ALGORITHM VISUALIZER

PROJECT REPORT

OF MINOR PROJECT

MASTER OF COMPUTER APPLICATIONS

SUBMITTED BY

TANU JAISWAL
Batch Year– 2019-22
Enrolment No.–U1949014

PROJECT GUIDE- ER. SHREYA AGARWAL

Centre of Computer Education & Training
Institute of Professional Studies
University of Allahabad, Prayagraj
Uttar Pradesh

RECOMMENDATION

The dissertation entitled Algorithm Visualizer submitted by Tanu Jaiswal is satisfactory account of the bonafide work done under my supervision is recommended towards fulfilment for the award of Master of Computer Application (MCA) degree by University of Allahabad (UoA).

Date:

Project Guide:

Er. Shreya Agarwal

ACKNOWLEDGEMENT

This project report is based on '**ALGORITHM VISUALIZER**'. I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.
I am highly indebted to **Er. Shreya Agarwal** for her guidance and constant supervision as well as for providing necessary information regarding the project & also for her support in completing the project. I would like to express my gratitude towards my parents & members of Institute of professional **Studies (**IPS), University of Allahabad for their kind co-operation and encouragement which help me in completion of this project.
I would like to express my special gratitude and thanks to my classmates and teachers for giving me such attention and time.
My thanks and appreciations also go to my Friends in developing the project and people who have willingly helped me out with their abilities.

Tanu Jaiswal
MCA, 5th Semester
Enrolment No.- U1949014

CERTIFICATE

This is to certify that **Tanu Jaiswal,** Student of **MCA 3RD Year** of **Institute of Professional Studies, University of Allahabad** has completed her project entitled **ALGORITHM VISUALIZER** under my guidance in the academic year (2021-22).

She has taken proper care and shown utmost sincerity in completing this project. Her Work is satisfactory. I wish her all the best for her bright future. I certify that this project is up to my expectations and as per the guidelines.

This application package is the original one and is never submitted somewhere for the same purpose.

ER. SHREYA AGARWAL (PROJECT GUIDE)

PATICULARS

INTRODUCTION

From the beginning of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. Among the authors of early sorting algorithms around 1951 was *Betty Holberton* who worked on *ENIAC* and *UNIVAC*. *Bubble Sort* was analysed as early as 1956. Asymptotically optimal algorithms have been known since the mid-20th century - new algorithms are still being invented, with the widely used Timsort dating to 2002, and the library sort being first published in 2006.

Comparison sorting algorithms have a fundamental requirement of $\Omega(n \log n)$ comparisons (some input sequences will require a multiple of $n \log n$ comparisons, where n is the number of elements in the array to be sorted). Algorithms not based on comparisons, such as counting sort, can have better performance.

Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as big O notation, divide and conquer algorithms, data structures such as heaps and binary trees, randomized algorithms, best, worst and average case analysis, time-space trade-offs, and upper and lower bounds.

Sorting small arrays optimally (in fewest comparisons and swaps) or fast (i.e. taking into account machine specific details) is still an open research problem, with solutions only known for very small arrays (<20 elements). Similarly optimal (by various definitions) sorting on a parallel machine is an open research topic.

Sorting algorithms can be classified by:

- Computational Complexity- Best, worst and average case behavior in terms of the size of the list. For typical serial sorting algorithms, good behavior is $O(n \log n)$, with parallel sort in $O(\log^2 n)$, and bad behavior is $O(n^2)$. Ideal behavior for a serial sort is $O(n)$, but this is not possible in the average case. Optimal parallel sorting is $O(\log n)$; Swaps for "in-place" algorithms.
- Memory usage (and use of other computer resources). In particular, some sorting algorithms are "in-place". Strictly, an in-place sort needs only $O(1)$ memory beyond the items being sorted; sometimes $O(\log n)$ additional memory is considered "in-place".

- Recursion: Some algorithms are either recursive or non-recursive, while others may be both (e.g., merge sort).
- Stability: stable sorting algorithms maintain the relative order of records with equal keys (i.e., values).
- Whether or not they are a comparison sort. A comparison sort examines the data only by comparing two elements with a comparison operator.
- General method: insertion, exchange, selection, merging, *etc.* Exchange sorts include bubble sort and quicksort. Selection sorts include cycle sort and heapsort.
- Whether the algorithm is serial or parallel. The remainder of this discussion almost exclusively concentrates upon serial algorithms and assumes serial operation.
- Adaptability: Whether or not the pre-sortedness of the input affects the running time. Algorithms that take this into account are known to be adaptive.
- Online: An algorithm such as Insertion Sort that is online can sort a constant stream of input.

Baeker was the first well-known visualization, ad hoc visualizations existed long before. The first recognized system for creating algorithm animations was BALSA in 1984.

Algorithms are a fascinating use case for visualization. To visualize an algorithm, we don't merely fit data to chart; there is no primary dataset. Instead, there are logical rules that describe behaviour.
Algorithm visualization can be defined as the use of images to convey some useful information about algorithms. That information can be a visual illustration of an algorithm's operation, of its performance on different kinds of inputs, or of its execution speed versus that of other algorithms for the same problem.
There are two principal variations of algorithm visualization:

1. Static Algorithm Visualization
2. Dynamic Algorithm Visualization, also called **algorithm animation**

To accomplish this goal, an algorithm visualization uses graphic elements points, line segments, two- or three-dimensional bars, and so on to represent some "interesting events" in the algorithm's operation.
Algorithm visualization illustrates how algorithms work in a graphical way. It mainly aims to simplify and deepen the understanding of algorithms operation. Specifically, it demonstrates the instances on the functioning of the algorithms and bring the information about statistics.

ALGORITHM VISUALISER

PROJECT ABOUT:

Algorithm Visualizer (A web based application that animates the complex working of computer sorting algorithms)

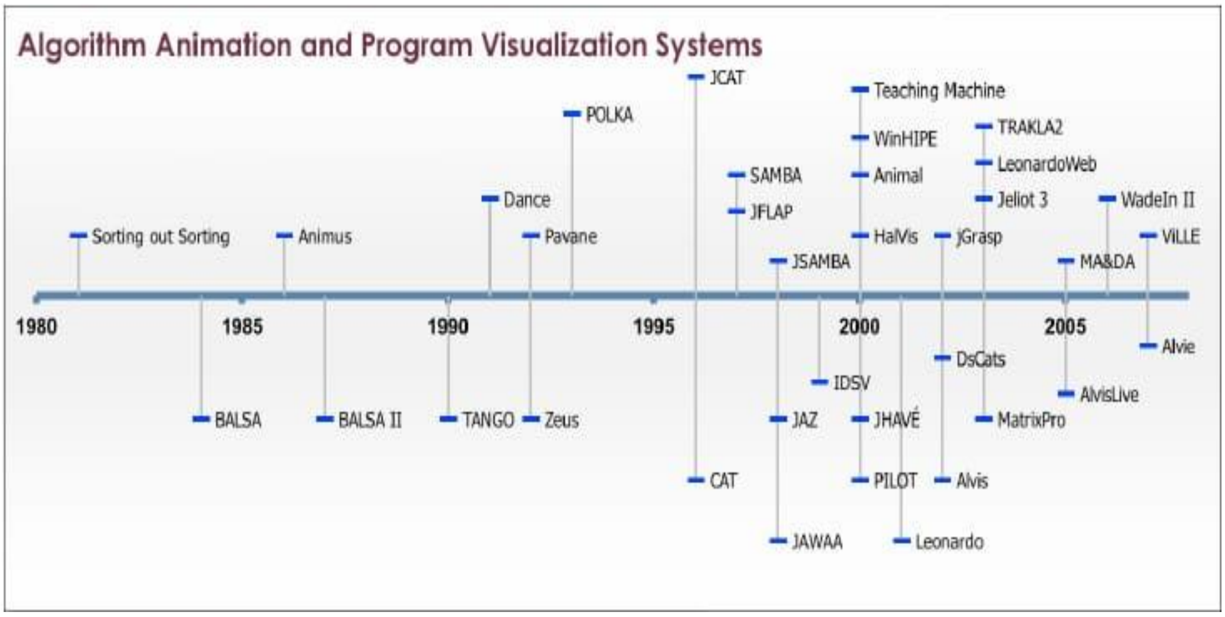
In addition to the mathematical and empirical analysis of algorithms, there is yet third way to study algorithms. It is called algorithm visualisation and can be defined as the use of images to convey some useful information about algorithms.

That information can be a visual illustration of an algorithm's operation, of its performance on different kinds of inputs, or of its execution speed versus that of other algorithm for the same problem.

To accomplish this goal, an algorithm visualization uses graphics element points, line segments, two- or three-dimensional bars, and so on to represent some "interesting events" in the algorithm's operation.

Under the Algorithm Visualizer, a generation of new array will be set by the users end along with the number of bars & speed (speed is directly proportional to the number of vertical bars).

Thereafter selecting one among the given sorting algorithm, the bars will visualize in a methodology of either top to down approach (first number of generated arrays will be compared to the bottom one) or vice-versa. For this animated visualization, you must set the whole process under "Sort!". Figure1.



OBJECTIVE(S)

1. As sorting is the classic subject in Computer Science, Sorting Algorithms illustrate many creative approaches to problem solving and these approaches can be applied to solve other problems.

2. For practicing fundamental programming techniques using selection statements, loops, methods and arrays, this is one of the best reasons for the visualization as it makes easier acknowledge.

3. By studying the variety of algorithms available for the same task (sorting), you can explore how to analyze worst-case time complexity, average-case time complexity, worst-case extra space, average-case extra space.

4. The main objective of this project is to help beginners to be able to visualize the basic algorithms and get a better understanding of the underlying operations.

5. This aims to simplify and deepen the understanding of algorithms functioning. (Because a deeper understanding of human perception of images will be required before the true potential of algorithm visualization is fulfilled.)

6. This project will act as a tool, made with certain Front-End languages for visualizing sorting algorithms. Our purpose is to portray the algorithms so the user can understand how a computer “move some pieces” to achieve his goal, having sorted data at the end!

SCOPE OF THE PROJECT

Algorithms and data structures as an essential part of knowledge in a framework of computer science, they have their stable position in computer science curriculum, since every computer scientist and every professional programmer should have the basic knowledge from the area. With the increasing number of students in Central European's higher education systems in last decades, introduction
of appropriate methods into the process of their education is also required. Our scope here is the higher education in the field of computer science.

The focus of this meta-study is on algorithm visualization effectiveness. We adopt a standard definition of algorithm visualization: a subclass of software visualization with illustrating computer algorithms in terms of their high-level operations, usually for the purpose of enhancing computer science students' understanding of the algorithms' procedural behaviour.
The notion of effectiveness, as it will be used here, concerns the union of humans and technology within the context of a scenario of use. Associated with such a scenario of use is:
(a) a particular objective to be fulfilled (e.g. 'learn how the target algorithm works');
(b) a particular individual or group having that objective;
(c) a particular algorithm visualization artifact that will be enlisted; and
(d) a particular target algorithm to be visualized.
Within this context, effectiveness thus responds to the question: To what extent does the algorithm visualization artifact assist the individual or group in fulfilling the objective

Amongst the most popular methods currently being investigated are algorithm visualizations and animations (hereafter referred to as algorithm visualizations). It is true that students can learn algorithms without using an algorithm visualization. But based on the age-old adage "a picture speaks more than thousand words," many researchers and educationists assume that students would learn an algorithm faster and more thoroughly using an algorithm visualization. There are other benefits that are assumed by those creating these systems.

a) A visualization can hope to convey dynamic concepts of an algorithm.
b) Studying graphically is assumed to be easier and more fun for many students than reading "dry" textbooks because an important characteristic of the algorithm visualizations is that they are (seemingly) more like a video game or an animated movie. By making an algorithm visualization

more similar to these popular forms of entertainment, instructors can use it to grab students’ attention.

c) Algorithm visualizations provide an alternative presentation mode for those students who understand things more easily when presented to them visually or graphically as compared to textually. In theory, these students would benefit a lot if visual systems are used to explain algorithms to them. Of course, the concept of using algorithm visualizations is not new because most instructors have always used graphics (slides Chapter1: Introduction 2 and pictures) to teach their courses. Algorithm visualizations could be viewed as more sophisticated pictures, slides, and movies.
d) Visualizations can help instructors to cover more material related to a specific algorithm in less time. Instructors can demonstrate the behaviour of an algorithm on both small and large data sets of elements using large screen projection devices in class. This provides students with a broader perspective about the working and the mechanism of that algorithm, time complexity, and differences in its behaviour in response to different data sets.
e) Students can use algorithm visualizations to explore behaviour of an algorithm on data sets that the students generate after the lecture is over in a more homework-like setting .

INPUT DATA AND VALIDATION

- For sorting, primarily you must create or generate new array by clicking on the button as giving input to the application for further computing.

- Secondary you choose the numbers of the graphical vertical animated bar and speed. The more graphical bars, faster will be speed.

- Avoid selecting the greater number of bars, if you are beginner to understand the algorithm working and its operation.

- During picking the particular sorting algorithm you must choose carefully as it won’t stop in between during computing/processing.

- On instant you pick “SORT”, processing will go on and you cannot pause the process as the reason is all the listening features of button will be paralyzed.

- All the steps must go in sequence for smooth working for this web-based application.

REQUIREMENT & SOFTWARE ANALYSIS

❖ Technology Stack

- Front-End Languages- HTML, CSS, JAVASCRIPT, BOOTSTARP
- Front -End Frameworks- BOOTSTRAP (CSS & JAVASCRIPT)

❖ Software Requirements

- Operating System:
  - Windows OS-
    Windows XP, Windows 7(Ultimate, & Enterprise), Windows 8(or 8.1), Windows 10 and later on versions.
  - MacOS-
    MacOS X (10.2 -10.15) 32/64-bit PowerPC, MacOS 11(Big Sur, 64-bit intel and ARM), MacOS 11.5 (Air M1, Big Sur, 64-bit intel and ARM), MacOS 12(Monterey, 64-bit intel and ARM) and later.
- Visual Studio Code
- Web Browser such as Safari, Google Chrome, Opera, Mozilla Firefox, Microsoft Edge (or Edge Chromium), Internet Explorer, to host on local server.

❖ Hardware Requirements

- Windows:
  - Processor- i3
  - Hard Disk- 5GB
  - Memory- 4GB RAM
- MacBook
  - Processor- Apple MacBook Air M1 chip OR Intel chip
  - Hard Disk- 5GB
  - Memory- 4/8GB RAM

❖ Software Analysis

- MacOS Monterey Version 12.1
- IDE: Visual Studio Code

TECHNOLOGIES AND METHOLOGIES USED

The implementation of this project is a combination of HTML5 (Hypertext Markup Language 5), JavaScript, CSS (Cascading Style Sheets), Bootstrap and React.

Following are the description about the factors that are concluded:

- **HTML** is used for describing the structure of Web pages. It has been used widely and effectively in this project, here are the few factors that are included:
    - Web Pages Development
    - Responsiveness
    - Offline Capability Usage

- **CSS** is used for describing the presentation of Web pages, including colors, layout, and fonts. Few implementing advantages of CSS are given below:
    - Compatibility
    - Image File Handling
    - Dynamic Templates

- **JavaScript** is used to add dynamic behavior to the webpage and add special effects to the webpage. There are certain purposes, due to which dynamic interactivity of web applications interprets full-fledged programming, given below:
    - Validation Purpose
    - Web Server Based
    - Interactive Behavior

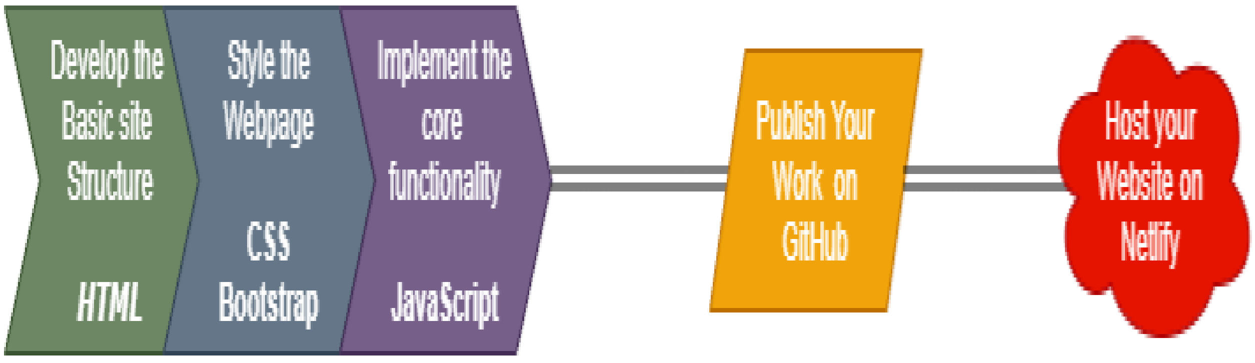
- **Bootstrap** is an HTML, CSS & JS Library that focuses on simplifying the development of informative web pages (as opposed to web apps). What basically Bootstrap package contains? List:
    - Scaffolding
    - JavaScript Plugins
    - Components like navbar, iconography, etc.

