CHAPTER – 1

Introduction

One of the most vital parts of computers and programming is algorithms. Algorithms can be used/followed by programmers for consistency in problem solving. There are various types of sorting algorithms with their distinct implementations and strategies. But it can be quite hard to comprehend these algorithms. The main goal of the project is to create a program which would serve as a tool for understanding how the most known sorting algorithms work.

The algorithm that are used in the proposed system to visualize are Bubble sort, Quick sort, Insertion sort, Selection sort, Merge sort. Sorting visualizer will be built using Html, CSS and JavaScript. Html and CSS will be used to create the frontend. JavaScript will be used to provide functionality. Depending on the user choice, the data will be run on the corresponding sorting algorithm.

1.1 Motivation

The project required us to use web programming and we were taught various sorting algorithms in class. But we had never applied it anywhere other than our lab programs. Hence we decided combine sorting algorithms with web programming and visualize certain sorting algorithms for a better and clear understanding of their working.

1.2 Problem Statement:

To create a web application using HTML, CSS and JavaScript that visualizes sorting algorithms and allows users to modify array size and sorting speed.

CHAPTER-2

Literature Survey

2.1 Existing System

Programs and algorithms are taught in schools/colleges theoretically. In most cases it is hard to understand exactly how an algorithm works. Students tend to learn the algorithms by heart. Thus there is a need for better understanding of these algorithms by some means.

2.2 Proposed system

The user can generate a new array, select its size and also adjust the sorting speed, following which he/she can click on any of the given sorting algorithms to visualize it. The visualization is done with the help of CSS divs (bars). At any point the divs that are being used/checked/worked on are clearly highlighted using different colours and after a div has been sorted into its correct position it is denoted by the green colour.

During the visualization the user can change the speed but the rest of the buttons will be disabled until the visualization is complete.

In this way, Sorting Visualizer provides a clear and better understanding of various sorting algorithms.

CHAPTER-3

System Requirement

3.1 Hardware Requirements

- Processor : Intel i5 processor or AMD ryzen 5 processor.
- Speed : 1.2 GHz.
- RAM Size : 8 GB.

