DECLARATION

We hereby declare that the project entitled **"Sorting Visualizer"** submitted for 6th Semester Project work as per MSBTE Curriculum of Computer Technology Course and the project has not formed the basis for the award of any degree, fellowship or any other similar titles.

Place: Almala

Date:

1) Mr. Kulkarni Gandharv N. 1910950187

2) Mr. Shete Chinmay A. 1910950163

ACKNOWLEDGEMENT

We would regard this project as the culmination of efforts put by various persons during this academic year. We express the whole hearted thanks to our guide **Prof. Sanghita Deb** for such priceless and affectionate guidance throughout the project, without which this report would only be a dream.We also express our sincere thanks to Head of Department Computer Technology **Prof. Ambulge S.S** for providing all necessary prerequisites. We express our deepest regards towards the staff members and friends for their constant support. We express whole hearted thanks to our principal of VAPM, Almala College **Prof. Dharashive P.S** for providing all necessary infrastructures, labs, etc.

By:

1) Mr. Kulkarni Gandharv N. 1910950187

2) Mr. Shete Chinmay A. 1910950163

CONTENTS

CONTENT 1

1. PROJECT INTRODUCTION:

1.1 Introduction:

Sorting algorithms are used to sort a data structure according to a specific order relationship, such as numerical order or lexicographical order.

This operation is one of the most important and widespread in computer science. For a long time, new methods have been developed to make this procedure faster and faster.

There are currently hundreds of different sorting algorithms, each with its own specific characteristics. They are classified according to two metrics: space complexity and time complexity.

Those two kinds of complexity are represented with asymptotic notations, mainly with the symbols O, Θ, Ω, representing respectively the upper bound, the tight bound, and the lower bound of the algorithm's complexity, specifying in brackets an expression in terms of n, the number of the elements of the data structure.

Most of them fall into two categories:

- Logarithmic

The complexity is proportional to the binary logarithm (i.e., to the base 2) of n.

An example of a logarithmic sorting algorithm is Quick sort, with space and time complexity O(n × log n).

- Quadratic

The complexity is proportional to the square of n.

An example of a quadratic sorting algorithm is Bubble sort, with a time complexity of O(n2).

Space and time complexity can also be further subdivided into 3 different cases: best case, average case and worst case.

Sorting algorithms can be difficult to understand and it's easy to get confused. We believe visualizing sorting algorithms can be a great way to better understand their functioning while having fun!

Learning is a necessity in life, since his birth till the ends. As a human, we learn to be able to achieve our independence and to adapt to various environmental changes. Nowadays,

due to globalization and proliferation of technology, computer education in Indonesia becomes great demand for students. One of the favourite subjects in computer education is Algorithm and Programming, which is now held in many courses as main subject. The subject is studied by first grade students, because it as the basic knowledge to be implemented in computer programming. Students have to understand algorithm clearly if they want to be a good programmer. Sorting is a common algorithm studied in information technology, computer science, and engineering. In computer science particularly algorithm, selection sorting algorithm is an algorithm for sorting a series of data. This concept is difficult to be understood for students who study computer science, especially creating coding in programming language. Therefore, one of the solutions is to provide the students algorithm visualization. Many of the phenomena treated in engineering are dynamic and/or three dimensional; often these phenomena are very difficult to be represented in the conventional print media, however it is anticipated that visualization has the capacity to do a much better job. Visualization is gaining its popularity to implement in education institutions either in distance learning or in blended learning. The development of algorithm visualization should be able to observe the condition of student concerned, due to the changes in paradigm of learning which is from teacher-centred learning to student-centred learning.

Algorithm visualization uses computer graphics or multimedia to show the actions of an algorithm step by step. In this research, the algorithm visualization is used to help students understand the concept of selection sort algorithm and creating coding to visualize it. Our goal is to develop algorithm visualization of Selection Sorting Algorithm. The visualization shows how all data move to the proper position in order to be sorted. It can be easily understood by students how the algorithm should be implemented in coding.

What is SORTING?

In computer science and mathematics, a sorting algorithm is an algorithm that puts elements of a list in a certain order [3]. The most-used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) that require sorted lists to work correctly; it is also often useful for canonicalizing data and for producing human-readable output. More formally, the output must satisfy two conditions: (1) The output is in non-decreasing order (each element is no smaller than the previous element according to the desired total order); and (2) The output is a permutation, or reordering, of the input. Since the early 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. 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, randomized algorithms, and case analysis.

What is VISUALIZATION?

Visualization is presenting data or information that used in science, engineering, medical, business, and others. Numerical simulations often produce data files that is contains data values. The data are converted to a visual form that is helpful for the user in analysing his/her problem. Visualization for science, engineering, and medical presents graphical form information presentation. And the term business visualization is used in connection with data sets related to business, management, social, industry, and other non-scientific areas.

1.2 Requirement Specification:

First validate the idea by doing a low-level implementation (Proof of concept) of the components involved in the project.

• Get more clarity around the unknowns. E.g.: XML vs HTML and why to use HTML5/CSS3 not the other versions, advantages of using Bootstrap.

• Get a better understanding of the stages involved in the project. E.g.: By doing a proof of concept you will understand that there are multiple stages such as creating the basic layout, styling it and implementing the functionalities.

❖ Software Requirement:

Sr. No	Resource	Quantity
1	VS Code (Editor)	1
2	Brower (e. g. Chrome, Mozilla Firefox, Safari etc.)	1
3	Important Plugins	1
4	Operating System (Windows 7/8/10/11, Linux, MacOS)	1

❖ Hardware Requirement:

Sr. No	Resource	Specification	Quantity
1	Computer or Laptop	1. i3 or above processor 2. 4 GB or above Ram	1

❖ Prerequisite(s):

HTML, CSS, JavaScript (introductory level).

CONTENT 2

2. EXISTING AND PROPOSED SYSTEM

2.1 Project Context:

We have learnt sorting algorithms like bubble sort, selection sort, insertion sort, quick sort. But often we fail to understand the core idea of a particular algorithm maybe because we are unable to visualize how they work. So, the most important thing to understand about these algorithms is visualization.

That's why we are making this project to let everyone understand how these algorithms work and through this project you also will get a deep understanding of such sorting algorithms.

This project will guide you step by step to complete this project and at the end of this project you will have an immense grip on some core concepts of JavaScript as well. Adding this project on your resume will showcase your skills and add a great value to your profile.

This project is a good start for beginners and a refresher for professionals who have dabbled in data structures and algorithms using JavaScript before and also web developers. The methodology can be applied to showcase any algorithm of one's choosing, so feel free to innovate!

2.2 Aim of System:

Sorting Visualizer will be displaying the working mechanism of various sorting algorithms like, Bubble Sort, Selection Sort, Insertion Sort, Quick Sort and Merge Sort. The main objective of developing this Visualizer is to make a learner comfortable in learning these techniques quickly and easily.

2.3 Objectives of System:

Create a web application using HTML, CSS, JavaScript to visualize how various sorting algorithms work. This project's functionality will be similar to this application

2.4 Constraints:

Scope: Sorting Visualizer is a web app for visualizing a bunch of different sorting algorithms Like Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort With the functionality of (Speed Control) and (Array Size Control).

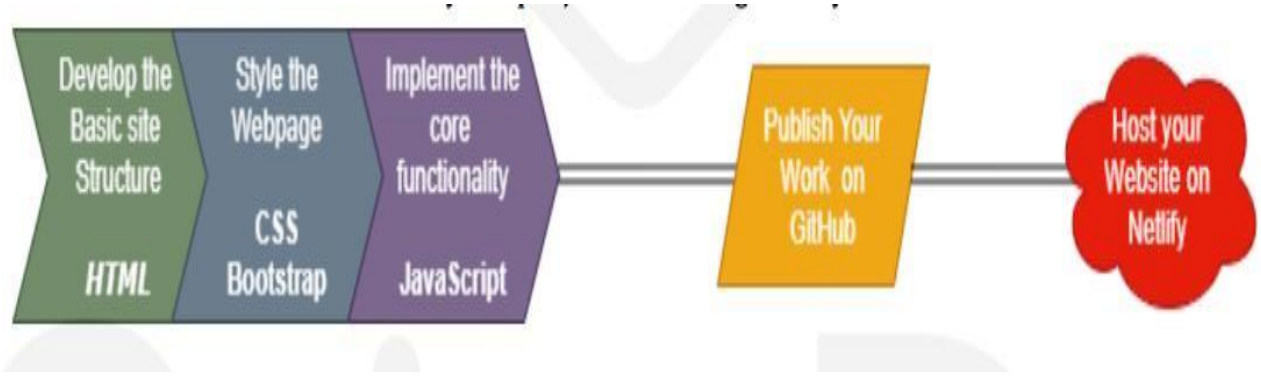
2.5 Complete System Design and Analysis:

- **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.

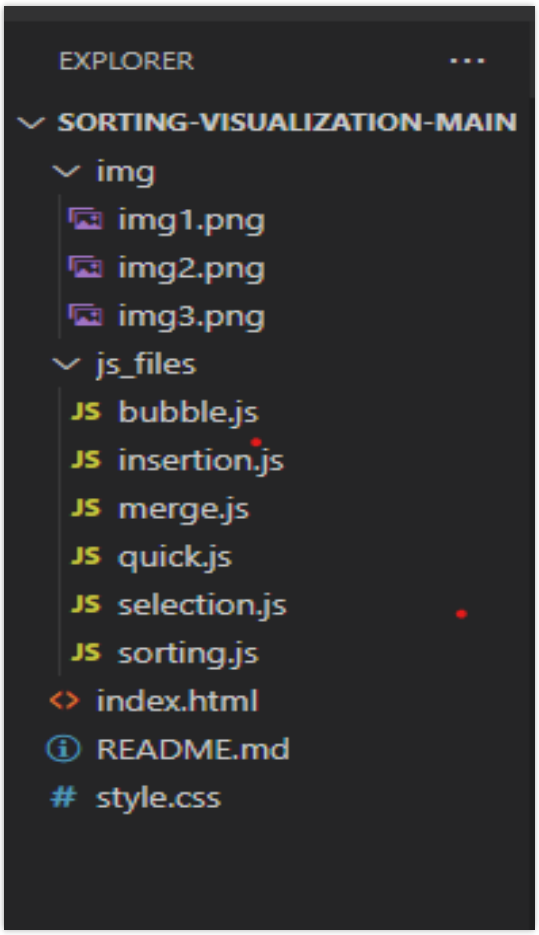
• Publish to GitHub and host your project live using Netlify.



•At the end of this project you will have built an app which would function like this.