SYSTEM DESIGN

The implementation of this project is a combination of HTML5 (Hypertext Markup Language 5), JavaScript, CSS (Cascading Style Sheets), & Bootstrap. There is only one project folder that contains the various index.html and a client folder. The client folder contains the java script bundle of library and modules and some icon file named as .ico. Index.html file has reference to the significant files contained in client folder. As of now, the preferred browser is Google Chrome and Safari using live server in Visual studio code, as I only performed rigorous testing in this environment. Due to quick tests showed possible use in Google Chrome and Safari. High-Level Approach:

- Creating the website's User Interface (UI) using HTML, CSS and enhancing it further using Bootstrap; without actually implementing any of the app's core features.
- Implementation of animations, effects and core functionalities (sorting algorithms) using JavaScript.
- Optional: Publish to GitHub and host your project live using Netlify.

Figure2.


Develop the Basic site Structure
HTML
Style the Webpage
CSS
Bootstrap
Implement the core functionality
JavaScript
Publish Your Work on GitHub
Host your Website on Netlify

One large refactors later, the code now resembles a Model-View-Controller architecture. Although, due to its functional nature, it has many more individualized functions that update the instance variables and Boolean values, thereby directly updating the View and Controller.

A simplified diagram of the Model-View-Controller relationship is below in Figure 3.

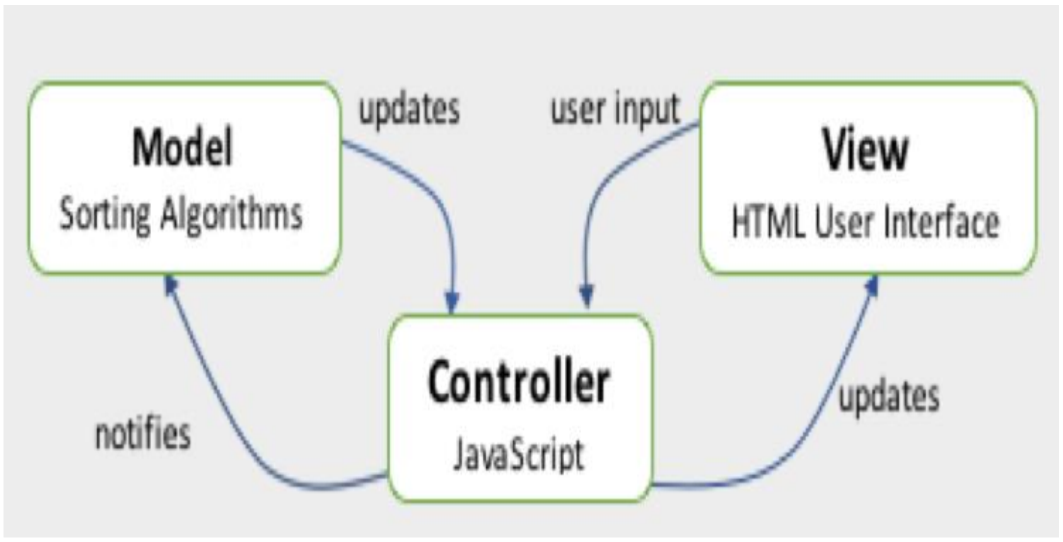
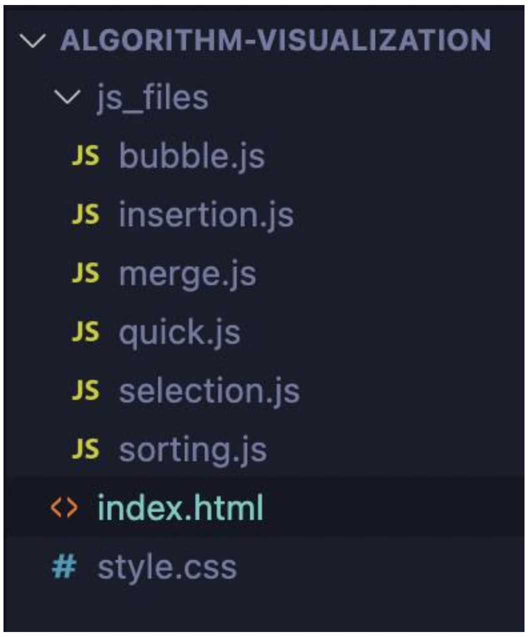


Figure 3: Model-View-Controller diagram of code.



```
ALGORITHM-VISUALIZATION
  js_files
    bubble.js
    insertion.js
    merge.js
    quick.js
    selection.js
    sorting.js
  index.html
  style.css
```

Figure 4.

SYSTEM ARCHITECTURE OF THE PROJECT

The back-end code is comprised of HTML5, CSS, and JavaScript. All three types of code are contained in one folder called Agorithm-Visualiser and index.html file can be run solely from this folder. One of the advantages of HTML 5 is that it is not necessary to include different types of web languages in a single file but I prefer file keeping separately in folder. This is good practice for readability and keeping related code together. However, I decided to separate the code for two reasons: 1) to increase the portability of the project by only needing to worry about one project setup and 2) where in the project file, the change in coding languages is distinctly marked and therefore does not significantly reduce readability. But, I will state the fact that the ability to put more than one web language in a single file is an example of an RIA (Rich Internet Application).

Below in Figure is an illustration of how the three coding languages relate and communicate with each other

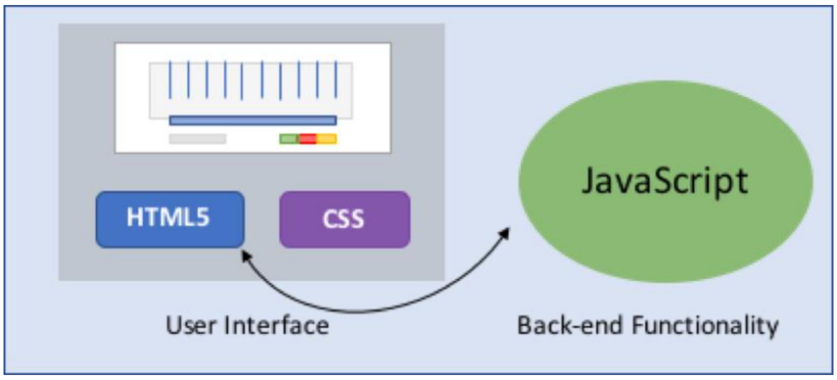


Figure 5. Architectural Diagram

As you can see, there are no major components besides the three coding languages. Most websites have tools or scripts that require a server on the back-end (like PHP), but it is not necessary in this case since JavaScript runs right in the user's browser. HTML5 and CSS are used for the interface. The HTML5 communicates with the JavaScript code and vice versa to launch the appropriate algorithms and update the interface accordingly, as seen with a single, bidirectional arrow.

Throughout the project, the code for the HTML5 and CSS did not change much. As the JavaScript was modified from a functional programmingfocus to a more object-oriented one, the parts of the HTML5 that did change were the function calls for each button. All of the back-end interaction is abstracted to the various buttons for selecting algorithms and running the animation.

PROJECT SCHEDULING

An elementary timeline chart for the development of the plan is given below:

	Sept	**Oct**	**Nov**	**Dec**	**Jan**
Requirement Gathering	■				
Analysis		■			
Designing			■		
Coding				■	
Implementation					■
	W_2 W_3 W_4	W_1 W_2 W_3 W_4	W_2 W_3 W_4	W_1 W_2 W_3	W_1 W_2

Here, $W_{i's}$ represents the week of the months.

ACTIVITY DIAGRAMS OF THE PROJECT

FLOW CHART OF THE CODING

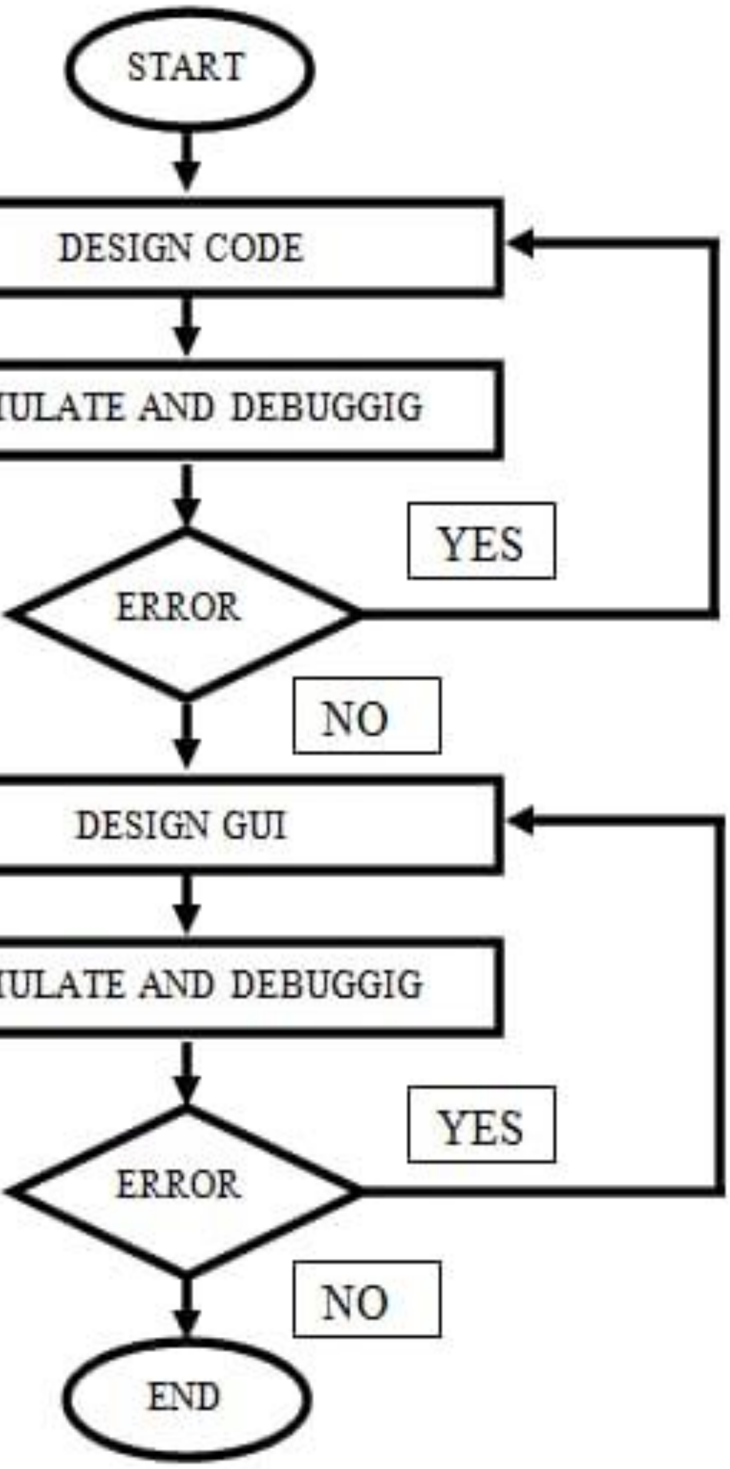


Figure 6.

DATA FLOW DIAGRAM

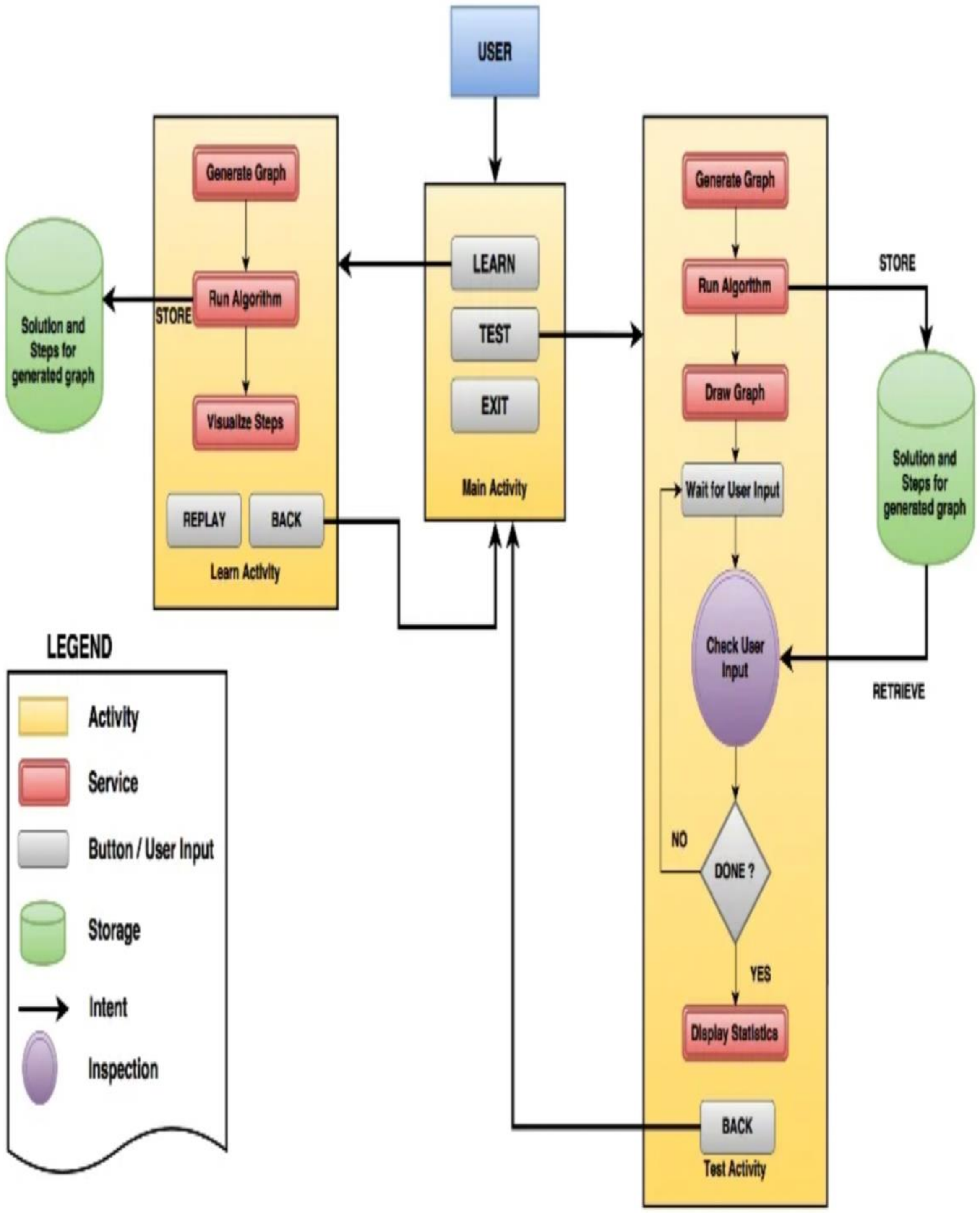


Figure 7. Application's Flow Diagram

CLASS/OBJECT DIAGRAM

The class/object relationships between components are shown on the next page in Figure 5. The main module represents the global scope in the html file between the <script></script> tags. The variables and methods listed in it are accessible by all components.