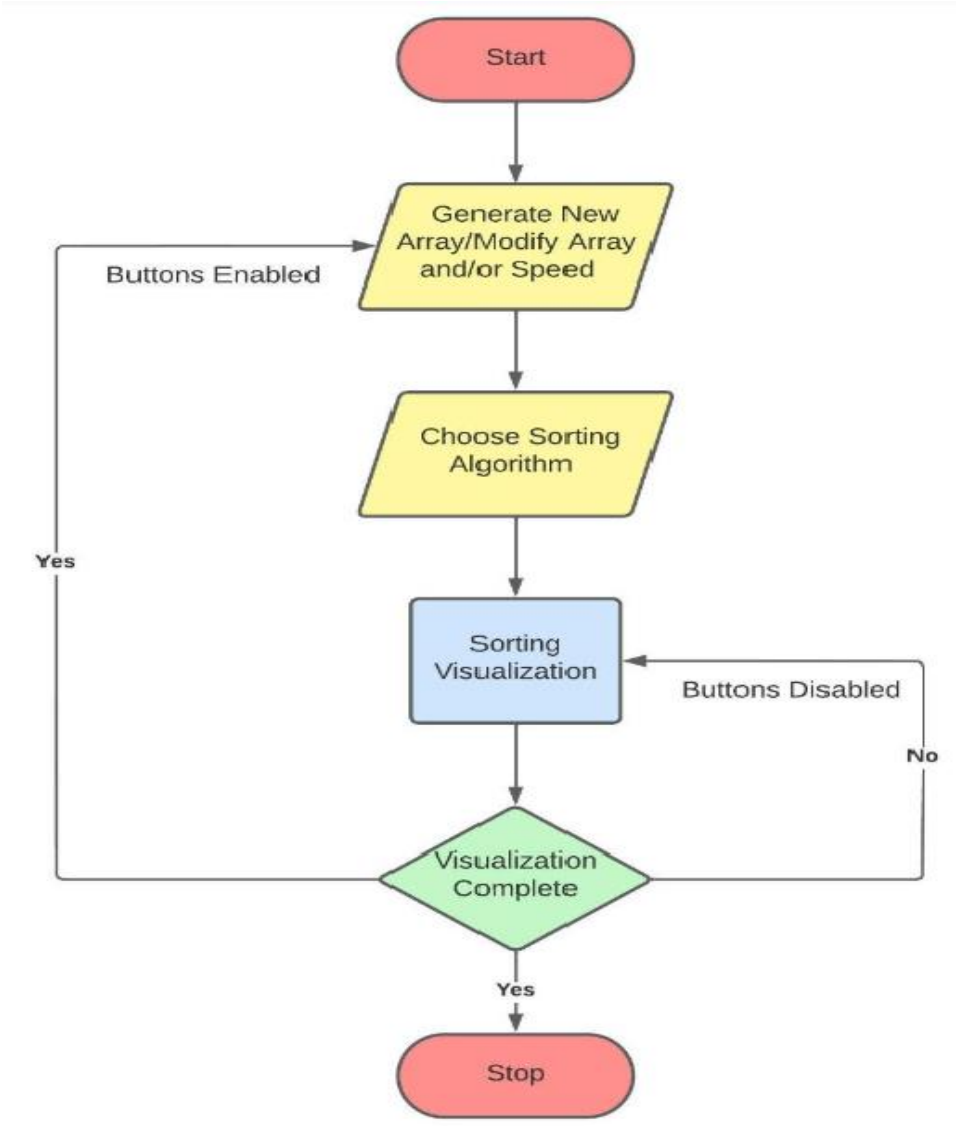
3.2 Software Requirements

- Operating system : Windows 10 or mac os.
- Tools used : Visual Studio code and prompt.
- Developing language used : html, CSS, java script language**.**

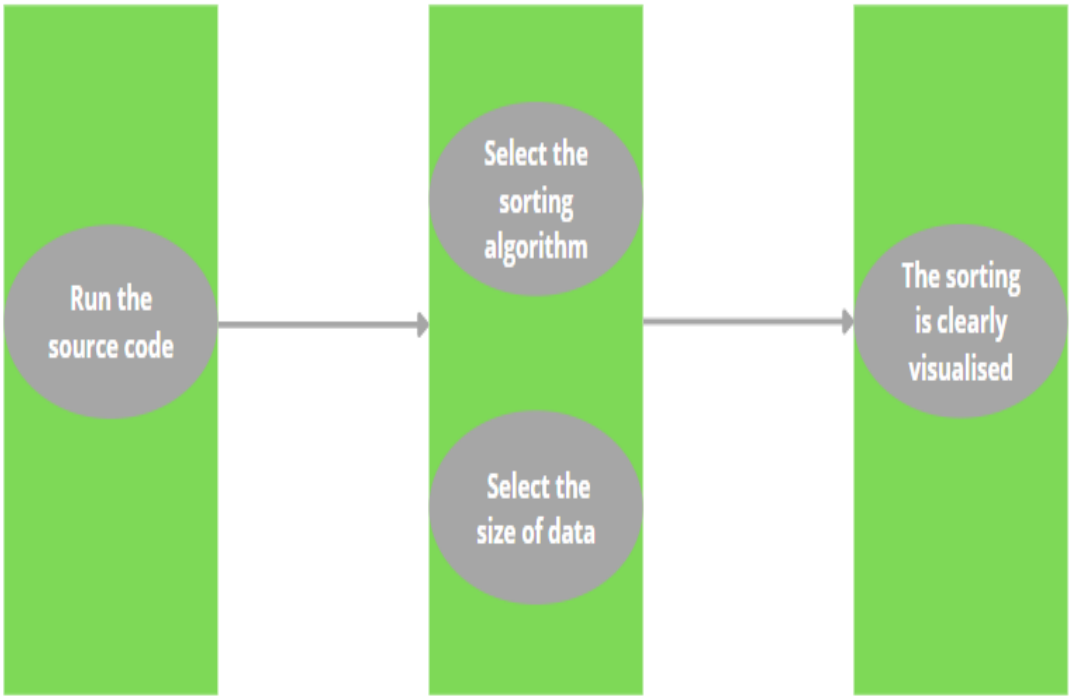
CHAPTER-4

System Design

4.1 Flow chart



4.2 Architectural Design



CHAPTER-5

Implementation

5.1 System modules

The project is implemented using HTML, CSS and JavaScript.
There are 6 JavaScript modules in the project:

- bubble.js – to store the bubble sort algorithm and call functions to enable and disable buttons.
- insertion.js – to store the insertion sort algorithm and call functions to enable and disable buttons.
- merge.js – to store the merge sort algorithm and call functions to enable and disable buttons.
- quick.js – to store the quick sort algorithm and call functions to enable and disable buttons.
- selection.js – to store the selection sort algorithm and call functions to enable and disable buttons.
- sorting.js – to define various functions to enable/disable buttons, create new array, delete existing array, adjust array size and sorting speed and select a sorting algorithm.

