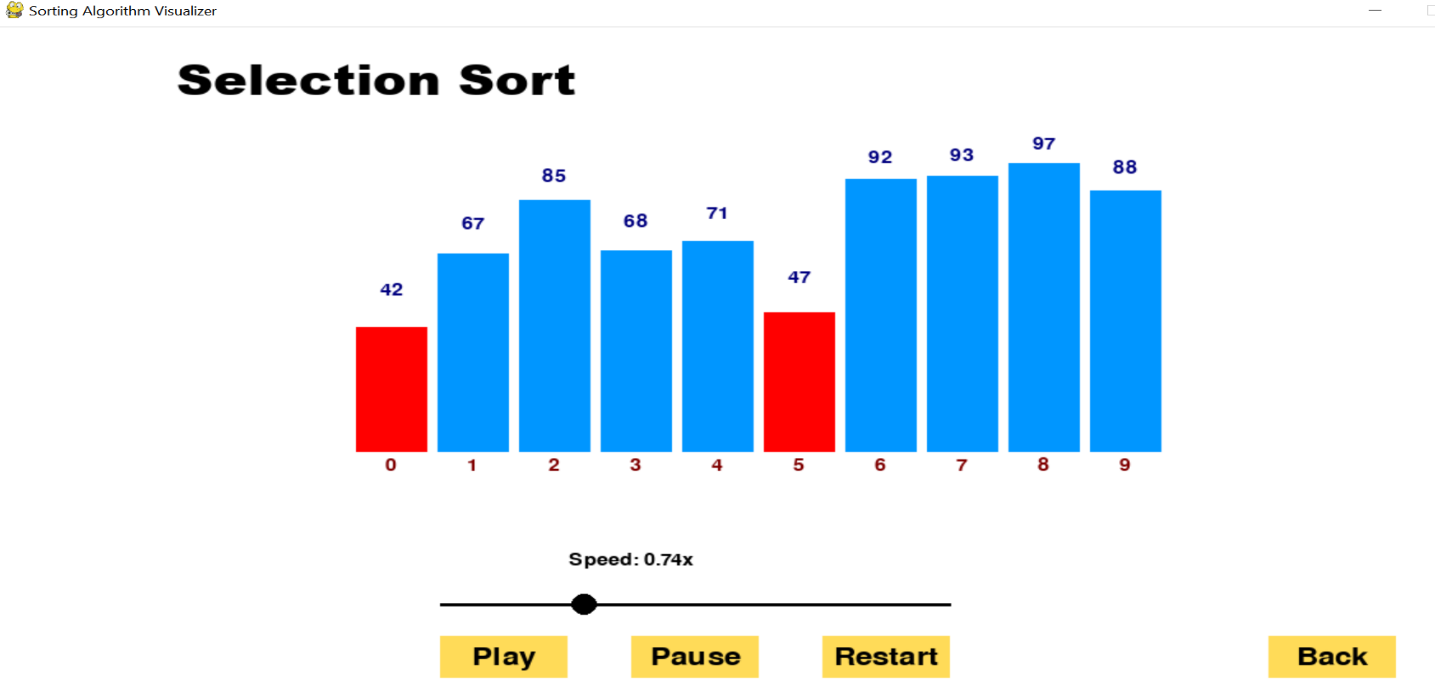
****

Fig:5 Selection Sort Page

1. **Merge Sort**

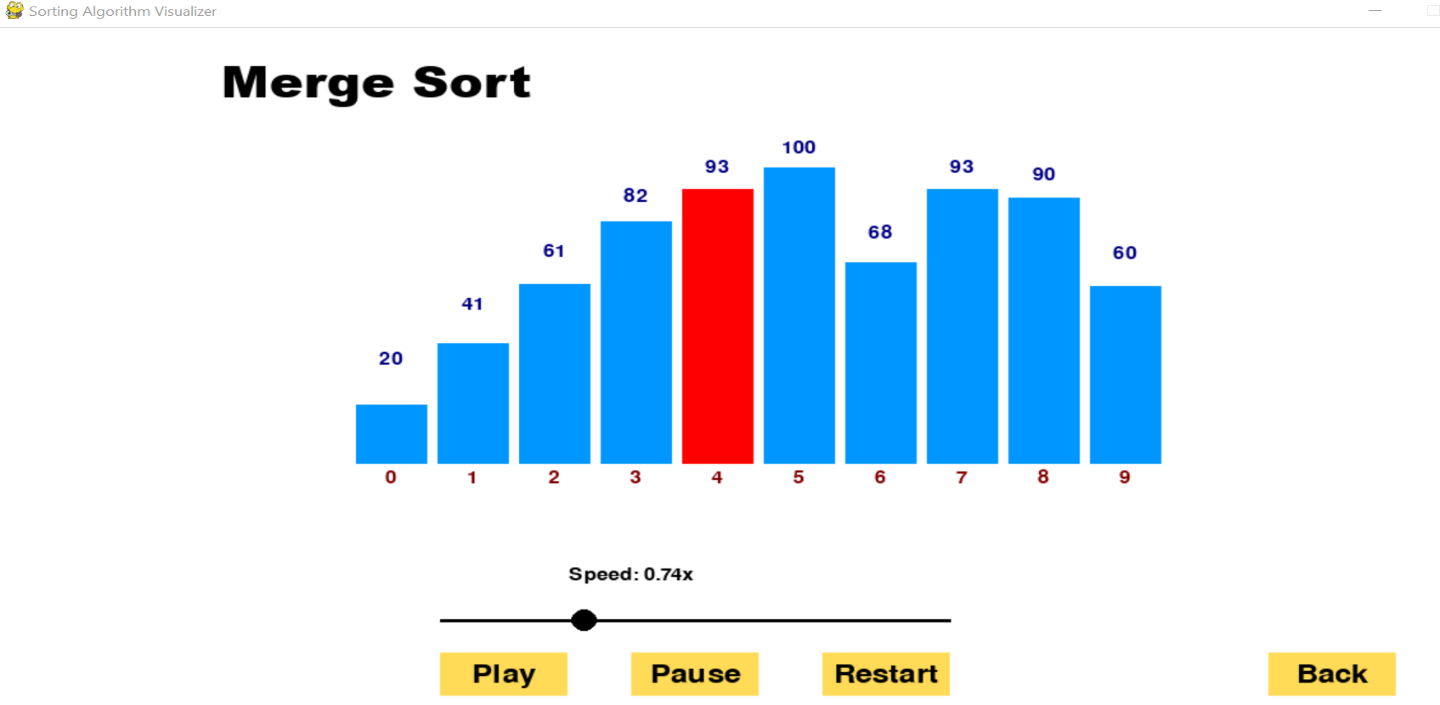
****

Fig:6 Merge Sort Page

1. **Quick Sort**

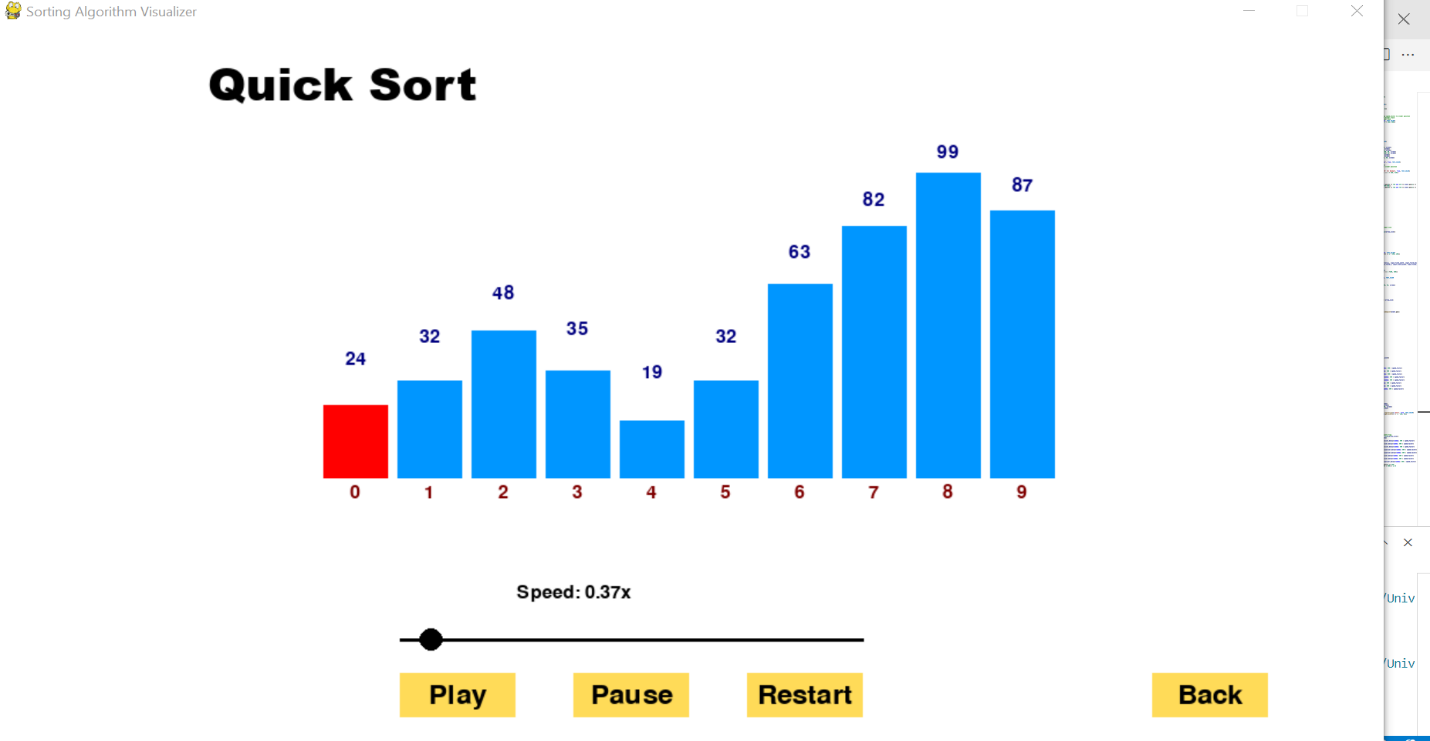
****

Fig:7 Quick Sort Page

1. **Radix Sort**

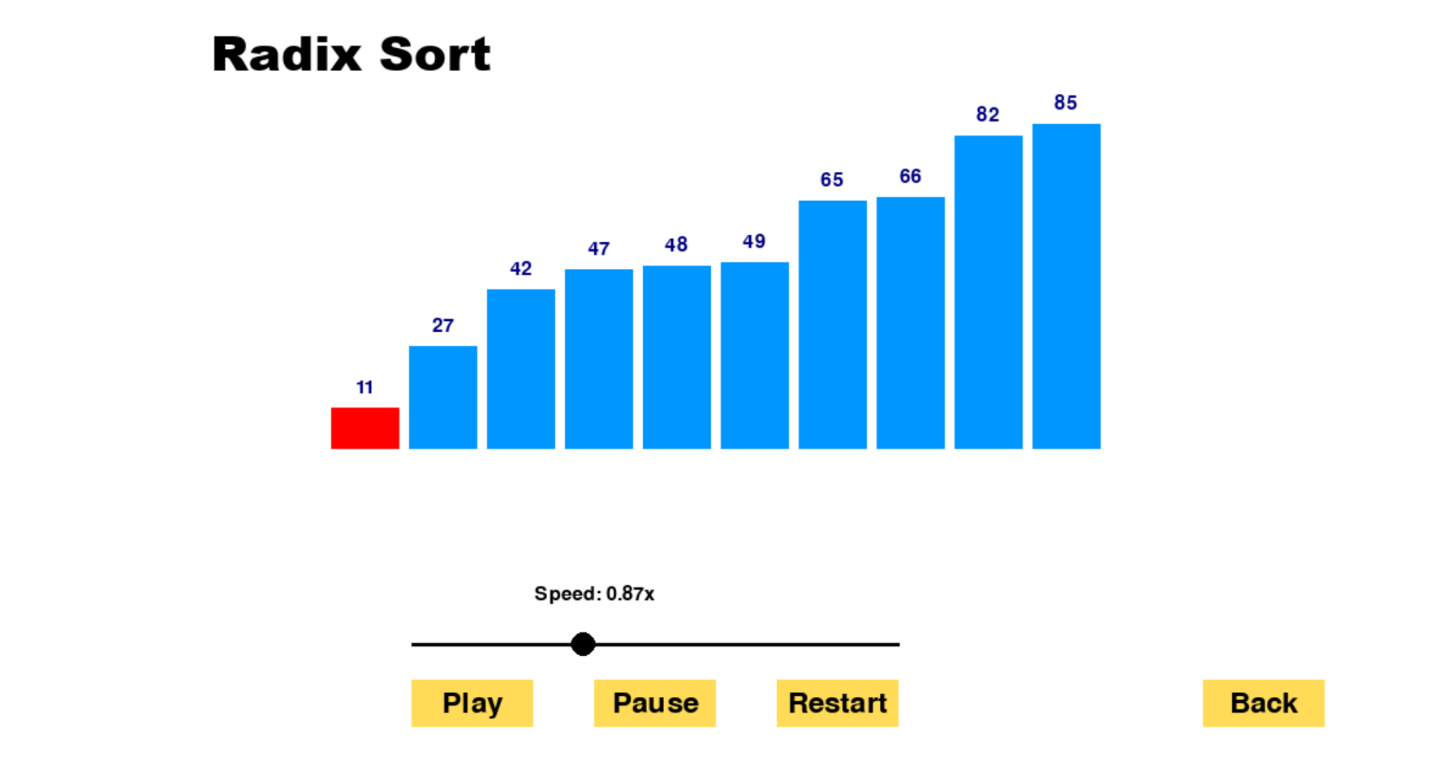
****

Fig:8 Radix Sort Page

1. **Counting Sort**

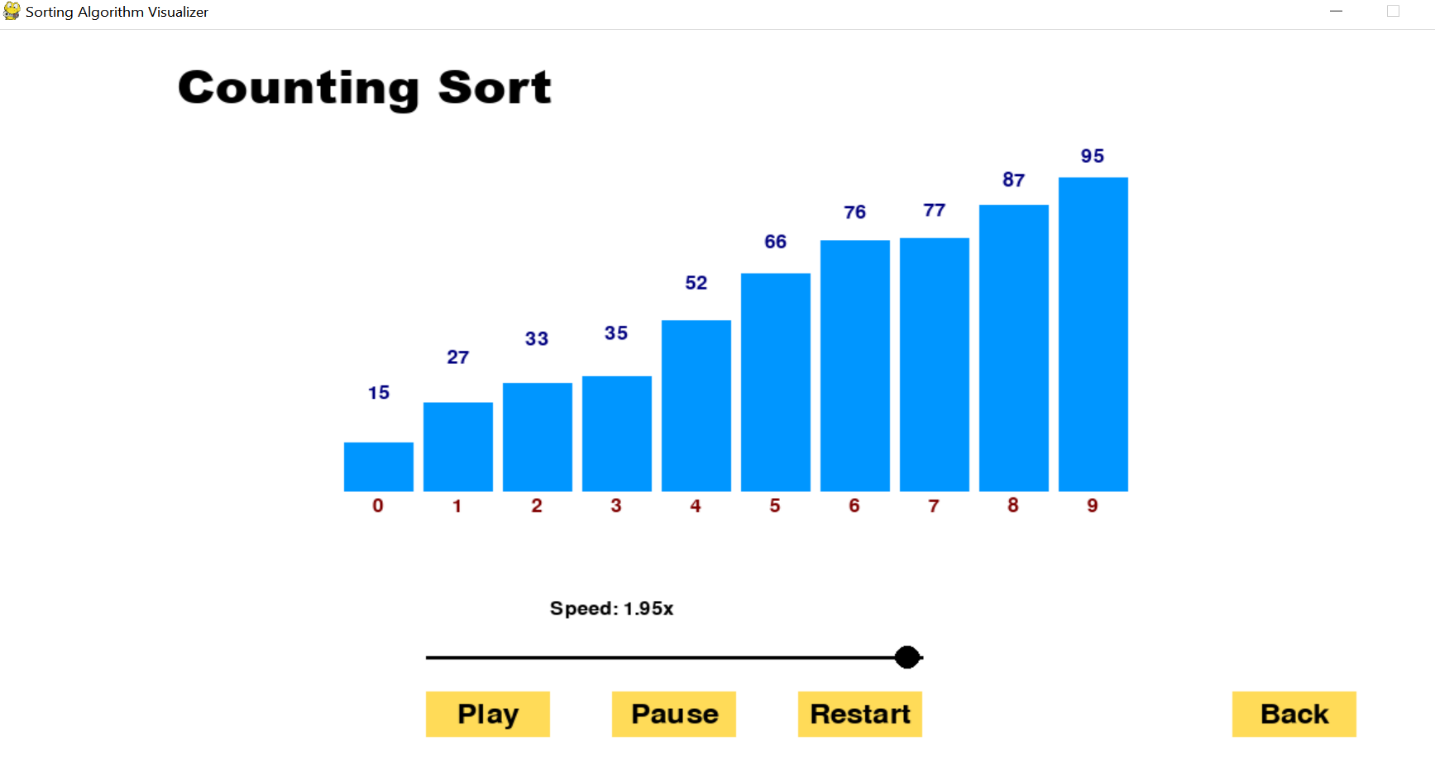