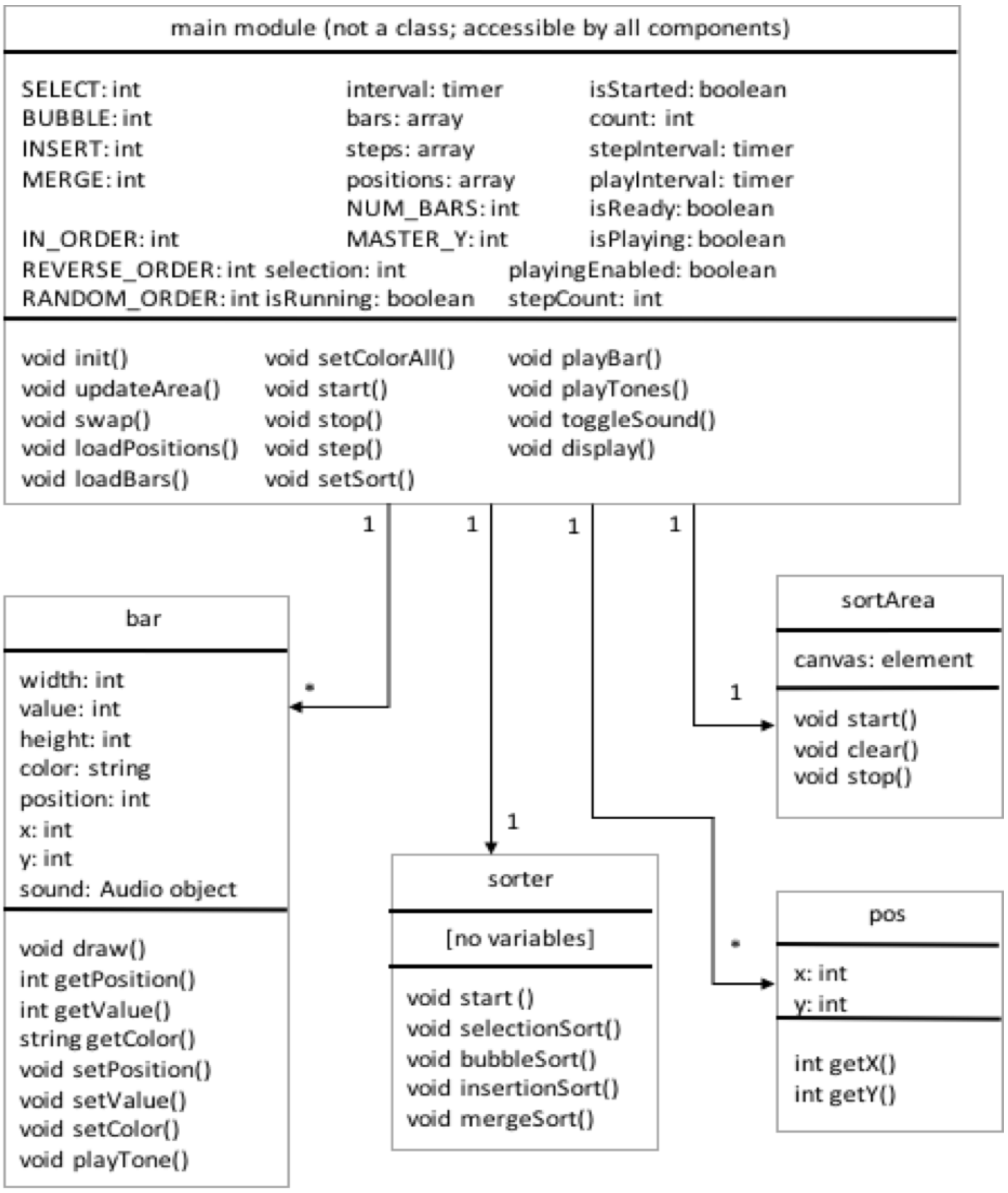


Figure 8: Class/object diagram of instance variables and respective functions in animation tool

USER DIAGRAM

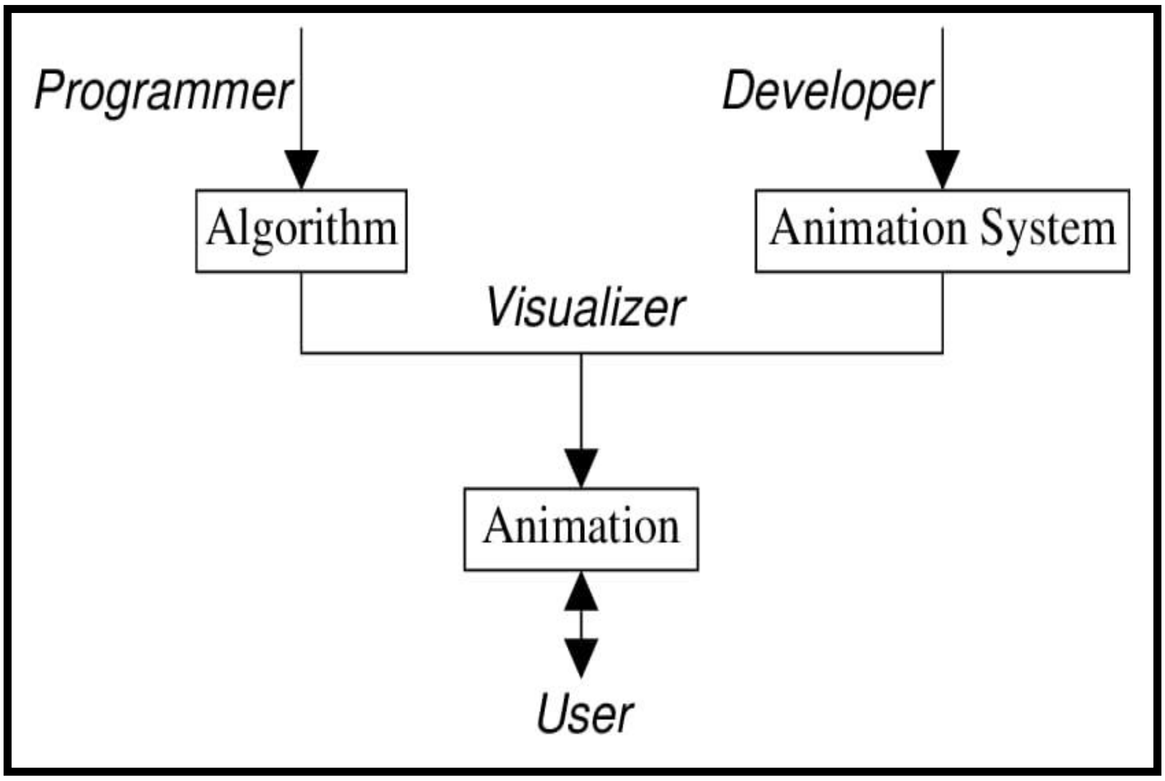


Figure 9. Schematic view of user roles

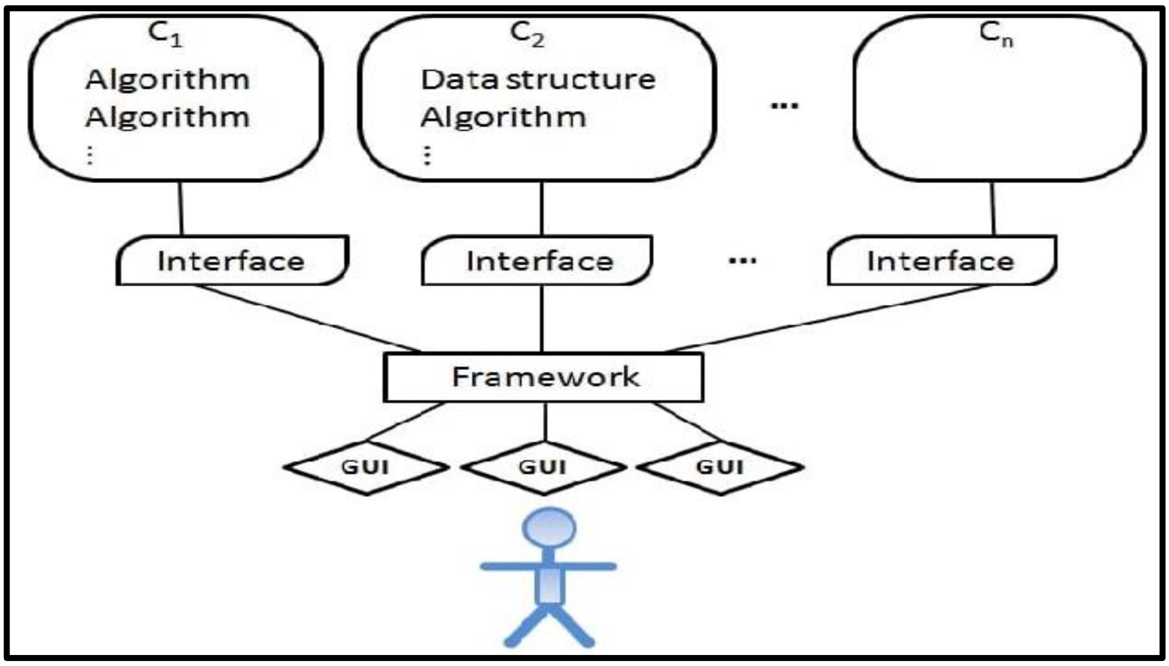


Figure 10. An illustration of the architecture

SOFTWARE IMPLEMENTATION

The implementation of this project is a combination of HTML (Hypertext Markup Language), JavaScript, and CSS (Cascading Style Sheets). There is a folder called Algorithm-Visualizer that contains the **index.html** (which is an HTML file), **style.css** (a cascading style sheet) and a **js_files** folder that contains the multiple JavaScript file name of: bubble.js, insertion.js, merge.js, quick.js, and selection.js.

As of now, the preferred browser is Mozilla Firefox as I only performed rigorous testing in this environment. However, quick tests showed possible use in Google Chrome and Safari.

The organization of the code follows both object-oriented and functional programming concepts. Originally, the design was almost completely functional, where only three objects were used: one to control the canvas that displayed the animation, another to represent a piece of data, or “bar” object (blue rectangle with dynamically changing height and position), and a final one to represent the positions that each bar moved to, or “pos” objects.

Some instance variables and Boolean values were used to keep track of the algorithm selected and when to animate, but this resulted in a heavily integrated mass of function calls that was difficult to upkeep.

One large refactor later, the code now resembles a Model-View-Controller architecture. Although, due to its functional nature, it has many more individualized functions that update the instance variables and Boolean values, thereby directly updating the View and Controller.

CODING

HTML FILE

```
<!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+nYRRuWl
olflfl" crossorigin="anonymous">
    <link rel="stylesheet" href="style.css">
    <link rel="shortcut icon" type="image/ico" href="js_files/icon.ico">
</head>
<header>
    <h1 align="center">Algorithm Visualizer</h1>
    <nav>
        <div class="row">
            <div class="col gap-2 d-sm-flex" id="newArray">
                <button type="button" class="btn btn-outline-success btn-dark
newArray">Generate 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-outline-primary btn-dark
bubbleSort">Bubble Sort</button>
                <button type="button" class="btn btn-outline-primary btn-dark
selectionSort">Selection Sort</button>
```

```
                <button type="button" class="btn btn-outline-primary btn-dark
insertionSort">Insertion Sort</button>
                <button type="button" class="btn btn-outline-primary btn-dark
quickSort">Quick Sort</button>
                <button type="button" class="btn btn-outline-primary btn-dark
mergeSort">Merge Sort</button>
            </div>
        </div>
    </nav>
</header>

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

    <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/tY9Ul7kGkf1S0CWuKcCD38l8YkeH8z8QjE0GmW1gYU
5S9FOnJ0"
        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 FILE

```css
body {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 20px;
    padding: 0 20px 30px 0;
    line-height: 1.4;
}
.flex-container{
    margin-top: 20px;
    display: flex;
    flex-wrap: nowrap;
    width: 100%;
    height: 500px;
    justify-content: center;
    transition: 2s all ease;
}

.flex-item{
    background: cyan;
    border: 1pt solid black;
    width: 10px;
    transition: 0.1s all ease;
}

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

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

JAVASCRIPT FILES: BUBBLE SORTING

```
async function bubble() {
    console.log('In bubbe()');
    const ele = document.querySelectorAll(".bar");
    for(let i = 0; i < ele.length-1; i++){
        console.log('In ith loop');
        for(let j = 0; j < ele.length-i-1; j++){
            console.log('In jth loop');
            ele[j].style.background = 'blue';
            ele[j+1].style.background = 'blue';
            if(parseInt(ele[j].style.height) > parseInt(ele[j+1].style.height)){
                console.log('In if condition');
                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 SORTING

```
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';
    }
}
```

MERGE SORTING

```
//let delay = 30;
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 SORTING

```
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 SORTING

```
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

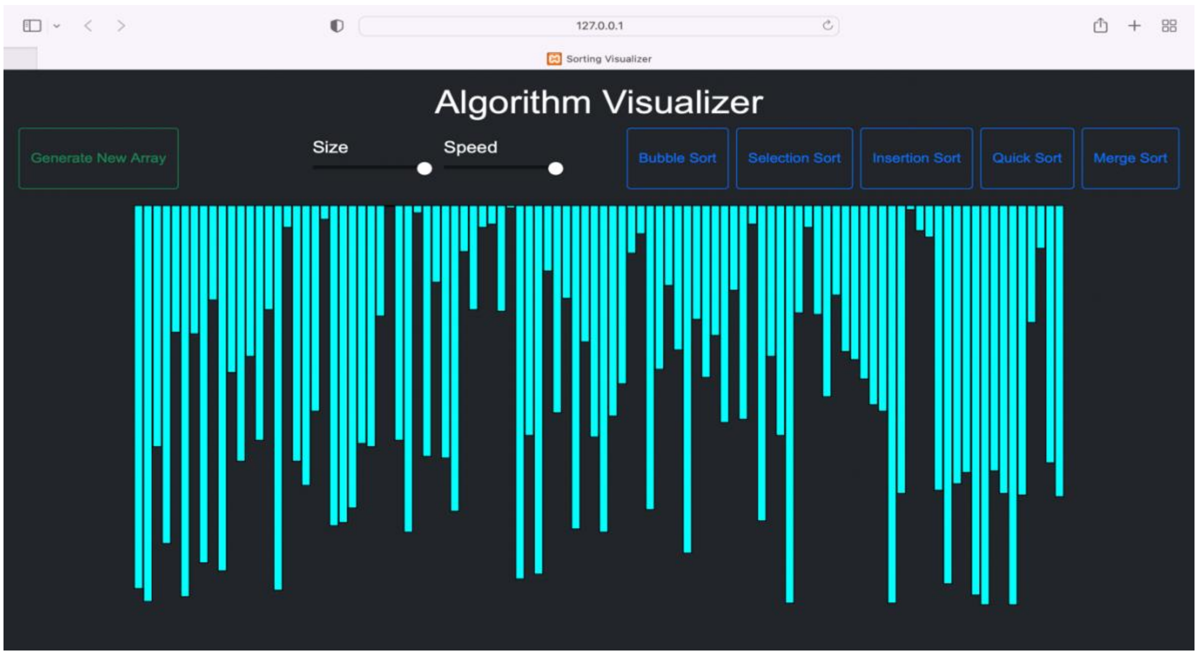
```
// swap function util for sorting algorithms takes input of 2 DOM elements
with .style.height feature
function swap(el1, el2) {
    console.log('In swap()');

    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 so animations of sorting, takes input
time in ms (1000 = 1s)
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);
});
```

