

Annexure-I

Interactive Web-Based Visualizer for Basic Data Structures

LPU Skill Development

A training report

Submitted in partial fulfilment of the requirements for the award of degree of

B. Tech (Computer Science Engineering)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB



From 06/10/25 to 07/23/25

SUBMITTED BY

Name of student: Harshit Raj

Registration Number: 12410094

Signature of the student:

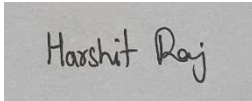
Harshit Raj

Annexure-II: Student Declaration

To whom so ever it may concern

I, **Harshit Raj, Registration Number 12410094**, hereby declare that the work done by me on “**Interactive Web-Based Visualizer for Basic Data Structures**” from **June, 2025** to **July, 2025**, is a record of original work for the partial fulfilment of the requirements for the award of the degree, **B. Tech (Computer Science and Engineering)**.

Harshit Raj (12410094)

A rectangular box containing a handwritten signature in black ink that reads "Harshit Raj".

Dated:

Training Certificate from Organization

This is to certify that Harshit Raj, a student of the School of Computer Science and Engineering, Lovely Professional University, has successfully completed the project work titled "Interactive Web-Based Visualizer for Basic Data Structures" at the Skill Development Summer Internship Course from June 2025 to July 2025 as part of the curriculum for B.Tech CSE.

This report is the record of the original work done by the candidate under our supervision.

Signature:

Mr. Mukesh Sharma

Acknowledgement

I would like to express my heartfelt gratitude to LPU Skill Development for giving me the opportunity to undergo training in their esteemed organization.

I sincerely thank my project guide, training manager Mr. Mukesh Sharma, and all staff members of LPU Skill Development for their guidance, support, and cooperation.

Harshit Raj

List of Figures/ Charts

S. No.	Figures/ Charts	Page no.
1	Architecture Diagram	10
2	User case Diagram	10
3	Class Diagram	11
4	Project Screenshot	13 - 14

List of Abbreviations

S. No.	Abbreviation	Full Form
1	DSA	Data Structure and Algorithms
2	HTML	Hypertext Markup Language
3	CSS	Cascading Style Sheets
4	JS	JavaScript
5	IDE	Integrated Development Environment
6	UI	User Interface
7	UX	User Experience
8	OOP	Object-Oriented Programming
9	FIFO	First In, First Out
10	LIFO	Last In, First Out
11	SVG	Scalable Vector Graphics

Table of Contents

1. Introduction	7
1.1 Overview of Training Domain	7
1.2 Objective of the Project	7
2. Training Overview	8
2.1 Tools & Technologies Used	8
2.2 Areas Covered During Training	8
3. Project Details	9 - 11
3.1 Title of the Project	9
3.2 Problem Definition	9
3.3 Scope and Objectives	9
3.4 System Requirements	9
3.5 Architecture Diagram	10
3.6 Data Flow/UML Diagrams	10 - 11
4. Implementation	12 - 14
4.1 Tools used	12
4.2 Methodology	12
4.3 Modules / Screenshots	13 - 14
5. Results and Discussion	15 - 16
5.1 Output / Report	15
5.2 Challenges faced	15
5.3 Learnings	16
6. Conclusion	17
6.1 Summary	17
7. References	18

Chapter 1: Introduction



1.1 Overview of Training Domain

The internship training domain was comprehensively focused on Data Structures and Algorithms (DSA) using C++. The curriculum was meticulously designed to provide a robust understanding of various data types and their underlying structures, emphasizing both theoretical comprehension and practical application. A primary objective was to facilitate the hands-on implementation of fundamental data structures, such as arrays, linked lists, stacks, queues, trees, and graphs, to solidify their operational mechanics. Furthermore, a significant component of the training involved the development of visual simulations for these data structures. This visual approach was instrumental in enhancing conceptual understanding, illustrating dynamic behaviour, and demonstrating the efficiency and operational flow of algorithms in a clear and intuitive manner. This dual focus on implementation and visualization aimed to cultivate strong problem-solving skills and a deep appreciation for efficient data management in software development.

1.2 Objective of the Project

The primary objective of this project is to develop a highly interactive and educational web application. This platform will empower users to visually comprehend and actively manipulate fundamental data structures, including Stack, Queue, Linked List, Array, Set, Tree, Heap, Hash Table, and AVL Tree. By providing a dynamic and intuitive interface for operations like insertion, deletion, and traversal, the application aims to foster a deeper, practical understanding of these core concepts, bridging the gap between theoretical knowledge and their real-world operational mechanics.