The html document was designed using bootstrap and CSS.

5.2 Code and implementation

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Sorting Visualizer</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous">
   <link rel="preconnect" href="https://fonts.googleapis.com">
   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
   <link href="https://fonts.googleapis.com/css2?family=Passion+One&display=swap"
rel="stylesheet">
   <link rel="stylesheet" href="style.css">
</head>

<body class="p-3 mb-2 bg-dark text-white">

  <h1 align="center">Sorting Visualizer</h1>
  <nav>
     <div class="row">
        <div class="col gap-2 d-sm-flex" id="newArray">
           <button type="button" class="btn btn-success newArray">New Array</button>
        </div>
        <div class="col" id="input">
```

```
            <span id="size">Size
                <input id="arr_sz" type="range" min="5" max="100" step=1 value=60>
            </span>
            <span id="speed">Speed
                <input id="speed_input" type="range" min="20" max="300" stepDown=10 value=60>
            </span>
        </div>
        <div class="col gap-2 d-sm-flex justify-content-end">
            <button type="button" class="btn btn-warning bubbleSort">Bubble Sort</button>
            <button type="button" class="btn btn-warning selectionSort">Selection Sort</button>
            <button type="button" class="btn btn-warning insertionSort">Insertion Sort</button>
            <button type="button" class="btn btn-warning quickSort">Quick Sort</button>
            <button type="button" class="btn btn-warning mergeSort">Merge Sort</button>
        </div>
    </div>
  </nav>

    <div id="bars" class="flex-container"></div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
b5kHyXgcpbZJO/tY9Ul7kGkf1S0CWuKcCD38l8YkeH8z8QjE0GmW1gYU5S9FOnJ0"
crossorigin="anonymous"></script>
    <script src="js_files/sorting.js"></script>
    <script src="js_files/bubble.js"></script>
    <script src="js_files/insertion.js"></script>
    <script src="js_files/merge.js"></script>
    <script src="js_files/quick.js"></script>
    <script src="js_files/selection.js"></script>
</body>
</html>
```

CSS:

```
body {
    font-size: 20px;
    padding: 0 20px 30px 0;
    line-height: 1.4;
    background-image: linear-gradient(black,#270140);
    background-attachment: fixed;
}

h1 {
  font-family: 'Passion One', cursive;
  font-size: 4em;
}

.flex-container{
    margin-top: 20px;
    display: flex;
    flex-wrap: nowrap;
    width: 100%;
    height: 500px;
    justify-content: center;
    transition: 2s all ease;
}

.flex-item{
    background: #e5a3f7;
    border: 1pt solid black;
    width: 20px;
    transition: 0.1s all ease;
}
```

```
.row{
    display: grid;
    grid-template-columns: 1fr 1fr 2fr;
}

#input{
    display: flex;
    padding: 10px;
    justify-content: space-around;
}
```

JavaScript:

bubble.js:

```
async function bubble() {

    const ele = document.querySelectorAll(".bar");
    for(let i = 0; i < ele.length-1; i++){

        for(let j = 0; j < ele.length-i-1; j++){

            ele[j].style.background = 'blue';
            ele[j+1].style.background = 'blue';
            if(parseInt(ele[j].style.height) > parseInt(ele[j+1].style.height)){

                await waitforme(delay);
                swap(ele[j], ele[j+1]);
            }
```

```
            ele[j].style.background = 'cyan';
            ele[j+1].style.background = 'cyan';
        }
        ele[ele.length-1-i].style.background = 'green';
    }
    ele[0].style.background = 'green';
}

const bubSortbtn = document.querySelector(".bubbleSort");
bubSortbtn.addEventListener('click', async function(){
    disableSortingBtn();
    disableSizeSlider();
    disableNewArrayBtn();
    await bubble();
    enableSortingBtn();
    enableSizeSlider();
    enableNewArrayBtn();
});
```

insertion.js:

```
async function insertion(){
    console.log('In insertion()');
    const ele = document.querySelectorAll(".bar");
    // color
    ele[0].style.background = 'green';
    for(let i = 1; i < ele.length; i++){
        console.log('In ith loop');
        let j = i - 1;
        let key = ele[i].style.height;
        // color
```