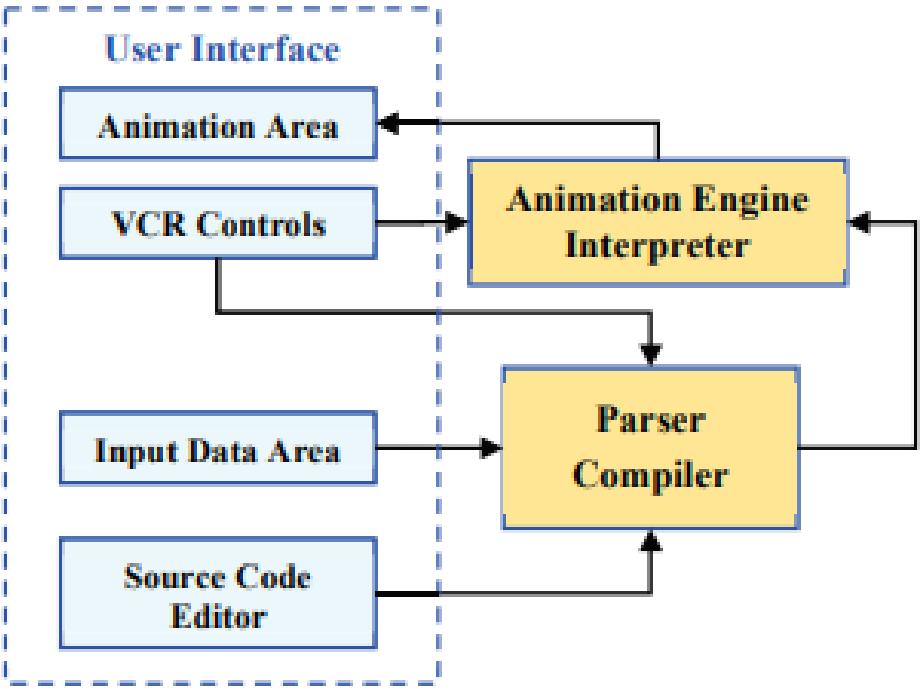
• This is a typical JavaScript Project, so you need a code editor like VScode (recommended), Atom, Sublime text, etc. with necessary plugins.

• Then create an appropriate project folder with essential files. It’s a good practice to follow the suggested file structure (shown below).



2.6 Architecture and implementation:

There are three building components in SORTING VISUALIZER: a) the user interface part, b) the parser/compiler and c) the animation engine. The user interface is actually a simple web page written in HTML5. The animation engine uses JavaScript and the HTML5 Canvas element. We have also used the Code Mirror JavaScript component that provides a code editor in the browser and has a rich programming API and a CSS theme system which makes it highly customizable. The compiler unit of SORTING VISUALIZER receives the source code as an input and produces an intermediate code which is executed by the animation engine. A predefined skeleton code for each algorithm is included in the animation engine. The parser/compiler scans the student's algorithm and fills the gaps in the code. This requires that the student has firstly selected the algorithm to be visualized from SORTING VISUALIZER list. Then he can do any modifications to the predefined code (e.g., a condition statement).



However, the student is not allowed to fundamentally change the nature of the algorithm. This system restriction was set because every algorithm has a unique animation; therefore, the animation engine should "know" in advance the critical parts of the algorithm to be animated. The prototype supports 2D smooth animations of array algorithms. It can easily visualize a new category (family) of algorithms just by adding a new visual data type, in the form of a JavaScript class, using methods which define the interesting events in student's code. In addition, the system can be extended and support other programming languages, like Pascal, C or Python. Therefore, the advantages of SORTING VISUALIZER, comparing to other similar AV systems, can be summarized as following:

• It provides customized visualizations that depend on the nature of the given algorithm.

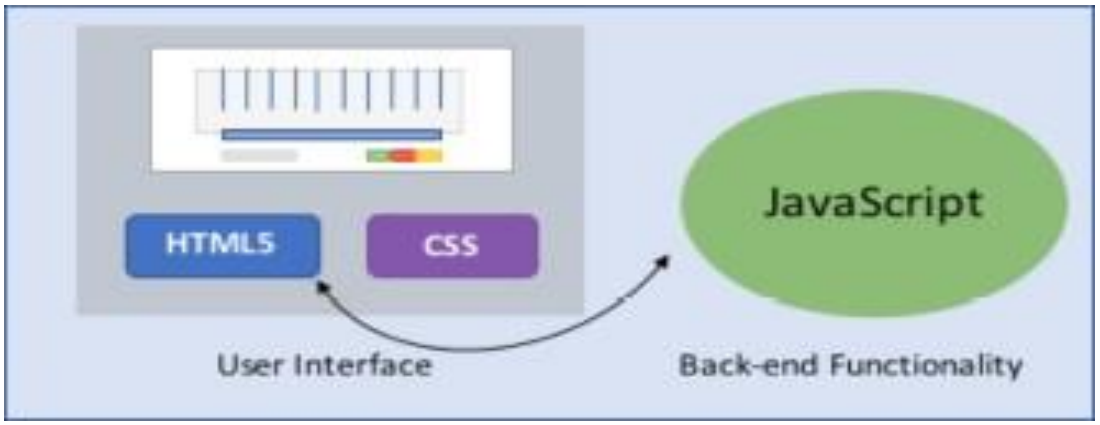
• It allows user to change input data in order to test algorithm behaviour in specific cases.

• It allows modification of the source code and step-by-step experimentation of the corresponding animation.

• It detects some categories of common students' errors, e.g., index value out of array bounds. • It is easy to use and offers full platform compatibility due to the web-based architecture.

2.6.1 System Architecture:

The back-end code is comprised of HTML5, CSS, and JavaScript. All three types of code are contained in one .html file and can be run solely from this file. One of the advantages of HTML 5 is that it is not necessary to include different types of web languages in a single file. Therefore, each type could have been separated, making a total of three files (plus the miscellaneous sound and image files). This is good practice for readability and keeping related code together. However, I decided not to separate the code for two reasons: 1) to increase the portability of the project by only needing to worry about one project file instead of three, and 2) where in the project file, the change in coding languages is distinctly marked and therefore does not significantly reduce readability. Also, 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.

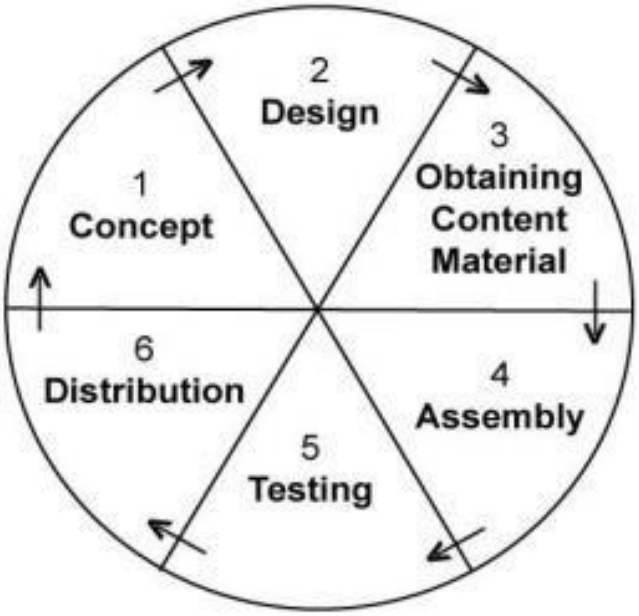


As you can see, there are no major components besides the three coding languages. Most websites have tools or scripts that require a server on the back-end (like PHP), but it is not necessary in this case since JavaScript runs right in the user's browser. HTML5 and CSS are used for the interface. The 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 programming focus 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.

2.7. RESEARCH DESIGN

This section presents the method of developing selection sorting algorithm visualization that is used in this research. This research uses Multimedia Development Life Cycle (MDLC) [9]. Authoring is somewhat like making a feature film, a movie, and there are many steps to the process. Multimedia Development Life Cycle, a typical multimedia systems development, may involve the following six major steps presented in Figure, as follows:

1. Concept. The objective for the project is defined, and the type of the application is specified. In the film production, this is the stage at which the producer decides the kind of application to take and the subject to be.



2. Design. This is the process of deciding in detail what will be in the project and how it will be presented. This stage includes script writing, storyboarding, making navigation structure and some design steps.

3. Obtaining content material. During this stage all the data, audio, video and images for the project are collected in appropriate digital formats. In the course material, this would be the production stage, where all the scenes for the multimedia application are set up.

4. Assembly. In this step, the overall of the project is built, the visualization of selection sorting algorithm is assembled, and any interactive features are built. The tool for this stage of authoring is Adobe Flash.

5. Testing. During testing, the application is run and checked to confirm that it performs exactly what the author has intended. We have performed our experiment in two phases, preliminary testing and main field testing. At the first phase, the system is evaluated by peers and experts. The subjects research for implementation of the revised model in the main field testing are informatics students.

6. Distribution. In this step, the application is reproduced and delivered to students for their use. The distribution can be form application files that can be run on a mobile device.

2.8 MULTIMEDIA

Multimedia is a combination of text, images, sound, animation, and video delivered via computer or electronic and digital equipment. Using together multimedia elements such as images and animation that are equipped with sound, video clips, and text, will be able give clear meaning to those who need it.

The word multimedia is a combination derived from multiple and media. We define digital multimedia as any combination of text, graphic (still and animated), sound, and motion video delivered to the user by a computer. The computer is an intrinsic part of multimedia. All these elements - text, graphics, sound, and video - are either computer-generated or transmitted through a computer. Multimedia systems are used in education, presentations, information kiosks, and gaming industry. The power of computer allows users to interact with the programs. Since interactivity is such a powerful concept, many experts in the field of multimedia consider interactivity as an integral part of multimedia.

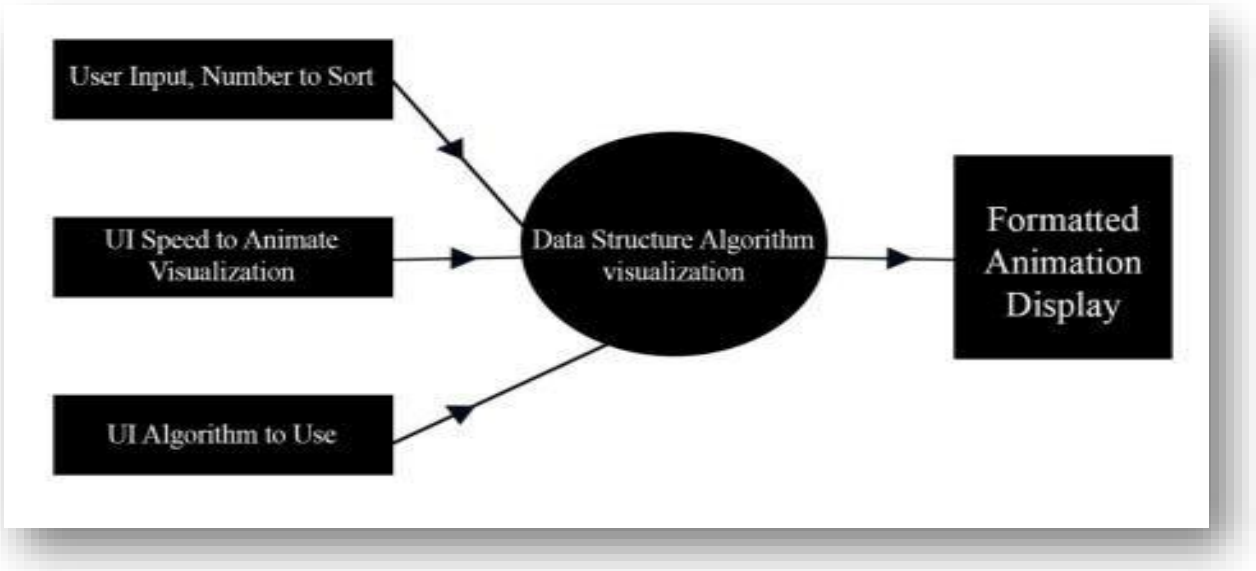
In a multimedia system, if the user has the ability to control the delivered elements and timing, the system is called an interactive system. There are different devices to provide end-user interactivity. Almost all tools today support the use of keyboard and mouse, button, and even touch screen. Buttons are on-screen objects that will produce some response when the end user clicks the mouse or touches them. The pushbutton control in Windows dialogue boxes is an example of a button. Authoring of buttons involves defining the button appearance on screen, the location, and the action when clicked. Assembly tools that support buttons will provide features to do all three things.

