A Dissertation work entitled on

DATA ALGO VISOR

A Project Report submitted in partial fulfilment of the requirements for the award
of the degree of
Bachelor's of Engineering
in
Information Technology
By

N.Nikila (160620737026)
N.Kavya (160620737027)
Vimudha Mahesekar (160620737044)

Under the guidance of
Dr. Malathi Latha
Associate Professor

Stanley College of Engineering & Technology for Women (Autonomous)
Chapel Road, Abids, Hyderabad – 500001
(Affiliated to Osmania University, Hyderabad, Approved by AICTE, Accredited by NBA and NAAC with 'A' grade)
2021-2022

Stanley College of Engineering & Technology for Women (Autonomous)
Chapel Road, Abids, Hyderabad – 500001
(Affiliated to Osmania University, Hyderabad, Approved by AICTE, Accredited by NBA and NAAC with 'A' grade)
2021-2022

CERTIFICATE

This is to certify that the project entitled "Data Algo Visor" submitted by N.Nikila (160620737026) , N.Kavya (160620737027) , Vimudha Mahesekar (160620737044) ,the students of Department of Information Technology, Stanley College of Engineering and Technology for Women, in partial fulfilment of the requirements for the award of the degree of Bachelor's of Engineering with Information Technology as specialisation is a record of bonafide work carried out by them during academic year 2022-2023.

Internal Guide **HOD** **External Examiner**

Stanley College of Engineering & Technology for Women (Autonomous)
Chapel Road, Abids, Hyderabad – 500001
(Affiliated to Osmania University, Hyderabad, Approved by AICTE, Accredited by NBA and NAAC with 'A' grade)
2021-2022

DECLARATION

We declare that the work reported in the thesis entitled "Data Algo Visor" is a record of the work done by us in the Department of Information Technology in Stanley College of Engineering and Technology for Women, Hyderabad.
No part of the thesis is copied from books / journals / internet and wherever referred the same has been duly acknowledged in the text. The reported data is based on the project work done entirely

by us and not copied from any other source.

N.Nikila (160620737026)
N.Kavya (160620737027)
Vimudha Mahesekar (160620737044)

Stanley College of Engineering & Technology for Women (Autonomous)
Chapel Road, Abids, Hyderabad – 500001
(Affiliated to Osmania University, Hyderabad, Approved by AICTE, Accredited by NBA and NAAC with 'A' grade)
2021-2022

ACKNOWLEDGEMENTS

We take this opportunity to convey our heartfelt gratitude to each and every one who has supported us in every way or the other during the course of our project. From the very core of my heart, we
would like to express our sincere gratitude to our internal guide Dr.Malathi Latha for her supervisory guidance. We express our profound gratitude to project coordinator Dr.Malathi Latha and Dr.Srinivasu Badugu, HOD of Information Technology.We are always grateful to our peers and friends who have always encouraged us and guided us whenever we needed assistance.

Vision of the Institute and Department

The Vision of the STLW:

Empower Women; Impact the World
Empowering girl students through professional education integrated with values and character to make an impact in the World.

The Mission STLW, in pursuance of its vision:

M1: Providing quality engineering education for girl students to make them competent and confident to succeed in professional practice and advanced learning.
M2: Establish state-of-art-facilities and resources to facilitate world class education.
M3: Integrating qualities like humanity, social values, ethics, and leadership in order to encourage contribution to society.

Vision of the Information Technology Department:

Empowering girl students with the contemporary knowledge in Information Technology, for their success in life

Mission of the Information Technology Department:

M1: Providing quality education and excellent environments for students to learn and practice various latest hardware, software and firmware platforms.
M2: To establish industry oriented training integrated with opportunities for team work, leadership.
M3: To groom students with values, ethics and social activities

Programme Educational Objectives

PEO1: Graduates shall have enhanced skills and contemporary knowledge to adapt new software and hardware technologies for professional excellence, employment and Research.
PEO2: Proficient in analyzing, developing and solving engineering problems to assist life-long learning and to develop team work.
PEO3: To inculcate self-confidence, acquire professional and ethical attitude, infuse leadership qualities, impart proficiency in soft-skills, and the ability to relate engineering with social issues.

Pos and PSOs of IT Dept

PROGRAMME OUTCOMES:

1) **Engineering knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the conceptualization of engineering models.

2) **Problem Analysis:** Identify, formulate, research literature and solve complex engineering problems reaching substantiated conclusions using first principles of mathematics and engineering sciences.

3) **Design/development of solutions:** Design solutions for complex engineering problems and design systems, components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations.

4) **Conduct investigations of complex problems:** Conduct investigations of complex problems including design of experiments, analysis and interpretation of data, and synthesis of information to provide valid conclusions.

5) **Modern Tool Usage:** Create, select and apply appropriate techniques, resources, and modern engineering tools, including prediction and modeling, to complex engineering activities, with an understanding of the limitations.

6) **The engineer and society:** Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.

7) **Environment & sustainability:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

8) **Ethics:** Demonstrate understanding of the societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to engineering practice.

9) **Individual and Teamwork:** Understand and commit to professional ethics, responsibilities, and norms of engineering practice.

10) **Communication:** understand the impact of engineering solutions in a societal context, demonstrate knowledge of, and need for sustainable development.

11) **Project Management and Finance:** Demonstrate a knowledge and understanding of management and business practices, such as risk and change management, and understand their limitations.

12) **Lifelong Learning:** Recognize the need for, and have the ability to engage in independent and life-long learning

PROGRAMME SPECIFIC OUTCOMES:

PSO1: **Skilled Professional:** Ability to apply technical skills and involve in the Creation, maintenance and use of Computer, Computer Networks and Computer Information Systems.

PSO2: **Research Capability:** Ability to pursue research with academic excellence and core competence skills.

ABSTRACT

The project aims to enhance student's understanding of data structure operations and various algorithms. The application is meticulously designed using HTML5,CSS3 and JavaScript adhering to professional standards and best practices. Its user-friendly interface ensures seamless navigation, while the utilisation of cutting-edge web technologies facilitates an optimal learning experience.

This educational platform empowers learners with practical insights and theoretical knowledge, enabling them to master essential DSA concepts effectively. The application simulates operations on data structures and algorithms like Sorting, Searching, Trees, Path Finding, etc Through this web-based platform, users can gain clear insights into data structures and algorithms, including searching, sorting, insertion, and deletion. The project combines practical simulation with theoretical explanations, offering an effective and efficient approach to learning data structures.

By combining visual engagement with algorithmic exploration, our project aims to revolutionize the way algorithms are learned and understood. With a focus on accessibility, accuracy, and versatility, our platform provides educators, students, and self-learners with a powerful tool to enhance their algorithmic proficiency. As computer science continues to evolve, our algorithm visualization project stands as a significant contribution to cultivating a generation of programmers who can confidently navigate complex algorithms and harness their potential in real-world applications.

CONTENTS

List of Figures

CHAPTER 1

1.INTRODUCTION

Aim behind implementation of this project to make a clear understandability of various algorithms of data structures. Using a web page this will simulate the data structure operations such as searching, sorting, insertion, deletion etc. In array, stack, queue, and linked list as well. Thus our web page provides effective and efficient knowledge of data structures. This also provides some theoretical knowledge regarding the data structure.

