GHULAM ISHAQ KHAN INSTITUTE OF ENGINEERING SCIENCES AND TECHNOLOGY - GIKI

PATTERNS IN ACTION

‘PROJECT REPORT’

CS-221 (DSA)

Proposal Submitted to:

Sir Qasim Riaz
Faculty of Computer Sciences and Electrical Engineering - FCSE

Submitted By:

Team Members:	**Section:** BSCS
Muhammad Haris	2023428
Mumtaz Ali	2023559
Daanish Ahmad Mufti	2023171

Semester Project Documentation (DSA – CS – 221)

Semester Project Title: Patterns in Action (Algorithm Visualizer)

Student Details:

	Student Name	**Student Reg #**	**Student Degree**
Student-1	Daanish Ahmad Mufti	2023171	BSCS
Student-2	Muhammad Haris	2023428	BSCS
Student-3	Mumtaz Ali	2023559	BSCS

1. Main Features

1. Specific Data Structure based sorting algorithm selection.
2. Random Data Selection for Basic Data Structures (Arrays, Stacks, Queues, LinkedList) using Sorting Algorithm Implementation /Specified Data Selection for advanced Data Structures (Graphs, Trees and HashMap) performing specific functions.
3. Speed, Color and Sound customization for making attractive front-end application
4. Operation Selection on Advance Data Structures (Graphs, Trees and HashMap); major operations automatically being performed in Basic Data Structures (Arrays, Stacks, Queues, LinkedList) that are using Sorting Algorithm Implementation.
5. Time Complexity Analysis reported in the detail table below.
6. Complete Data Structures and Algorithm visualization, utilization for making an end-to-end project, helping students/end-users gain more insights into the whole course.

2. Types of Users

1. Students/Teachers/General Users

- Ability to select Algorithm based on a specific Data Structure for visualization.
- Ability to load custom data or generate random data depending on the type of data structure/algorithm selected.
- Ability to select specific operations for Advance Data Structures (Insertion/Deletion etc.) like graphs, hashing etc.
- Interactive speed control for better understanding.
- Visual and audio feedback for operations.
- Covering all visualization while addressing the use of each specific function of the data structure used in that sorting technique e.g. insert, delete, push, pop etc., thus providing intricate details to that Data Structure.

Thus, this program is helpful for instructors to showcase data structures and algorithms to students in class to help them understand better in class and get more insights of how each data structure works to optimize memory and space.

2. Developers

- Access to codebase for modifications.
- Ability to add new algorithms and more visualization of Data Structures and their functioning's.
- Capability to modify visualization parameters.

Therefore, developers can use the program to integrate in greater projects or university websites that want to highlight the foundations of the course content they are offering, the project is open source on GitHub thus anyone can fork it to enhance and use in relevant other projects.

3. Detailed Feature Implementation

1. Basic Data Structures Implemented on Visualization of Sorting Algorithms

- Real-time animation of sorting process using Data Structures, building interest of end user to analyze how the Sorting functions are being performed to implement such visually fascinating tasks thus eventually leading them to the Data-Structure based backend whose learning will enhance the user's overall problem-solving skills.
- Color-coded elements showing current data structure operations being performed and Speed control for better understanding and Sound effects for operations
- Performance timer for execution time analysis.
- Arrays, Linked List, Stacks, Queue and Recursion operations utilized to implement Sorting Visually.
- Covering basic functions of the above-mentioned data structures; such as insert, search and delete in linked list and arrays and push/queue, pop/dequeue in stacks and queue.
- Covering advanced functions of Graph, BST and Hash-map Data structures like traversals, deletion etc.

2. Binary Search Tree Operations

- Interactive node insertion and deletion visualizing how trees form/de-form properly during these operations thus giving a better understand of code, logic building and overall concept of BST.
- Tree traversal visualization, describing how traversal like In-order Traversal moves across the nodes following the order of (Left-Data-Right), thus visually enhancing the ability to depict the proper functioning of Data Structure.
- Dynamic repositioning of nodes with Clear visual representation of tree structure.