CONTENT 3

3. LOGICAL DESIGN OF THE PURPOSED SYSTEM

3.1 Dataflow Diagram:

DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. Data FlowDiagram can be represented in several ways. The DFD belongs to structured-analysis modelling tools. Data Flow diagrams are very popular because they help us to visualize the major steps and data involved in software-system processes.



Data objects represented by labelled arrows and transformation are represented by circles also called as bubbles. DFD is presented in a hierarchical fashion i.e., the first data flow model represents the system as a whole. Subsequent DFD refine the context diagram (level 0 DFD), providing increasing details with each subsequent level.

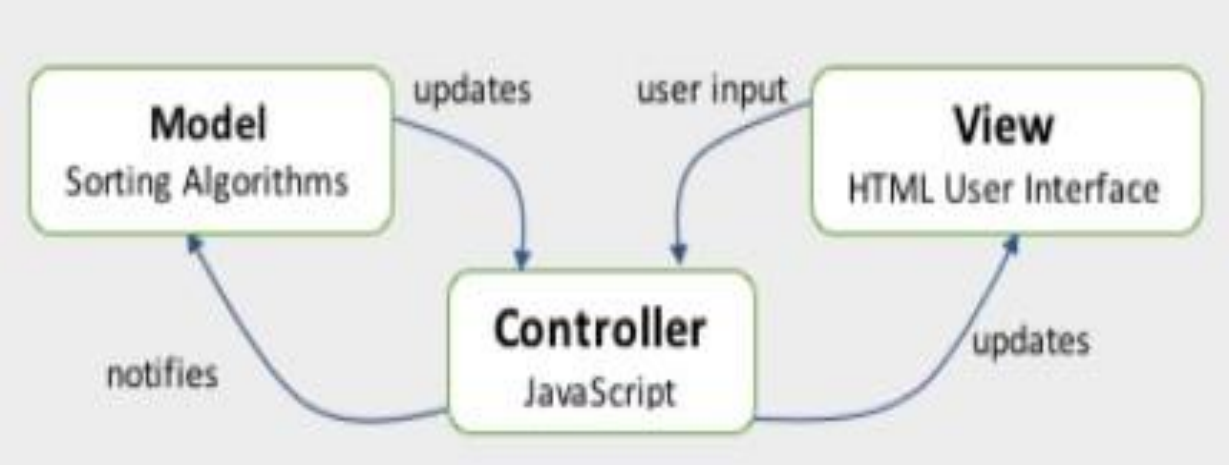
The DFD enables the software engineer to develop models of the information domain & functional domain at the same time. As the DFD is refined into greater levels of details, the analyst performs an implicit functional decomposition of the system. At the same time, the DFD refinement results in a corresponding refinement of the data as it moves through the process that embody the applications.

3.2 Model-View-Controller diagram of code:

The implementation of this project is a combination of HTML5 (Hypertext Markup Language 5), JavaScript, and CSS (Cascading Style Sheets). There is only one project file that contains the code and is an HTML file. The only addition to the main HTML file is the individual sound files to support the sound animation feature (saved as ".m4a" audio files).As of now, the preferred browser is Mozilla Firefox as I only performed rigorous testing inthis environment. However, quick tests showed possible use in Google Chrome and Safari.

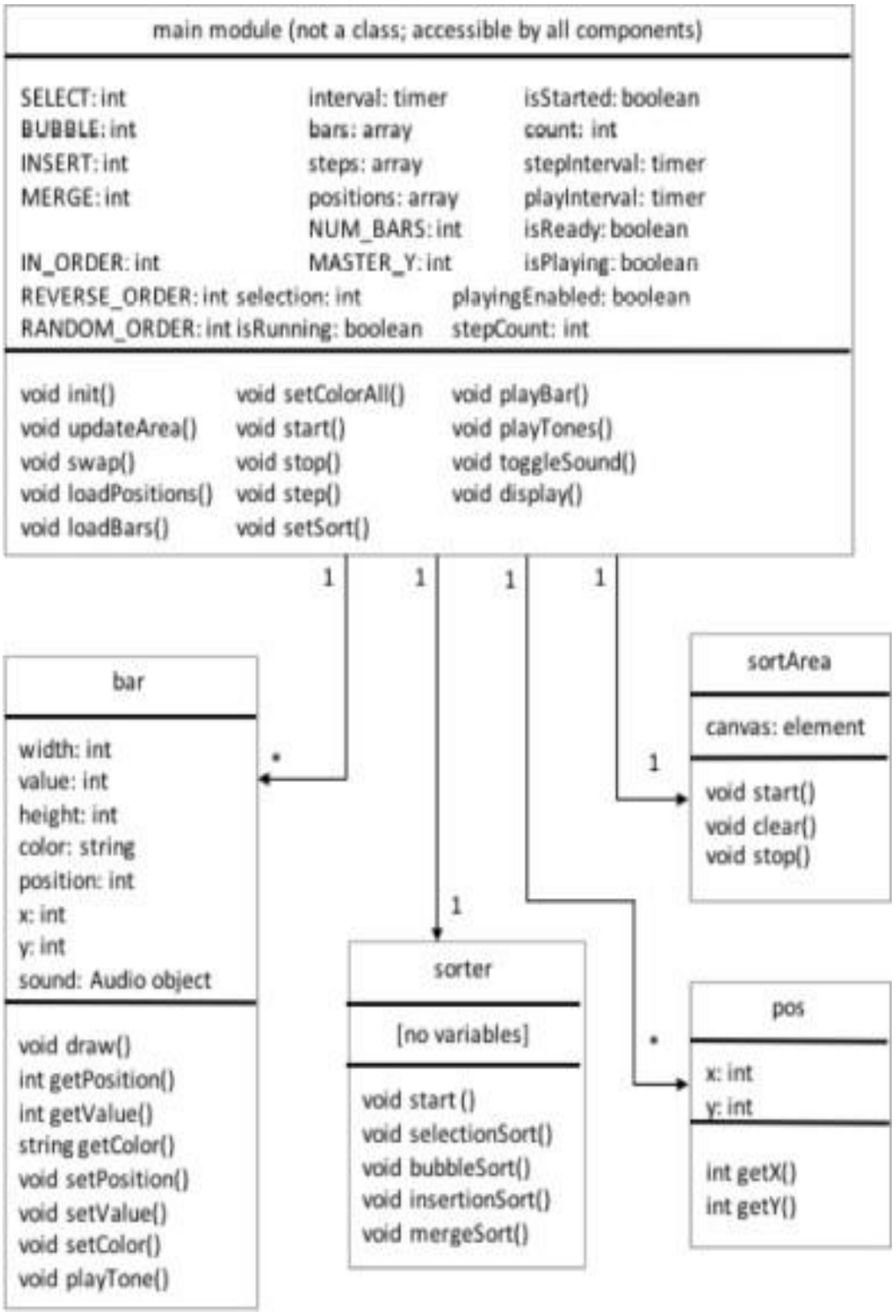
The organization of the code follows both object-oriented and functional programming concepts. Originally, the design was almost completely functional, where only three objects were used: one to control the canvas that displayed the animation, another to represent a piece of data, or "bar" object (blue 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 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 shown below.

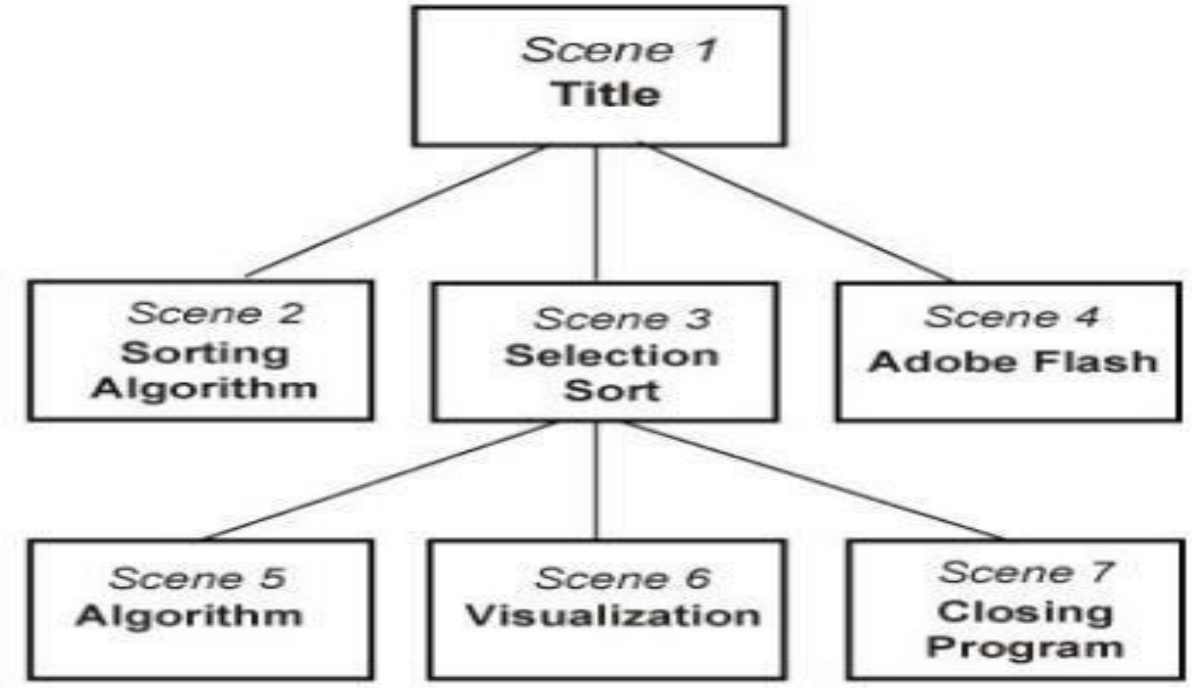


3.3 Class/object diagram of instance variables and respective functions in animation tool:

The class/object relationships between components are shown on the next page in Figure 9. The main module represents the global scope in the html file between the tags. The variables and methods listed in it are accessible by all components.