1.1 MOTIVATION

The motivation behind creating an algorithm visualizer stems from the desire to enhance the learning experience and comprehension of intricate algorithms and data structures. By incorporating visual elements and interactive features, these tools provide an intuitive understanding of complex processes, enabling learners to follow the flow of algorithms and their operations more easily.

1.2 PROBLEM DESCRIPTION

The problem description for this project involves addressing the challenge of facilitating effective learning and understanding of complex algorithms and data structures. Conventional learning methods often rely on textual explanations, which may be challenging for some learners to grasp abstract concepts. As a result, there is a need to create an interactive and visually engaging educational platform that employs cutting-edge web technologies to simulate operations on data structures and algorithms such as sorting, searching, trees, and path finding.

1.3 OBJECTIVE

Designed with meticulous attention to professional standards and best practices, the web-based application offers a user-friendly interface, ensuring seamless navigation and engagement. By combining theoretical explanations with practical simulations, the project aims to empower learners with both practical insights and theoretical knowledge, enabling them to master essential DSA concepts effectively. Through interactive simulations of algorithm operations like sorting, searching, trees, and path finding, users can observe step-by-step behaviours and gain clear insights into algorithm functioning.

1.4 BACKGROUND

Traditional teaching methods often rely on textual explanations, which may hinder students' comprehension of abstract concepts. To address this, the project aims to create an interactive web-based application using HTML5, CSS3, and JavaScript, adhering to professional standards and best practices. By employing cutting-edge web technologies, the algorithm

visualizer offers practical simulations and visualisations for sorting, searching, trees, and path finding operations.

1.5 REPORT OVERVIEW

The report provides an overview of the algorithm visualizer project, a web-based educational platform designed to enhance students' understanding of data structures and algorithms. The project utilises HTML5, CSS3, and JavaScript, adhering to professional standards and best practices to create a user-friendly interface. The algorithm visualizer covers operations like sorting, searching, trees, and path finding, encouraging engagement and fostering problem-solving skills. The report emphasises the project's objective of bridging the gap between theory and real-world application, empowering users with a deeper understanding of algorithms. Additionally, it highlights the significance of the visualisations in promoting effective learning and debugging techniques.

CHAPTER 2

2.LITERATURE SURVEY:

The literature survey for the algorithm visualizer project explores existing research and tools related to algorithm visualisation and interactive learning platforms for data structures and algorithms. The survey also reviews prior works on web-based educational platforms using HTML5, CSS3, and JavaScript to simulate algorithm operations. Additionally, it highlights the potential benefits of interactive learning and visualisation in promoting problem-solving skills and facilitating knowledge transfer. Overall, the literature survey serves as a foundation for the project, guiding its development to build upon existing research and best practices in the field of algorithm visualisation and educational technology.

2.1 EXISTING SYSTEM:

- **Virtual Labs**

Virtual labs are online educational platforms that provide a simulated environment for students to conduct experiments and explore various scientific concepts. These digital laboratories replicate real-world lab experiences, allowing users to manipulate virtual equipment, collect data, and observe outcomes. By offering a risk-free and accessible way to engage with experiments, virtual labs bridge the gap between theoretical knowledge and practical skills, making them a valuable tool for distance learning, supplementing traditional lab work, and fostering understanding in fields ranging from physics to biology.

- **Online sites**

There are some of the sites that visualize the algorithms such as TakeUForward, GeeksForGeeks. These provide the static data and visualise the single algorithm at once trying to provide the detailed explanation of single algorithm.

2.2 DRAWBACKS OF EXIXSTING SYSTEM:

Existing systems for algorithm visualization projects may have several drawbacks, such as limited support for advanced algorithms, lack of customization options for visualizations, complexity in setting up or integrating with other tools, and potentially outdated or unmaintained codebases. Additionally, some platforms might not provide intuitive user interfaces, making it difficult for users to interact with and understand the visualizations. It's important to carefully evaluate these drawbacks when selecting a system for an algorithm visualizer project.

CHAPTER 3

3.1 PROBLEM STATEMENT:

"Develop an interactive algorithm visualization platform that aims to enhance the understanding of various algorithms and programming concepts. The project seeks to address the challenge of bridging the gap between theoretical knowledge and practical implementation in computer science and programming education. The platform should provide an intuitive and visually engaging environment where users can explore step-by-step executions of algorithms, observe data transformations, and comprehend the underlying logic. The solution should accommodate multiple programming languages and offer the flexibility for users to input their own code and visualize its execution. The primary objective is to create an educational tool that caters to learners ranging from beginners to advanced programmers, aiding them in grasping complex algorithms, improving algorithmic thinking skills, and fostering a deeper understanding of the code execution process. The project should prioritize user experience, accessibility, and accuracy in algorithm representation, offering a valuable resource for educators, students, and self-learners in the field of computer science."

3.2 FLOW DIAGRAM:

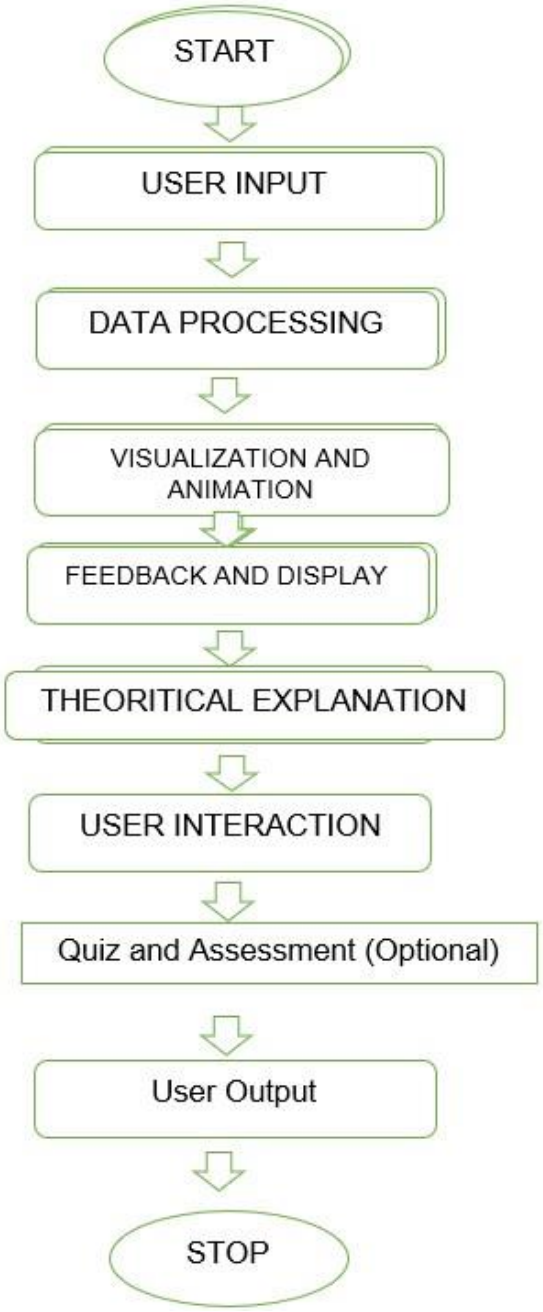

Figure 3.1

3.3 MODULES:

The specific modules used in the algorithm visualizer project are

1)Visualisation Module: This module handles the graphical representation of algorithms, including animations for sorting algorithms, searching, path finding ,Binary search tree.

