Sorting and Searching Visualizer Report

Introduction

Background

Data structures are fundamental components in computer science, crucial for efficiently storing, organizing, and manipulating data. Understanding data structures is essential for solving complex computational problems. Sorting and searching algorithms, in particular, are foundational techniques that are widely used in various applications, from databases to artificial intelligence.

Objective

The objective of this project is to create an interactive visualizer for sorting and searching algorithms using HTML, CSS, and JavaScript. This visualizer aims to provide a clear and intuitive understanding of how these algorithms work, step by step. By visualizing the internal processes of sorting and searching algorithms, users can gain deeper insights into their functionality and performance.

Significance

This visualizer serves as a valuable educational tool for students, educators, and professionals. It helps in demystifying the complex operations involved in sorting and searching, making it easier to grasp key concepts. By enhancing the learning experience, this project contributes to better comprehension and application of data structures and algorithms in real-world scenarios.

Problem Definition and Requirements

Problem Definition

Understanding the behavior and performance of sorting and searching algorithms can be challenging, especially when dealing with large datasets or complex algorithms. Traditional methods of learning through textbooks or static diagrams may not effectively convey the dynamic nature of these processes. There is a need for an interactive tool that can visually demonstrate the step-by-step execution of sorting and searching algorithms, allowing users to see the immediate impact of each operation.