3. HashMap Operations

- Interactive visualization of hash table with bucket-based collision handling using chaining
- Real-time insertion of elements with automatic bucket assignment based on modulo hashing
- Color-coded visualization showing elements and their bucket assignments
- Dynamic chain visualization within buckets for colliding elements
- Clear functionality to reset the hash table
- User-friendly input interface for adding new elements
- Visual representation of load factor and collision handling

4. Graph Operations with BFS Visualization

1. Interactive graph structure with nodes and edges visualization
2. Real-time Breadth-First Search (BFS) traversal animation
3. Color-coded nodes showing visited (blue), current (green), and unvisited (white) states
4. Click-based BFS initiation from any selected node
5. Visual representation of adjacency list-based graph structure
6. Edge visualization showing connections between nodes
7. Animation controls with timing delays for better understanding

4. <u>Requirements Breakdown</u>

1. Software Requirements

- C++ compiler with C++11 support
- SFML Graphics Library
- Windows Operating System
- Minimum 4GB RAM

- Graphics card with basic 2D support

2. **Implementation Files**
   - Main program file: *.cpp
   - Header files: SFML headers
   - Resource files:
   - Sound effects (*.wav)
   - Textures (*.png)
   - Font files (*.ttf)

5. Features to Codding Matrix

Sr #	Feature Name	DSA Concept Used	Operation Performed	Complexity Analysis (Approximate)	No. of Variables & Objects Created	Functions Created	Line of Code Written
1	Bubble Sort	Arrays	Comparison & Swapping	$O(n^2)$	~10 variables, 1 array	bubbleSort(), drawArray()	~50
2	Selection Sort	Doubly Linked List	Node Traversal & Swapping	$O(n^2)$	~15 variables, DLL class	selectionSort(), swap ()	~100
3	Insertion Sort	Dynamic Stack & Array	Push, Pop & Insertion	$O(n^2)$	~12 variables, stack array	insertionSort()	~80
4	Merge Sort	Arrays, Recursion	Divide & Merge	O(n log n)	~15 variables, temp array	mergeSort(), merge()	~120
5	Quick Sort	Arrays, Stack	Partitioning & Recursion	O(n log n)	~10 variables, indices vector	quickSort()	~70
6	BST Visualization	Binary Search Tree	Insert, Delete, Traversal	O(log n) - average	BST class, Node struct	Multiple tree operations	~200
7	HashMap Visualization	Hash Table with Chaining	Insert, Clear, Hash Function	O(1) average, O(n) worst	HashMap class, Bucket vectors	initializeGUI(), addElement(), clearHashmap()	~200
8	Graph Visualization	Graph + Queue	BFS Traversal, Node/Edge Management	O(V + E)	Node/Edge structs, Queue	startBFS(), processBFS(), addNode()	~250