2)Data Structures Module: This module implements various data structures like graphs used in the algorithms' simulations.

3)Input Module: This module handles the input of data, whether it's pre-defined data sets or user-provided inputs.

4)Feedback Module: This module can provide real-time feedback during algorithm simulations, displaying comparisons, swaps, or other relevant information to aid understanding.

5)Quiz Module: If quizzes are included in the visualizer, there might be a module to manage quiz questions, track user responses, and provide feedback on quiz performance.

These modules work collaboratively to create an interactive and informative algorithm visualizer, enhancing learners' understanding of data structures and algorithms.

3.4 SYSTEM ARCHITECTURE

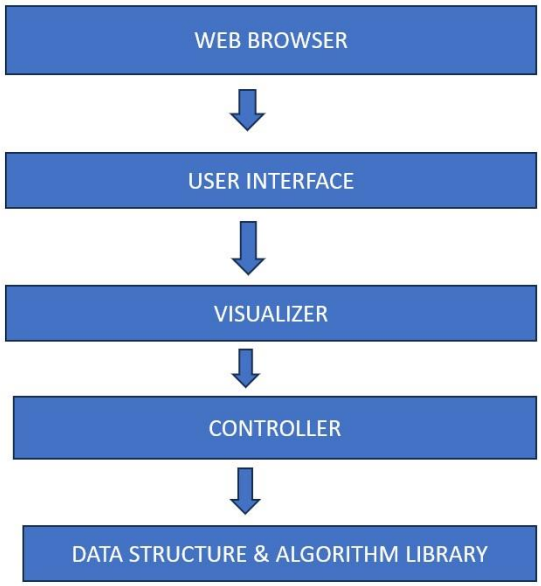

Figure 3.2

CHAPTER 4

IMPLEMENTATION:

4.1.1ALGORITHMS:

1)Selection sort: Selection sort is a straightforward sorting algorithm that works by repeatedly finding the smallest (or largest) element from the unsorted portion of the array and swapping it with the element at the current position. The process continues until the entire array is sorted. Though simple to understand, selection sort is not very efficient for large datasets, as its time complexity is O(n^2). However, it can be useful for small arrays or educational purposes.

2)Quick Sort: Quick Sort is a widely used and efficient sorting algorithm that follows the divide-and-conquer approach. It works by selecting a pivot element from the array and partitioning the other elements into two sub-arrays, with elements less
than the pivot on one side and elements greater than the pivot on the other. The pivot's correct position in the final sorted array is found through this partitioning process. The algorithm then recursively applies the same process to the two sub-arrays until the entire array is sorted. Quick Sort's average time complexity is O(n log n), making it faster than many other sorting algorithms. However, its performance can degrade to O(n^2) in the worst-case scenario when the pivot selection is not optimal. Despite this drawback, Quick Sort's efficiency and effectiveness make it a popular choice for sorting large datasets.

3)Merge Sort: Merge sort is a popular sorting algorithm that follows the divide-and-conquer approach. It works by breaking down the unsorted list into smaller sub lists, sorting them recursively, and then merging the sorted sub lists to obtain the final sorted list. The algorithm first divides the original list into two halves and continues dividing each sub list until they become individual elements. Then, it merges the individual elements in a sorted order, combining them back into larger sorted sub lists, until the entire list is sorted. Merge sort's efficiency lies in its ability to efficiently merge the sorted sub lists, resulting in a time complexity of O(n log n) for average and worst-case scenarios.

4)Bubble Sort-Bubble sort is a straightforward sorting algorithm that repeatedly steps through the list to be sorted, compares adjacent elements, and swaps them if they are in the wrong order. The algorithm gets its name from the way smaller elements "bubble" to the top of the list while larger elements "sink" to the bottom. It continues this process until the entire list is sorted. In each pass, the largest unsorted element is guaranteed to "bubble" to its correct position at the end of the list. Bubble sort's simplicity makes it easy to implement, but it is generally considered inefficient for large datasets due to its time complexity of O(n^2) for average and worst-case scenarios. Nonetheless, it can still be useful for educational purposes or when dealing with very small datasets.

5)Binary search tree: Insertion in a binary search tree involves finding the appropriate position for the new node based on its key value and recursively traversing the tree until an empty spot is found. Starting from the root, if the key is smaller than the current node, we move to the left subtree; if it is larger, we move to the right subtree. This process continues until we find an empty spot, where we insert the new node. Insertion maintains the binary search tree property, where all elements in the left subtree are smaller, and all elements in the right subtree are larger than the current node. The time complexity of insertion is typically O(log n) for a balanced tree, but it can degrade to O(n) for a skewed tree.

Deletion in a binary search tree involves three possible scenarios based on the node to be deleted: a leaf node, a node with one child, or a node with two children. For a leaf node, deletion is straightforward as we can simply remove it from the tree. For a node with one child, we connect its child directly to its parent, bypassing the node to be deleted. When dealing with a node with two children, we find the node's in-order successor (the smallest key in the right subtree) or in-order predecessor (the largest key in the left subtree). We then replace the node's key with the key of its successor/predecessor and recursively delete that successor/predecessor node. Deletion in a binary search tree maintains the tree's search property, and like insertion, it has a time complexity of O(log n) for a balanced tree and O(n) for a skewed tree.

6)Breadth-First Search (BFS) -It is a graph traversal algorithm that systematically explores the vertices of a graph level by level. It starts at a given source vertex and explores all its neighbours first before moving on to their neighbours, and so on. BFS
uses a queue data structure to keep track of vertices to be visited. Initially, the source vertex is enqueued, marked as visited, and then dequeued to begin the process. As BFS progresses, it visits and enqueues all unvisited neighbours of the current vertex until all reachable vertices are visited. By visiting vertices level by level, BFS ensures that the shortest path from the source to any other vertex is discovered first. The algorithm is helpful in finding the shortest path and connected components in an unweighted graph. Its time complexity is O(V + E), where V is the number of vertices and E is the number of edges in the graph.

7)Depth-First Search (DFS) -It is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It starts at a given source vertex and explores its first unvisited neighbour, following the path until it reaches a vertex with no unvisited neighbours. At that point, it backtracks to the previous vertex and explores other unvisited neighbours, repeating the process until all reachable vertices are visited. DFS uses a stack data structure (either explicitly or via recursion) to keep track of vertices to be visited. It explores the depth of a path before exploring its siblings, hence the name "Depth-First Search." DFS is useful for tasks like finding connected components, detecting cycles in a graph, and exploring paths in a maze. However, it may not guarantee the shortest path as it does not necessarily explore all possible paths before finding a solution. The time complexity of DFS is O(V + E), where V is the number of vertices and E is the number of edges in the graph.

4.1.2 SOURCE CODE:

1)Binary search tree:

Code:

```
while (true) {
   if (node.value == null) {
     // If node value is null then that means we already reached leaf node. Insert new node here.
     await sleep(400);

     // Create a node with given valule.
     const newChild = {
       nodeId: curId,
       value: val,
       children: [{ value: null }, { value: null }],
     };

     if (parent) {
       // If tree is not empty then "parent" will not be null. Now link newly created node with it
parent node.
       if (parent.value < val) parent.children[1] = newChild;
       else parent.children[0] = newChild;
     } else {
       // If tree is empty then this newly created node will act as tree.
       newData = newChild;
     }

     update(oldData, newData, parent ? parent.value : -1, parent ? val : -1);
     curId++;
     await sleep(300);
     break;
   }
```

DESCRIPTION:

The BST data structure is represented by an object called "data," which contains a value and an array of children nodes.