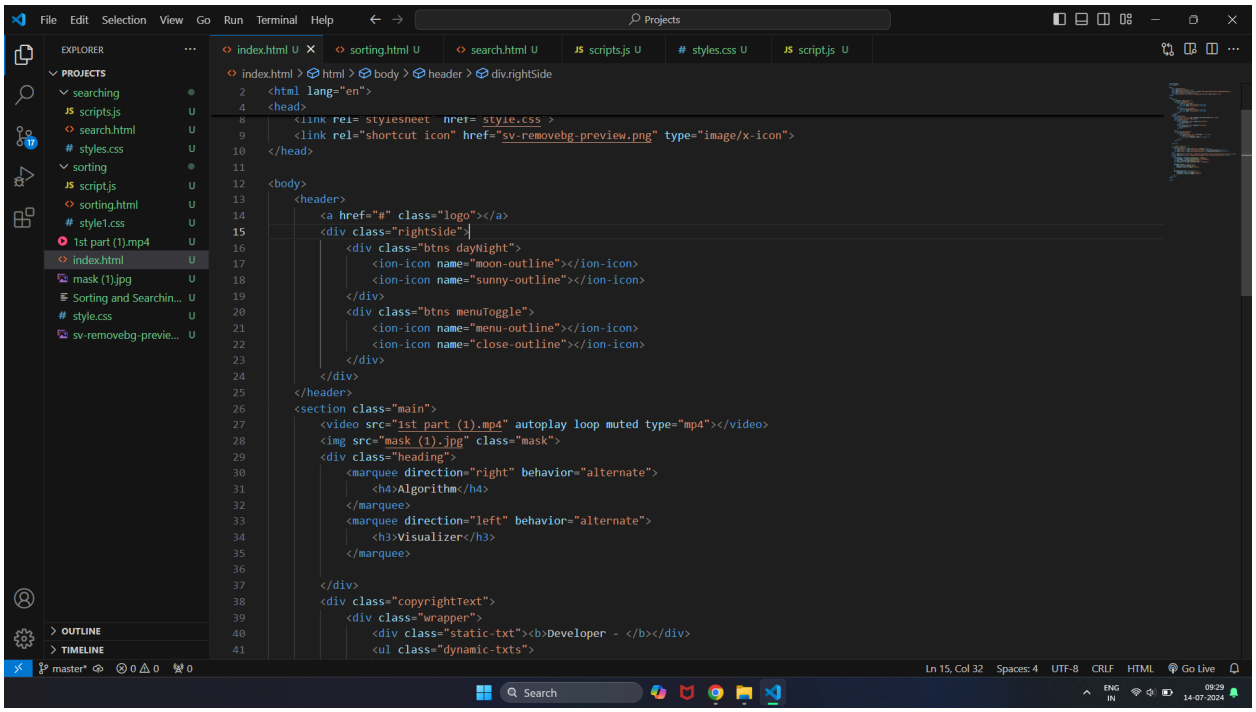
Requirements

1. **User Interface**:
    - A clean and intuitive interface to select different sorting and searching algorithms.
    - Controls to start, pause, and reset the visualization.
    - Options to adjust the speed of the visualization and input data size.
2. **Sorting Algorithms**:
    - Implementation of various sorting algorithms such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort.
    - Visual representation of each step in the sorting process.
3. **Searching Algorithms**:
    - Implementation of common searching algorithms such as Linear Search and Binary Search.
    - Visual representation of the search process, highlighting comparisons and the current search range.
4. **Data Handling**:
    - Capability to generate random data sets of varying sizes.
    - Option for users to input custom data sets for visualization.
5. **Performance Metrics**:
    - Display of performance metrics such as the number of comparisons, swaps, and execution time for each algorithm.
6. **Responsiveness**:
    - Ensure the visualizer is responsive and works seamlessly across different devices and screen sizes.
7. **Code Documentation and Usability**:
    - Well-documented code for ease of understanding and future modifications.
    - Instructions and guidelines for users to effectively interact with the visualizer.

This project, by meeting the outlined requirements, aims to bridge the gap between theoretical knowledge and practical understanding of sorting and searching algorithms, making the learning process more engaging and effective.

Code and Website Preview:


Algorithm Visualizer
127.0.0.1:5500/index.html
Developer - ABHIJEET SINGH
Algorithm
Visualizer


File Edit Selection View Go Run Terminal Help
Projects
EXPLORER
PROJECTS
searching
scripts.js
search.html
styles.css
sorting
script.js
sorting.html
style1.css
1st part (1).mp4
index.html
mask (1).jpg
Sorting and Searchin...
style.css
sv-removebg-previe...
OUTLINE
TIMELINE
index.html U
sorting.html U
search.html U
scripts.js U
styles.css U
script.js U
index.html > html > body > header > div.rightSide
<html lang="en">
<head>
<link rel="stylesheet" href="style.css">
<link rel="shortcut icon" href="sv-removebg-preview.png" type="image/x-icon">
</head>
<body>
<header>

<ion-icon name="moon-outline"></ion-icon>
<ion-icon name="sunny-outline"></ion-icon>

<ion-icon name="menu-outline"></ion-icon>
<ion-icon name="close-outline"></ion-icon>

</header>
<section class="main">
<video src="1st part (1).mp4" autoplay loop muted type="mp4"></video>
<img src="mask (1).jpg" class="mask">

<marquee direction="right" behavior="alternate">
<h4>Algorithm</h4>
</marquee>
<marquee direction="left" behavior="alternate">
<h3>Visualizer</h3>
</marquee>

Developer -
<ul class="dynamic-txts">
master*
Ln 15, Col 32
Spaces: 4
UTF-8
CRLF
HTML
Go Live
Search
ENG
IN
09:29
14-07-2024

Search Visualizer

Enter size of array: 20 Generate Array
Enter value to search: 20 Linear Search Binary Search

0	6	16	20	31	32	36	54	54	59	75	75	82	85	89	89	95	96	96	97

Value 20 found at index 3

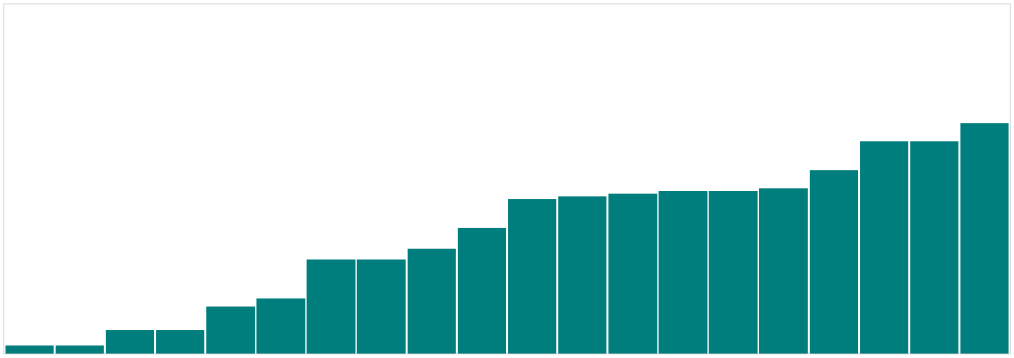
Algorithm Code

```
// C++ implemntation for Binary Search
        int binarySearch(const vector<int>& arr, int value) {
    int left = 0;
    int right = arr.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;  // Calculate the mid index to avoid overflow

        if (arr[mid] == value) {
            return mid;  // Return the index where the value is found
        } else if (arr[mid] < value) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;  // Return -1 if the value is not found
}
```

Time Complexity

Time Complexity: O(log n)

Array Size: 20 Generate Array Quick Sort Start Sorting



C++ code

```
// C++ implementation of QuickSort
// Function to swap two elements
void swap(int* a, int* b) {
```