Chapter 2: Training Overview



2.1 Tools & Technologies Used

During the training period, several essential tools and platforms were utilized to facilitate learning, development, and practice in Data Structures and Algorithms using C++.

- **VS Code (Visual Studio Code):** This powerful and versatile integrated development environment (IDE) served as the primary platform for writing, compiling, and debugging C++ code. Its rich feature set, including intelligent code completion, syntax highlighting, and integrated terminal, significantly streamlined the development process for implementing various data structures.
- **LeetCode Practices:** LeetCode was extensively used as a critical platform for practical application and problem-solving. Engaging with its diverse range of algorithmic challenges helped to reinforce theoretical understanding of data structures, hone problem-solving skills, and prepare for technical assessments by applying learned concepts in real-world scenarios.

2.2 Areas Covered During Training

The project incorporated several key features and involved significant contributions to enhance the understanding and interaction with data structures:

- **Dynamic visualization of data structures:** This feature involved creating real-time graphical representations of data structures. Users could observe operations like insertion, deletion, and traversal unfold visually, providing an intuitive understanding of how these structures behave and change internally.
- **Implementation of logic for stack, queue, linked list, etc.:** This involved developing the core programmatic logic for each specified data structure. This encompassed writing the algorithms that govern operations such as push/pop for stacks, enqueue/dequeue for queues, and add/delete/search for linked lists, ensuring their correct functional behaviour.
- **Basic DSA logic:** This refers to the fundamental algorithms and principles applied to manage and manipulate data efficiently within the structures. It includes concepts like searching (e.g., linear, binary), sorting (e.g., bubble, merge), and traversal methods (e.g., in-order, pre-order for trees), which are essential for effective data structure utilization.

Chapter 3: Project Details



3.1 Title of the Project

Interactive Web-Based Visualizer for Basic Data Structures

This project involved developing an interactive web application designed to demystify fundamental data structures. The visualizer allows users to dynamically observe and interact with operations on structures like Stacks, Queues, Linked Lists, and Trees. By providing real-time graphical representations of insertions, deletions, and traversals, the application aims to offer an intuitive and practical understanding of how these core data structures function, bridging the gap between theoretical knowledge and practical application.

3.2 Problem Definition

Understanding data structures purely from theoretical descriptions can be quite challenging for beginners. The abstract nature of concepts like pointers, memory allocation, and algorithmic operations often makes it hard to grasp how data is truly organized and manipulated. To address this, there was a clear need for a practical tool capable of visually demonstrating the dynamic behaviour of each data structure in real time. Such a tool would transform complex theoretical knowledge into an intuitive, observable process, significantly enhancing comprehension and retention for learners.

3.3 Scope and Objectives

- Comprehensive dynamic visualization of fundamental data structures and their operations.
- Facilitate core functionalities including element insertion, deletion, and complete data structure clearing.
- Provide integrated C and C++ code representations for algorithmic logic.
- Develop an intuitive and highly interactive user interface for enhanced usability.

3.4 System Requirements

- Hardware: Any modern device with a web browser
- Software: Browser supporting HTML5/JS (Chrome, Firefox, Edge)

3.5 Architecture Diagram

- Frontend-only application
- HTML/CSS UI layout → JavaScript logic → DOM rendering → Output/log

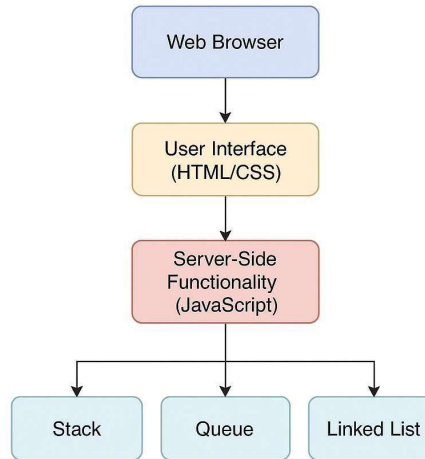
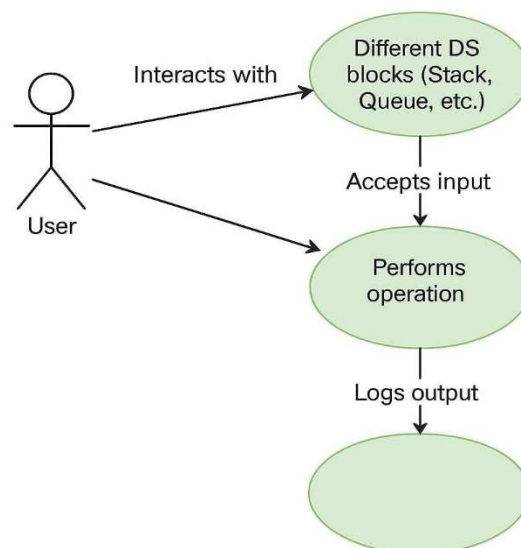