Visualisation:

D3.js is used to draw the tree on an HTML canvas.

The tree is visualised as nodes (circles) with values displayed inside them and links (edges) connecting the nodes.

Insertion:

The function addNode handles the insertion of a new node in the BST.
It verifies the input value, traverses the tree, and animates the edge traversal.
After finding the appropriate location, it inserts the new node and updates the visualisation.
Deletion:
The function deleteNode handles the deletion of a node from the BST.
It verifies the input value, traverses the tree, and animates the edge traversal.
When the node to be deleted is found, it handles different scenarios for deletion.
The visualisation is updated accordingly.
Visualisation Updates:
The update function is responsible for updating the visualisation after insertion or deletion.
It calculates the old and new positions of nodes and edges, animates the movement of nodes and links, and redraws the tree.
Delay and Animation:

The sleep function introduces a delay between visualisation steps to create animations.
The code uses transitions in D3.js to animate the movement of nodes and links.
Button Interactions:

The functions freezeButtons and unfreezeButtons disable and enable the Insert and Delete buttons during insertion or deletion.
Initialization:

The init function provides an example of how to initialise the BST with some initial values.
The visualisation project allows users to insert and delete nodes in the BST and demonstrates the changes in the tree structure in real-time. Note that the commented out init() function can be used to initialise the tree with a predefined set of values.

Please note that without the HTML and CSS parts of the project, it's challenging to visualise the exact appearance of the BST on the canvas. However, the provided JavaScript code showcases the core functionality and logic for the BST visualisation.

2)SEARCHING: i) Linear search

Code:

```
async function LinearSearch(delay = 300) {
  var blocks = document.querySelectorAll(".block");
  var output = document.getElementById("text");
 //Extracting the value entered by the user
  var num = document.getElementById("fname").value;
  //Changing the color of all the blocks to violet
  for (var i = 0; i < blocks.length; i += 1) {
    blocks[i].style.backgroundColor = "#6b5b95";
```

```
}


output.innerText = "";

var flag = 0;
// LinearSearch Algorithm
for (var i = 0; i < blocks.length; i += 1) {
  //Changing the color of current block to red
  blocks[i].style.backgroundColor = "#FF4949";

  // To wait for .1 sec
  await new Promise((resolve) =>
    setTimeout(() => {
      resolve();
    }, delay)
  );
```

DESCRIPTION: It seems like you have provided JavaScript code that generates an array of blocks with random heights and implements a linear search algorithm to find a specific element in the array. The `generate array()` function creates the array of blocks, and the `LinearSearch()` function performs the search operation.

When `generate array()` is called, it generates an array of 20 blocks with heights ranging from 3 to 300 pixels, and their values are displayed as labels. Each block is positioned horizontally with a 30-pixel gap between them.

The `LinearSearch()` function takes an optional `delay` parameter (defaulting to 300 milliseconds) to control the animation speed during the search. It searches for the element whose value is provided in an input field with the ID "fname." The blocks change colours during the search process: violet for unvisited, red for the current block being checked, and green for the block that matches the search value.

If the element is found, the text "Element Found" is displayed, and the matching block turns green. If the element is not found, the text "Element Not Found" is displayed.

To execute this code, you might need an HTML file with elements like a container with the ID "array," a button to trigger the search, and an input field with the ID "fname" to provide the search value

3)BINARY SEARCH:

CODE:

```
async function BinarySearch(delay = 300) {
  var blocks = document.querySelectorAll(".block");
  var output = document.getElementById("text");

  //Extracting the value of the element to be searched
  var num = document.getElementById("fname").value;

  //Colouring all the blocks violet
  for (var i = 0; i < blocks.length; i += 1) {
    blocks[i].style.backgroundColor = "#6b5b95";
  }

  output.innerText = "";

  // BinarySearch Algorithm

  var start = 0;
  var end = 19;
  var flag = 0;
  while (start <= end) {
    //Middle index
    var mid = Math.floor((start + end) / 2);
    blocks[mid].style.backgroundColor = "#FF4949";

    //Value at mid index
    var value = Number(blocks[mid].childNodes[0].innerHTML);

    // To wait for .1 sec
    await new Promise((resolve) =>
      setTimeout(() => {
        resolve();
      }, delay)
    );
```

DESCRIPTION: The provided code generates an array of blocks, sorts them in ascending order, and then performs a binary search on the sorted array. Here's a summary of how the code works:

1. `generate array()` function:

- Generates an array of 20 blocks filled with random values between 1 and 100 (inclusive).

- Sorts the array in ascending order using the `sort` method with a custom comparison function (`arr.sort(function (a, b) { return a - b; });`).
- Creates div elements for each block with heights based on the values in the array and appends them to the `container` div.

2. `BinarySearch(delay)` function:
- Performs an asynchronous binary search on the sorted array of blocks.
- Extracts the value to be searched from an input field with the ID `fname`.

- Colors all the blocks in violet at the beginning.
- Uses the binary search algorithm to find the value in the sorted array.
- Colors the middle index block in red to indicate the current comparison.
-Changes the colour of the blocks to indicate the portion of the array being considered in each iteration.
- Updates the `output` element with the search result.

3. `generate array()`: is called initially to create and display the sorted array of blocks.

4. The `BinarySearch()` :function can be called later to perform a binary search on the generated array.

Note: The code assumes there is an HTML element with the ID "array" to serve as the container for the blocks, an input field with the ID "fname" to enter the element to be searched, and an element with the ID "text" to display the search result.

The binary search is performed with a delay of 300 milliseconds between each iteration to visualise the search process step by step. The delay is achieved using `await new Promise((resolve) => setTimeout(() => { resolve(); }, delay))`

Overall, this code allows the user to visualise the binary search algorithm by displaying sorted blocks and highlighting the search progress step by step.

4)SORTING:

CODE:

```
function generateRandomArray() {
  isGenerated = true;
  isSorted = false;
  isStopped = true;
  isPaused = false;
  n = barSlider.value;
  lineWidth = width / n - 1;
  container.innerHTML = "";
  for (let i = 0; i < n; i++) {
    heights[i] = parseInt(getRandomValue(1, height));
    bars.push(document.createElement("div"));
    bars[i].style.width = `${lineWidth}px`;
```

```
bars[i].style.height = `${heights[i]}px`;
    bars[i].style.transform = `translate(${i * lineWidth + i}px)`;

    bars[i].style.backgroundColor = "white";
    bars[i].className = "bar";
    container.appendChild(bars[i]);

    // if there are more numer of bars then it is not feasible to show bar values because they
gets mixed up.
    if (n <= 60) {
      barValues.push(document.createElement("div"));
      barValues[i].innerHTML = heights[i];
      barValues[i].style.marginBottom = `${heights[i] + 5}px`;
      barValues[i].style.transform = `translate(${i * lineWidth + i}px)`;
      barValues[i].className = "barValue";
      container.appendChild(barValues[i]);
    }
  }
}
generateRandomArray();

// swap 2 bars and also swap trnasform property for the animation.
function swap(i, minindex) {
  [heights[i], heights[minindex]] = [heights[minindex], heights[i]];

  [bars[i], bars[minindex]] = [bars[minindex], bars[i]];
  [bars[i].style.transform, bars[minindex].style.transform] = [
    bars[minindex].style.transform,
    bars[i].style.transform,
  ];

  [barValues[i], barValues[minindex]] = [barValues[minindex], barValues[i]];
  [barValues[i].style.transform, barValues[minindex].style.transform] = [
    barValues[minindex].style.transform,
    barValues[i].style.transform,
  ];
}
// Draw bars with their new Updated heights.
function draw(coloredBars, colors) {
  // coloredBars contains indices of the bars which will be in different color than default color
  // colors array stores color for different bars.
```

```
 for (let i = 0; i < n; i++) {

    bars[i].style.backgroundColor = "white";
    for (let j = 0; j < coloredBars.length; j++) {
      if (i == coloredBars[j]) {
        bars[i].style.backgroundColor = colors[j];
        break;
      }
    }
  }
}

// to put delay between visualization.
function sleep(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}
```