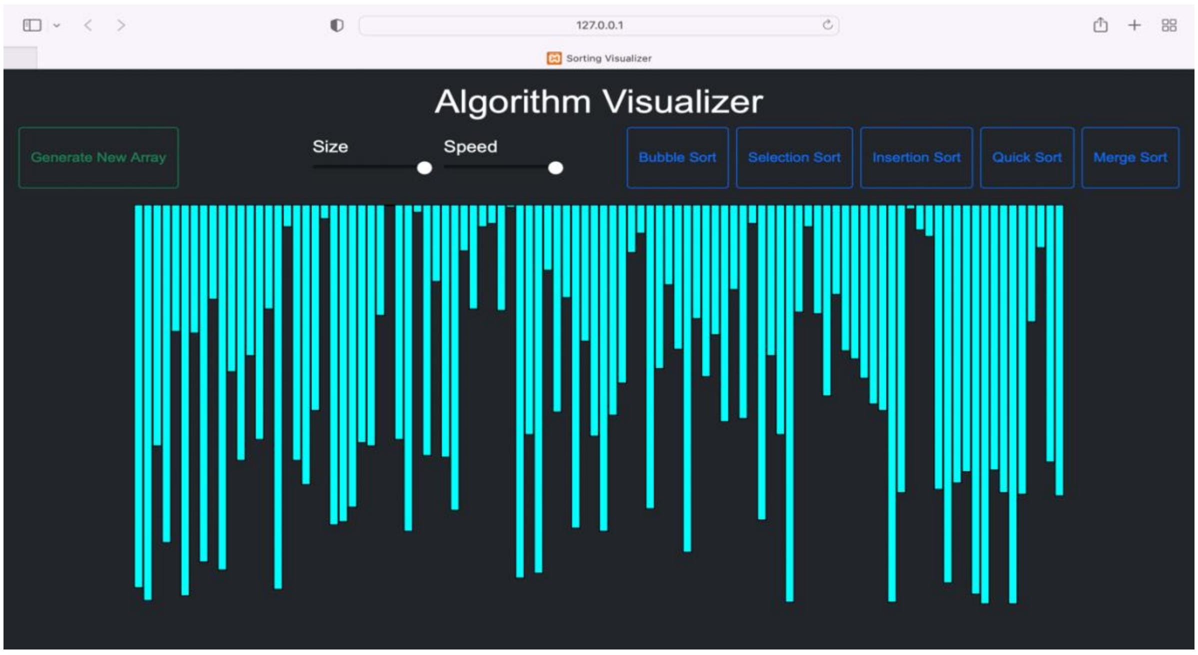
SCREENSHOTS

HOME PAGE

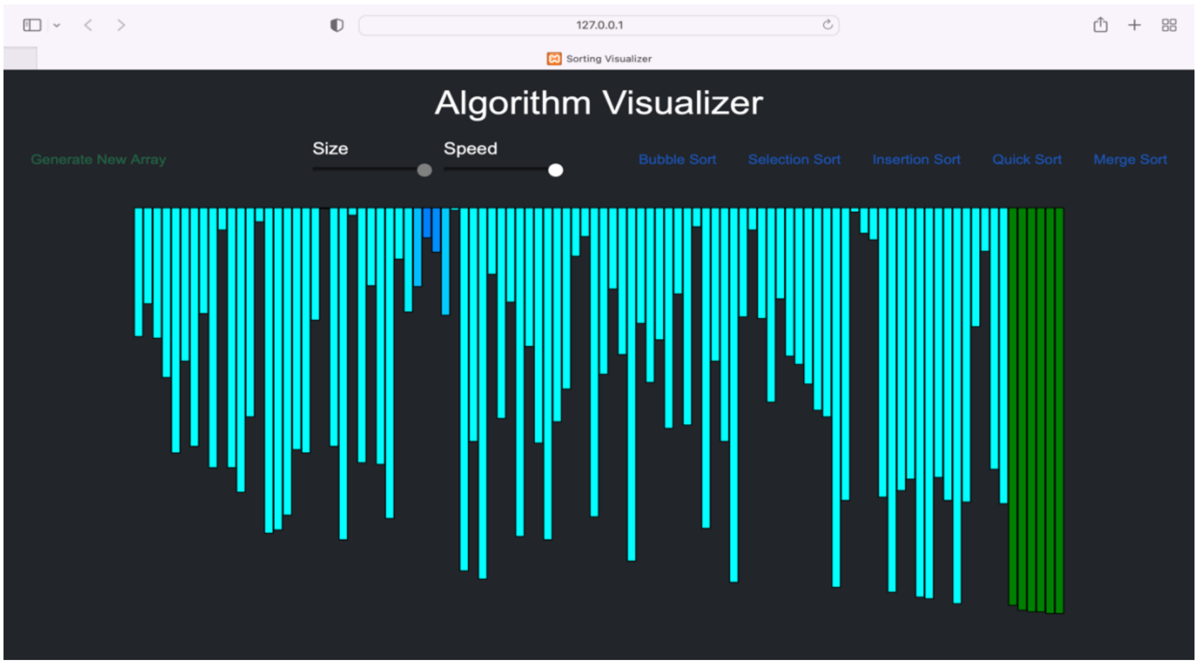


BUBBLE SORT

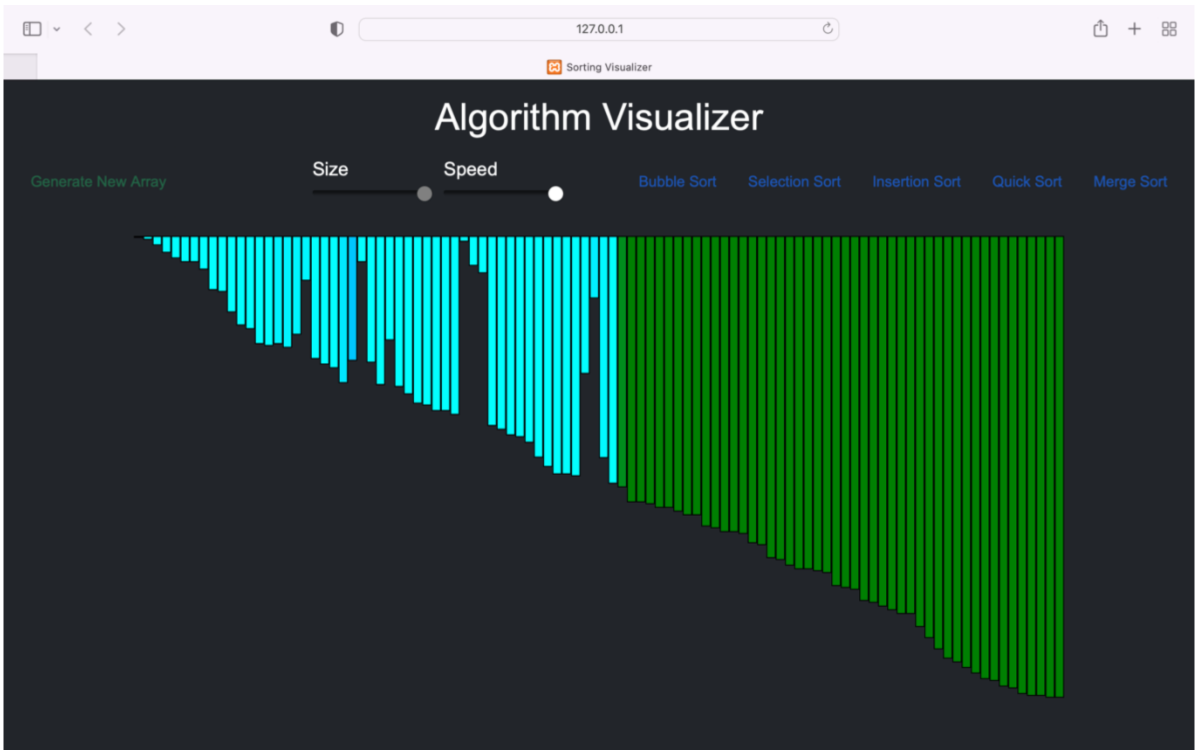
Step-1



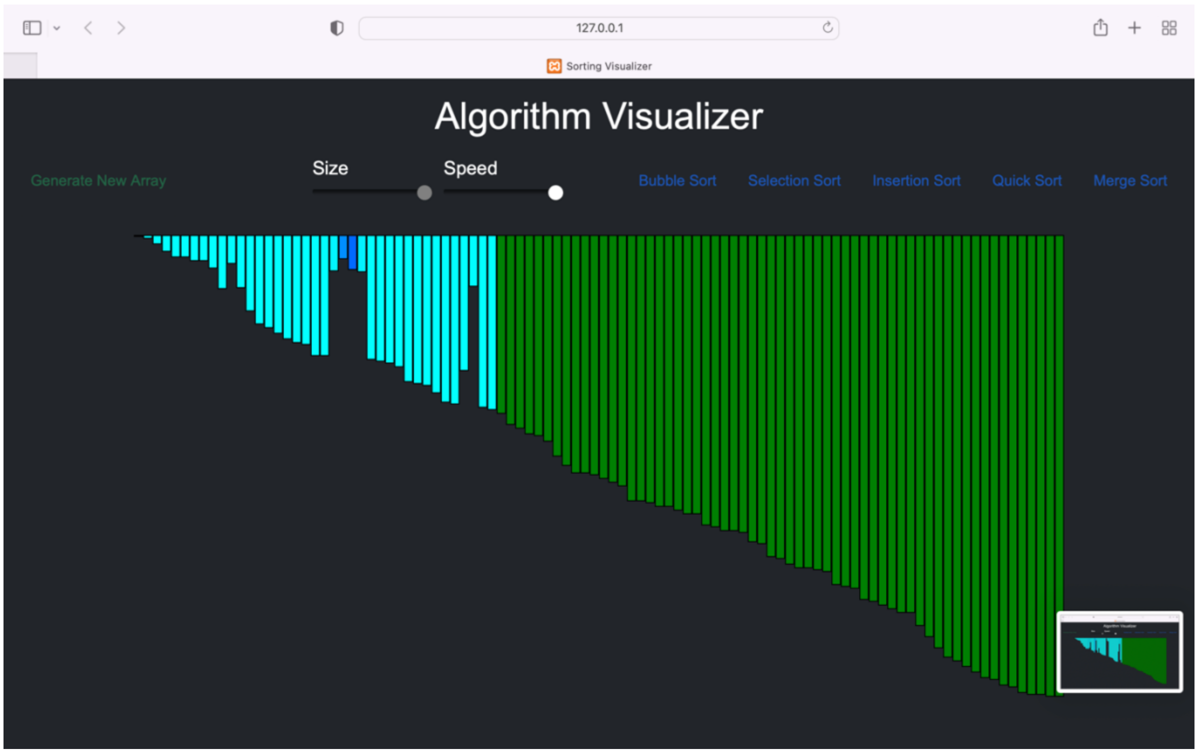
Step-2


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

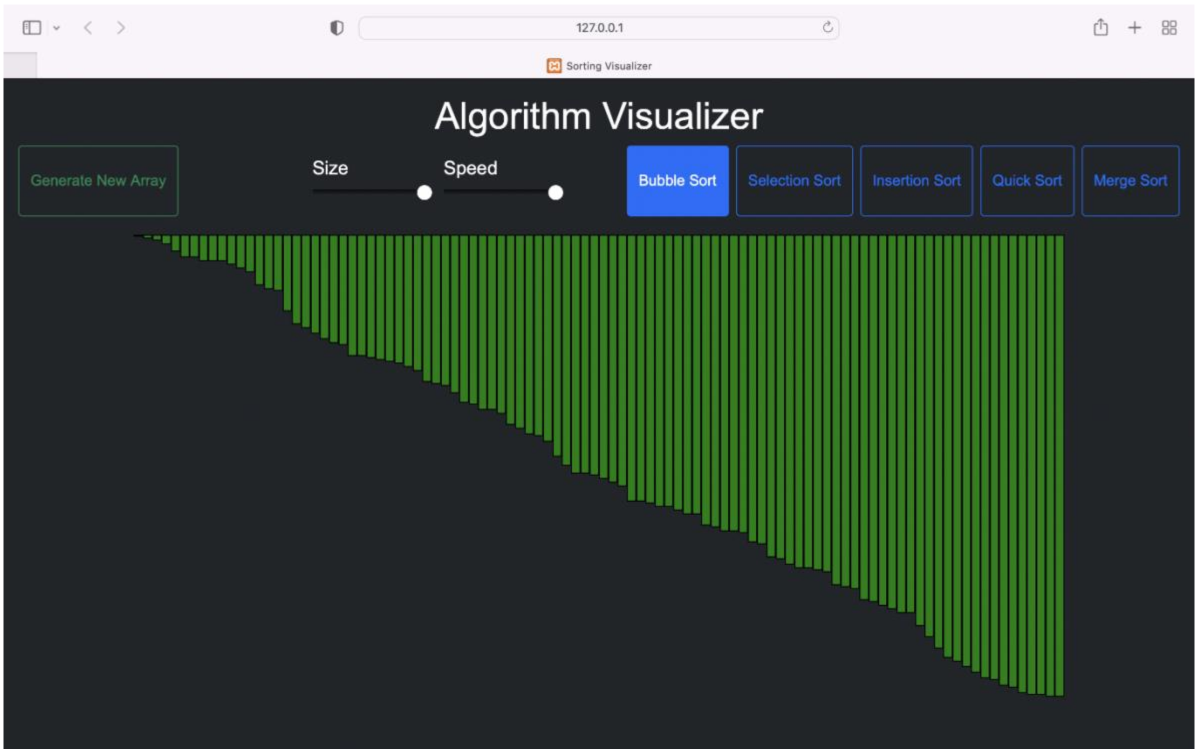
Step-3


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-4



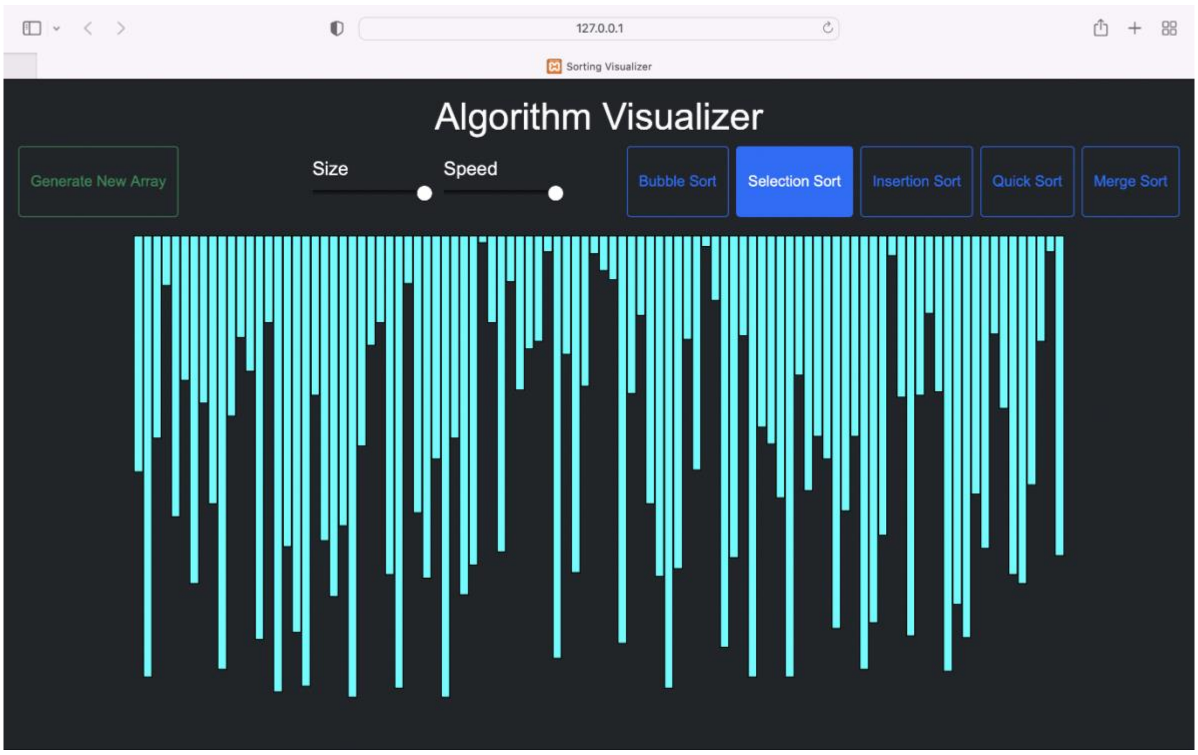
Step-5



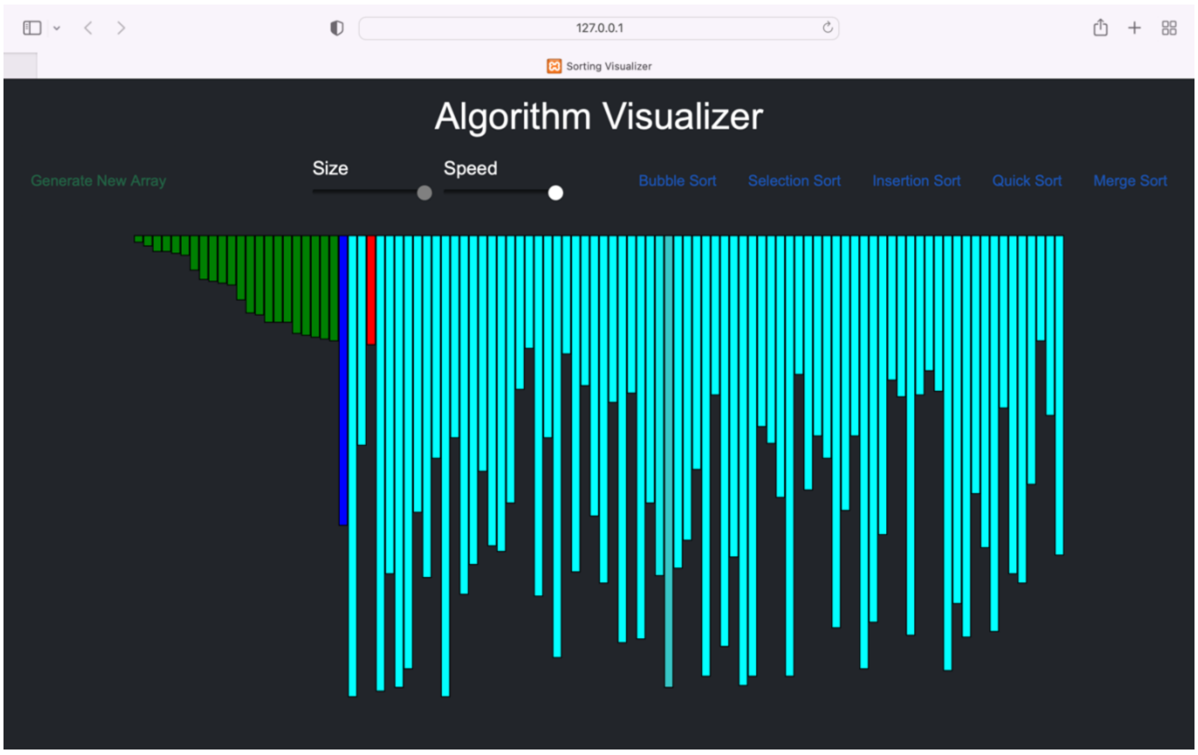
SORTED!!!