3.4 Navigation structure:



The step in developing a navigation structure and storyboard of the selection sorting algorithm visualization. First, it establishes lateral thought processes, helping to break down the navigation structures that are usually embedded in traditional approaches to course delivery. Second, it can result in an overview based on quite abstract design, which in turn generates fresh implementation. Third, it provides a storyboard for identifying relationships between the components. Navigation structure [10] is essential to design an interactive multimedia application as shown in Figure.

CONTENT 4

4. SYSTEM ANALYSIS

4.1 System Design:

As this tool is designed to be a learning aid in the classroom, this has presented ethical "Do Algorithm Animations Assist Learning?": An Empirical Study and Analysis". Our group in Almala study were randomly separated into two groups. In one, first member was studied the material using conventional methods (i.e., textbook). In the other, the second member had both the textbook and an animation tool to assist them when learning the material. This separation is necessary for more reliable results that will hopefully help improve teaching and learning for many students, but this comes at the cost of giving the participants in the study different experiences. Although our results proved a non-significant difference in the post-tests of both groups, it still created a noticeable learning gap, giving the students that used the animation tool an advantage over the others. Given that we had already reported these results and I am fond of fair practices, I decided not to follow his approach and made my post-test into a short survey that asked questions about what they recently learned in class about algorithms and how that changed when exploring my animation tool. An included 5-point rating scale gave the opportunity to judge if they believed the tool helped their current understanding at all. The survey was also voluntary, so it was up to the students' discretion to participate. The animation tool, however, was integrated into their class lecture via my presentation and time dedicated afterward to use the tool. Therefore, all the students were exposed to the tool instead of a select few, which avoided any academic disadvantage.

Key-design features

Content:

SORTING VISUALIZER was designed to support instruction and students' learning activities in the context of DAPE course, according to the goals set by the curriculum. A wide range of algorithms with arrays are included in SORTING VISUALIZER and supported by dynamic visualizations: bubble sort, insertion sort (in various versions), selection sort, quick sort, linear search, binary search, and merging of two sorted arrays. Figure 1 shows a screenshot of SORTING VISUALIZER presenting the visualization of merging algorithm.

This algorithm has a linear complexity and produces a sorted array that contains the elements of two given sorted arrays. The student can insert new input data to the source arrays or modify the pseudocode of the algorithm in order to investigate and reflect on its logic.

Visualization features: SORTING VISUALIZER represents an array by a series of contiguous cells that can move during execution of the algorithm, in order to help students built efficient mental models and overcome their difficulties. The actions that are critical for the inherent logic of an algorithm are described as interesting events (e.g., swapping two array elements in sorting). Every algorithm has a different visualization in order to point out its interesting events. During algorithm execution, SORTING VISUALIZER highlights the source code command corresponding to the current event. This helps students to link a programming command with the corresponding graphical presentation. Another important feature of the system is the animation of the control variables (I, j, k) in a loop command (e.g., FOR) which are visualized as arrows pointing to the respective array element. Control variables play also the role of array indices and, due to their automated change of value, they are considered as a source of difficulties for the students.

Interface and functionality: The user interface of SORTING VISUALIZER consists of a browser window divided into three main components:

• The source code editor (left pane), that shows the algorithm that is visualized and highlights the line that is currently executed. The student is allowed to make modifications and restart the visualization.

• The animation area (right pane) where the animation of the algorithm is displayed, i.e., the constant changes of the graphical representations associated with the operation of the current algorithm command. The critical algorithmic parameters are highlighted or visualized by smooth transitions of objects.

• The control panel (down centre pane) includes operation buttons that support student-visualization interaction. There are four main control operations, e.g., start, pause, restart, and step-by-step execution. Input boxes help students to modify the input data of source arrays A, B in order to test algorithm execution for various data sets. The down-left slider aims to control the animation speed.

User control and engagement:

The critical design principle for an effective algorithm visualization system is to promote students' engagement, not only during the execution of sample algorithms, but also through experimentation by modifying algorithm's code and input data. Using SORTING VISUALIZER, the student can select the algorithm that he/she wants to visualize. During the execution of the algorithm, the system allows users to modify some parts of the code and, at the same time, to watch the corresponding visualization on the screen. The dynamic features above offer enhanced opportunities to the students to explore the behaviour of the algorithm and to identify the critical logical steps and the structure of the algorithm.

Web features:

Most web-based visualization systems are based on java applets. Therefore, users need to install the proper java runtime environment; following they download and launch the particular applet. Java applications are heavier and demanding than scripts. SORTING VISUALIZER is designed as a fully web-based system written in HTML5 and JavaScript. Therefore, by simply visiting the hosting URL, it runs in any platform, operating system, browser or device. Educators and students do not need to install any additional software or download any special-purpose package. It is also fully client-based so there is no overload due to the communication with any server.

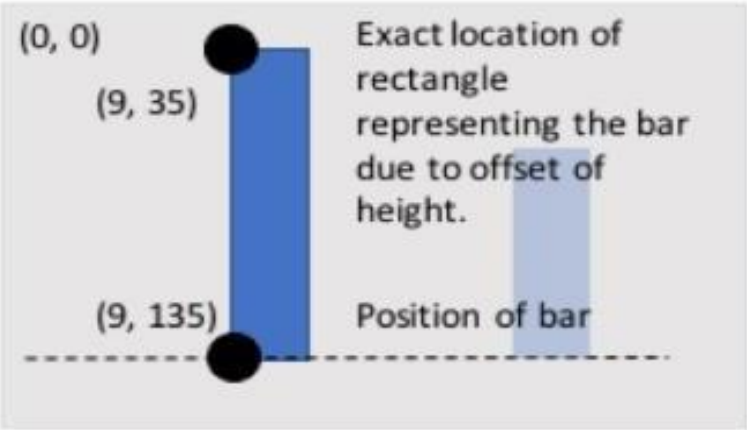
The Controller

The Controller consists of all visible buttons on the webpage. These buttons are coded in HTML and directly call a JavaScript method when pressed. The CSS is used for stylizing the appearance, layout, color, and effects of the buttons, along with the canvas. The methods manipulate the state of the Model and perform the necessary updates to prepare and update the View for animation. The buttons are segregated into four groups. The top, blue-bordered buttons are the sorting algorithm buttons and interact with the sorter object directly. After they are called, the Model builds a steps array that is then available for animation. This animation can be controlled by the bottom right buttons: Start, Stop, and Step. These update the current state of the bar graph by pulling new values from the step array. The View visualizes these updates and the result is the bars changing position based on what algorithm was selected. The Step button is the only button in this group that updates the bars directly. The Start and Stop buttons only control a timer that either calls the Step button at a predetermined speed (every quarter of a second) or stops the timer, respectively. The bottom-left blue-bordered buttons also directly control the bars. They change the initial position of

the bars to be in order, reversed, or ran dom. The positions of the bars do not actually change. Instead, each button simply loads an array of values that reflects the ordering desired. The View instantaneously updates this change on the canvas.

The View

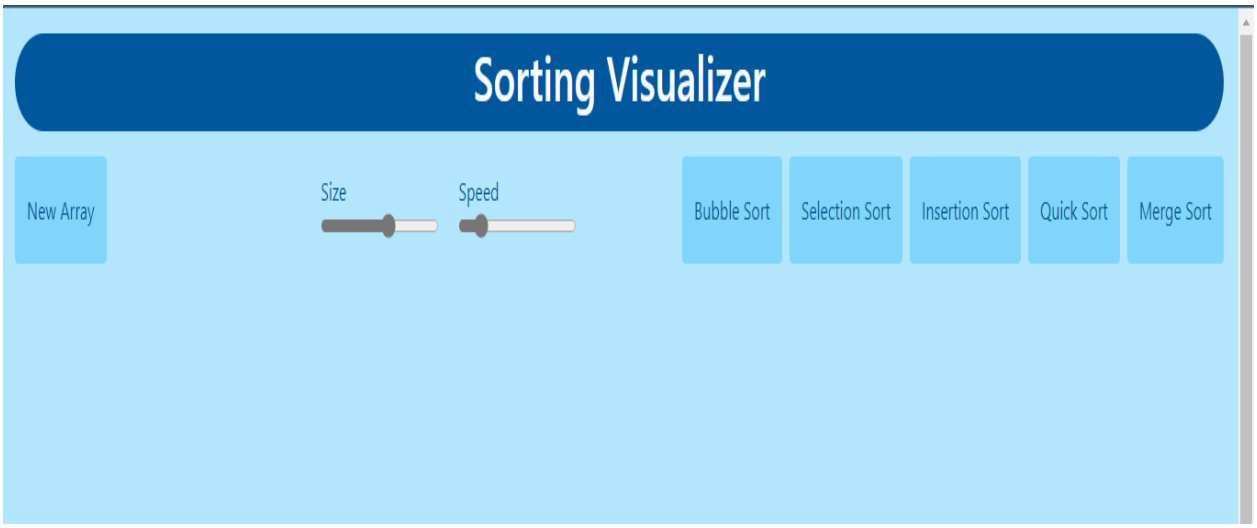
The View is composed of three objects, called sort area, bar, and pos. These objects do not belong to any specific object called "View," but instead operate in the space allocated by the tags in the .html file. You can think of this space as the "main" function, which is the first function called in a program. Sort area is the object that generates the bar graph and runs constantly with a timer to keep the individual bars updated. Therefore, when "Step" is called, it updates the current values of the bars to the new ones based on the steps array (discussed later). Then, the sort area will show the changes by redrawing the rectangles with different heights that reflect the new values at the next iteration of the timer. The bars update sixty times a second, so the change invoked by the "Step" button is seen instantly. I based the timer interval on a game tutorial I studied on the W3Schools website when learning JavaScript. The bar object is used to represent each piece of data in the sort area. It includes the attributes color, value, position, height, and sound. By keeping a separate array called bars for the current bars in the bar graph, it is then easy to change any or all the attributes by iterating over the array as necessary. The In Order, Reverse, and Random buttons do this by quickly iterating over the array to update the bars to a new data configuration. Related to the bar object is the pos object (which is short for position). The canvas area that sorts area updates is an x-y coordinate grid of pixels. To make positioning the bars easier, I created this object to assign a coordinate pair to a position number (1-32). Therefore, if I wanted to change the position of a bar, I would only specify the position number. Then, the bar will look up the exact coordinates from that number and move there. For example, position one consists of the coordinate pair (9, 135), which is the bottom left-hand corner of the bar. However, the rectangle object that each bar is drawn from must be given the top left-hand corner coordinates, so the height must also be considered to reposition the bar correctly. Figure 10 shows how the coordinate points relate to the positioning of a bar on the canvas. This feature was used in the earlier version of the program extensively, but after the refactor, it is now only used in the initial setup of the bars when the page loads.