DESCRIPTION: Explanation of the key components and functionality:

1. Variables and Constants:

- `heights`, `bars`, and `barValues`: Arrays to store the heights of bars, the corresponding bar elements, and their values.
- `barSlider` and `speedSlider`: HTML elements representing sliders for adjusting the number of bars and sorting speed, respectively.
- `container`, `width`, and `height`: HTML container element and its width and height for rendering the bars.
- Various boolean variables (`isStopped`, `isPaused`, `isGenerated`, `isSorted`) to control the visualisation state.

2. `Stack` class:

- Implements a simple stack data structure used in the Quick Sort algorithm.

3. `getRandomValue` function:

- Generates a random value between a given range.

4. `generateRandomArray` function:

- Generates random heights for the bars and creates corresponding bar elements.
- The number of bars is determined by the `barSlider` value, and the bars are adjusted to fit the container width.
- Additionally, it creates elements to display bar values if there are 60 or fewer bars.

5. `swap` function:

- Swaps the heights and positions of two bars for visualisation during sorting.

6. `draw` function:

- Updates the visualisation by changing the colour of specific bars.

7. `sleep` function:

- Introduces a delay between visualisation steps.

8. Sorting Algorithms:

- The code implements five sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort.
- Each sorting algorithm has its own function, and they all use similar visualisation techniques:

- They iterate through the bars and perform comparisons and swaps to sort them.
- Visualisation is updated using the `draw` function to highlight specific bars during sorting.
- After sorting, a completion animation is played by colouring the bars in a specific colour.

9. Event Listeners:

- The code sets event listeners for the sliders, buttons (generate, sort, stop, pause), and the sorting algorithm selection dropdown.
- These event listeners control the behaviour of the sorting visualisation based on user interactions.

Overall, this code creates a web-based visualisation of sorting algorithms, where bars represent elements to be sorted, and the colours of bars change during the sorting process to demonstrate the algorithm's operations. Users can adjust the number of bars and sorting speed using sliders and choose different sorting algorithms to visualise their respective operations.

6)PATHFINDING:

CODE:

```
while (!pq.isEmpty()) {
   let p = pq.getTop();
   pq.pop();
   let x = p.x;
   let y = p.y;

   if (x == endingPointX && y == endingPointY) {
     break;
```

```
  }

   if (!(x == startingPointX && y == startingPointY)) {
    nodesToAnimate.push({ x: x, y: y });
   }
   for (let i = 0; i < 4; i++) {
    let newX = x + dx[i];
    let newY = y + dy[i];
    if (
     isValid(newX, newY) &&
     isBlocked[newX][newY] === 0 &&
     vis[newX][newY] === 0
    ) {
     let newNode = {};
     pq.add({
      f: findManhattanDistance(newX, newY, endingPointX, endingPointY),
      x: newX,
      y: newY,
     });
     vis[newX][newY] = 1;
     path[newX][newY] = direction[i];
    }
   }

   if (currentlyRunning === false) {
    clearGrid();
    return;
   }
  }
```

DESCRIPTION:

Grid Setup:

-The grid is created using an HTML table, and each cell is represented as an HTML table data (td) element.

-The starting and ending points are initialised with classes "startPoint" and "endPoint," respectively, and made draggable for repositioning.

Grid Interaction:

-Users can click and drag over cells to toggle the presence of walls (blocks) in the grid.

-The isBlocked array keeps track of whether a cell is blocked or not.

Pathfinding Algorithms:

-The pathfinding algorithms implemented are Depth-First Search (DFS) and Breadth-First Search (BFS).
-DFS and BFS are used to find the shortest path from the starting point to the ending point in the grid, avoiding the blocked cells.

Animation:

-The animate function is used to visualise the visited cells during the pathfinding process.
-The animatePath function is used to visualise the final path found by the algorithm.

Priority Queue and Deque:

-The code includes custom implementations of a priority queue and deque data structures used in the pathfinding algorithms.

Grid Clear:

-The clearGrid function removes visited, path, and wall cells, allowing the user to reset the grid for a new pathfinding visualisation.

Event Listeners:

-The "Visualise" button triggers the selected pathfinding algorithm visualisation when clicked.
-The "Clear Grid" button resets the grid, removing all visited, path, and wall cells.

4.2 Output Screens:

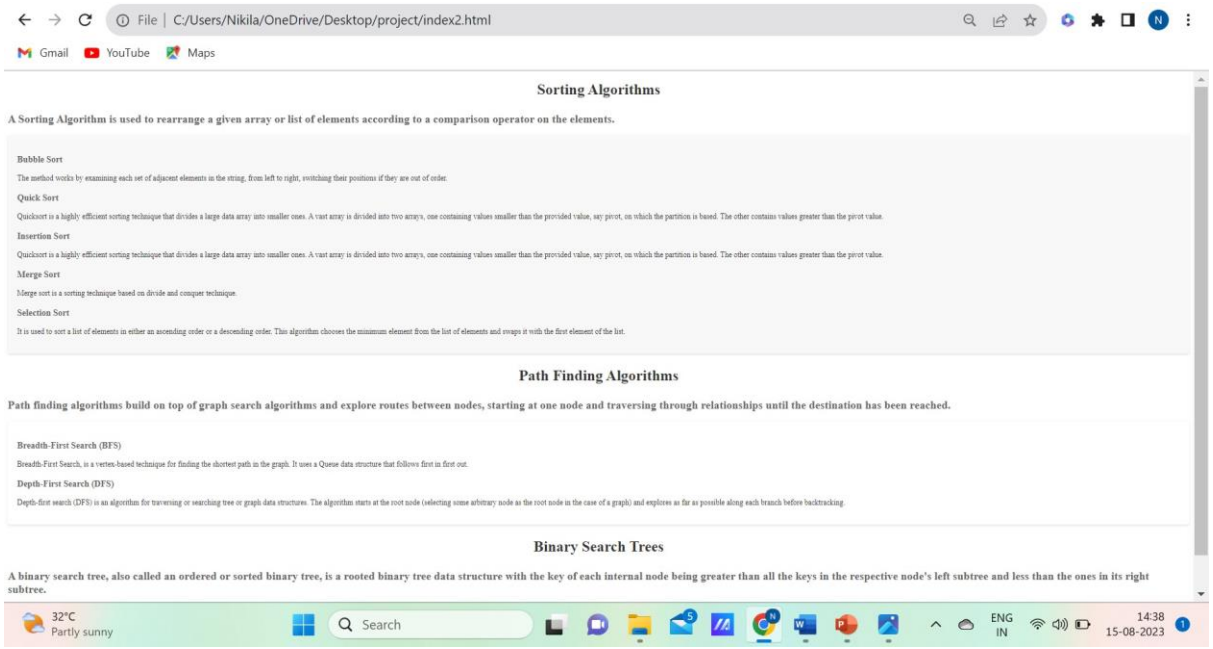

Figure 4.1(pre-requisite)