SELECTION SORT

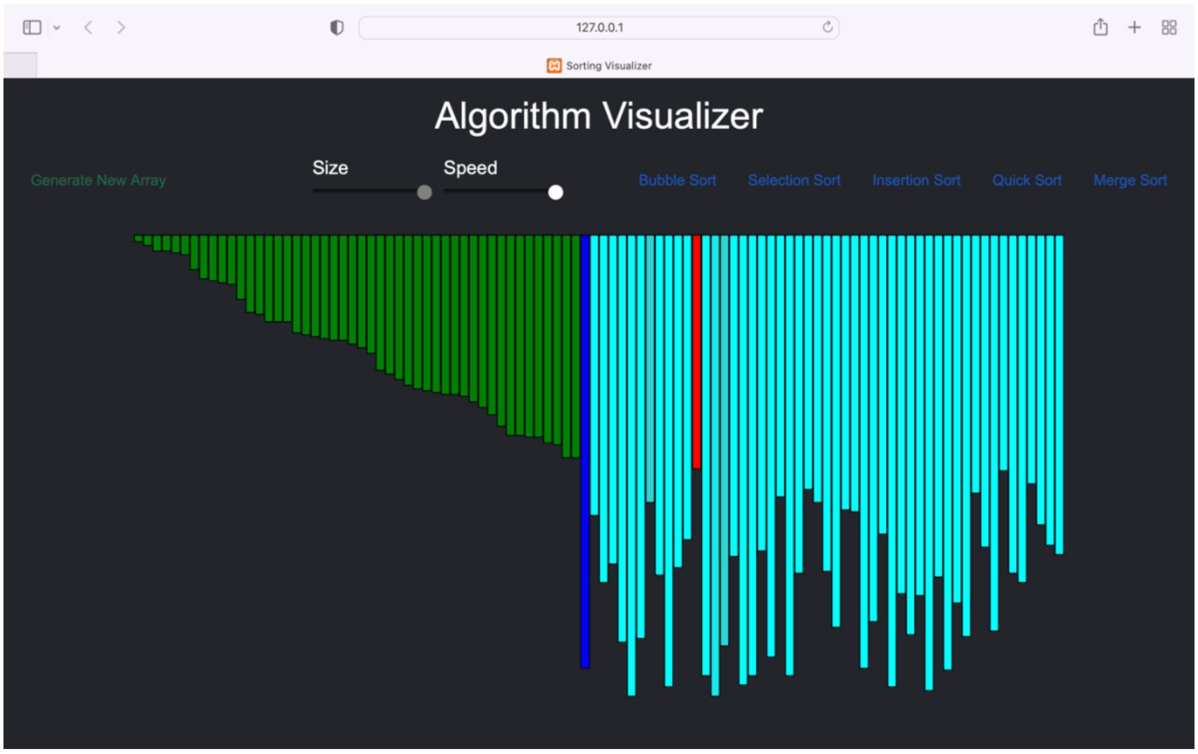
Step-1



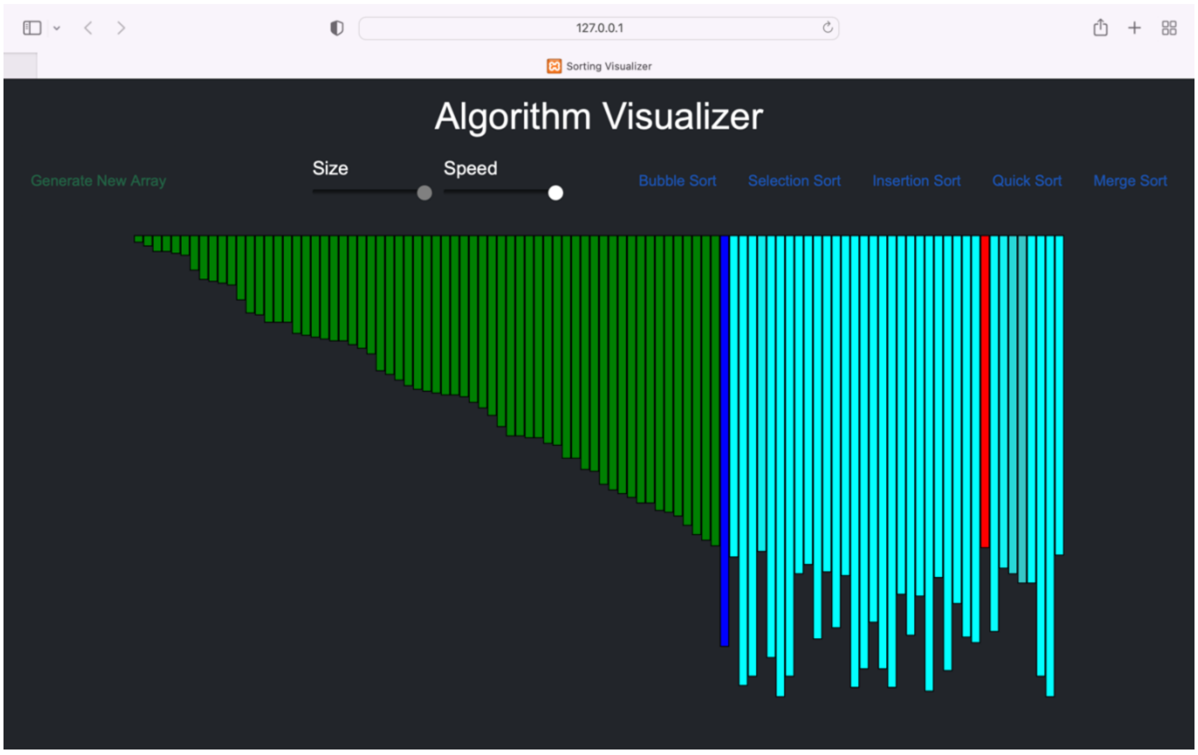
Step-2



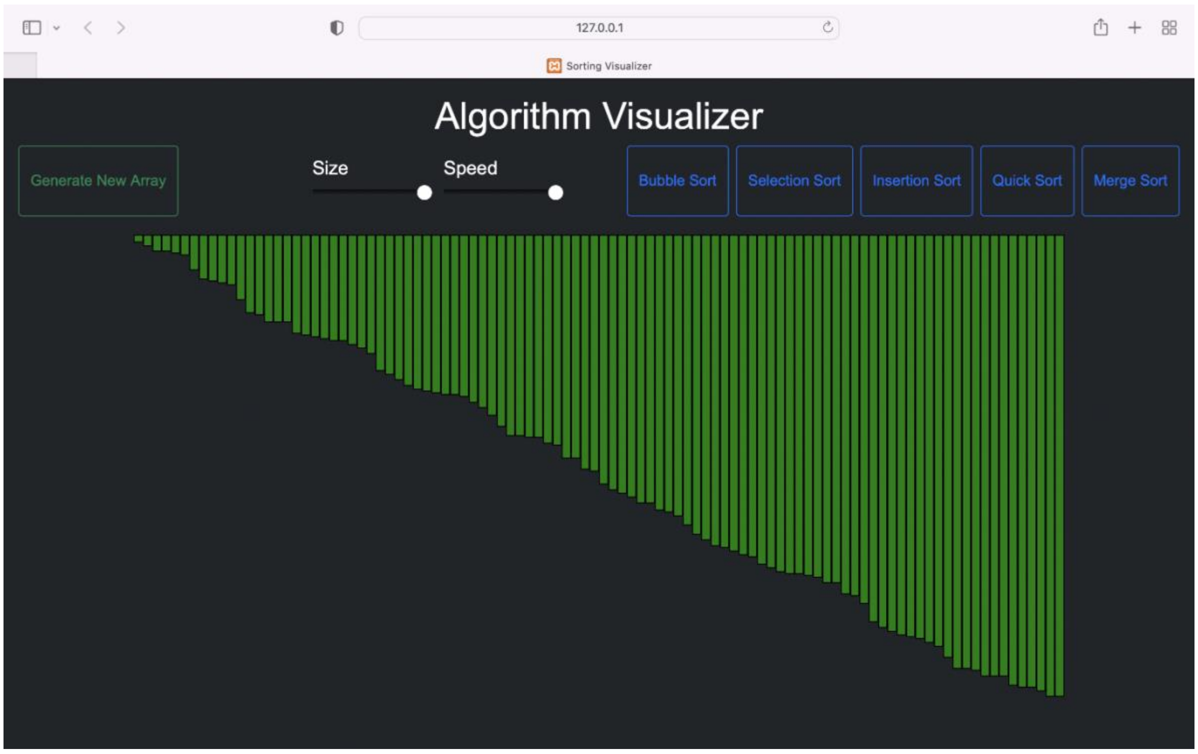
Step-3


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-4


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

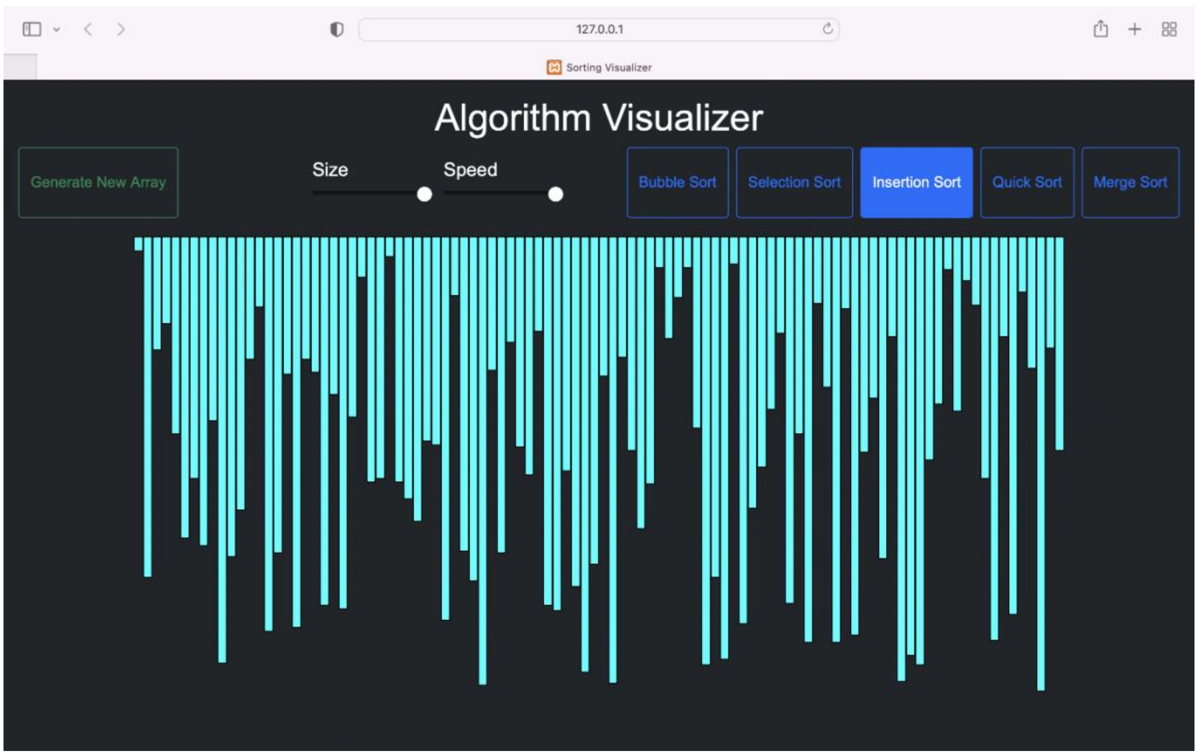
Step-5



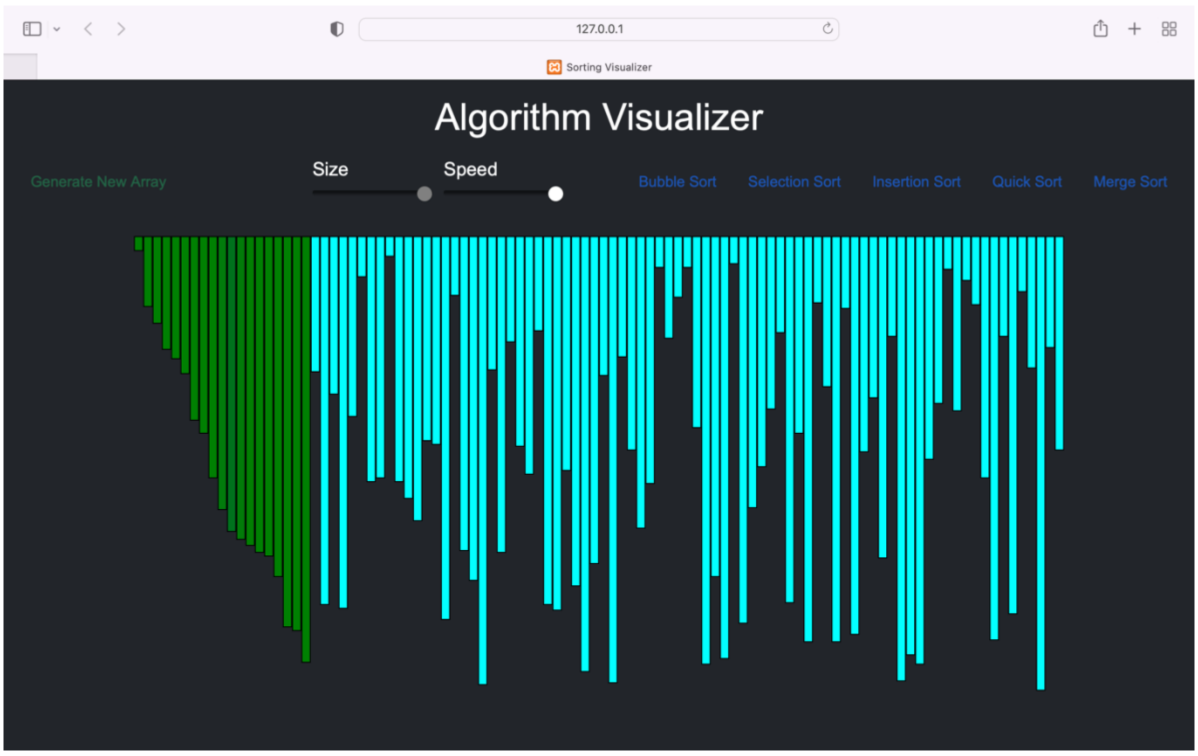
SORTED!!!