****

Fig:9 Counting Sort Page

1. **Bucket Sort**

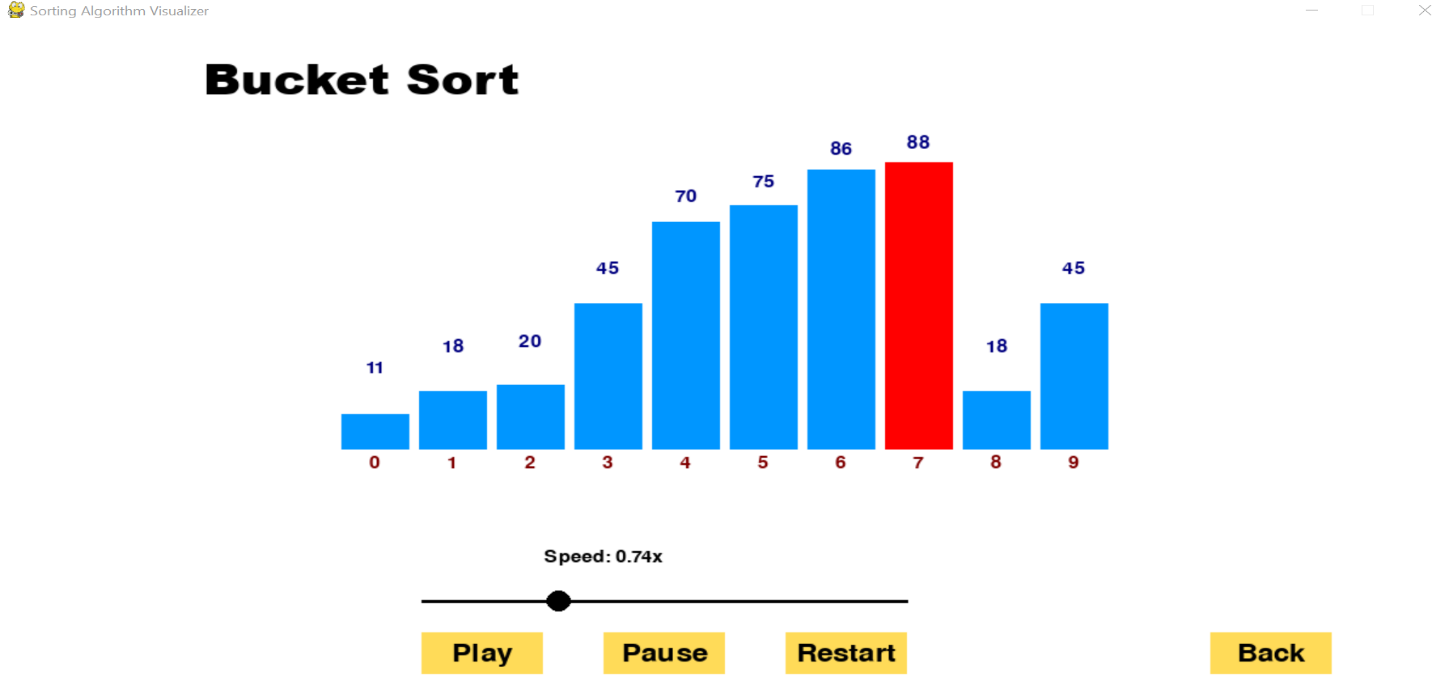
****

Fig:10 Bucket Sort Page

* + 1. **Data Structures**

1. **Stack**

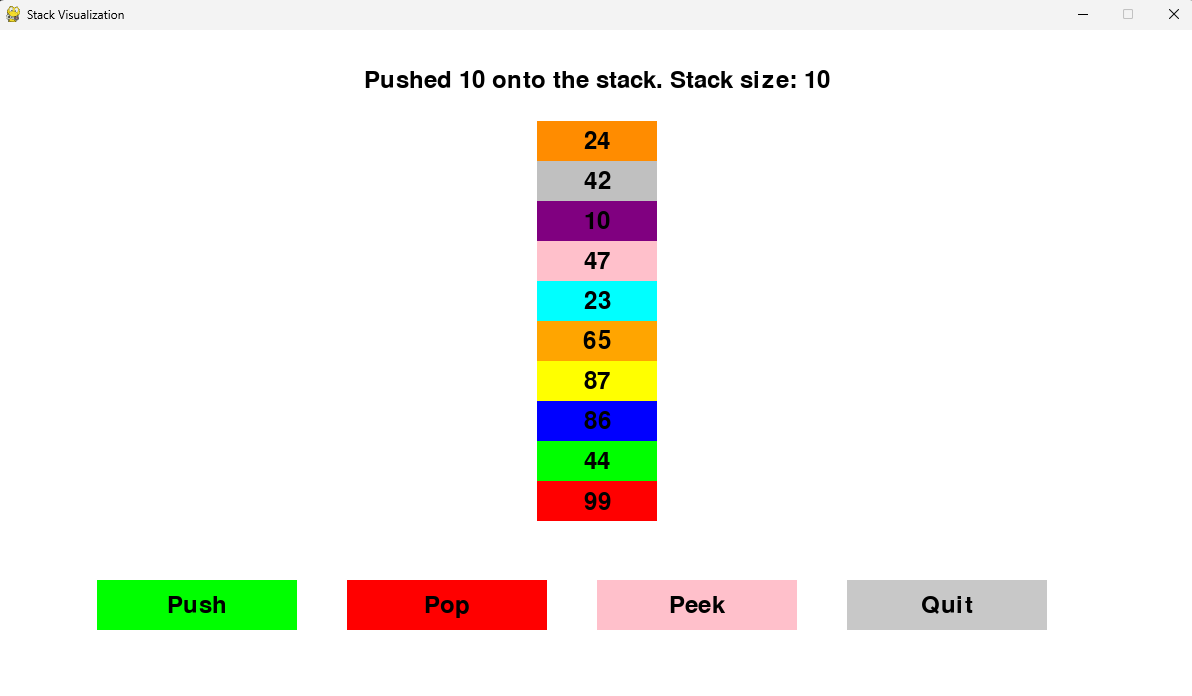
****

Fig:11 Stack

1. **Queue**

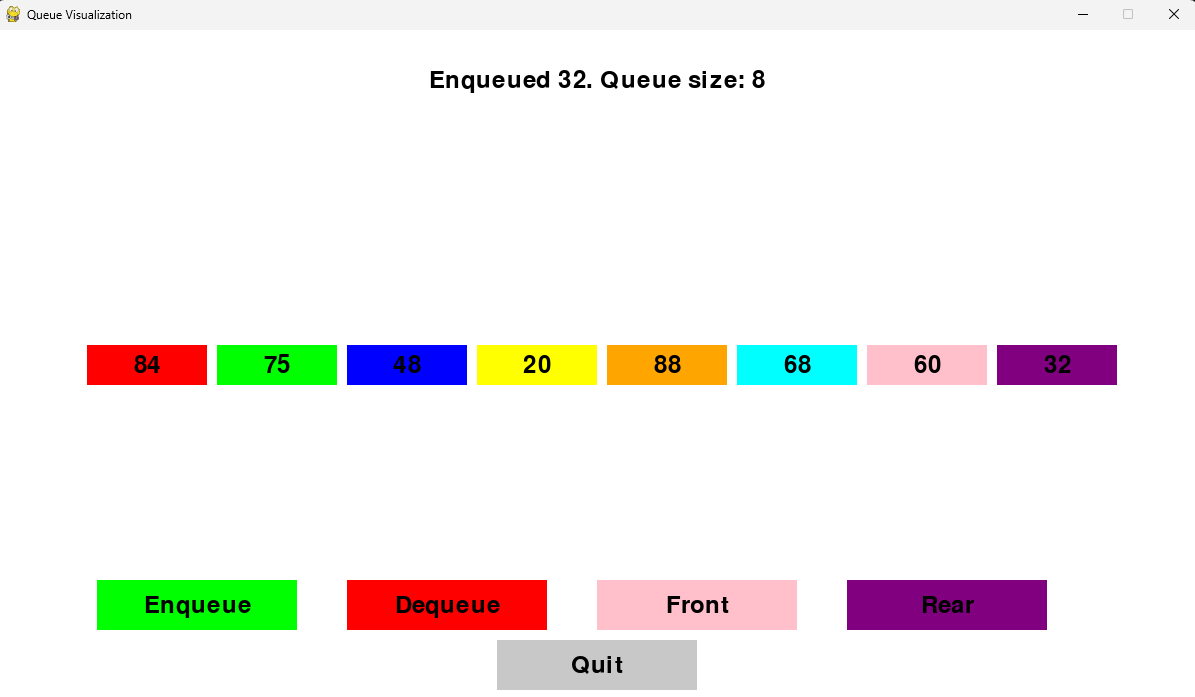
****

Fig:12 Queue

1. **Link List**

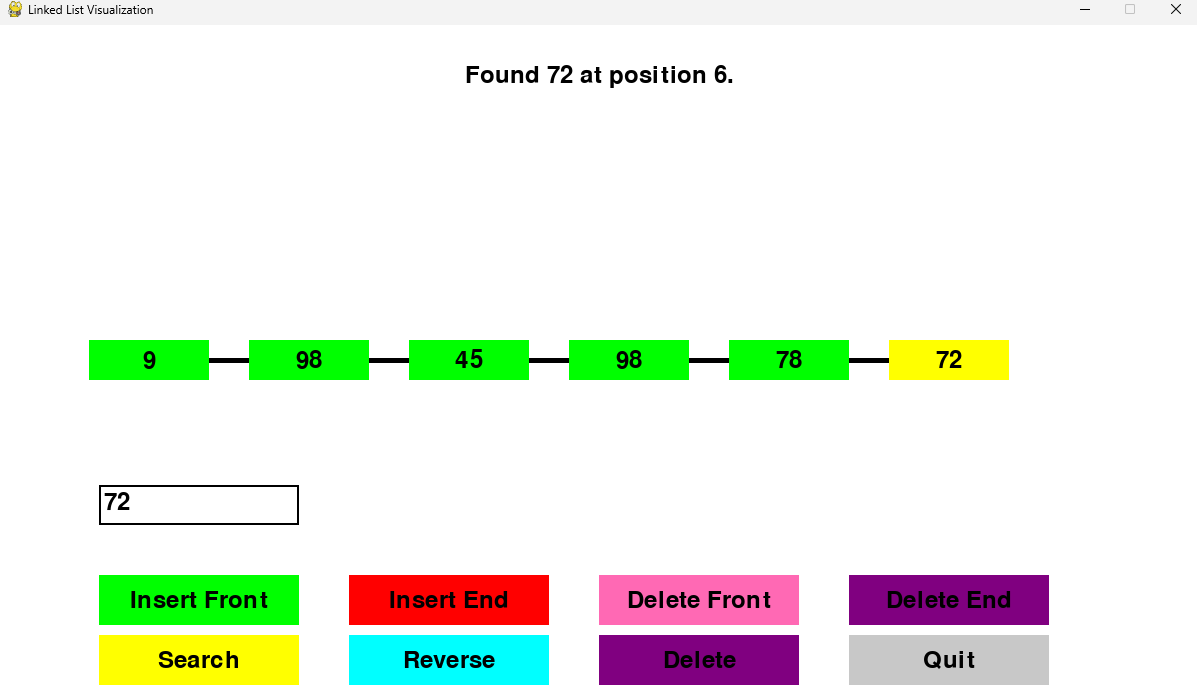
****

Fig:13 Link List

1. **Dequeue**

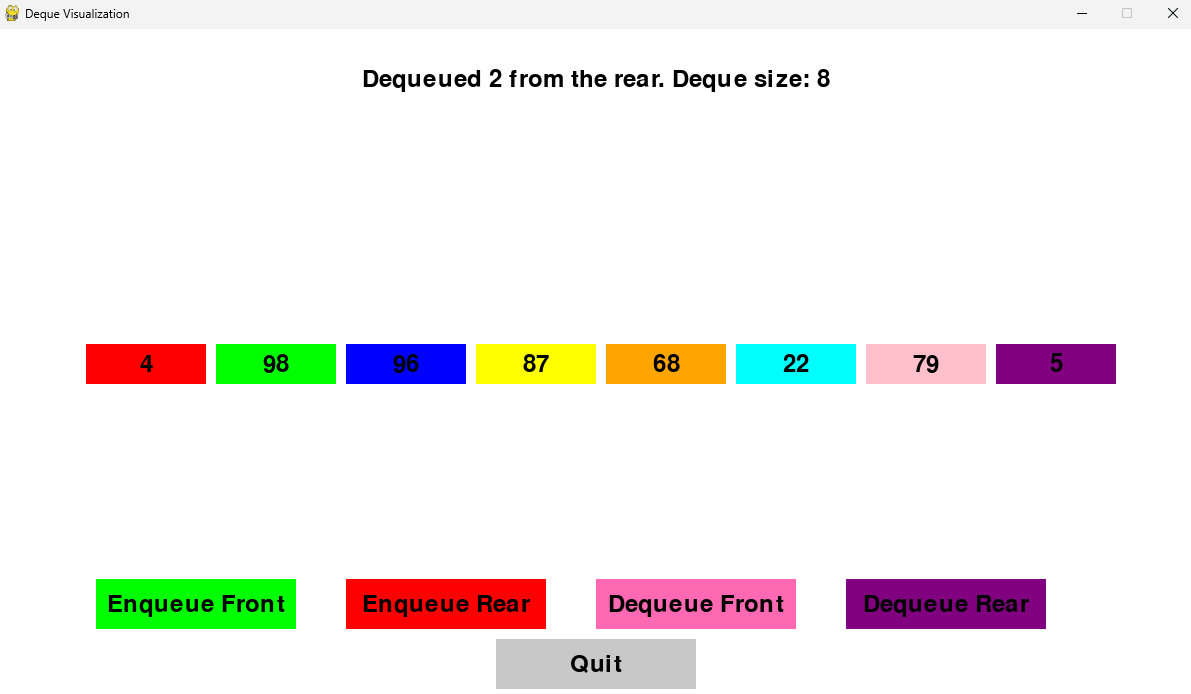