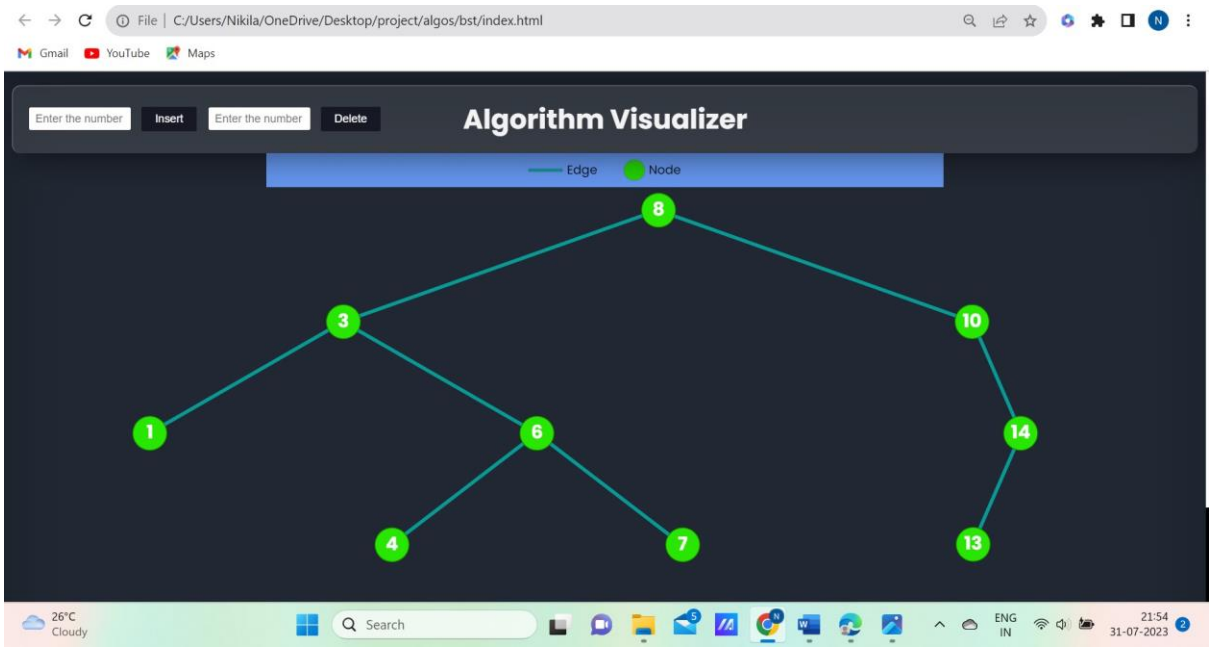

Figure 4.2(bst)