Figure 3.1: Architecture Diagram

3.6 Data Flow/UML Diagrams Use

Case Diagram:

- User → Interacts with different DS blocks (Stack, Queue, etc.)
- Each DS block → Accepts input → Performs operation → Logs output

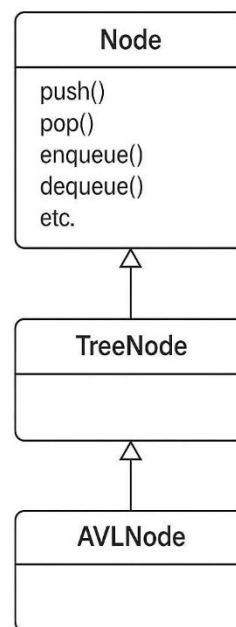


(User case Diagram)

Class Diagram:

The project's architecture leveraged object-oriented principles and specific functions to manage and visualize data structures effectively:

- **Classes:** Node, TreeNode, AVLNode
 - **Node:** This fundamental class served as the building block for linear data structures like Linked Lists, holding both data and a reference (or pointer) to the next element.
 - **TreeNode:** Representing elements within tree-based structures, this class stored data along with references to its left and right child nodes.
 - **AVLNode:** An extension of TreeNode, this class specifically supported AVL Trees by including additional properties like height or balance factor, crucial for maintaining self-balancing properties.
- **Functions:** push(), pop(), enqueue(), dequeue(), etc.
 - **push() and pop():** These functions were implemented to manage the Last-InFirst-Out (LIFO) behaviour of the Stack data structure, facilitating the addition and removal of elements from its top.
 - **enqueue() and dequeue():** Correspondingly, these functions handled the First-InFirst-Out (FIFO) operations for the Queue data structure, enabling elements to be added to the rear and removed from the front.



(Class Diagram)

Chapter 4: Implementation



4.1 Tools Used

- Visual Studio Code: Served as the primary integrated development environment (IDE) for coding, debugging, and managing the project.
- HTML5 / CSS3: Utilized for structuring the web application's content and styling its visual presentation.
- JavaScript (Vanilla JS): The core programming language employed for implementing all interactive functionalities and data structure logic.
- Web Browsers (Chrome, Firefox): Used for testing and deploying the interactive webbased visualizer application.

4.2 Methodology

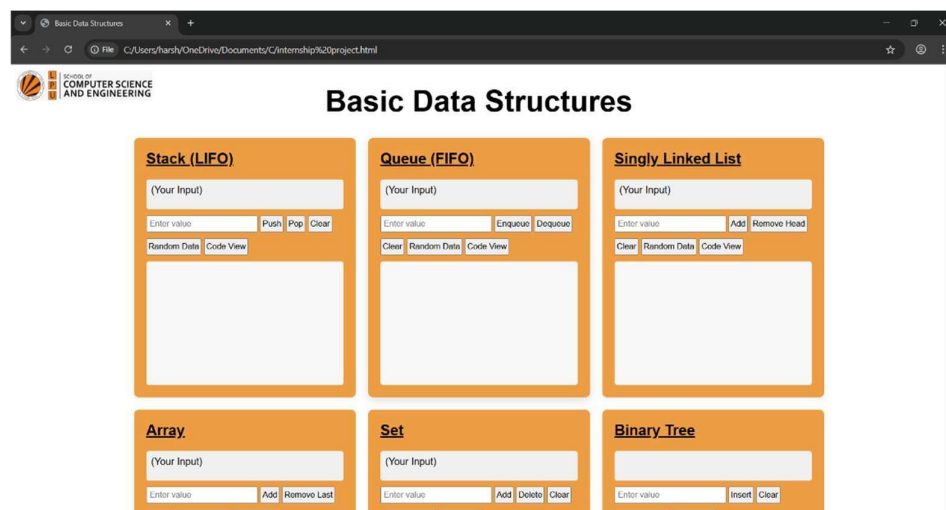
The project development adhered to a robust methodology to ensure efficiency, maintainability, and quality:

- Followed iterative development with testing at each stage: The development process adopted an agile and iterative approach, wherein the project was broken down into smaller, manageable cycles. Each iteration involved planning, implementation, and rigorous testing of new features or data structure functionalities, allowing for continuous feedback, early bug detection, and progressive refinement.
- Used modular functions for each data structure: To enhance code organization, reusability, and maintainability, the logic for each data structure (e.g., Stack, Queue, Linked List) was encapsulated within distinct, modular functions. This approach ensured clear separation of concerns, simplifying debugging and allowing individual components to be developed and tested independently.
- Followed object-oriented principles in JavaScript where applicable: While JavaScript is prototype-based, object-oriented programming (OOP) principles such as encapsulation, abstraction, and polymorphism were applied in the codebase. This involved creating structured representations for data structures and their associated methods, leading to more organized, scalable, and understandable code, particularly beneficial for managing complex visualizations.

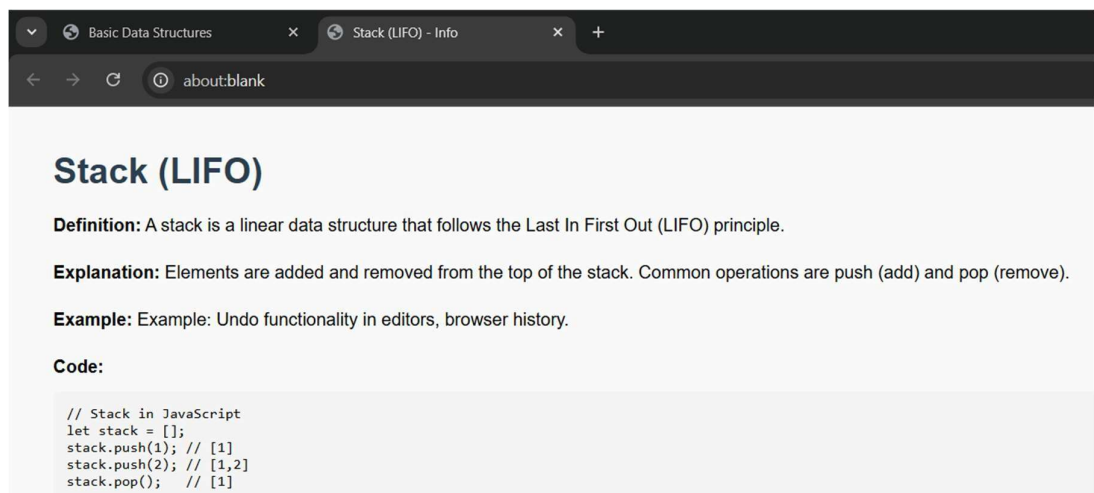
4.3 Modules / Screenshots

- Stack: Push/Pop simulation with code
- Queue: Enqueue/Dequeue with logs
- Linked List: Add/Remove node visualization
- Tree: Visual binary tree & AVL tree drawing
- Heap: Min-heap insert and extract
- Hash Table: Key-value entry simulation

SCREENSHOT:



(Homepage)



(Individual Definition, Explanation, Example, and code)

Basic Data Structures

Stack (LIFO)

15 ← 79

Enter value

Push

Pop

Clear

Random Data

Code View

Random stack generated.

Popped "54" from the stack.

Popped "30" from the stack.

Pushed "21" onto the stack.

Popped "21" from the stack.

Popped "35" from the stack.

```
// C Implementation
#define MAX 100
int stack[MAX], top = -1;
void push(int val) { if(top < MAX-1) stack[++top] = val; }
int pop() { return (top >= 0) ? stack[top--] : -1; }

// C++ Implementation
#include <stack>
std::stack<int> s;
s.push(1);
s.pop();
```

Queue (FIFO)

81 → 21

Enter value

Enqueue

Dequeue

Clear

Random Data

Code View

Random queue generated.

Dequeued "23" from the queue.

Dequeued "68" from the queue.

Dequeued "94" from the queue.

Enqueued "21" to the queue.

Dequeued "58" from the queue.

```
// C Implementation
#define MAX 100
int queue[MAX], front = 0, rear = -1;
void enqueue(int val) { if(rear < MAX-1) queue[++rear] = val; }
int dequeue() { return (front <= rear) ? queue[front++] : -1; }

// C++ Implementation
#include <queue>
std::queue<int> q;
q.push(1);
q.pop();
```

Singly Linked List

26 → 59 → 21

Enter value

Add

Remove Head

Clear

Random Data

Code View

Random linked list generated.