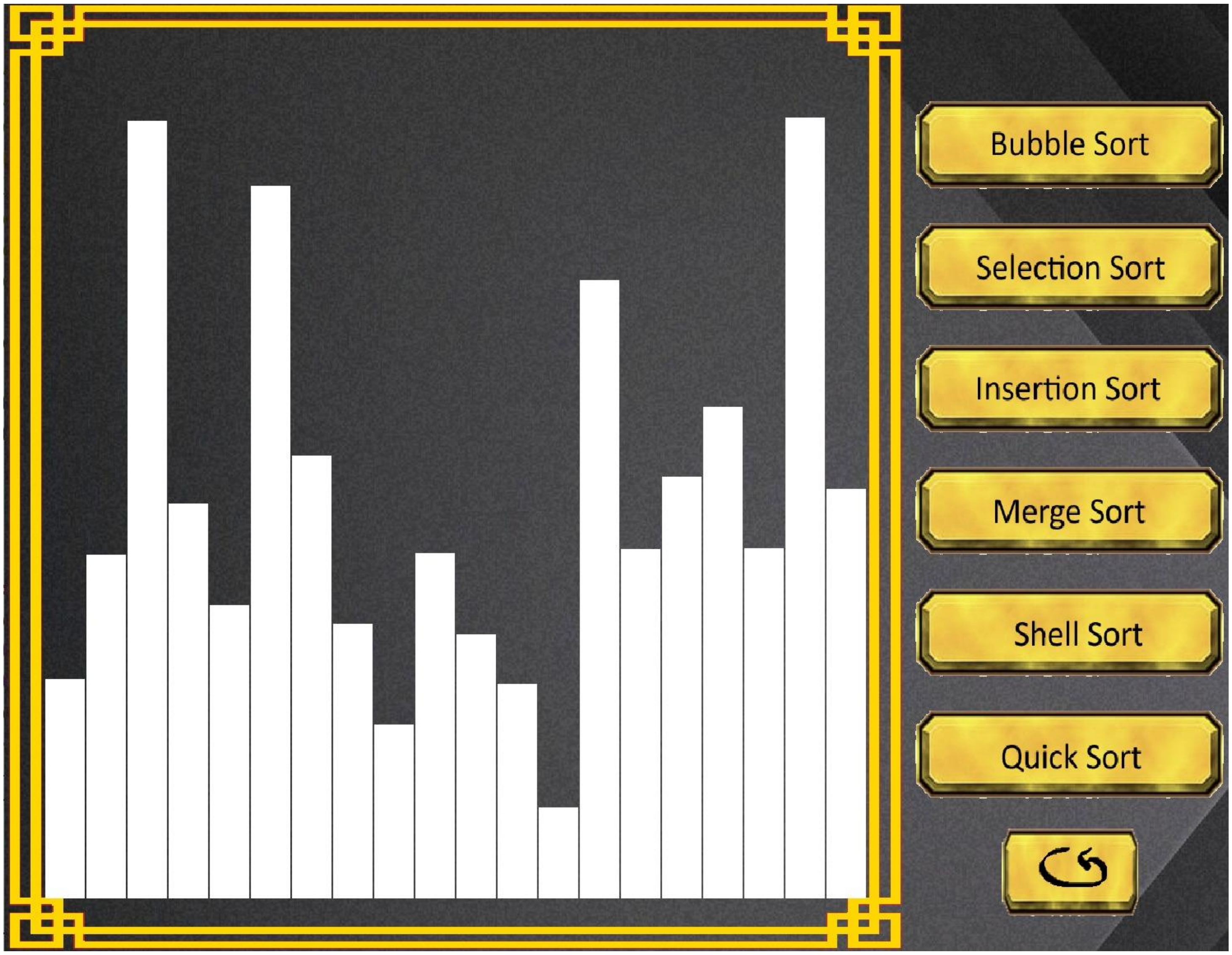
Note:

Graph Traversal (BFS/DFS)

Not directly feasible with an edge list. Traversals like BFS and DFS require efficient adjacency information for each vertex. To perform traversal efficiently, you would need to convert the edge list to an adjacency list, which takes O(V+E) time.