Figure 4.3

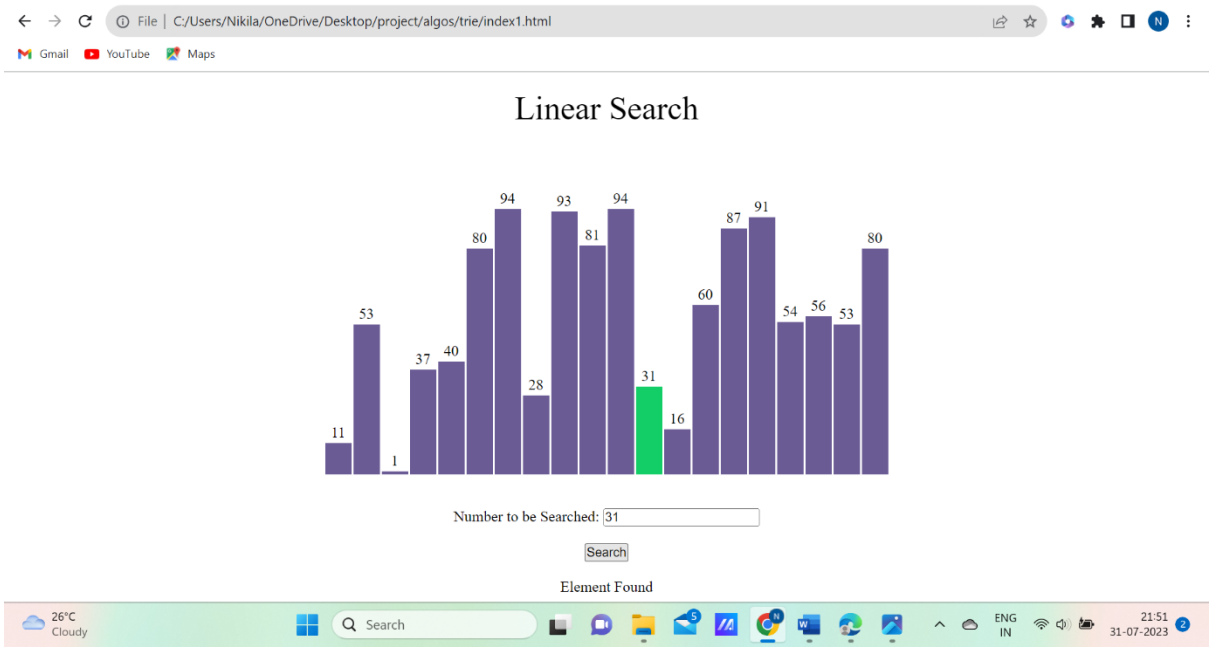

Figure 4.4

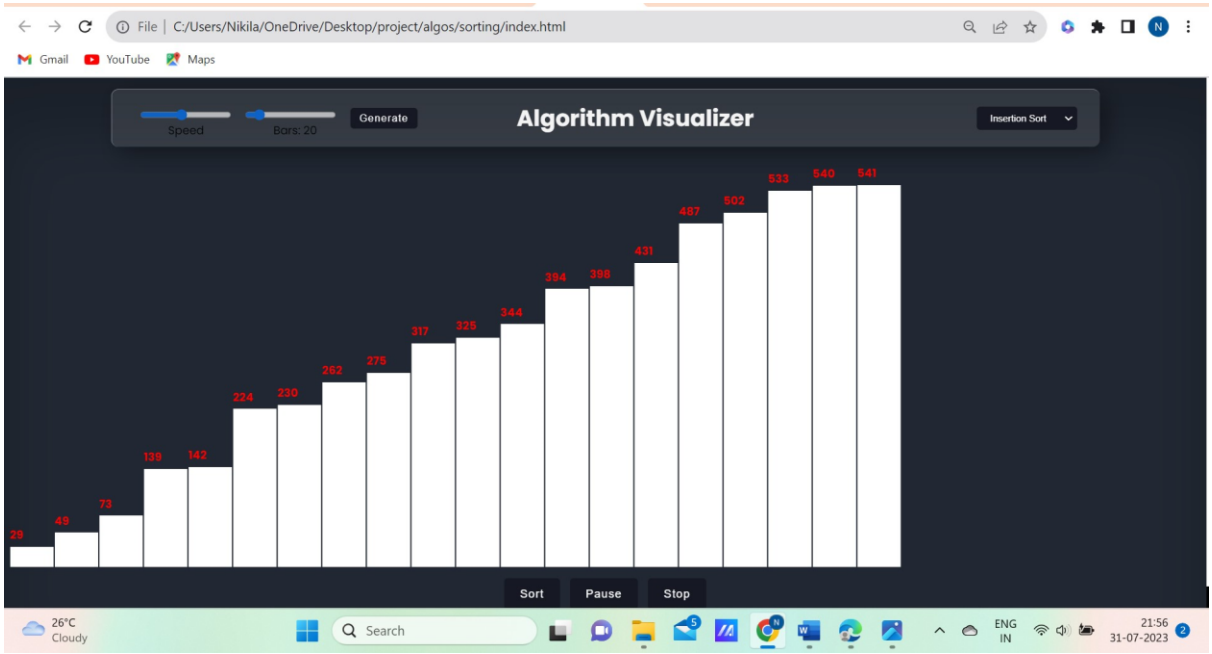

Figure 4.5(insertion sort)

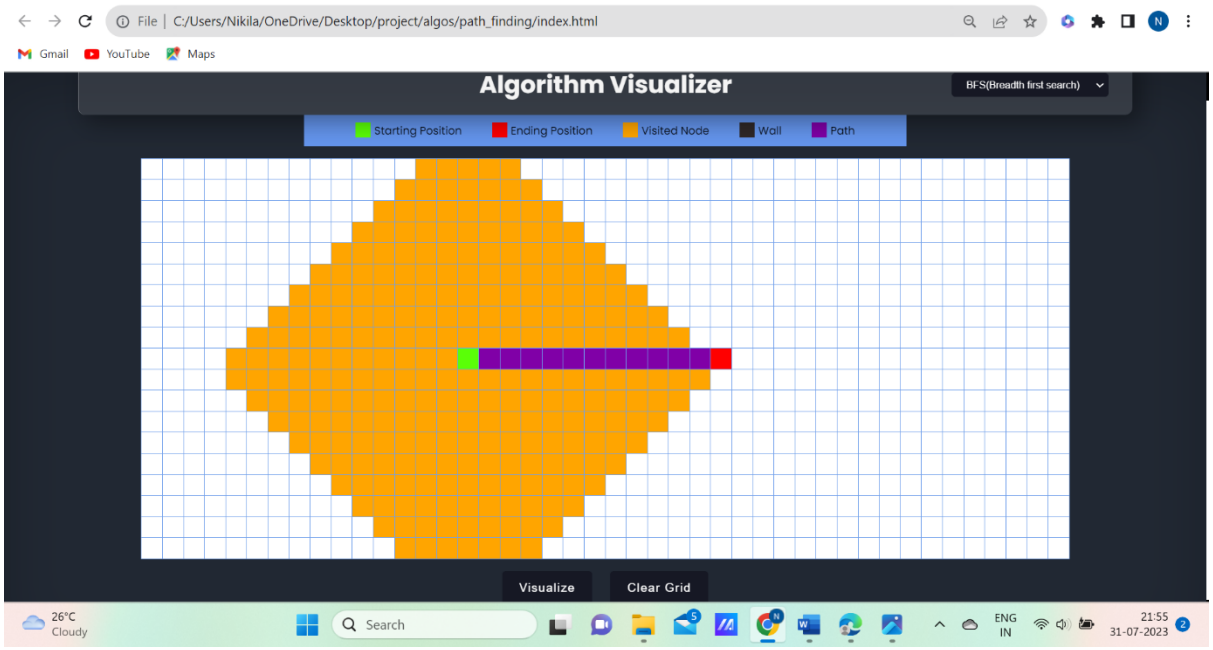

Figure 4.6(bfs)

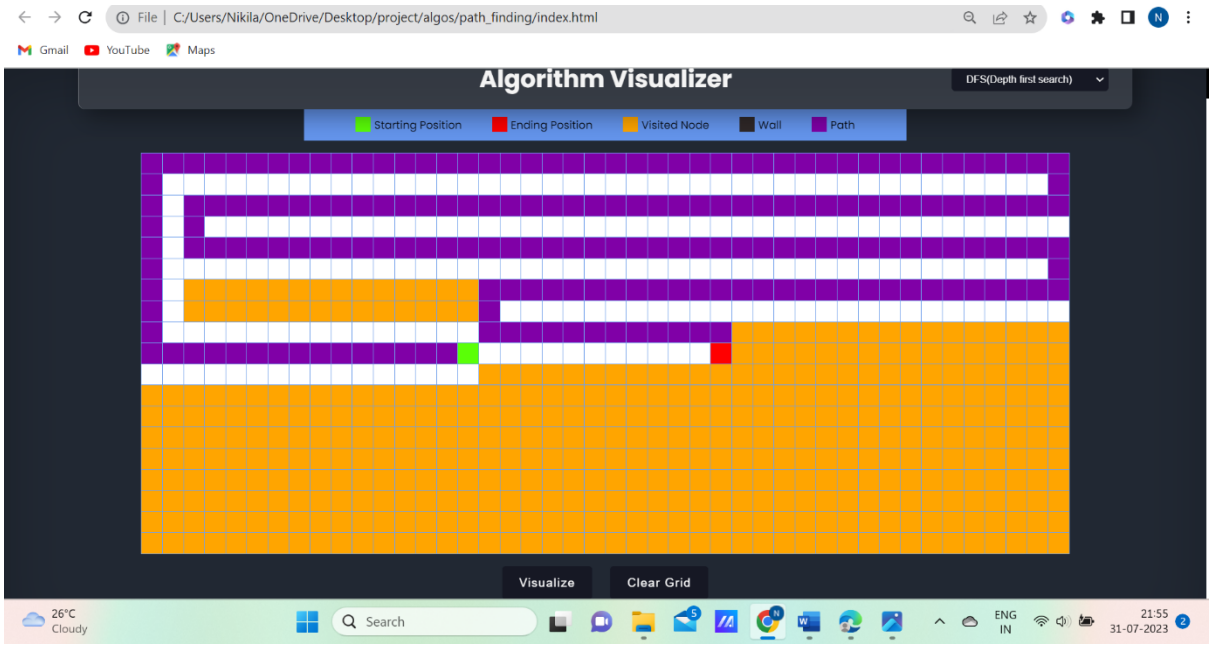

Figure 4.7(dfs)