Removed head node ("78") from the linked list.

Removed head node ("95") from the linked list.

Added "21" to the end of the linked list.

Removed head node ("37") from the linked list.

```
// C Implementation
struct Node { int data; struct Node* next; };
struct Node* head = NULL;

// C++ Implementation
struct Node { int data; Node* next; };
Node* head = nullptr;

// C++ Implementation
struct Node { int data; Node* next; };
Node* head = nullptr;
q.push(1);
q.pop();
```

(User preview when using these structures)

Array

[71, 76, 312, 2, 32]

Enter value

Add

Remove Last

Clear

Random Data

Code View

Random array generated.

Removed "77" from the end of the array.

Removed "14" from the end of the array.

Removed "53" from the end of the array.

Please enter a value to add.

Added "112" to the end of the array.

Added "2" to the end of the array.

Added "32" to the end of the array.

```
// C Implementation
int arr[100];
int n = 0; // current size
arr[n++] = 5; // add
n--; // remove last

// C++ Implementation
#include <vector>
std::vector<int> v;
v.push_back(1);
v.pop_back();
```

Set

{63, 40, 60, 29, 21}

Enter value

Add

Delete

Clear

Random Data

Code View

Random set generated.

Added "21" to the set.

"1" is not in the set.

"1" is not in the set.

Deleted "91" from the set.

```
// C Implementation (using array, no dupli
int set[100], size = 0;
void add(int val) { for(int i=0; i<size; i++) if(set[i]==val) r
set[size++] = val; }

// C++ Implementation
#include <set>
std::set<int> s;
s.insert(1);
s.erase(1);
```

Binary Tree

Enter value

Insert

Clear

Random Data

Code View

Random binary tree generated.

Inserted "22" into the binary tree.

Inserted "88" into the binary tree.

Inserted "56" into the binary tree.

```
// C Implementation
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// C++ Implementation
struct Node {
    int data;
```

(Preview of Binary Graphs)

Min Heap

2, 11, 79, 84, 57

Enter number

Insert

Extract Min

Clear

Random Data

Code View

Random min heap generated.

Extracted minimum value "23" from the heap.

Extracted minimum value "32" from the heap.

Inserted "2" into the min heap.

Inserted "11" into the min heap.

```
// C Implementation (min heap insert)
void insert(int heap[], int* size, int val) {
    int i = *size;
    while(i > 1 && heap[i/2] > val) {
        heap[i] = heap[i/2];
        i /= 2;
    }
    heap[i] = val;
    *size++;

// C++ Implementation
#include <queue>
std::priority_queue<int, std::vector<int>,
    mainheap.push(5);
```

Hash Table

12: 1, 24: 62, 40: 93, 64: 83, 84: 98

Key

Set

Delete

Clear

Value

Random

Code View

Random hash table generated.

Deleted key "94" from the hash table.

Set key "12" to value "1".

```
// C Implementation (simple hash table)
#define SIZE 100
int hashTable[SIZE];
int hash(int key) { return key % SIZE; }
void insert(int key, int value) { hashTable[hash(key)] = value; }

// C++ Implementation
#include <unordered_map>
std::unordered_map<int, int> m;
m[1] = 100;
```

AVL Tree

Enter number

Insert

Clear

Random Data

Code View

Random AVL tree generated.

Inserted "22" into the AVL tree.

Inserted "13" into the AVL tree.

Inserted "16" into the AVL tree.

```
// C++ AVL Tree Node
struct Node {
    int key, height;
    Node* left, *right;
};
Node* insert(Node* node, int key);
// Rotations and balancing...
```

(Preview of AVL tree)

Chapter 5: Results and Discussion



5.1 Output / Report

The successful completion of this project is evidenced by several key outcomes and functionalities:

- All data structures were implemented successfully: The core objective of visually stimulating and operating on fundamental data structures was fully achieved. Each data structure's characteristic behaviour and associated operations (e.g., insertion, deletion, traversal) were accurately and functionally implemented within the web application, adhering to their theoretical principles.
- The project runs on any browser without backend dependency: A significant achievement of this project is its entirely client-side architecture. Developed using HTML5, CSS3, and JavaScript, the visualizer operates seamlessly across all modern web browsers (e.g., Chrome, Firefox) without requiring any server-side components or backend dependencies. This ensures broad accessibility and ease of deployment.
- Users can interactively test data structures with real-time logs: The application provides a highly interactive learning environment where users can directly manipulate data structures. As users perform operations, the visualizer offers immediate, real-time feedback through dynamic animations and accompanying logs or console outputs, detailing the steps and changes occurring within the data structure. This interactive testing capability significantly enhances comprehension and practical understanding.