INSERTION SORT

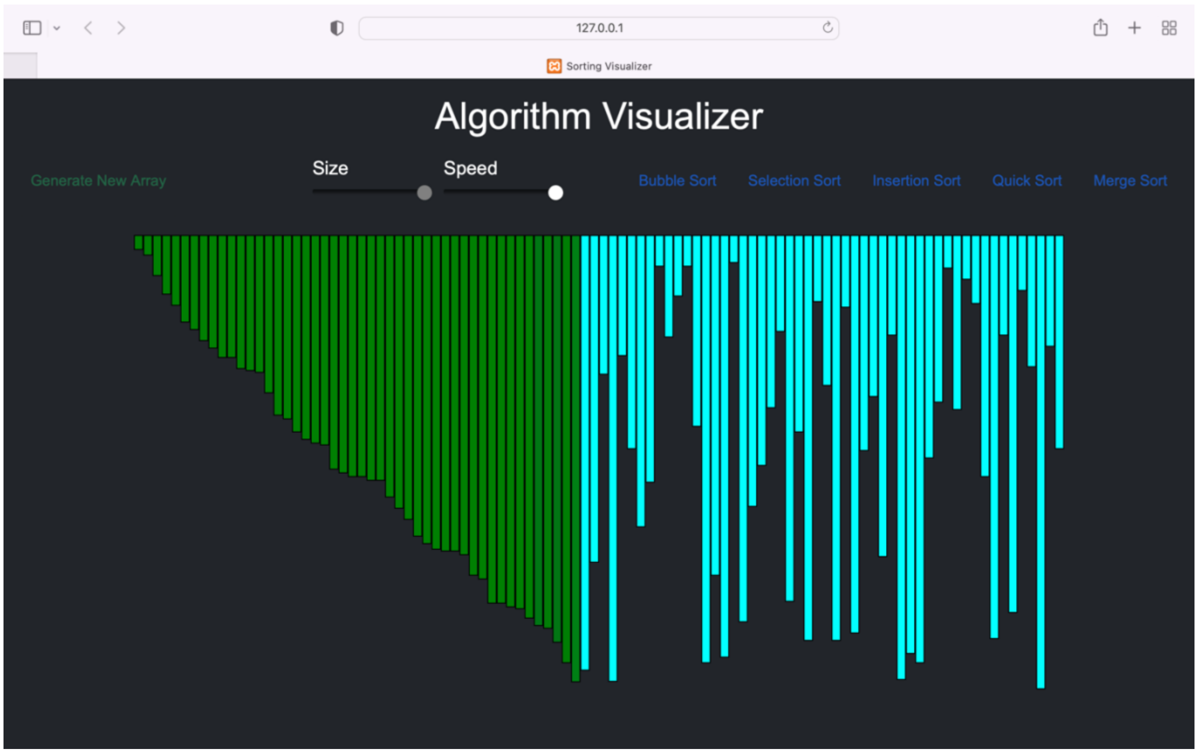
Step-1



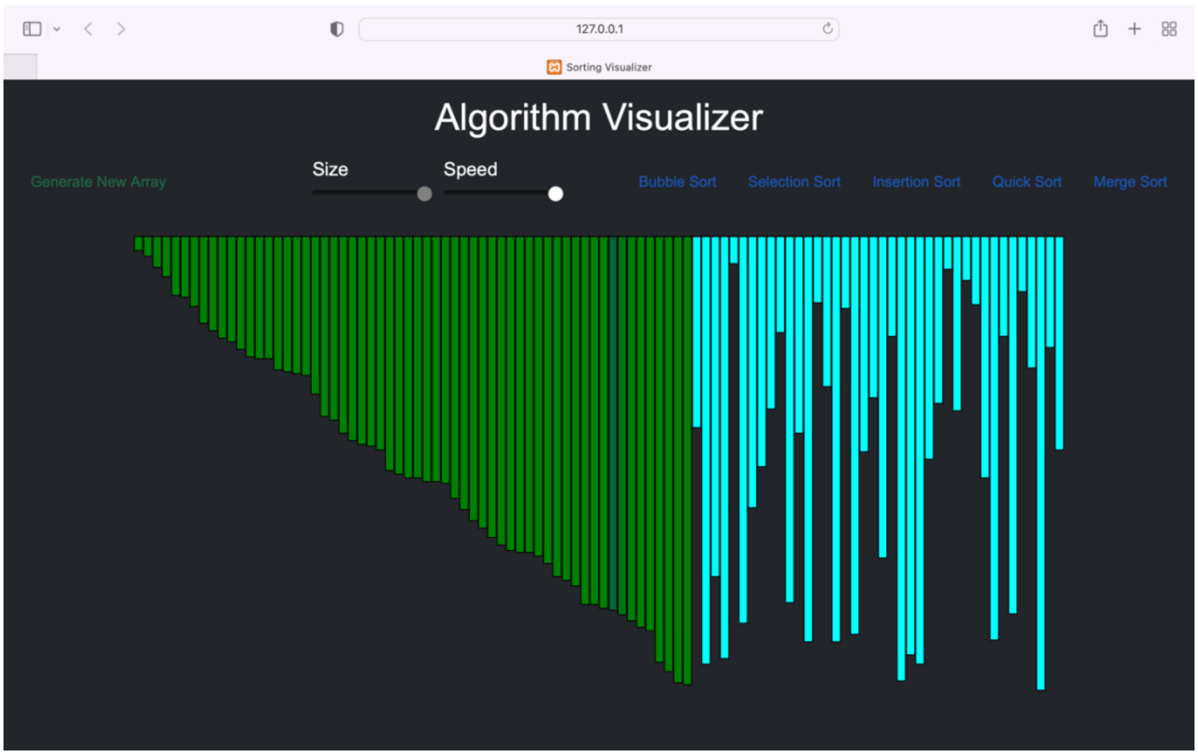
Step-2



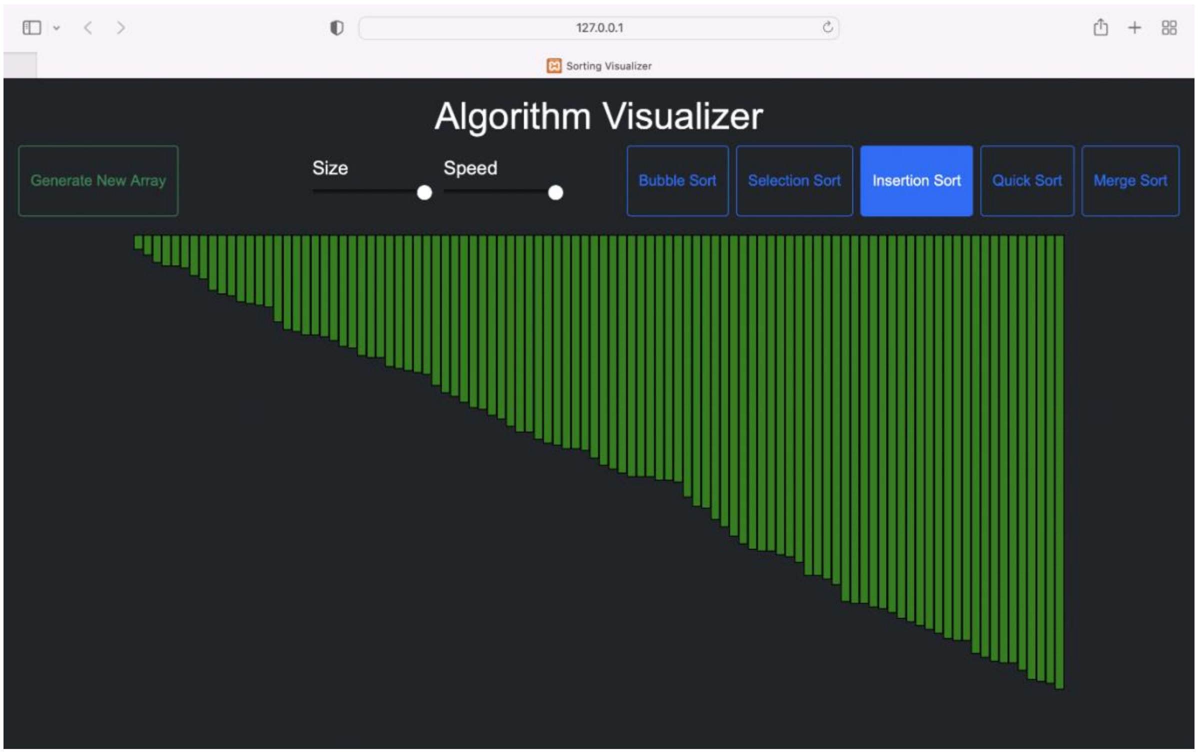
Step-3


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-4


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

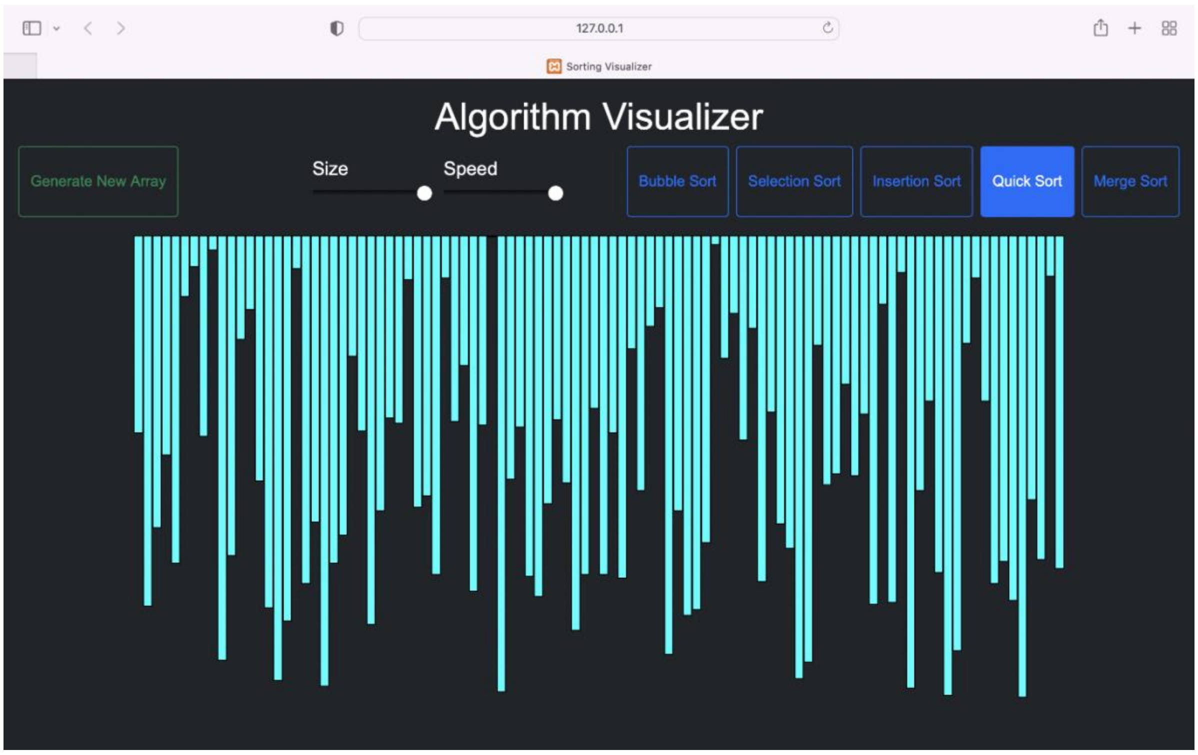
Step-5



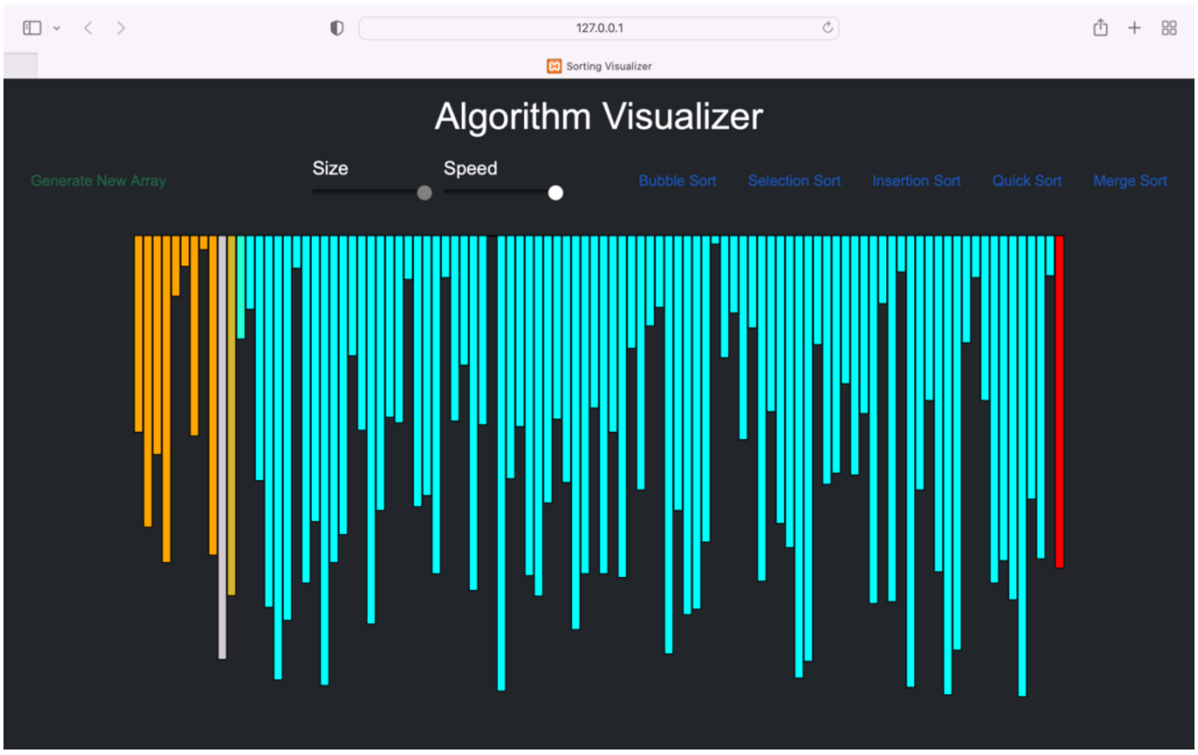
SORTED!!!