```
        ele[i].style.background = 'blue';

        await waitforme(delay);

        while(j >= 0 && (parseInt(ele[j].style.height) > parseInt(key))){
            console.log('In while loop');
            // color
            ele[j].style.background = 'blue';
            ele[j + 1].style.height = ele[j].style.height;
            j--;

            await waitforme(delay);

            // color
            for(let k = i; k >= 0; k--){
                ele[k].style.background = 'green';
            }
        }
        ele[j + 1].style.height = key;
        // color
        ele[i].style.background = 'green';
    }
}

const inSortbtn = document.querySelector(".insertionSort");
inSortbtn.addEventListener('click', async function(){
    disableSortingBtn();
    disableSizeSlider();
    disableNewArrayBtn();
    await insertion();
    enableSortingBtn();
```

```
    enableSizeSlider();
    enableNewArrayBtn();
});
```

merge.js:

```
async function merge(ele, low, mid, high){
    console.log('In merge()');
    console.log(`low=${low}, mid=${mid}, high=${high}`);
    const n1 = mid - low + 1;
    const n2 = high - mid;
    console.log(`n1=${n1}, n2=${n2}`);
    let left = new Array(n1);
    let right = new Array(n2);

    for(let i = 0; i < n1; i++){
        await waitforme(delay);
        console.log('In merge left loop');
        console.log(ele[low + i].style.height + ' at ' + (low+i));
        // color
        ele[low + i].style.background = 'orange';
        left[i] = ele[low + i].style.height;
    }
    for(let i = 0; i < n2; i++){
        await waitforme(delay);
        console.log('In merge right loop');
        console.log(ele[mid + 1 + i].style.height + ' at ' + (mid+1+i));
        // color
        ele[mid + 1 + i].style.background = 'yellow';
        right[i] = ele[mid + 1 + i].style.height;
    }
```

```
await waitforme(delay);
let i = 0, j = 0, k = low;
while(i < n1 && j < n2){
    await waitforme(delay);
    console.log('In merge while loop');
    console.log(parseInt(left[i]), parseInt(right[j]));

    // To add color for which two r being compared for merging

    if(parseInt(left[i]) <= parseInt(right[j])){
        console.log('In merge while loop if');
        // color
        if((n1 + n2) === ele.length){
            ele[k].style.background = 'green';
        }
        else{
            ele[k].style.background = 'lightgreen';
        }

        ele[k].style.height = left[i];
        i++;
        k++;
    }
    else{
        console.log('In merge while loop else');
        // color
        if((n1 + n2) === ele.length){
            ele[k].style.background = 'green';
        }
        else{
            ele[k].style.background = 'lightgreen';
```

```
        }
        ele[k].style.height = right[j];
        j++;
        k++;
    }
}
while(i < n1){
    await waitforme(delay);
    console.log("In while if n1 is left");
    // color
    if((n1 + n2) === ele.length){
        ele[k].style.background = 'green';
    }
    else{
        ele[k].style.background = 'lightgreen';
    }
    ele[k].style.height = left[i];
    i++;
    k++;
}
while(j < n2){
    await waitforme(delay);
    console.log("In while if n2 is left");
    // color
    if((n1 + n2) === ele.length){
        ele[k].style.background = 'green';
    }
    else{
        ele[k].style.background = 'lightgreen';
    }
    ele[k].style.height = right[j];
```

```
        j++;
        k++;
    }
}

async function mergeSort(ele, l, r){
    console.log('In mergeSort()');
    if(l >= r){
        console.log(`return cause just 1 elemment l=${l}, r=${r}`);
        return;
    }
    const m = l + Math.floor((r - l) / 2);
    console.log(`left=${l} mid=${m} right=${r}`, typeof(m));
    await mergeSort(ele, l, m);
    await mergeSort(ele, m + 1, r);
    await merge(ele, l, m, r);
}

const mergeSortbtn = document.querySelector(".mergeSort");
mergeSortbtn.addEventListener('click', async function(){
    let ele = document.querySelectorAll('.bar');
    let l = 0;
    let r = parseInt(ele.length) - 1;
    disableSortingBtn();
    disableSizeSlider();
    disableNewArrayBtn();
    await mergeSort(ele, l, r);
    enableSortingBtn();
    enableSizeSlider();
    enableNewArrayBtn();
});
```

quick.js:

```
async function partitionLomuto(ele, l, r){
    console.log('In partitionLomuto()');
    let i = l - 1;
    // color pivot element
    ele[r].style.background = 'red';
    for(let j = l; j <= r - 1; j++){
        console.log('In partitionLomuto for j');
        // color current element
        ele[j].style.background = 'yellow';
        // pauseChamp
        await waitforme(delay);

        if(parseInt(ele[j].style.height) < parseInt(ele[r].style.height)){
            console.log('In partitionLomuto for j if');
            i++;
            swap(ele[i], ele[j]);
            // color
            ele[i].style.background = 'orange';
            if(i != j) ele[j].style.background = 'orange';
            // pauseChamp
            await waitforme(delay);
        }
        else{
            // color if not less than pivot
            ele[j].style.background = 'pink';
        }
    }
    i++;
```

```
    // pauseChamp
    await waitforme(delay);
    swap(ele[i], ele[r]); // pivot height one
    console.log(`i = ${i}`, typeof(i));
    // color
    ele[r].style.background = 'pink';
    ele[i].style.background = 'green';

    // pauseChamp
    await waitforme(delay);

    // color
    for(let k = 0; k < ele.length; k++){
        if(ele[k].style.background != 'green')
            ele[k].style.background = 'cyan';
    }

    return i;
}

async function quickSort(ele, l, r){
    console.log('In quickSort()', `l=${l} r=${r}`, typeof(l), typeof(r));
    if(l < r){
        let pivot_index = await partitionLomuto(ele, l, r);
        await quickSort(ele, l, pivot_index - 1);
        await quickSort(ele, pivot_index + 1, r);
    }
    else{
        if(l >= 0 && r >= 0 && l <ele.length && r <ele.length){
            ele[r].style.background = 'green';
            ele[l].style.background = 'green';
```

```
        }
    }
}


const quickSortbtn = document.querySelector(".quickSort");
quickSortbtn.addEventListener('click', async function(){
    let ele = document.querySelectorAll('.bar');
    let l = 0;
    let r = ele.length - 1;
    disableSortingBtn();
    disableSizeSlider();
    disableNewArrayBtn();
    await quickSort(ele, l, r);
    enableSortingBtn();
    enableSizeSlider();
    enableNewArrayBtn();
});
```

selection.js:

```
async function selection(){
    console.log('In selection()');
    const ele = document.querySelectorAll(".bar");
    for(let i = 0; i < ele.length; i++){
        console.log('In ith loop');
        let min_index = i;
        // Change color of the position to swap with the next min
        ele[i].style.background = 'blue';
        for(let j = i+1; j < ele.length; j++){
            console.log('In jth loop');
```

```
            // Change color for the current comparision (in consideration for min_index)
            ele[j].style.background = 'red';

            await waitforme(delay);
            if(parseInt(ele[j].style.height) < parseInt(ele[min_index].style.height)){
                console.log('In if condition height comparision');
                if(min_index !== i){
                    // new min_index is found so change prev min_index color back to normal
                    ele[min_index].style.background = 'cyan';
                }
                min_index = j;
            }
            else{
                // if the currnent comparision is more than min_index change is back to normal
                ele[j].style.background = 'cyan';
            }
        }
        await waitforme(delay);
        swap(ele[min_index], ele[i]);
        // change the min element index back to normal as it is swapped
        ele[min_index].style.background = 'cyan';
        // change the sorted elements color to green
        ele[i].style.background = 'green';
    }
}

const selectionSortbtn = document.querySelector(".selectionSort");
selectionSortbtn.addEventListener('click', async function(){
    disableSortingBtn();
    disableSizeSlider();
    disableNewArrayBtn();
```

```
    await selection();
    enableSortingBtn();
    enableSizeSlider();
    enableNewArrayBtn();
});
```

sorting.js:

```
// swap function util for sorting algorithms takes input of 2 DOM elements with .style.height
feature
function swap(el1, el2) {
    console.log('In swap()');

    let temp = el1.style.height;
    el1.style.height = el2.style.height;
    el2.style.height = temp;

}

// Disables sorting buttons used in conjunction with enable, so that we can disable during sorting
and enable buttons after it
function disableSortingBtn(){
    document.querySelector(".bubbleSort").disabled = true;
    document.querySelector(".insertionSort").disabled = true;
    document.querySelector(".mergeSort").disabled = true;
    document.querySelector(".quickSort").disabled = true;
    document.querySelector(".selectionSort").disabled = true;
}