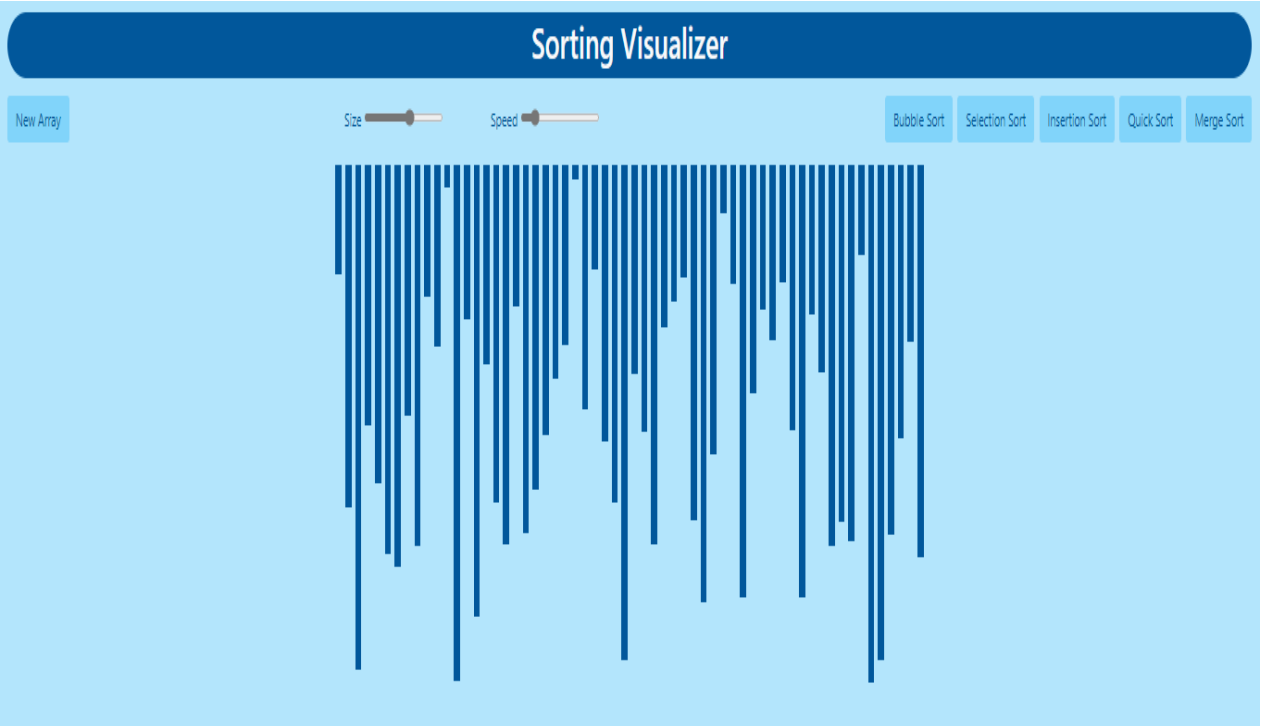
4.2 Screen shoot of our Implementation:

Since only HTML has been used the site should look something like this.

After styling using CSS and Bootstrap the site should look something like this.



After adding the bars, the site looks something like this.



CONTENT 5

5. CODING:

Category wise source code:

5.1 HTML Code:

```
1.<!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="stylesheet" href="style.css">
</head>
<body class="p-3">
    <header>
        <div class="head">
            <h1 class="text-center text-light">Sorting Visualizer</h1>
        </div>
        <nav class="mt-3">
            <div class="row">
```

```
                <div class="col gap-2 d-sm-flex" id="newArray">
<button type="button" class="btn newArray my-btn">New
Array</button>
                </div>
                <div class="col" id="input">
                    <span id="size" style="color: #01579B;">Size
<input id="arr_sz" type="range" min="5"max="100" step=1
value=60>
                    </span>
                    <span id="speed" style="color: #01579B;">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 my-btnbubbleSort">Bubble
Sort</button>
<button type="button" class="btn my-btn
selectionSort">Selection Sort</button>
<button type="button" class="btn my-btn
insertionSort">Insertion Sort</button>
<button type="button" class="btn my-btnquickSort">Quick
Sort</button>
<button type="button" class="btn my-btnmergeSort">Merge
Sort</button>
                </div>
            </div>
        </nav>
    </header>

    <div id="bars" class="flex-container"></div>
    <div class="text-center mt-5">

<input type="text" placeholder="Enter Custom Array"
id="customArr" class="customArr mx-3" >
<button type="button" id="customArrBtn" class="btn newArray my-
btn">Generate Custom Array</button>
        <br>
        <p class="mt-2">e.g 10,20,40,50,60</p>
    </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>
```

5.2. CSS Code:

```
body{
    background-color: #B3E5FC;
}
.head{
    background-color: #01579B;
    border-radius: 10rem;
    padding: .2rem;
}
.flex-container{
    margin-top: 20px;
    display: flex;
    flex-wrap: nowrap;
    width: 100%;
    height: 500px;
    justify-content: center;
    transition: .6s all ease;
}

.flex-item{
    background:#01579B;
    width: 10px;
    margin: 0 3px;
    transition: 0.1s all ease;
}
.row{
    display: grid;
    grid-template-columns: 1fr 1fr 2fr;
}
.customArr{
    background-color: transparent;
    border: none;
    border-bottom: 1px solid #01579B;
    width: 10rem;
    padding: .2rem;
    transition: .4s;
    color: #01579B;
}
.customArr::placeholder{
    color: #01589ba4;
```

```
    padding: .2rem;
}
.customArr:focus{
    outline: none;
    width: 25rem;
}

#input{
    display: flex;
    padding: 10px;
    justify-content: space-around;
}
.my-btn{
    background-color: #81D4FA;
    color: #01579B;
}
.my-btn:hover{
    background-color: #E1F5FE;
    color: #01579B;
}
```

5.3. JavaScript Files

5.3.1. Bubble Sort:

```
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();
});
```

5.3.2. Selection Sort:

```
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();
});
```

5.3.3. Insertion Sort:

```
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();
});
```

5.3.4. Quick Sort:

```
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();
});
```

5.3.5. Merge Sort:

```
//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();
});
```

5.3.6. sorting.js:

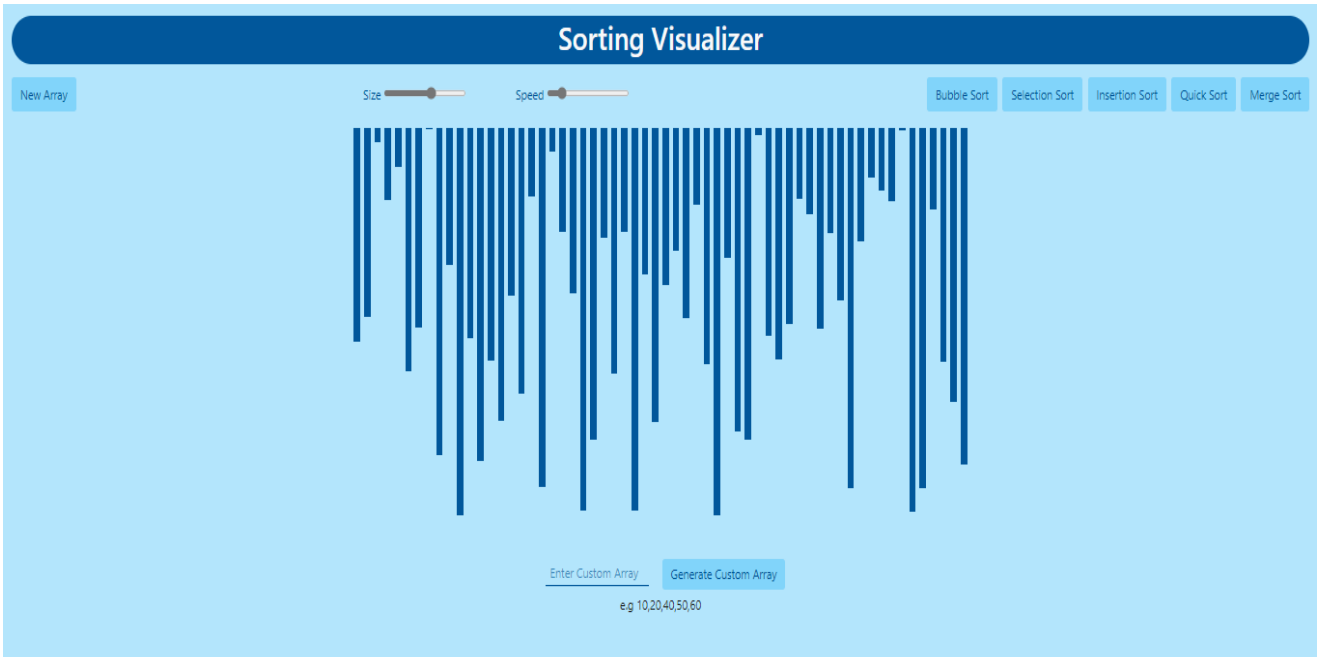
```
// swap function util for sorting algorithms takes input of 2 DOM elements
with .style.height feature
function swap(el1, el2) {
    console.log('In swap()');

    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;
```