****

Fig:14 Deque

1. **Tree**

* **Binary Search Tree**

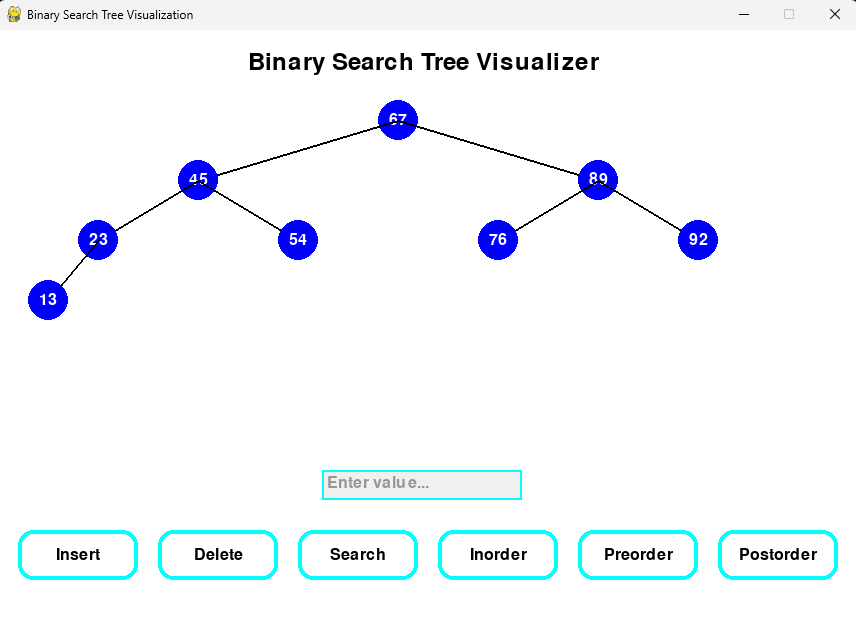
****

Fig:15 Binary Search Tree

* **AVL Tree**

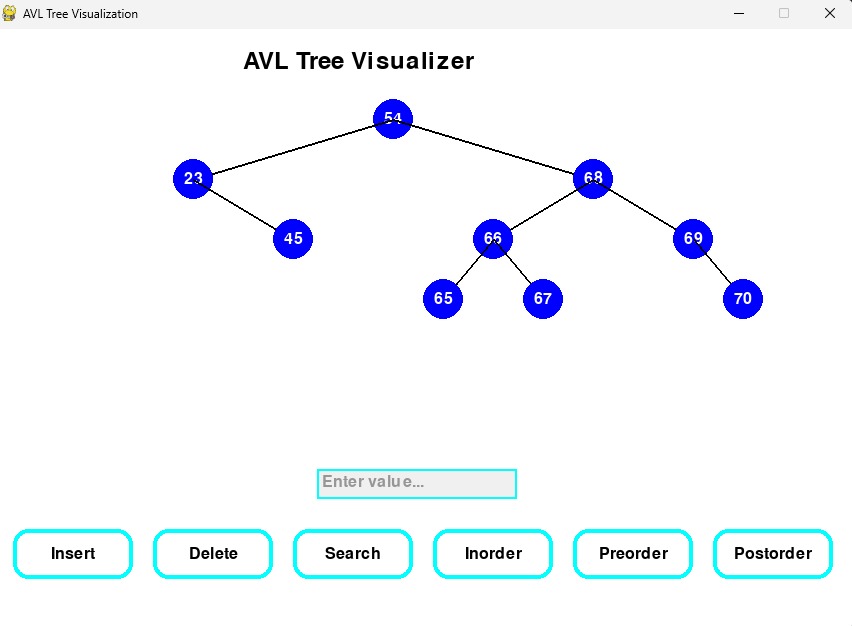
****

Fig:16 AVL Tree

* **Red-Black Tree**

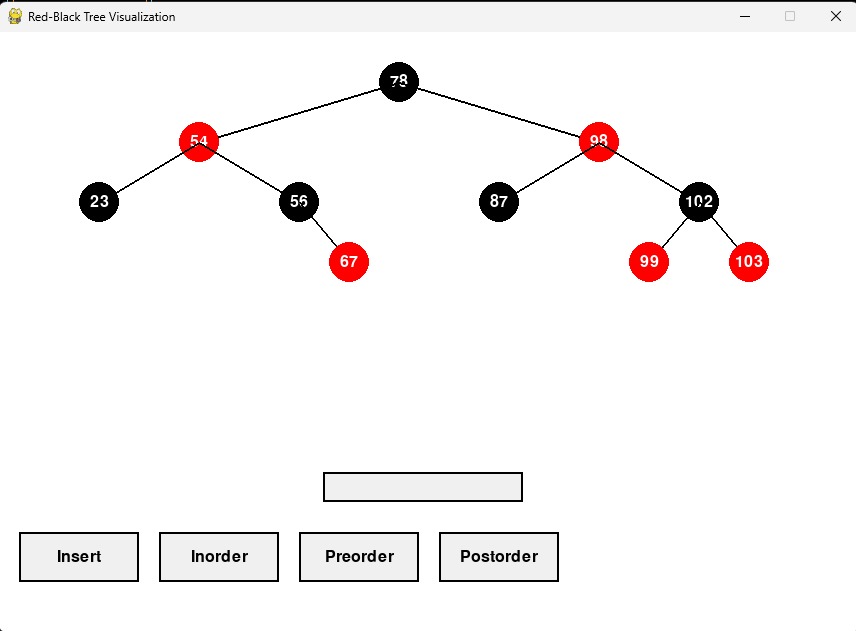
****

Fig:17 Red-Black Tree

1. **Graph**

* **BFS and DFS**

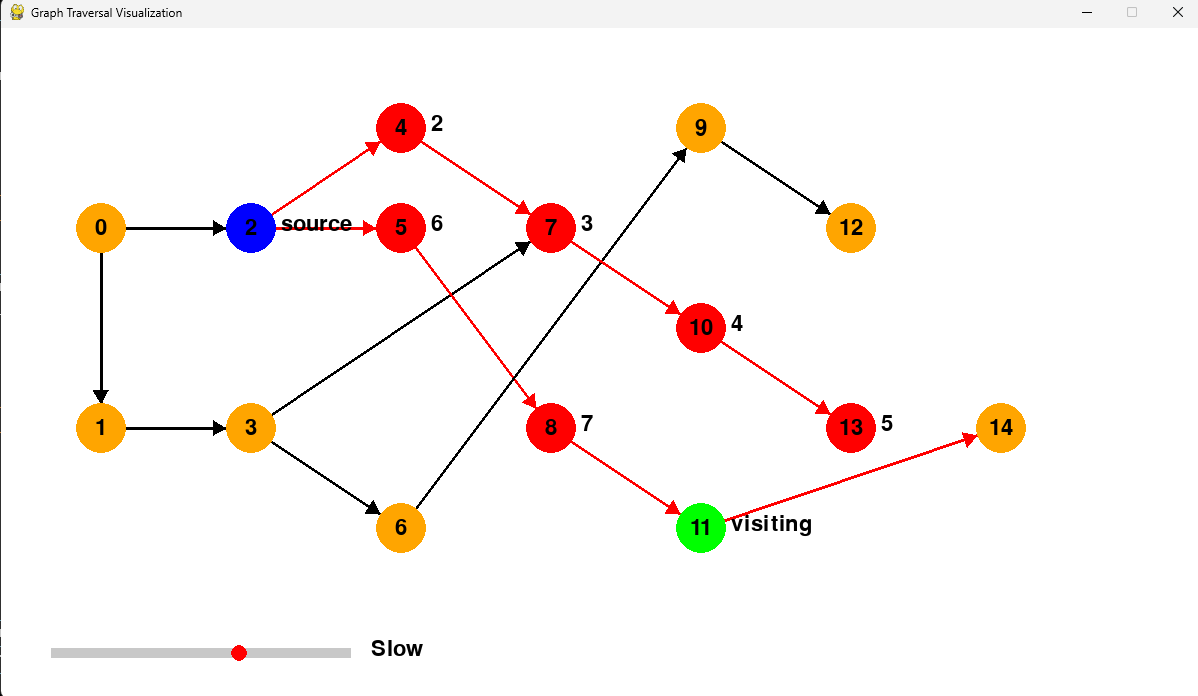
****

Fig:18 Graph Transversal using BFS and DFS