QUICK SORT

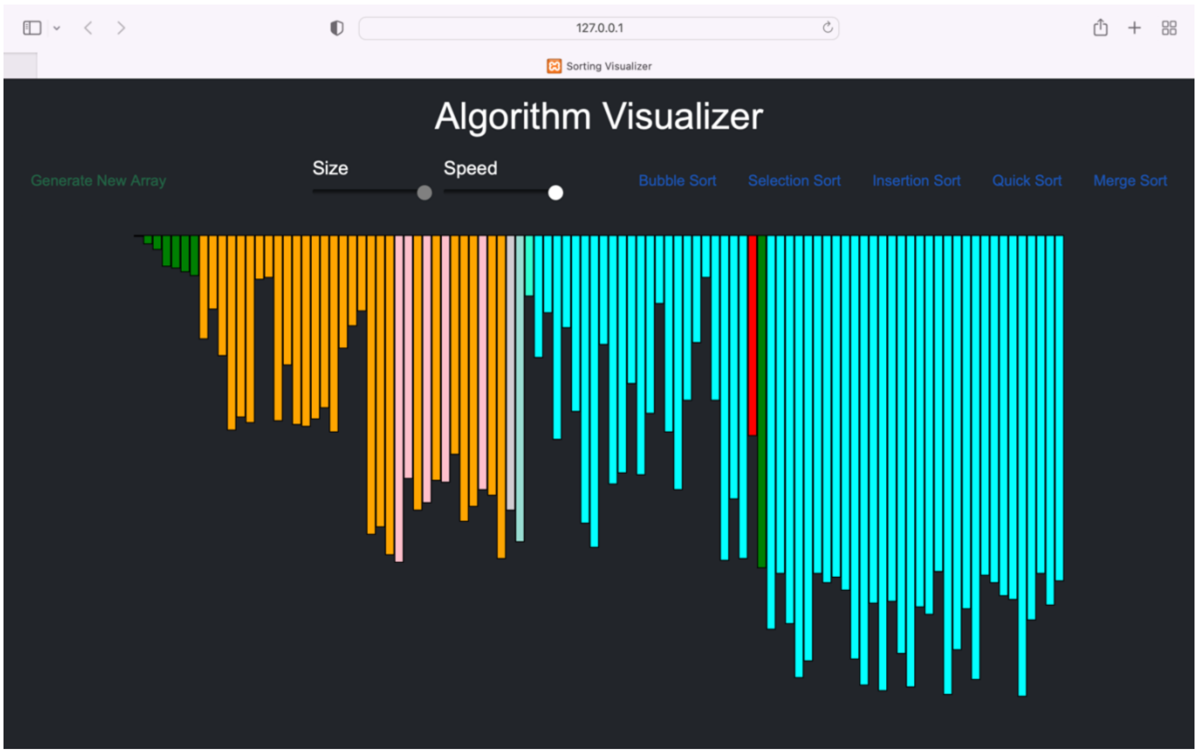
Step-1



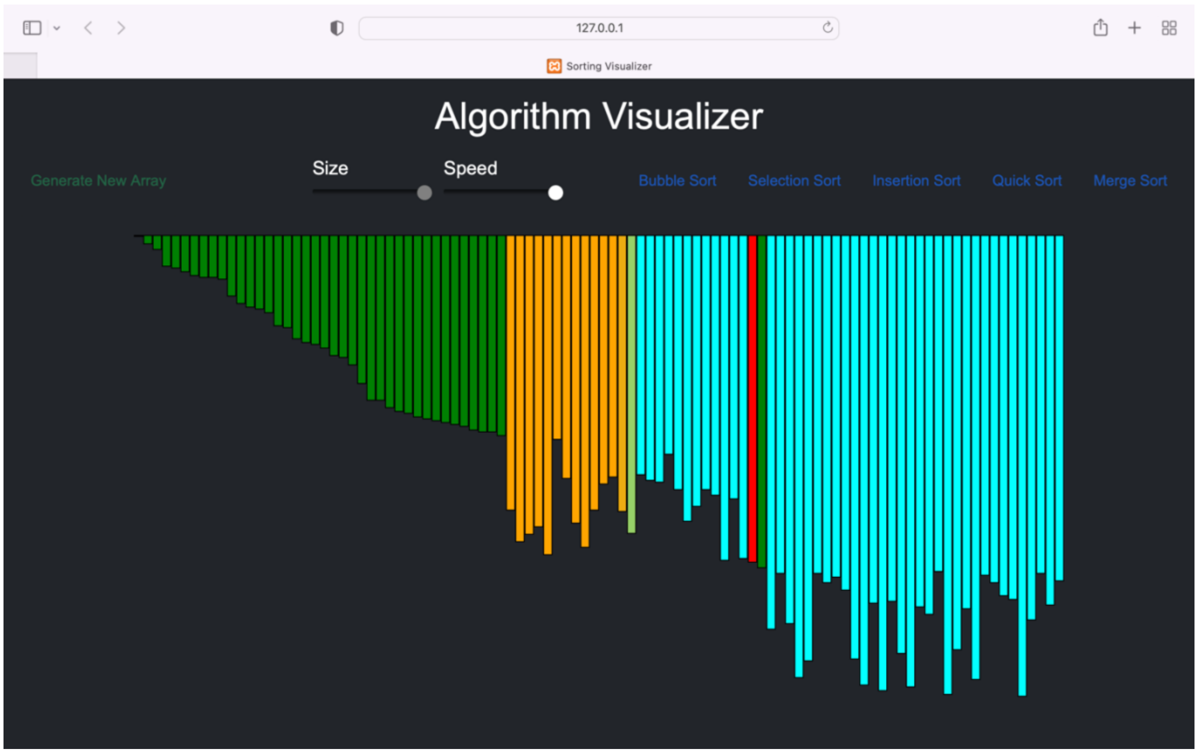
Step-2



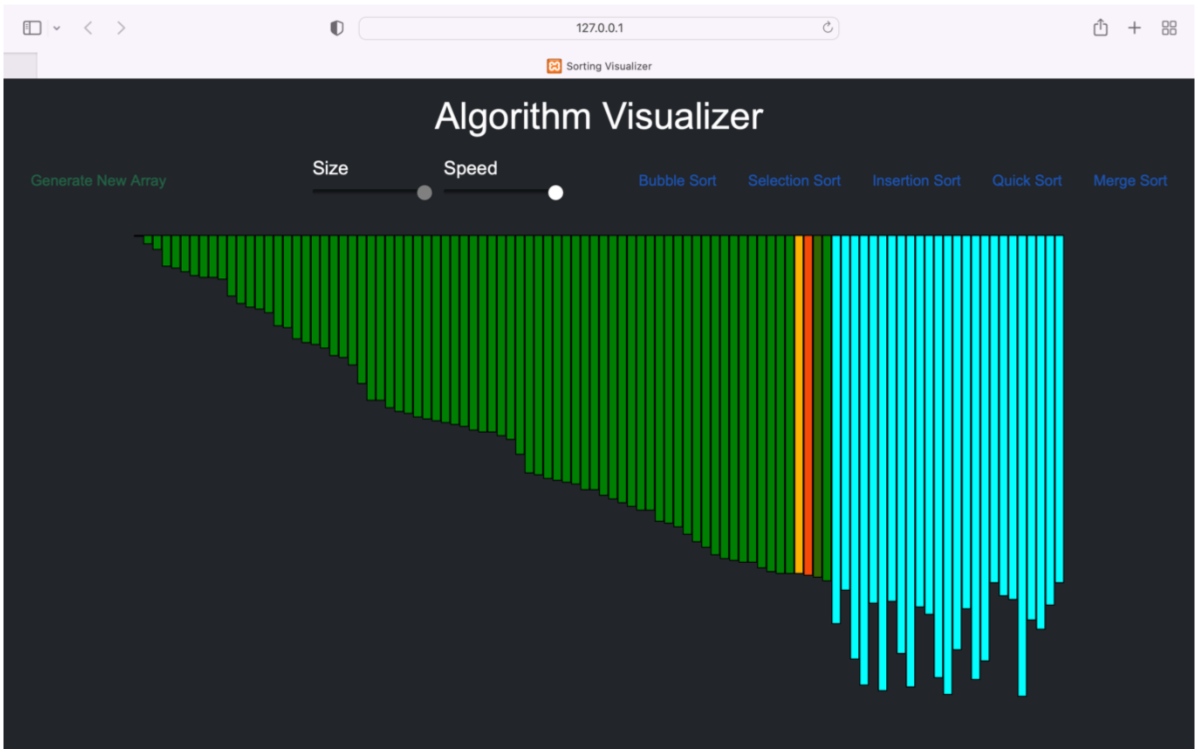
Step-3


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

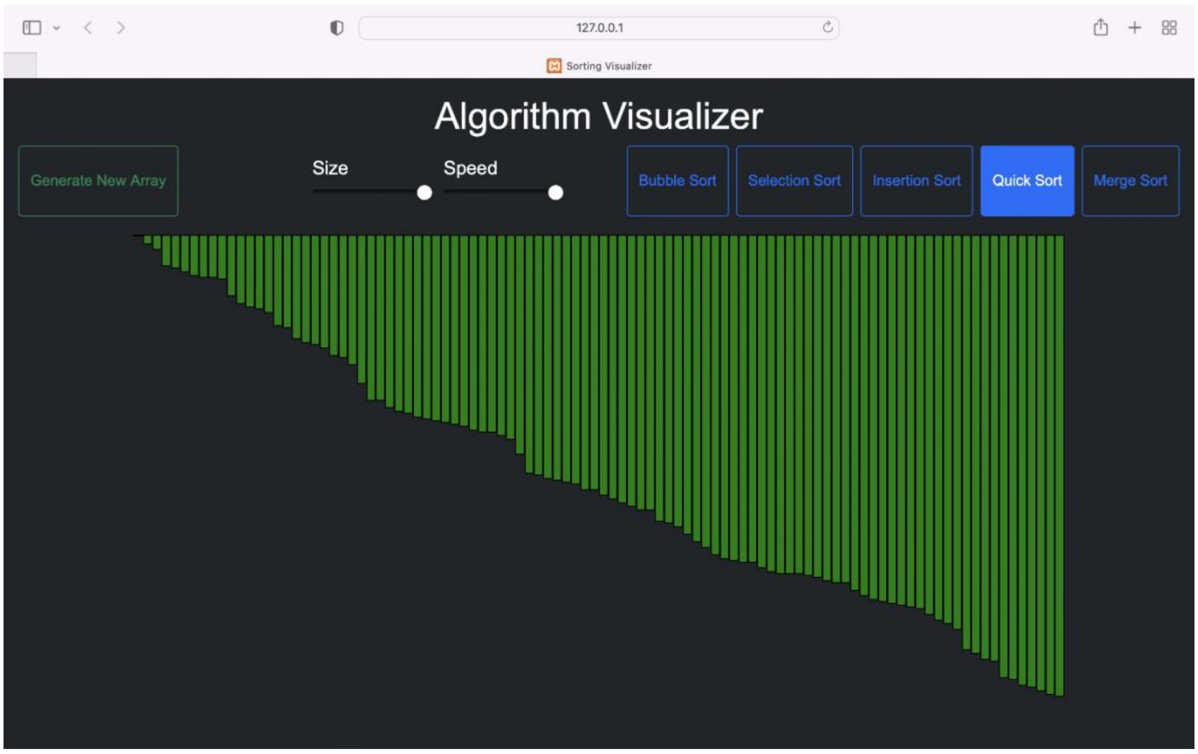
Step-4


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-5



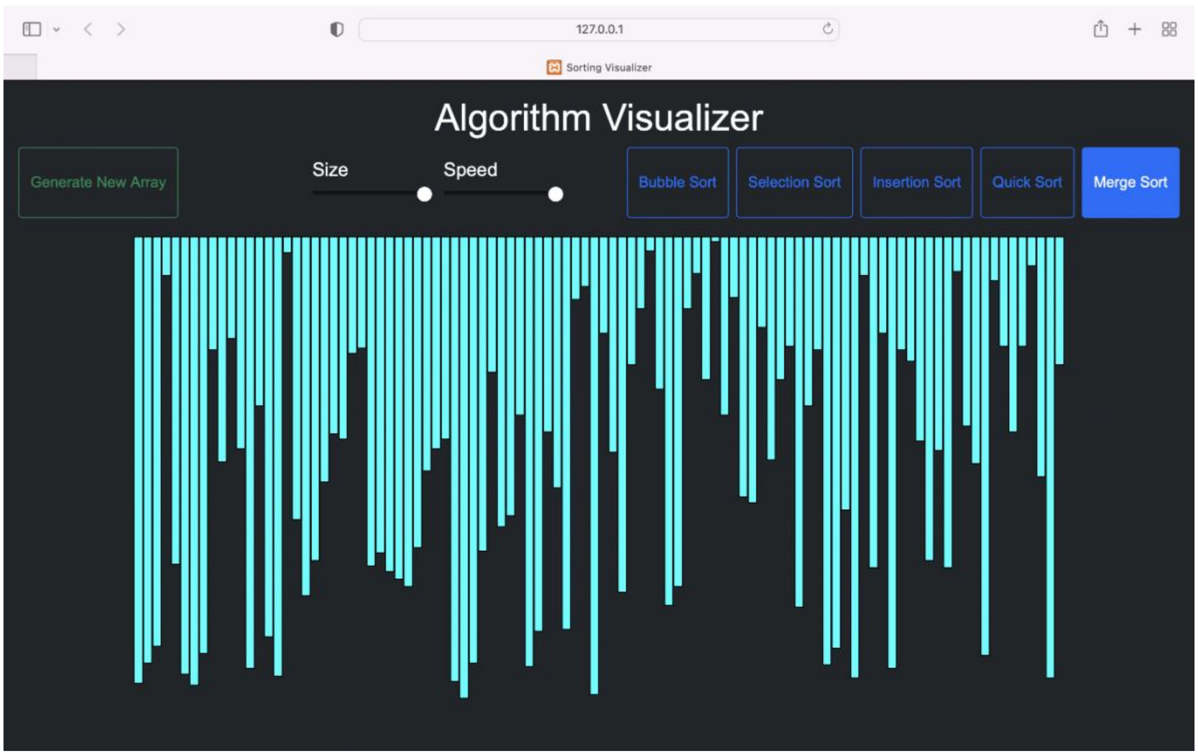
Step-6



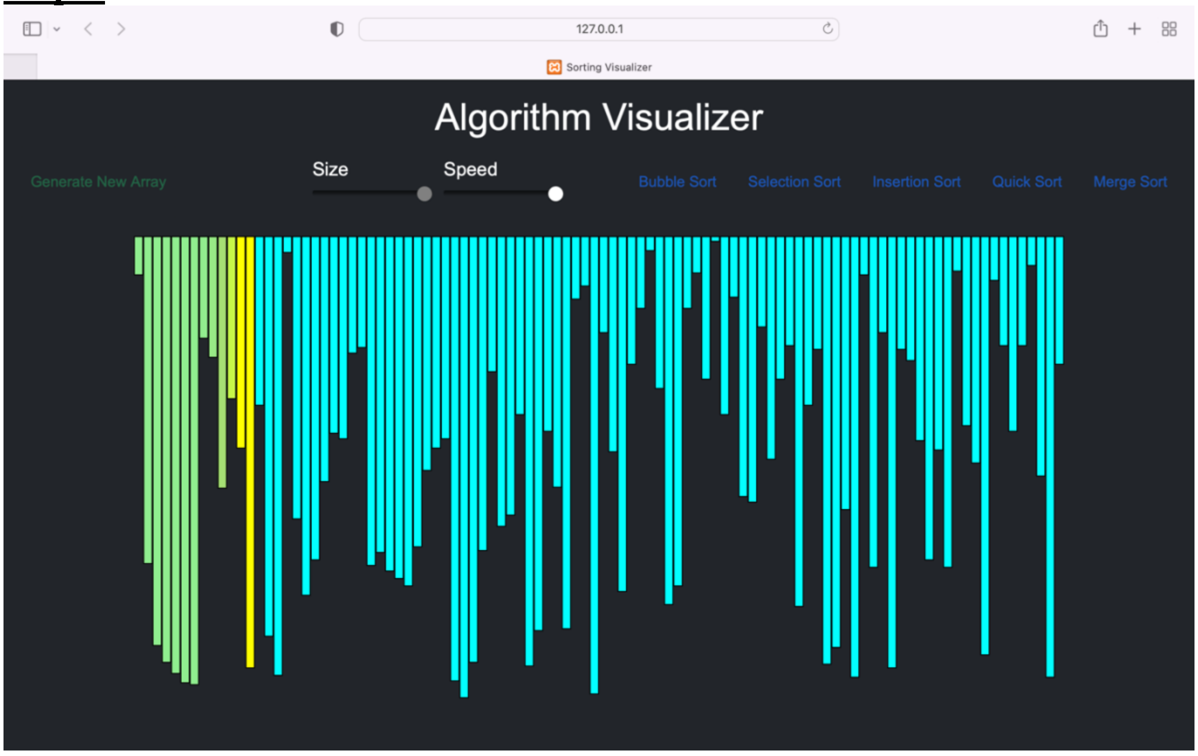
SORTED!!!