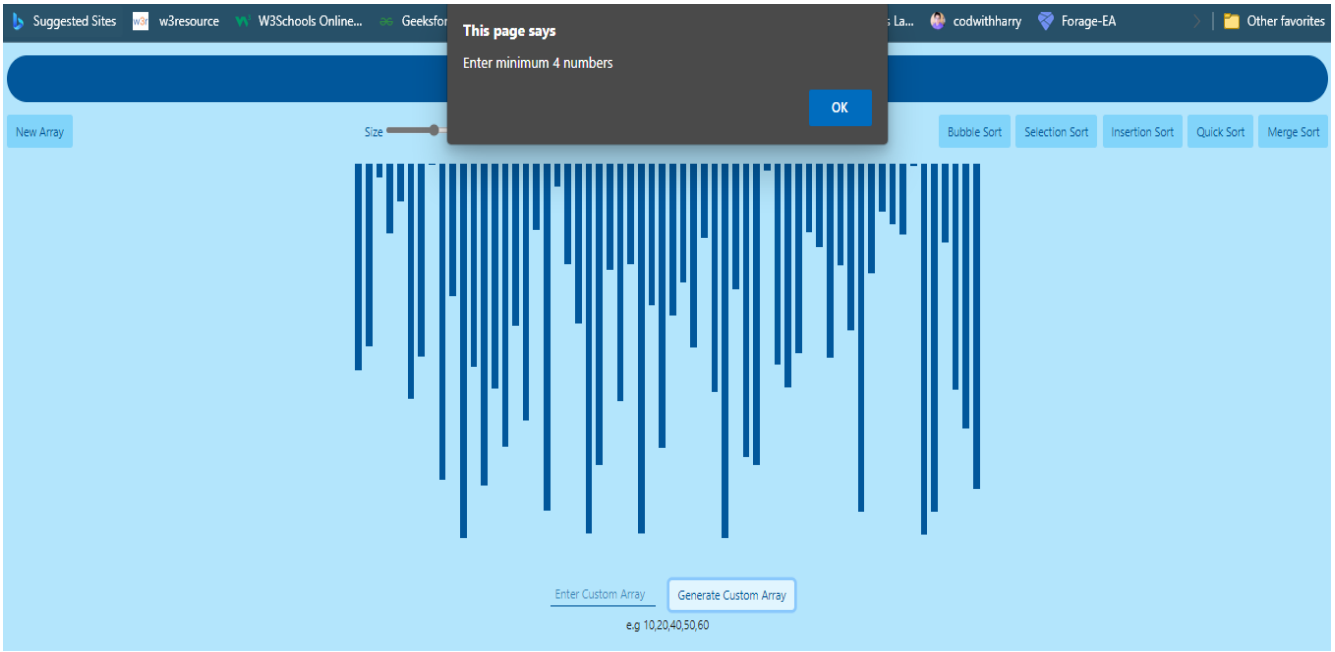
```
}

// 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 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));
```

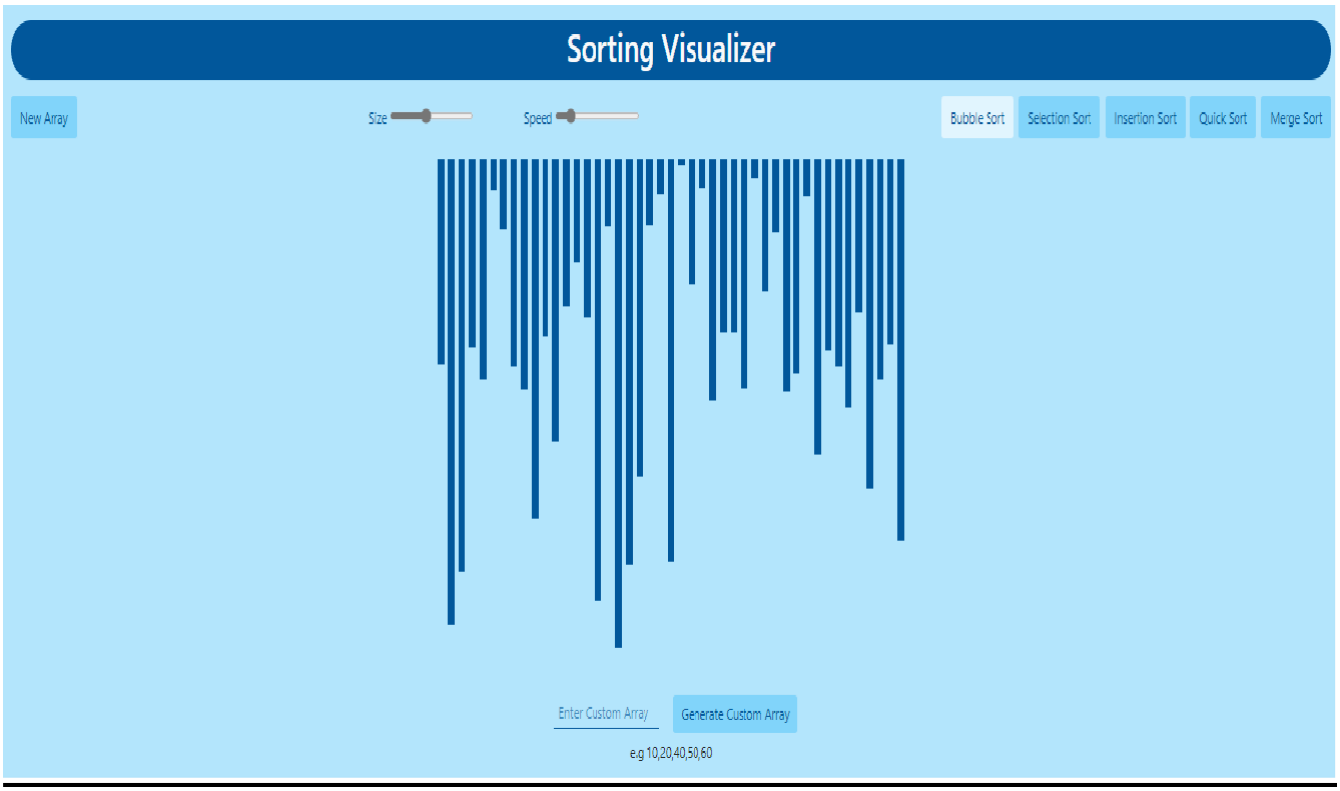
```
});

// 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 = [];
let customArray = [];

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

//To create custom array
let customArrayInput=document.getElementById("customArr");
let customArrBtn=document.getElementById("customArrBtn");
customArrBtn.addEventListener("click",createCustomArray)
function createCustomArray() {
    // calling helper function to delete old bars from dom
    deleteChild();

    // creating an array of custom numbers
    customArray = customArrayInput.value.split(",");

    if(customArray.length>3){
        // 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 < customArray.length; i++) {
        const bar = document.createElement("div");
        bar.style.height = `${customArray[i]*2}px`;
        bar.classList.add('bar');
        bar.classList.add('flex-item');
        bar.classList.add(`barNo${i}`);
        bars.appendChild(bar);
    }
}
    else{
        alert('Enter minimum 4 numbers')
```

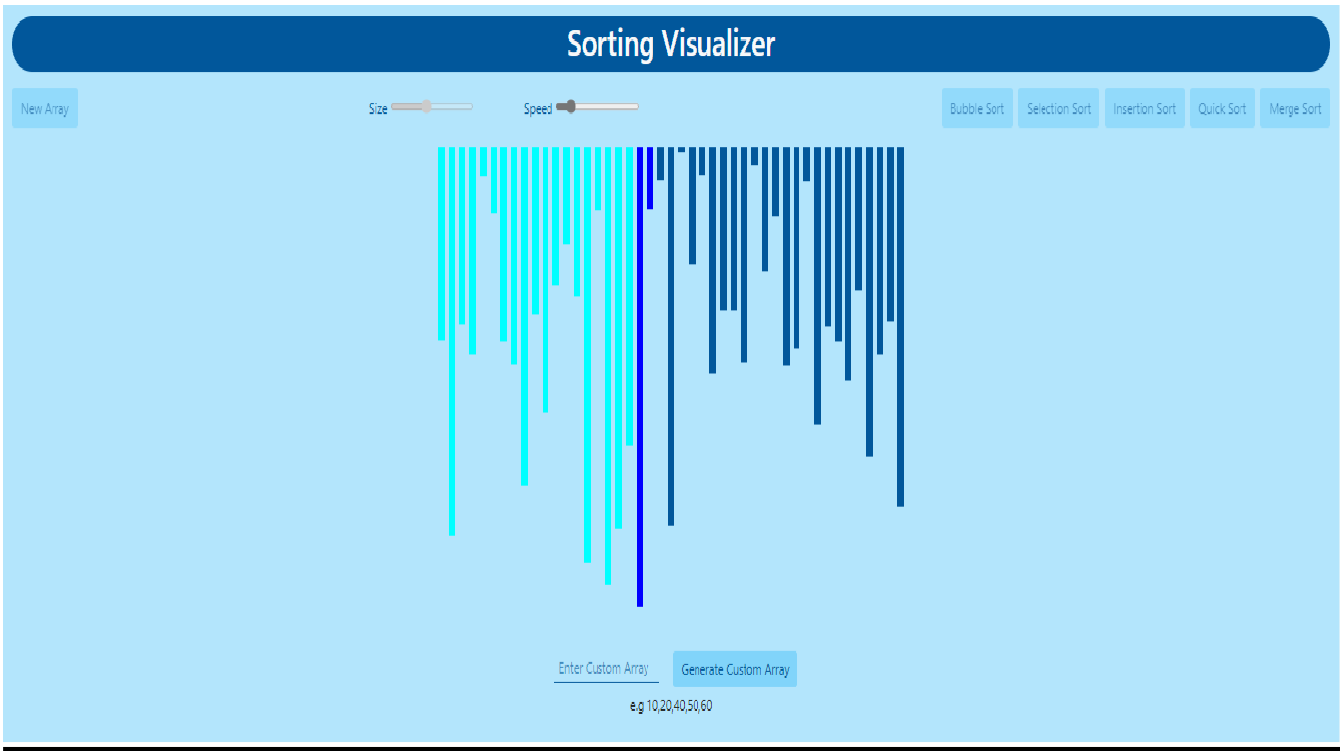
```
    }
}

// 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);
});
```

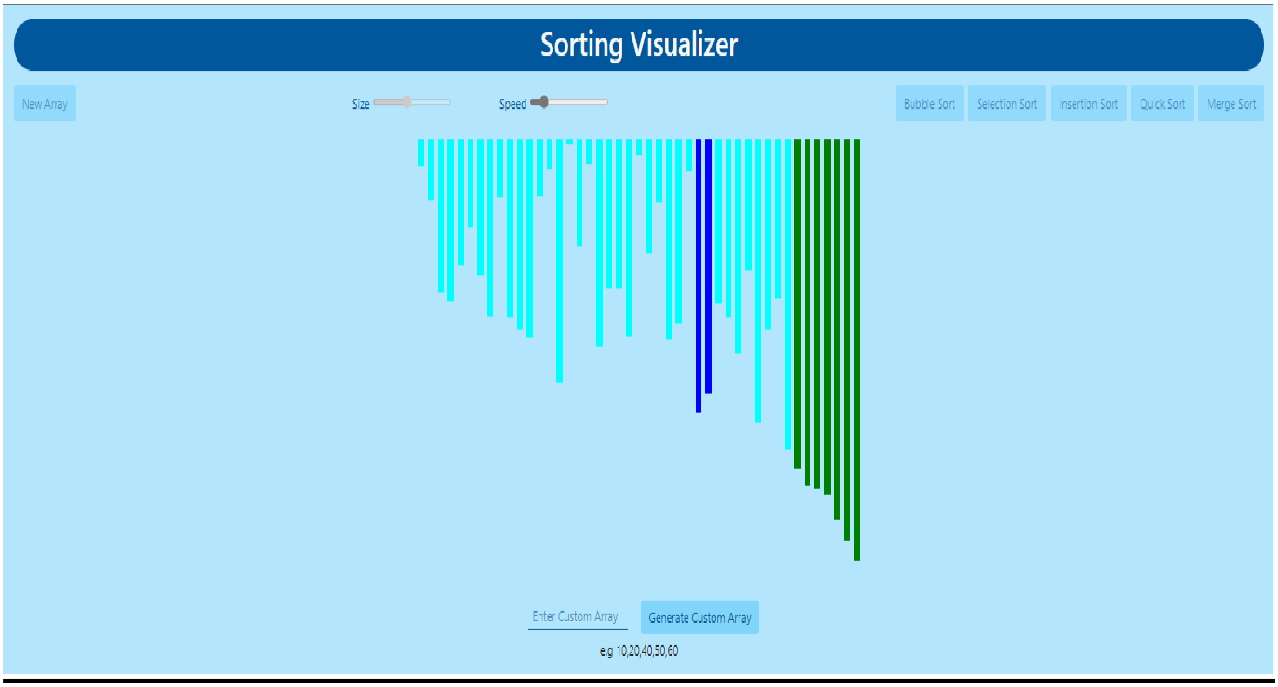
5.4 Output Of The Code:

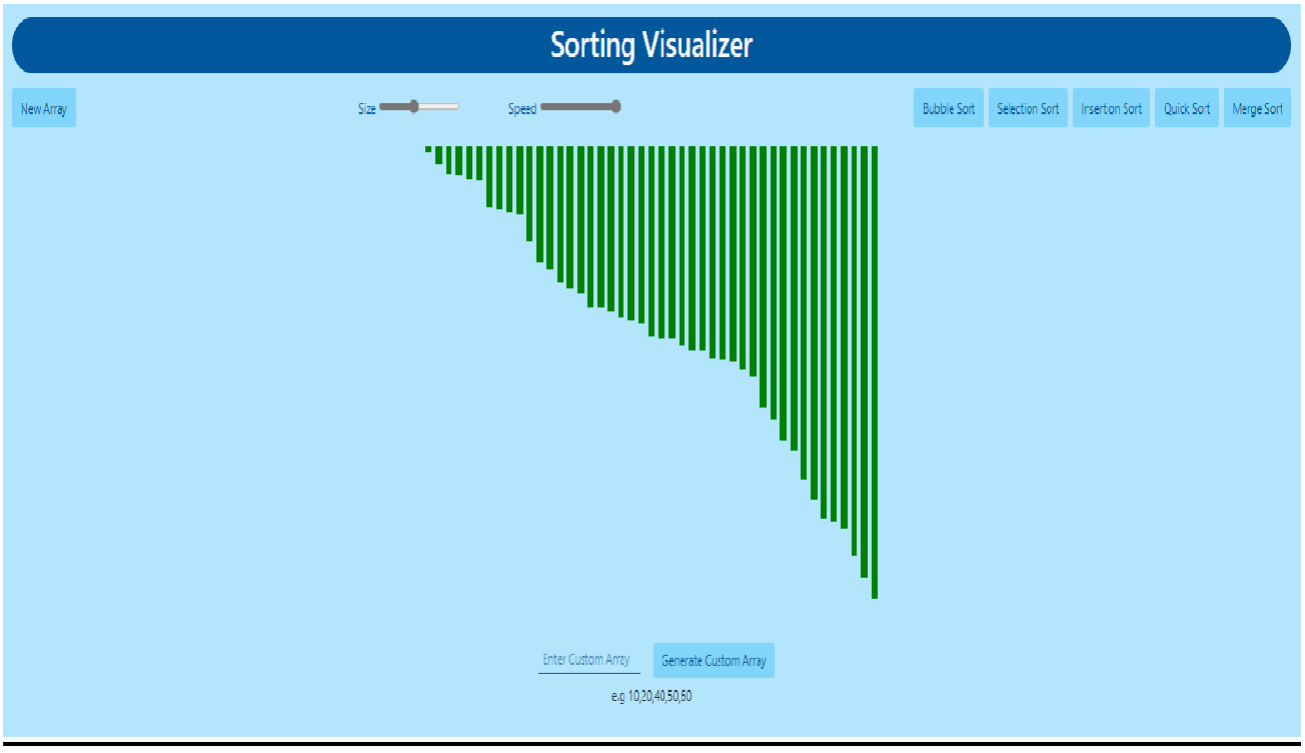


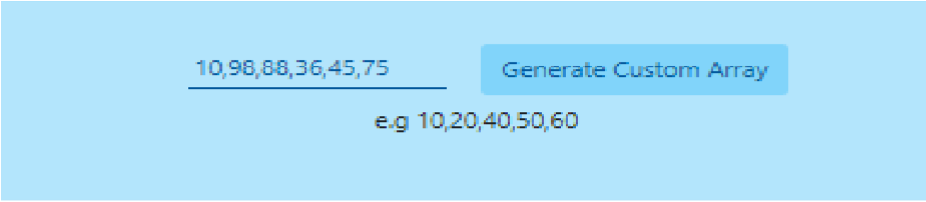


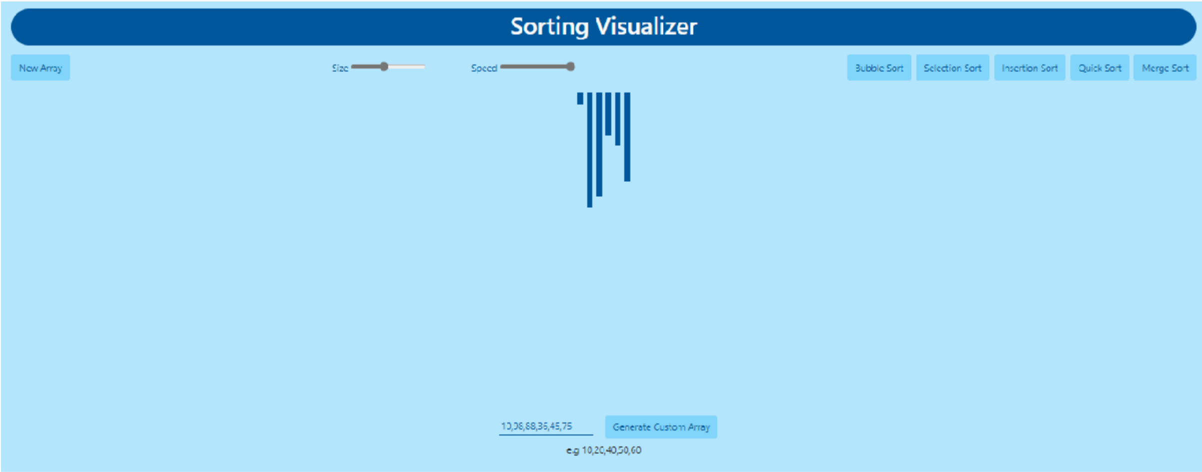

Sorting Visualizer
New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort
Enter Custom Array
Generate Custom Array
e.g 10,20,40,50,60

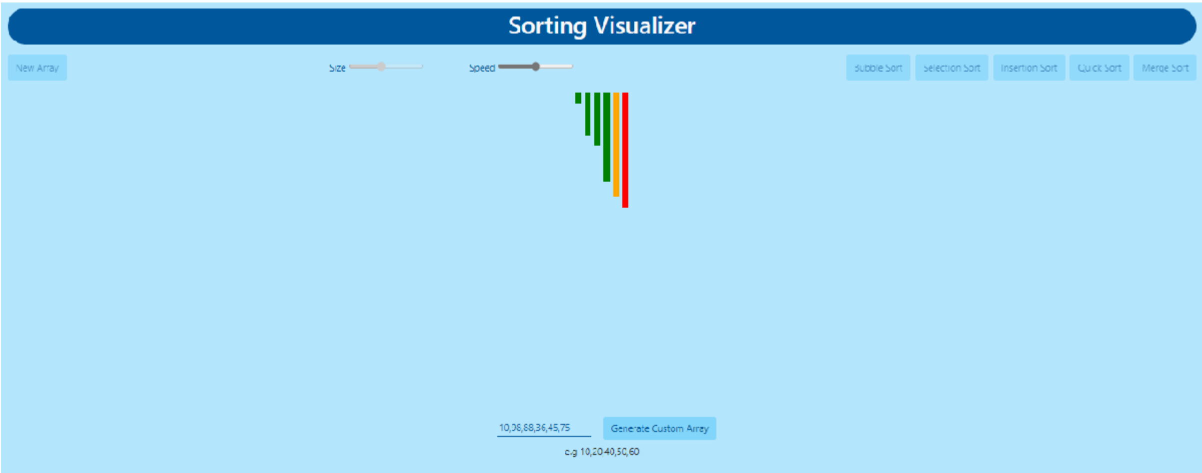

Sorting Visualizer
New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort
Enter Custom Array
Generate Custom Array
e.g 10,20,40,50,60

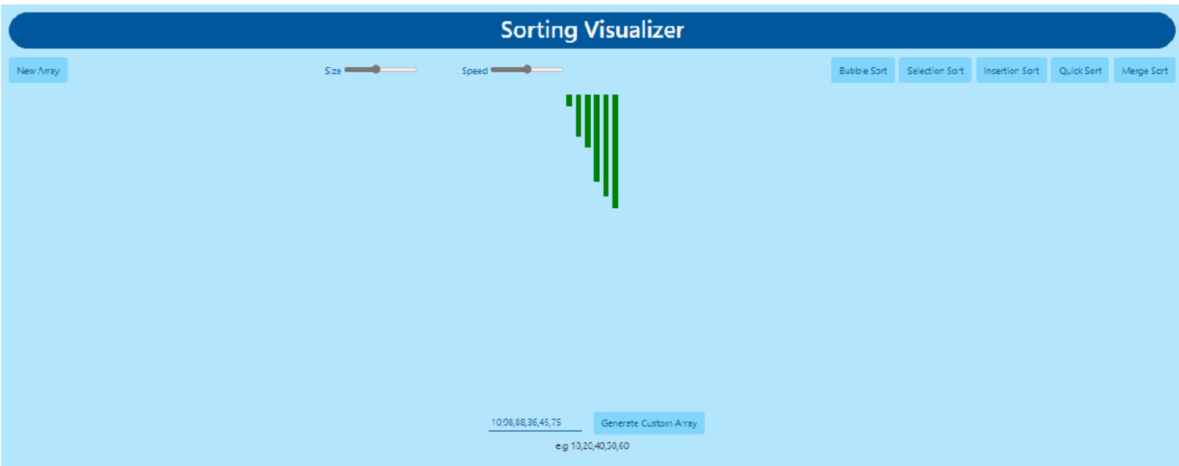

Sorting Visualizer
New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort
Enter Custom Array
Generate Custom Array
e.g 10,20,40,50,60


Sorting Visualizer
New Array
Size
Speed
Bubble Sort
Selection Sort
Insertion Sort
Quick Sort
Merge Sort
Enter Custom Array
Generate Custom Array
e.g 10,20,40,50,60


10,98,88,36,45,75
Generate Custom Array
e.g 10,20,40,50,60


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


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


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

CONTENT 6

6. SOFTWARE TESTING:

Testing the application is run and checked to confirm that it performs exactly what the objectives are performed. In the application, this is similar to screening, where the application or parts of it are viewed and approved by the lecturer or his or her assistance.