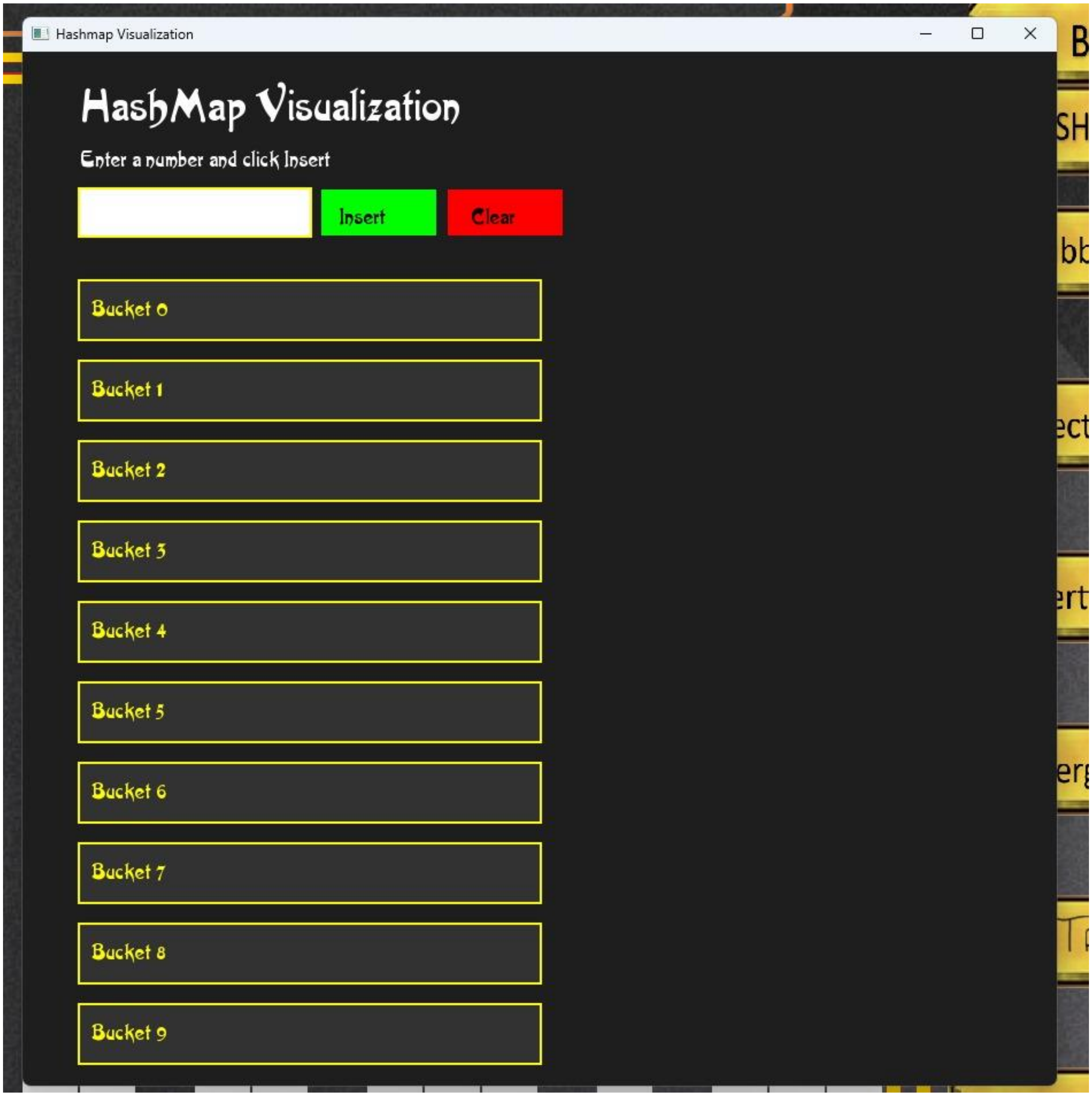
6. Project Screenshots

SCREENSHOT DEMONSTRATING VARIOUS SORTING OPTION THAT ARE BASED ON DATA STRUCTURES BACKEND



Note: These sorting' Algorithms are working through data-structures whose specifications are already described in the table above.