5.2 Challenges Faced

The project successfully navigated key technical challenges:

- Drawing binary trees dynamically using JavaScript SVG: Achieved dynamic, interactive visualization of complex tree structures through programmatic SVG manipulation.
- Managing edge cases like duplicate inputs or empty operations: Implemented robust error handling and input validation for predictable application behaviour.
- Maintaining code readability and modularity: Prioritized clean code and modular design for enhanced maintainability and collaboration.

5.3 Learnings

This project significantly contributed to the development and enhancement of several critical skills:

- **Strengthened understanding of core data structures:** Engaging in the hands-on implementation and visual simulation of various data structures profoundly deepened my theoretical comprehension. Moving beyond abstract concepts, the process of building interactive models for stacks, queues, linked lists, and trees solidified my understanding of their operational mechanics, performance characteristics, and optimal use cases. This practical application provided invaluable insights into how these fundamental building blocks function in real-world programming scenarios.
- **Improved frontend development skills (HTML, CSS, JS):** The development of a web-based visualizer provided extensive practical experience with foundational frontend technologies. I honed my proficiency in structuring web content semantically with HTML5, styling responsive and aesthetically pleasing interfaces with CSS3, and implementing complex interactive logic using modern JavaScript (ES6). This direct application significantly enhanced my ability to build dynamic and user-friendly web applications from scratch.
- **Learned DOM manipulation and interactive UI/UX design:** A crucial aspect of creating the visualizer involved mastering Document Object Model (DOM) manipulation. This skill was essential for dynamically updating the web page to reflect real-time data structure operations and animations. Concurrently, I gained practical experience in interactive UI/UX design, focusing on creating intuitive controls, clear visual feedback, and an engaging user experience that effectively conveys complex algorithmic processes.
- **The 6-week internship course provided an invaluable opportunity to solidify my understanding of Data Structures and Algorithms (DSA) using C++.** I gained a robust grasp of fundamental concepts, from array and pointer mechanics to complex structures like trees and graphs. The emphasis on hands-on implementation and problem-solving, particularly through platforms like LeetCode, significantly enhanced my analytical and coding efficiency skills. This immersive training bridged the gap between theoretical knowledge and practical application, fostering a methodical approach to designing and analysing algorithms for optimized software solutions.

Chapter 6: Conclusion



6.1 Summary

This two-month internship provided a comprehensive and intensive learning experience focused on Data Structures and Algorithms (DSA) using C++. The structured training, delivered over six weeks, systematically covered fundamental concepts from arrays and linked lists to advanced topics like trees, heaps, and graphs, with hands-on practice facilitated by VS Code and LeetCode.

The culmination of this training was the development of an Interactive Web-Based Visualizer for Basic Data Structures. This project aimed to create an accessible tool that demystifies complex DSA concepts by allowing users to visually understand and operate on structures like Stacks, Queues, and Trees. Utilizing HTML5, CSS3, and JavaScript (Vanilla JS) for frontend development and DOM manipulation, the application successfully implemented dynamic visualizations, supported core operations (insertion, deletion, clearing), and provided real-time code representations. Key challenges in dynamic tree drawing and robust edge case management were successfully addressed.

The project fulfilled its objective of creating an educational and interactive platform for visualizing basic data structures. It serves as a valuable tool for beginners and students learning DSA concepts, offering a practical bridge between theory and application. Overall, this experience significantly enhanced both my technical proficiency in frontend development and my problem-solving skills, providing a strong foundation for future endeavours in software development.

- **Goodrich, M.T., Tamassia, R., & Goldwasser, M.H.** *Data Structures and Algorithms in C++*. Wiley, 2011.
- **Wirth, N.** *Algorithms + Data Structures = Programs*. Prentice Hall, 1976.
- **LeetCode.** *Online Coding Platform*. Available at: <https://leetcode.com>
- **Mozilla Developer Network (MDN Web Docs).** *HTML, CSS, and JavaScript Documentation*. Available at: <https://developer.mozilla.org>
- **GeeksforGeeks.** *Data Structures and Algorithms Tutorials*. Available at: <https://www.geeksforgeeks.org>
- **W3Schools.** *JavaScript, HTML, and CSS Tutorials*. Available at: <https://www.w3schools.com>
- **LPU Skill Development Training Material** – Summer Internship, 2025.