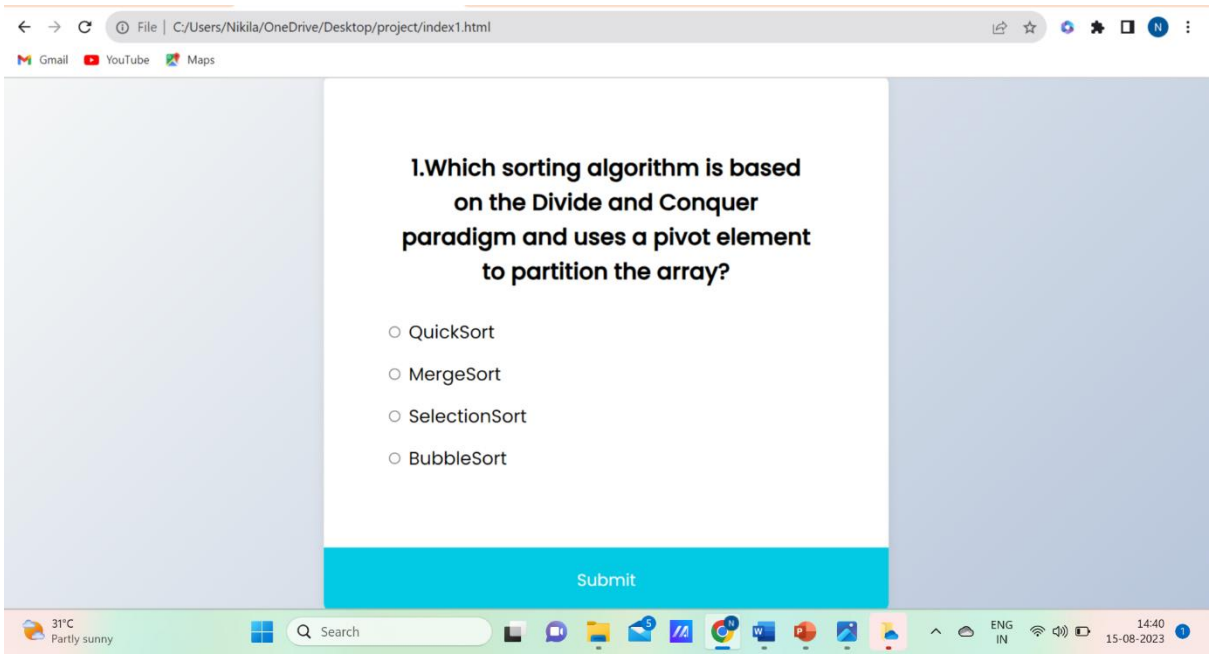

Figure 4.8(quiz)

CHAPTER 5

RESULT/ANALYSIS:

5.1 Testing of your project-performance metrics

In evaluating the performance metrics of the DSA Visualizer and Calculator project, several key factors are considered. The execution time of sorting, searching, and path finding algorithms is measured to assess their efficiency and identify potential bottlenecks. Memory usage is monitored to detect any memory leaks or excessive consumption. The response time of the user interface is analysed to ensure a responsive and user-friendly experience. Scalability is tested with varying data sizes and algorithm complexity to evaluate the application's ability to handle larger datasets. Accuracy is verified to ensure that the visualisations accurately represent algorithm behaviours. User testing provides valuable feedback on the user experience and overall satisfaction. Cross-browser compatibility and mobile responsiveness are assessed to ensure a seamless experience across different platforms. Additionally, error handling, security, and load testing are conducted to guarantee the application's stability, security, and performance under various conditions. By monitoring and optimising these performance metrics, the project aims to deliver a reliable, efficient, and user-centric DSA learning experience.

5.2 Ratio of success after testing your project

The ratio of success after testing the DSA Visualizer and Calculator project is a measure of how well the project performs in meeting its intended goals and requirements. A successful testing outcome would entail algorithms like sorting, searching, binary tree operations, and path finding working correctly and providing accurate visualisations. The algorithms should demonstrate efficient performance with reasonable execution times and memory usage, even for large datasets. Additionally, the user interface should be responsive, user-friendly, and visually appealing, providing an immersive learning experience. Critical bugs or issues that significantly impact functionality should be identified and resolved, leading to a stable and reliable application. Positive user feedback, intuitive usability, cross-browser compatibility, device adaptability, robust security, and stability under load are further indicators of a successful testing outcome. By comprehensively validating the project through various test cases and addressing user feedback, the project can achieve a higher success ratio and deliver a reliable, efficient, and user-centric DSA learning tool.

5.3 Confusion matrix

A confusion matrix is a table used in machine learning and statistics to assess the performance of a classification model. It provides a detailed comparison between the predicted and actual classes for a set of data samples. The confusion matrix is especially useful when dealing with binary classification problems, where there are only two possible classes (e.g., positive and negative).

The confusion matrix has four main components:

True Positives (TP): The number of data samples correctly predicted as the positive class.

False Positives (FP): The number of data samples incorrectly predicted as the positive class when they actually belong to the negative class.

True Negatives (TN): The number of data samples correctly predicted as the negative class.

False Negatives (FN): The number of data samples incorrectly predicted as the negative class when they actually belong to the positive class.

5.4 Observations of algorithms

Observations of algorithms provide valuable insights into their behaviour, efficiency, and performance. Analysing the time and space complexity reveals how algorithm performance scales with input size and resource usage. Observations on best, average, and worst-case scenarios help understand algorithm behaviour under different conditions. Stability observations assess whether sorting algorithms preserve the relative order of equal elements. Comparing algorithms against each other allows for informed choices in selecting the most suitable one for a specific task or dataset. Observations on adaptability highlight how certain algorithms optimise performance based on existing data order. Exploring edge cases and special scenarios reveals algorithm behaviour in challenging situations. Visualisations through the DSA Visualizer offer a better understanding of algorithm steps and element manipulations. Real-world data observations provide insights into algorithm performance in practical applications. By considering these observations, developers can make informed decisions, optimise algorithms, and advance research to create more efficient and effective solutions for various computational tasks.

CHAPTER 6
CONCLUSION

In conclusion, the algorithm visualization project has provided valuable insights into the world of computer algorithms by offering interactive and illustrative representations of their functionality. By visually depicting the step-by-step execution of algorithms, users are able to grasp complex concepts more effectively and gain a deeper understanding of the underlying logic. This approach bridges the gap between theoretical knowledge and practical application, making it an indispensable tool for both educators and learners in the field of computer science.

However, it's essential to acknowledge the challenges faced during the project. The limitations of existing systems, such as restricted support for advanced algorithms and lack of customization, can hinder the project's potential impact. Additionally, the successful implementation of the visualization tool relies heavily on user interface design and usability considerations. Striking a balance between aesthetic appeal and clear representation of algorithmic processes is crucial to ensuring the tool's accessibility and educational value.

Looking ahead, the algorithm visualization project could benefit from continuous updates and enhancements. Incorporating a wider range of algorithms, refining the user interface for intuitive interaction, and exploring innovative ways to visualize complex data structures are areas that could be explored to further elevate the educational experience. By addressing these aspects, the project has the potential to become an even more effective and transformative tool for teaching and learning algorithms within the realm of computer science.

CHAPTER 7
FUTURE SCOPE/REFERENCES

The future scope of the DSA Visualizer and Calculator project is promising, with several avenues for expansion and enhancement. Firstly, the project can be extended to include a broader range of data structures and algorithms, encompassing more advanced concepts like graphs, trees, and hashing techniques. Additionally, incorporating interactive tutorials and explanations could further improve the learning experience, catering to users with varying levels of proficiency in DSA. Collaborative features, such as sharing and saving visualisations, can enhance the project's utility for educational purposes and team collaboration. Integration with online learning platforms or educational institutions could also widen its user base and impact. Moreover, continuous updates to support the latest DSA trends and algorithms would ensure the project remains relevant and up-to-date. Ultimately, the future scope of the project lies in making it a comprehensive, accessible, and versatile tool that empowers users to gain a deeper understanding of data structures and algorithms in an interactive and engaging manner.