// Enables sorting buttons used in conjunction with disable
function enableSortingBtn(){
```

```
    document.querySelector(".bubbleSort").disabled = false;
    document.querySelector(".insertionSort").disabled = false;
    document.querySelector(".mergeSort").disabled = false;
    document.querySelector(".quickSort").disabled = false;
    document.querySelector(".selectionSort").disabled = false;
}

// Disables size slider used in conjunction with enable, so that we can disable during sorting and
enable buttons after it
function disableSizeSlider(){
    document.querySelector("#arr_sz").disabled = true;
}

// Enables size slider used in conjunction with disable
function enableSizeSlider(){
    document.querySelector("#arr_sz").disabled = false;
}

// Disables newArray buttons used in conjunction with enable, so that we can disable during
sorting and enable buttons after it
function disableNewArrayBtn(){
    document.querySelector(".newArray").disabled = true;
}

// Enables newArray buttons used in conjunction with disable
function enableNewArrayBtn(){
    document.querySelector(".newArray").disabled = false;
}

// Used in async function so that we can do animations of sorting
function waitforme(milisec) {
```

```
    return new Promise(resolve => {
        setTimeout(() => { resolve('') }, milisec);
    })
}

// Selecting size slider from DOM
let arraySize = document.querySelector('#arr_sz');

// Event listener to update the bars on the UI
arraySize.addEventListener('input', function(){
    console.log(arraySize.value, typeof(arraySize.value));
    createNewArray(parseInt(arraySize.value));
});

// Default input for waitforme function (260ms)
let delay = 260;

// Selecting speed slider from DOM
let delayElement = document.querySelector('#speed_input');

// Event listener to update delay time
delayElement.addEventListener('input', function(){
    console.log(delayElement.value, typeof(delayElement.value));
    delay = 320 - parseInt(delayElement.value);
});

// Creating array to store randomly generated numbers
let array = [];

// Call to display bars right when you visit the site
createNewArray();
```

```
// To create new array input size of array
function createNewArray(noOfBars = 60) {
    // calling helper function to delete old bars from dom
    deleteChild();

    // creating an array of random numbers
    array = [];
    for (let i = 0; i < noOfBars; i++) {
        array.push(Math.floor(Math.random() * 250) + 1);
    }
    console.log(array);

    // select the div #bars element
    const bars = document.querySelector("#bars");

    // create multiple element div using loop and adding class 'bar col'
    for (let i = 0; i < noOfBars; i++) {
        const bar = document.createElement("div");
        bar.style.height = `${array[i]*2}px`;
        bar.classList.add('bar');
        bar.classList.add('flex-item');
        bar.classList.add(`barNo${i}`);
        bars.appendChild(bar);
    }
}

// Helper function to delete all the previous bars so that new can be added
function deleteChild() {
    const bar = document.querySelector("#bars");
    bar.innerHTML = '';
```

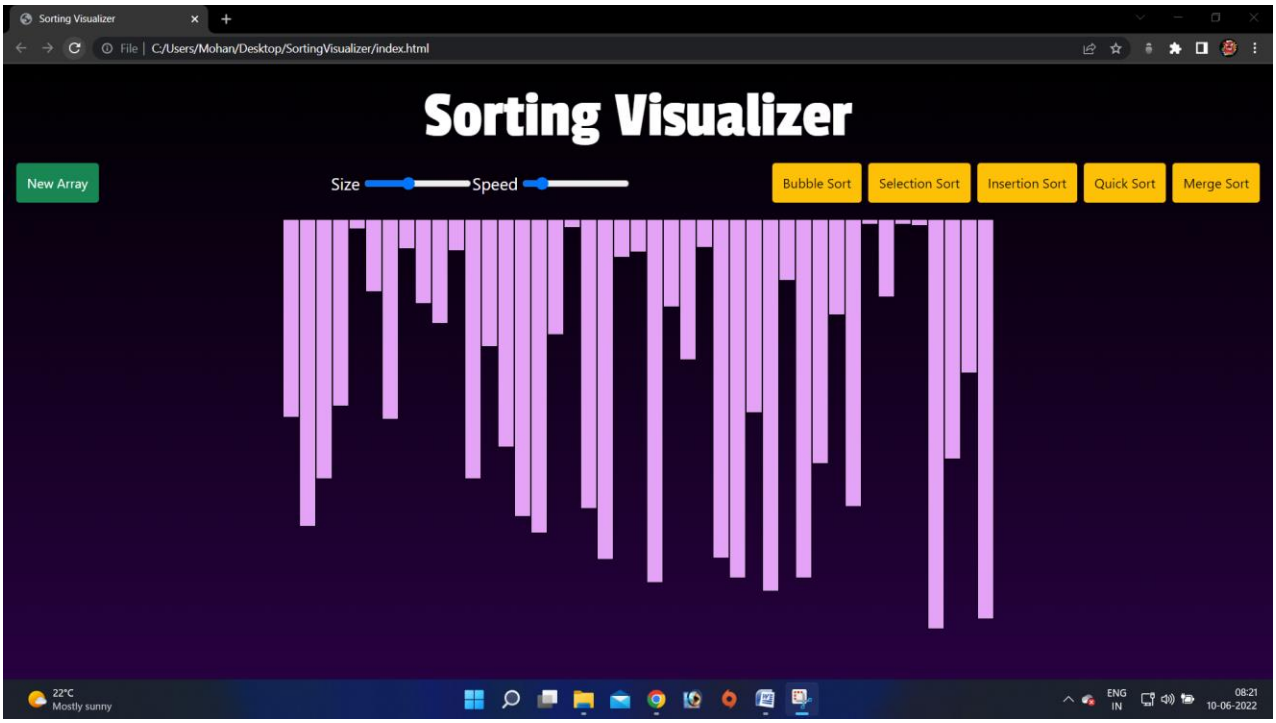
```
}