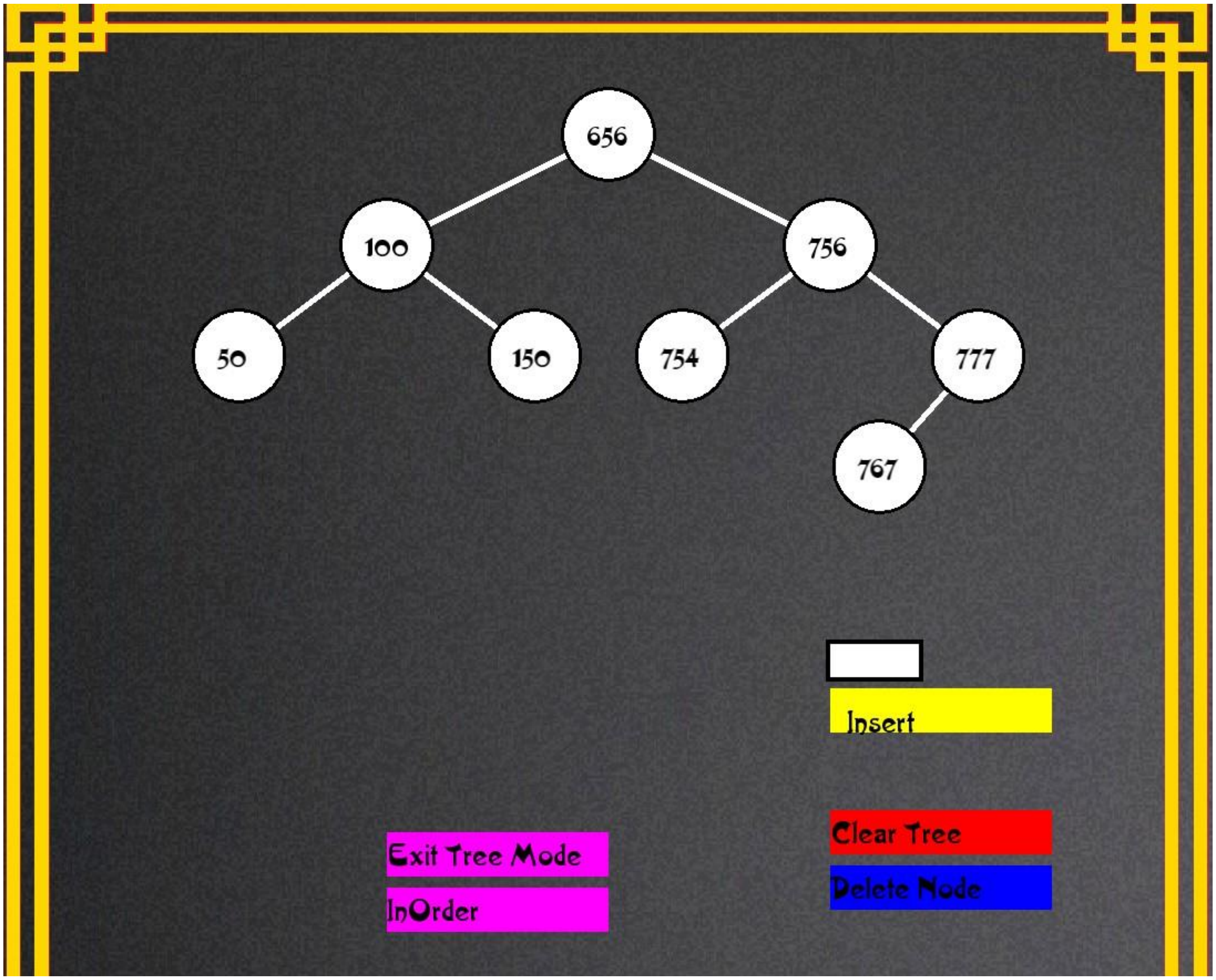
SCREENSHOT DEMONSTRATING HASHMAP VISUALIZATION AND OPERATIONS


Hashmap Visualization

HashMap Visualization

Enter a number and click Insert

<u>SCREEN SHOT DEMONSTRATING TREE DATA STRUCTURE AND OPERATIONS</u>



<u>SCREENSHOT VISUALIZING BFS TRAVERSAL IN GRAPHS WHERE USER CAN SELECT HIS OWN SPECIFIC START NODE</u>

Nodes: 7

Queue Size: 1
Click on any node to start BFS