1. **Hashing**

* **Linear Hashing**

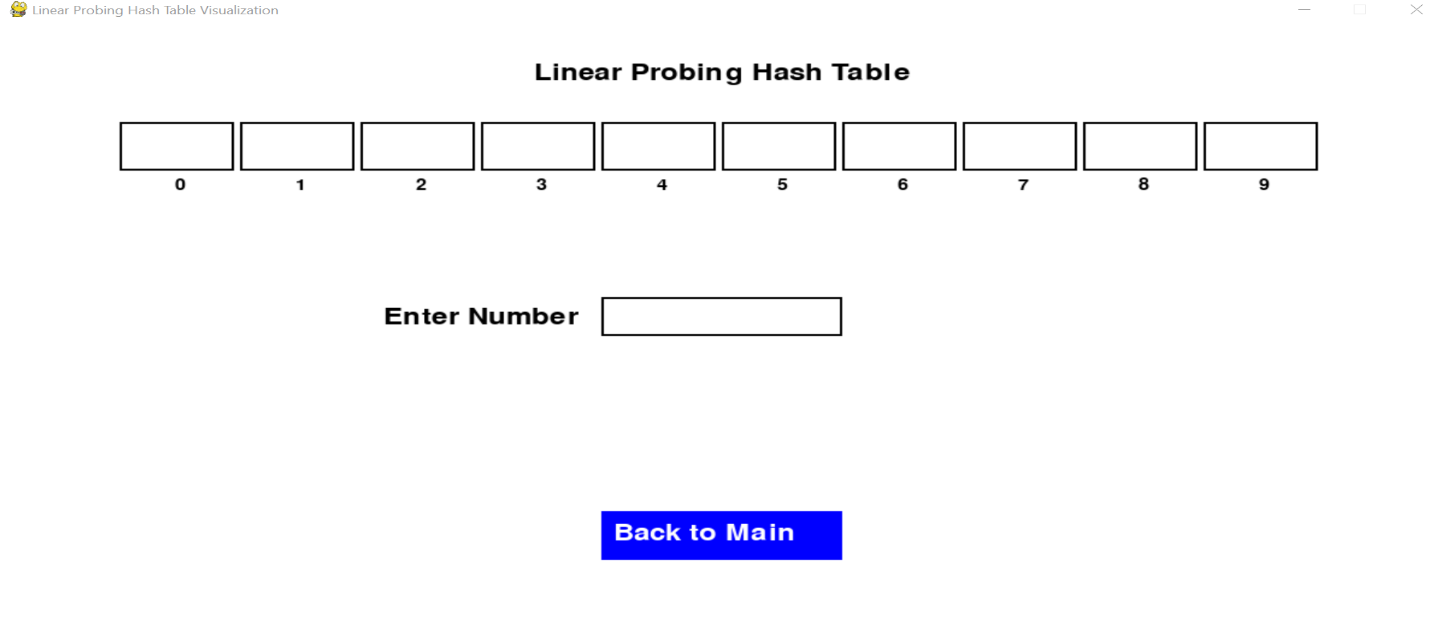
****

Fig:19 Linear Hashing

* **Quadratic Hashing**

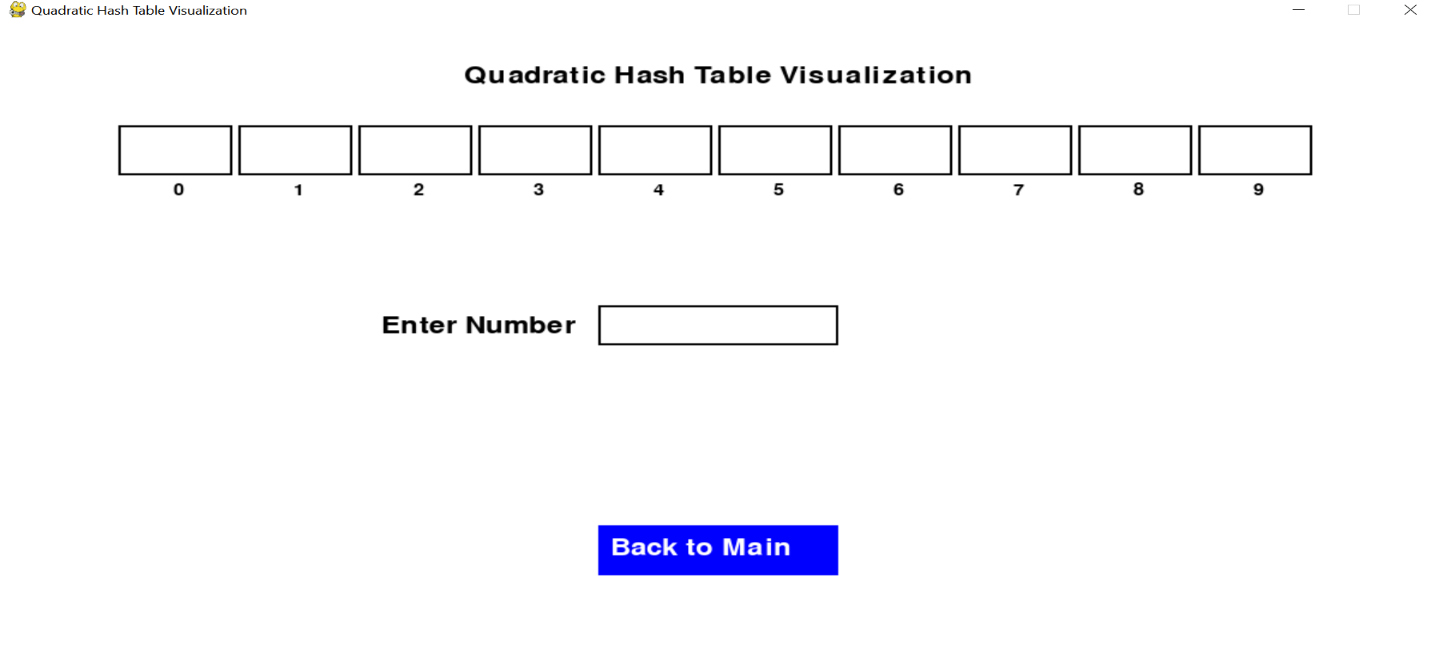
****

Fig:20 Quadratic Hashing

* **Double Hashing**

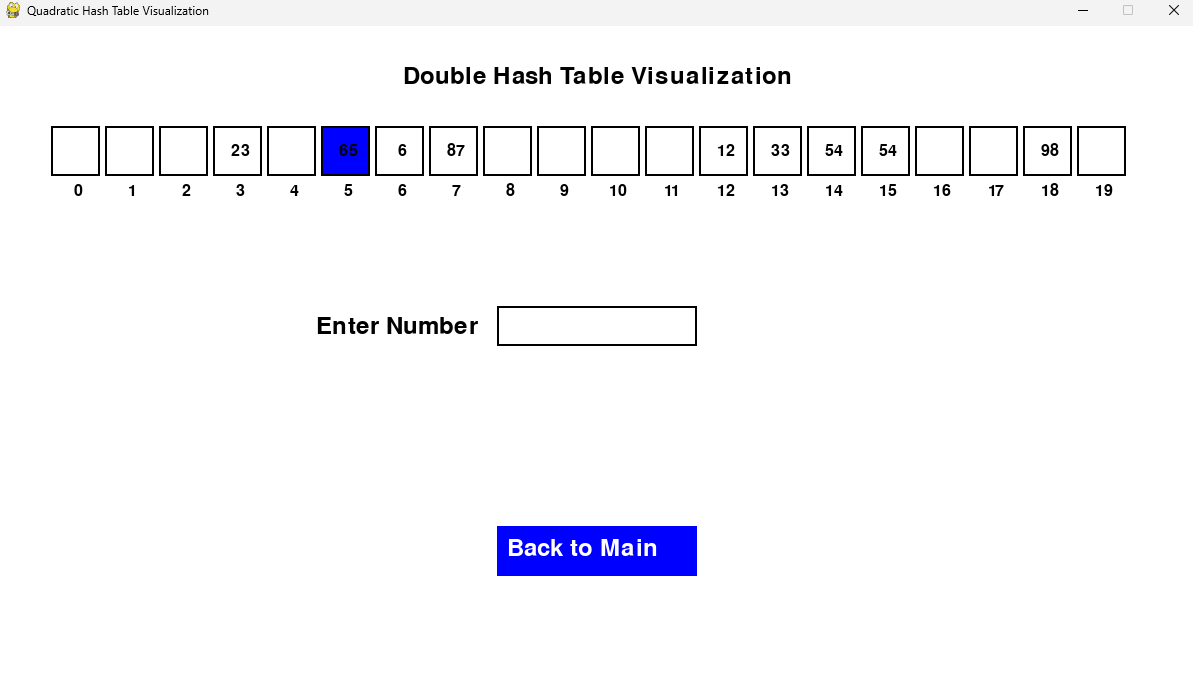
****

Fig:21 Double Hashing

1. **Array**

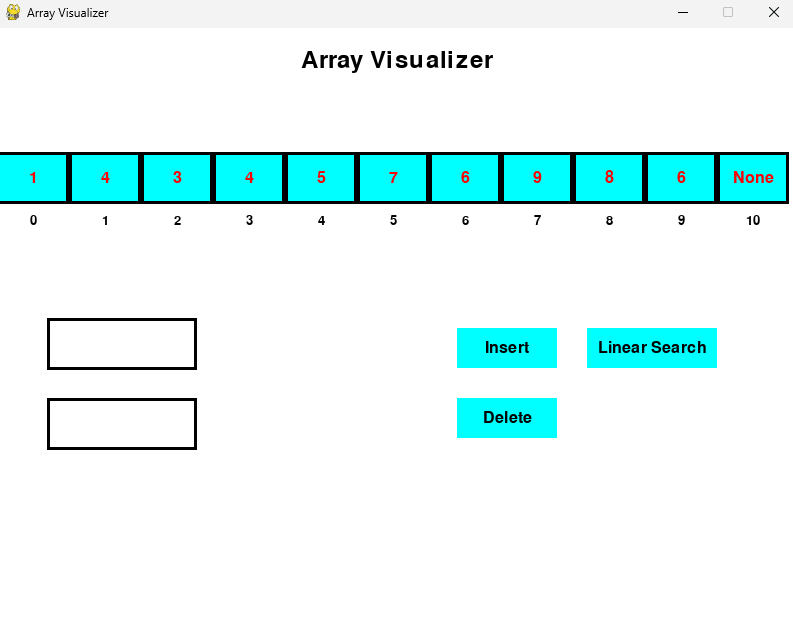
****

Fig:22 Array

* 1. **Challenges Faced**

One of the main challenges faced during the development of this project was balancing the depth of technical functionality with an intuitive and interactive user interface, ensuring the visualizer remained accessible to learners of all levels. Another significant challenge was optimizing the animations to accurately represent algorithm execution without compromising performance or clarity.

* 1. **Future Improvements**

The Data Structure and Algorithm Visualizer has significant potential

for improvement. Future improvements could include integrating

advanced algorithms as well as additional data structures, broadening

the learning scope for users. Moreover, introducing features for

algorithm customization, such as modifying parameters or creating

custom algorithms, would empower users to explore and experiment

further. Enhancements in the user interface, such as mobile

compatibility, could also make the visualizer more accessible and

engaging for a wider audience.

* 1. **Conclusion**

The Data Structure and Algorithm Visualizer is a powerful tool designed to bridge the gap between theoretical concepts and practical understanding in computer science education. By leveraging interactive animations and a user-friendly interface, it offers an engaging platform for learning core data structures and algorithms. With its support for a wide range of functionalities, from sorting and graph traversal to tree operations, the visualizer caters to students, educators, and professionals alike. This visualizer not only simplifies complex concepts but also inspires curiosity and a deeper appreciation for the computer science enthusiasts.