MERGE SORT

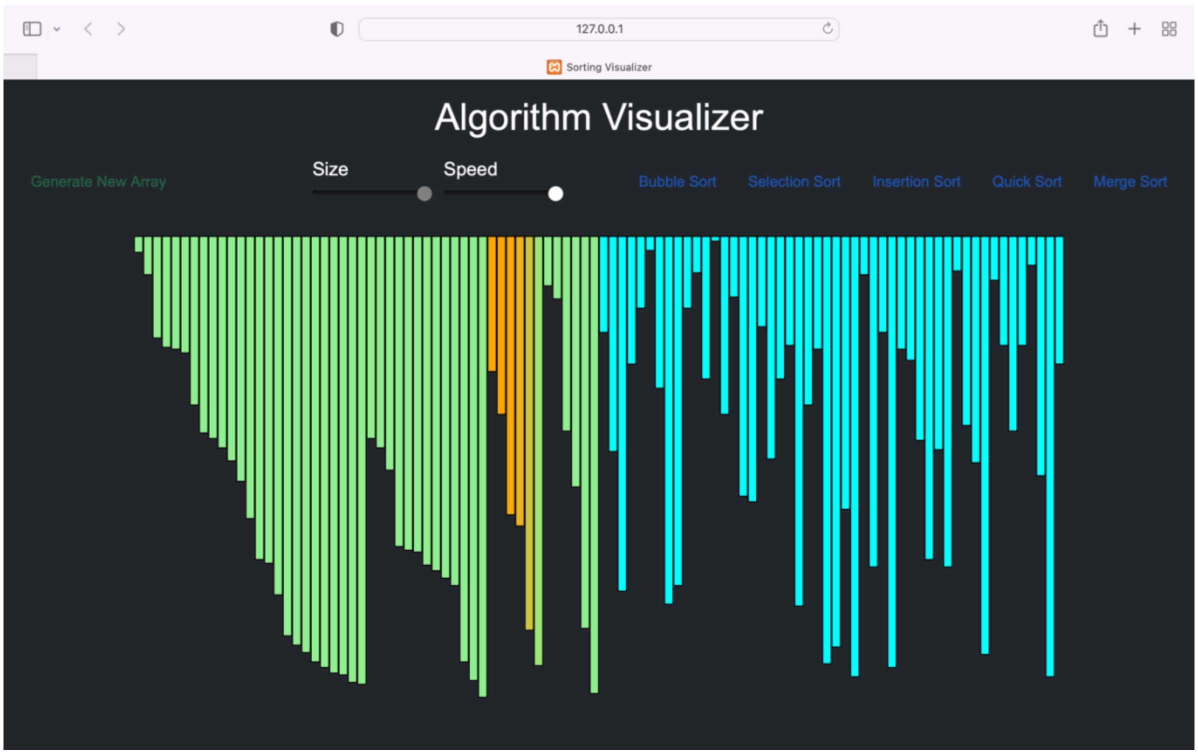
Step-1



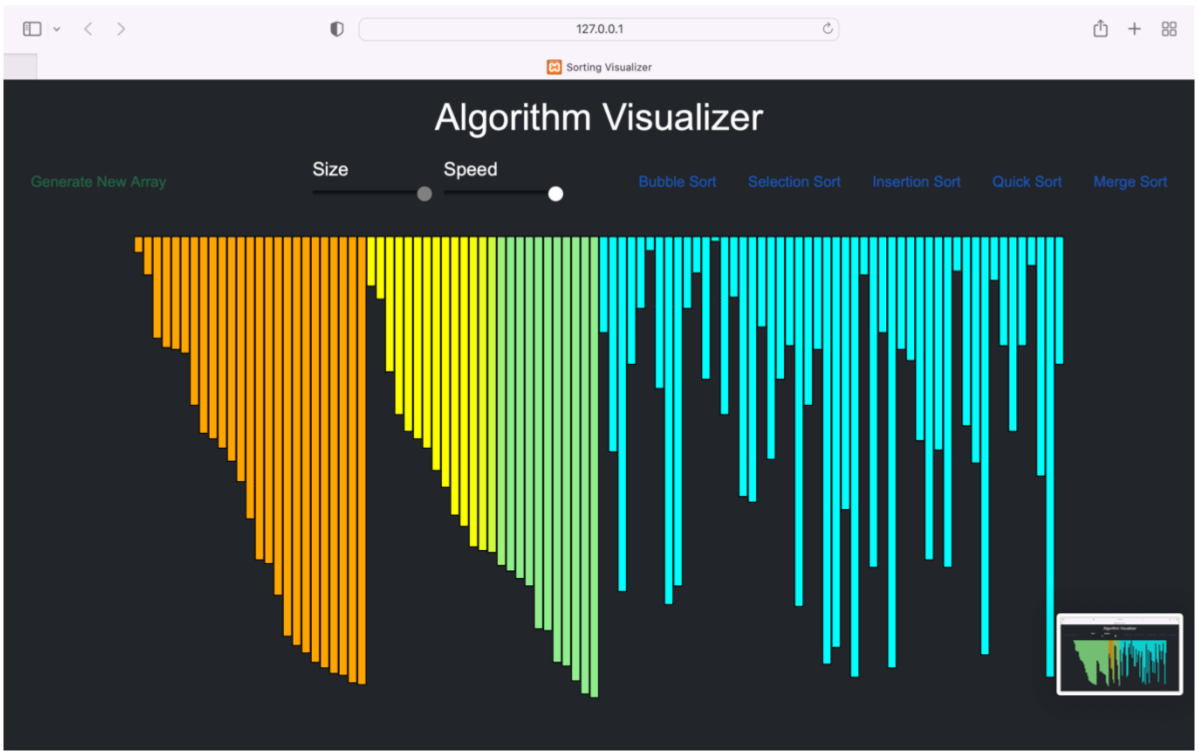
Step-2



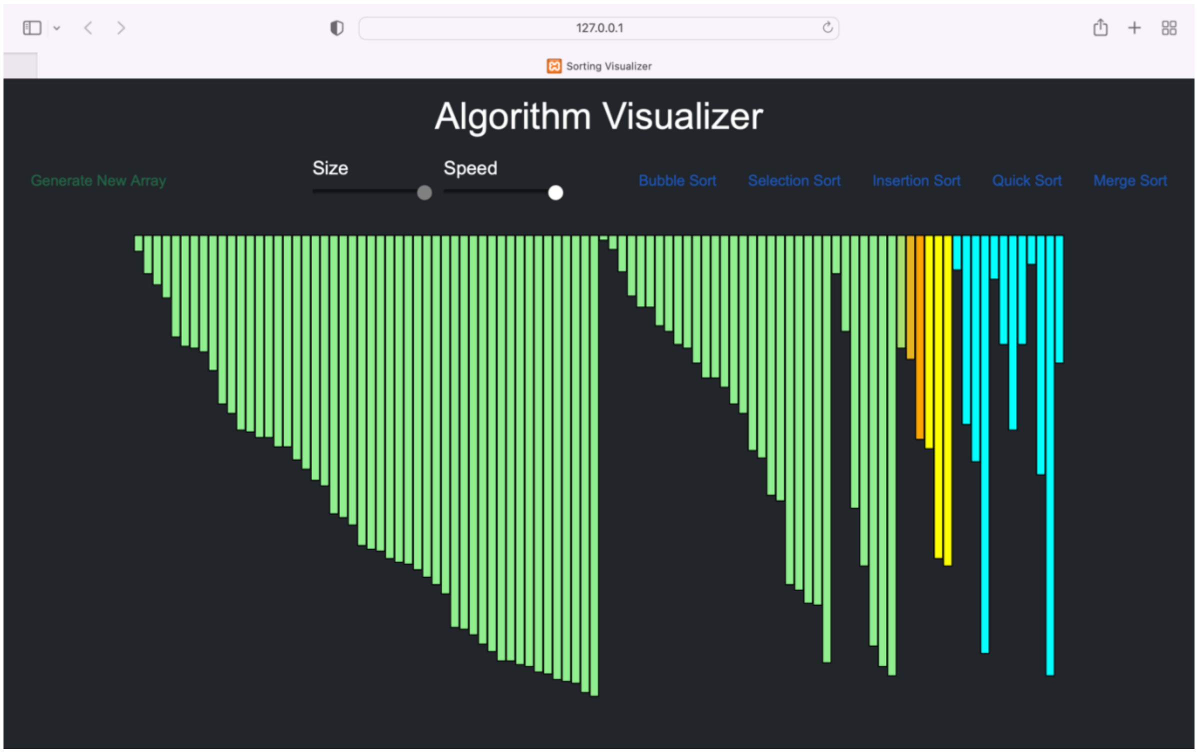
Step-3


127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-4

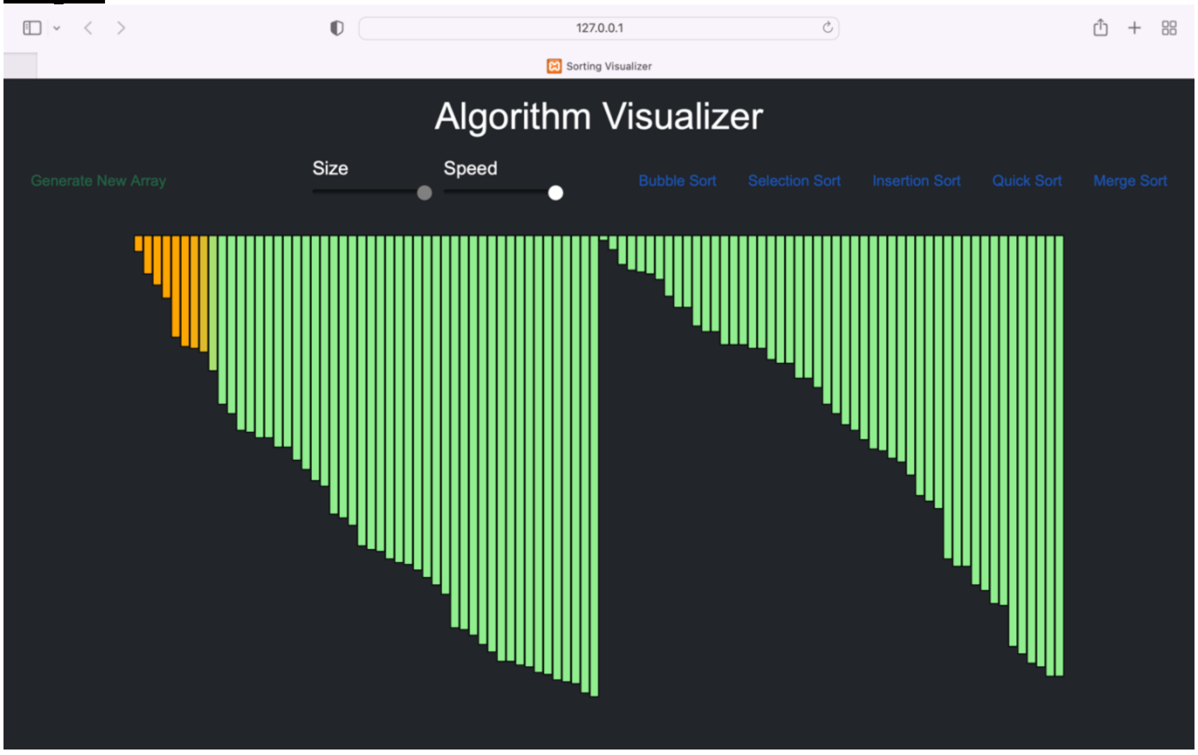

127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-5



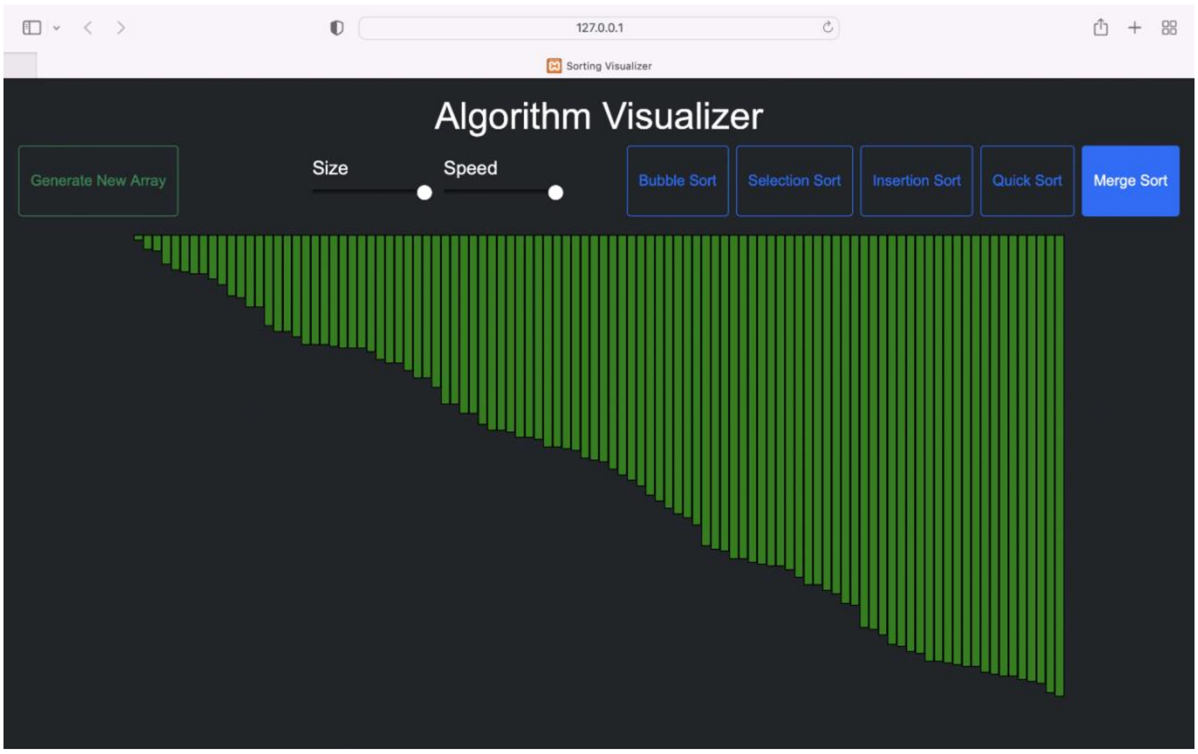
127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-6



127.0.0.1
Sorting Visualizer
Algorithm Visualizer
Generate New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort

Step-7



SORTED!!!

LIMITATIONS OF THE PROJECT

1. Algorithm visualizations can help understanding the principles, but do not replace the need to implement algorithms by students is a chosen programming language.

2. Another drawback of using algorithm visualizations within our subject is the lack of the tool offering required visualizations in a single package with the unified interface.

3. This Algorithm Visualizer contains only one among all types of computer algorithms called Sorting Algorithms. There are many of them of : Backtracking, Brute-Force, Branch and Bound, Divide and Conquer, Greedy, Simple Recursive, Dynamic, Randomized- algorithms.

CONCLUSION

This web-based animation tool called as Algorithm Visualiser, for viewing the following sorting algorithms functions in great part because of all the time and effort that I invested into it : Bubble; Selection; Insertion; Quick; Merge Sort.
While many good algorithm visualizations are available, the need for more and higher quality visualizations continues. There are many topics for which no satisfactory visualizations are available. Yet, there seems to be less activity in terms of creating new visualizations now than at any time within the past ten years. On the other hand, the theoretical foundations for creating effective visualizations appear to be steadily improving.
More papers are appearing regarding effective use of visualization in courses, and a body of work has begun to form regarding how to develop effective visualizations in the first place. Collectively, the community is learning how to improve. While more fundamental research on how to develop and use algorithm visualizations is necessary, we also simply need more implementors to produce more quality visualizations. And we need to encourage them to provide visualizations on under-represented topics.

BIBLIOGRAPGY AND REFERENCE

1. Reference Book: Algorithm Design: Foundations, Analysis, and Internet Examples, Authored by Roberto Tamassis, published by Wiley.
2. Research Paper on "A Meta-Study of Algorithm Visualization Effectiveness" CHRISTOPHER D. HUNDHAUSENn , SARAH https://www.cc.gatech.edu/~stasko/papers/jvlc02.pdf
3. Research Paper on Easy Chair Preprint No.5385, Algorithm Visualizer, Ashwani Kumar Singh, Danish Jamal and Pranjal Aggarwal https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiZq7jEpKLzAhUZ5nMBHVcCCOoQFnoECBAQAQ&url=https%3A%2F%2Feasychair.org%2Fpublications%2Fpreprint_download%2Ftw6K &usg=AOvVaw0bZunt72S0AKEJFLpws-s4
4. Honours Project Overview, Visualizing Sorting Algorithms, Brian Faria

   https://digitalcommons.ric.edu/honors_projects/?utm_source=digitalcommons.
   ric.edu%2Fhonors_projects%2F127&utm_medium=PDF&utm_campaign=PD FCoverPages
5. Sorting Algorithms- Wikipedia https://en.wikipedia.org/wiki/Sorting_algorithm
6. Algorithm Visualization: A Report on the State of the Field Clifford A. Shaffer Department of Computer Science Virginia Tech Blacksburg, VA; Matthew Cooper Department of Computer Science Virginia Tech Blacksburg, H. Edwards Department of Computer Science Virginia Tech Blacksburg, https://people.cs.vt.edu/shaffer/Papers/Cooper07.pdf
7. Using algorithm visualizations in computer science education Research Article Slavomír Šimonák ˇ file:///Users/tanujaiswal/Downloads/Using_algorithm_visualizations_in_computer_science.pdf
8. SORTING ALGORITHM VISUALIZER Shubham Nath*1, Jatin Gupta*2, Abhinav Gupta*3, Teena Verma*4 https://www.irjmets.com/uploadedfiles/paper/volume3/issue_7_july_2021/14236/1628083553.pdf