Software testing can be stated as the process of verifying and validating that software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

1. Verification: it refers to the set of tasks that ensure that software correctly implements a specific function.

2. Validation: it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Verification: “Are we building the product right?”

Validation: “Are we building the right product?”

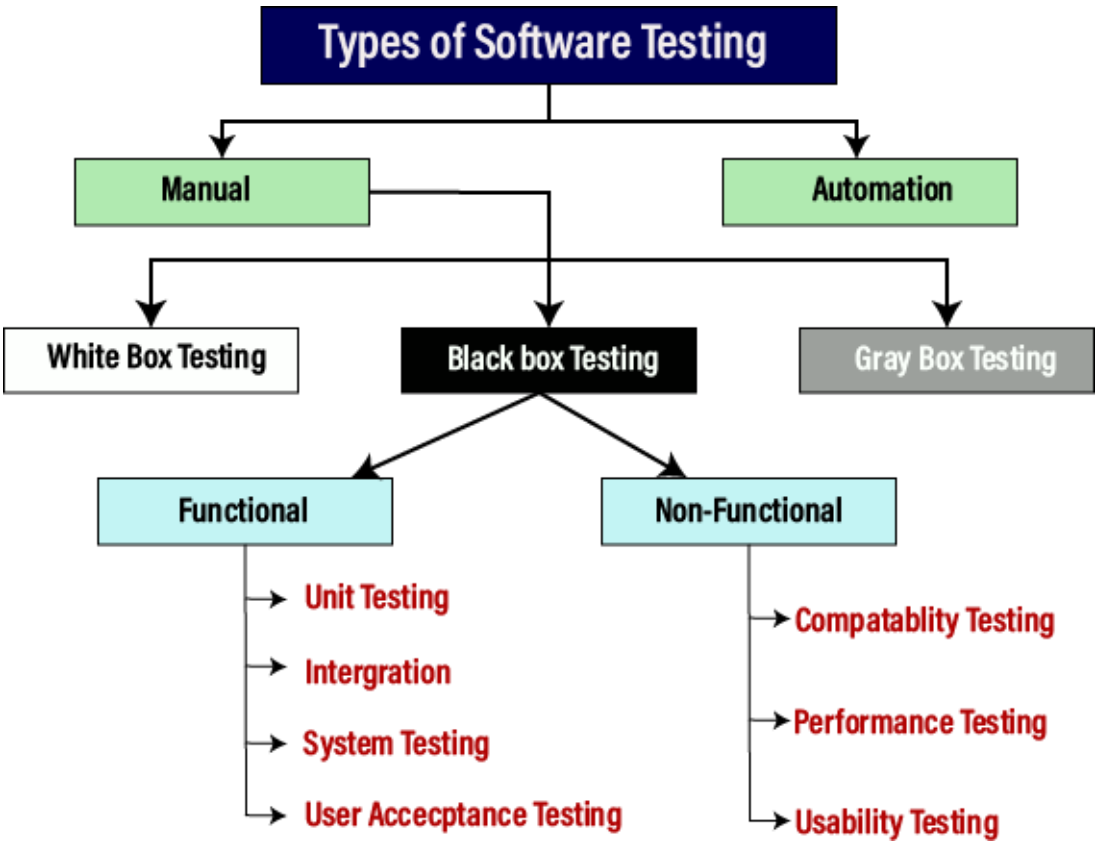
Types Of Testing:

Software Testing can be broadly classified into two types:

1. Manual Testing: Manual testing includes testing software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

2. Automation Testing: Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.



Functional Testing:

It is a type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification. In functional testing, each function tested by giving the value, determining the output, and verifying the actual output with the expected value. Functional testing performed as black-box testing which is presented to confirm that the functionality of an application or system behaves as we are expecting. It is done to verify the functionality of the application.

Functional testing also called as black-box testing, because it focuses on application specification rather than actual code. Tester has to test only the program rather than the system.

What is the process of functional testing?

Testers follow the following steps in the functional testing:

- Tester does verification of the requirement specification in the software application.
- After analysis, the requirement specification tester will make a plan.
- After planning the tests, the tester will design the test case.
- After designing the test, case tester will make a document of the traceability matrix.
- The tester will execute the test case design.
- Analysis of the coverage to examine the covered testing area of the application.
- Defect management should do to manage defect resolving.

What to test in functional testing?

The main objective of functional testing is checking the functionality of the software system. It concentrates on:

- Basic Usability: Functional Testing involves the usability testing of the system. It checks whether a user can navigate freely without any difficulty through screens.
- Accessibility: Functional testing test the accessibility of the function.
- Mainline function: It focuses on testing the main feature.
- Error Condition: Functional testing is used to check the error condition. It checks whether the error message displayed.

Types Of Functional Testing Used In Our Testing:

1. Unit Testing:

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with its performance.

A unit is a single testable part of a software system and tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or

Integration testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as integration testing.

Any application in this world will do functional testing compulsory, whereas integration testing will be done only if the modules are dependent on each other. Each integration scenarios should compulsorily have source→ data→destination. Any scenarios can be called as integration scenario only if the data gets saved in the destination.

Non-Functional Testing:

Non-functional testing is a type of software testing to test non-functional parameters such as reliability, load test, performance and accountability of the software. The primary purpose of non-functional testing is to test the reading speed of the software system as per non-functional parameters. The parameters of non-functional testing are never tested before the functional testing.

Non-functional testing is also very important as functional testing because it plays a crucial role in customer satisfaction.

For example, non-functional testing would be to test how many people can work simultaneously on any software.

Why Non-Functional Testing?

Functional and Non-functional testing both are mandatory for newly developed software. Functional testing checks the correctness of internal functions while Non-Functional testing checks the ability to work in an external environment.

It sets the way for software installation, setup, and execution. The measurement and metrics used for internal research and development are collected and produced under non-functional testing.

Non-functional testing gives detailed knowledge of product behavior and used technologies. It helps in reducing the risk of production and associated costs of the software.

Types of Non-Functional Testing Used in Our Project:

1. **Usability Testing**

Usability Testing is a significant type of software testing technique, which is comes under the non-functional testing.

It is primarily used in user-centred interaction design on order to check the usability or ease of using a software product. The implementation of usability testing requires an understanding of the application, as it is extensive testing.

Generally, usability testing is performed from an end-user viewpoint to verify if the system is efficiently working or not.

Checking the user-friendliness, efficiency, and accuracy of the application is known as Usability Testing.

The primary purpose of executing the usability testing is to check that the application should be easy to use for the end-user who is meant to use it, whereas sustaining the client's specified functional and business requirements.

When we use usability testing, it makes sure that the developed software is straightforward while using the system without facing any problem and makes end-user life easier.

In other words, we can say that Usability testing is one of the distinct testing techniques that identify the defect in the end-user communication of software product. And that's why it is also known as User Experience (UX) Testing.

It helps us to fix several usability problems in a specific website or application, even making sure its excellence and functionality.

The execution of usability testing certifies all the necessary features of a product, from testing the effortlessness of navigating a website and to validate its flow and the content in order to propose the best user experience.

Typically, the usability testing is executed by real-life users, not by the development team, as we are already aware of that the development team is the one who has created the product. Consequently, they fail to identify the more minor defects or bugs related to the user experience.

Note: It can be implemented in the Designing phase of the software development life cycle (SDLC) in order to helps us getting more clarity of the user's needs.

In Usability Testing, the user-friendliness can be described with the help of the following characteristics:

- Easy to understand
- Easy to access
- Look and feel
- Faster to Access
- Effective Navigation
- Good Error Handling

2. **Compatibility Testing:**

It is part of non-functional testing where the functionality of an application is checked on different software, hardware platforms, network, and browsers are known as compatibility testing.

Why we use compatibility testing?

Once the application is stable, we moved it to the production, it may be used or accessed by multiple users on the different platforms, and they may face some compatibility issues, to avoid these issues, we do one round of compatibility testing.

Types of Compatibility Tests:

Hardware: It checks software to be compatible with different hardware configurations.

Operating Systems: It checks your software to be compatible with different Operating Systems like Windows, Unix, Mac OS etc.

Software: It checks your developed software to be compatible with other software. For example, MS Word application should be compatible with other software like MS Outlook, MS Excel, VBA etc.

Network: Evaluation of performance of a system in a network with varying parameters such as Bandwidth, Operating speed, Capacity. It also checks application in different networks with all parameters mentioned earlier.

Browser: It checks the compatibility of your website with different browsers like Firefox, Google Chrome, Internet Explorer etc.

Devices: It checks compatibility of your software with different devices like USB port Devices, Printers and Scanners, Other media devices and Bluetooth.

Mobile: Checking your software is compatible with mobile platforms like Android, iOS etc.

Versions of the software: It is verifying your software application to be compatible with different versions of the software. For instances checking your Microsoft Word to be compatible with Windows 7, Windows 7 SP1, Windows 7 SP2, Windows 7 SP3.

CONTENT 7

7. CONCULSION

Through much time and effort, I have successfully created a working web-based animation tool for visualizing the following sorting algorithms: Selection Sort, Bubble Sort, Insertion Sort, and Merge/Insertion Sort. Even with its memory overhead, it received overall positive feedback from the students who explored it. I am not surprised that there was not a significant difference in learning the material, which reflects what I found in my previous research. There remains, however, a strong mindset to research and create animations like these to improve learning in the classroom, which I agree with completely. Learning how to code a web platform was challenging, and I thank the tutorials on W3Schools.com for getting me there. I had a previous internship where I updated the JavaScript on a webpage, but it was much more concise and did not involve objects and HTML5 for visualizations. The good news is that JavaScript is still one of the most popular web languages, so I am not too worried about another big refactor soon for a language update.

Through this sorting algorithm visualization has been presented. Some details about implementation of sorting algorithm in HTML, CSS and JavaScript programming have been described. The visualization and the interactivity have been well tested by us and indicates that:

1. Student understand easily the concept of sorting algorithm by looking at the visualization. Learning material text is more affective if it is provided with graphic, animation, or video to be learned by student.

2. Multimedia, particularly the visualization increases student motivation in algorithm learning and code creating, according to variety sorting method using particular programming language they have learned, i.e., C++, Java, Visual Basic, and their programming languages.

7.1 FUTURE SCOPE

- Not all the sorting algorithm are covered, so one can implement the other sorting algorithms in JavaScript file for visualization.
- One can make the new .html file and add the explanation of algorithm all algorithm with code.
- One can change the .css file update the design of project and make it more attractive.

CONTENT 8

8. BIBLOGRAPHY

1. https://www.crio.do/

2. https://www.javatpoint.com/

3. https://en.wikipedia.org/wiki/sorting_visualizer

4. https://digitalcommons.ric.edu/