// Selecting newarray button from DOM and adding eventlistener
const newArray = document.querySelector(".newArray");
newArray.addEventListener("click", function(){
    console.log("From newArray " + arraySize.value);
    console.log("From newArray " + delay);
    enableSortingBtn();
    enableSizeSlider();
    createNewArray(arraySize.value);
});
```

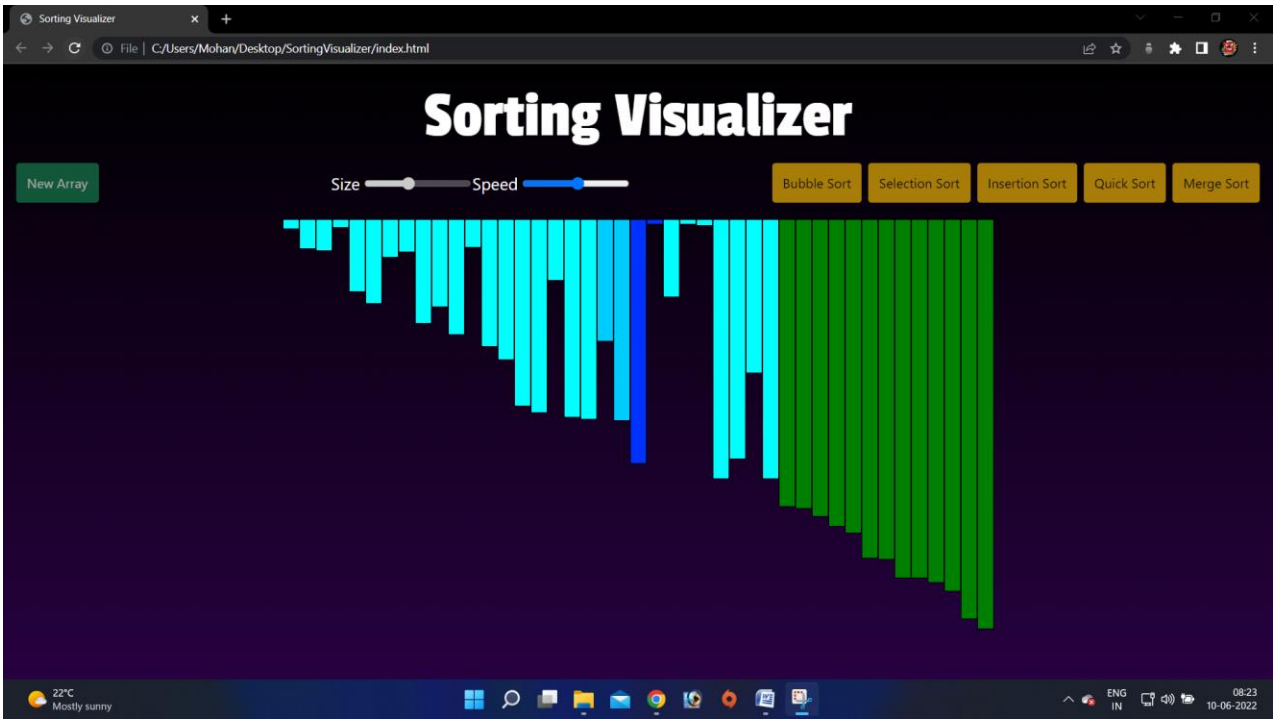
CHAPTER-6

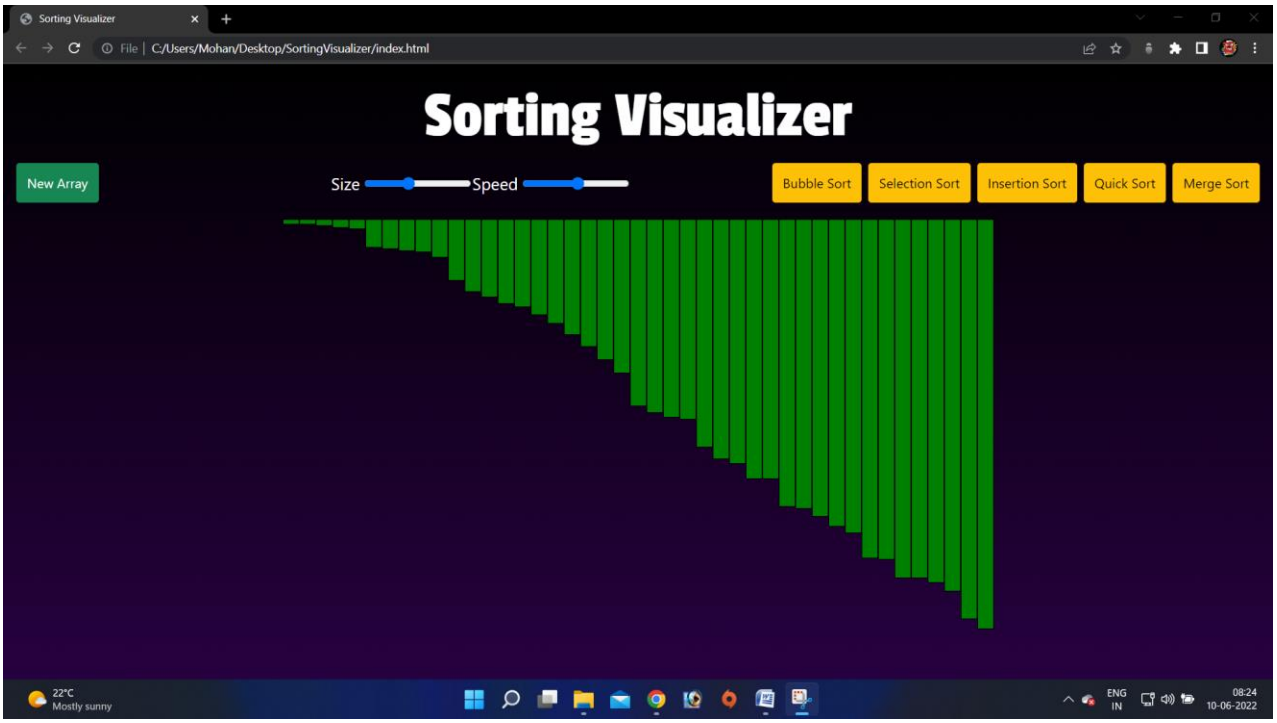
RESULT

6.1 Outcome

The users can now select any of the given sorting algorithms for its visualization.




Sorting Visualizer
File | C:/Users/Mohan/Desktop/SortingVisualizer/index.html
Sorting Visualizer
New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort
22°C
Mostly sunny
ENG
IN
08:23
10-06-2022


Sorting Visualizer
File | C:/Users/Mohan/Desktop/SortingVisualizer/index.html
Sorting Visualizer
New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort
22°C
Mostly sunny
ENG
IN
08:24
10-06-2022

6.2 Conclusion

- The Sorting Visualizer can be used to clearly visualize the working of various sorting algorithms.
- The user can also generate a new array, adjust the size of the array and the sorting speed.

6.3 Future Enhancement

- Visualize more sorting algorithms.
- Creating a game/application utilizing the sorting algorithms.
- Creating a 3D visualization of the same.

References

- Web programming building internet applications by Chris Bates.