Index

1 . Introduction

1.1 Company Profile

FountLab is customer-centric, technology-driven and process-oriented company. We solve industry-specific problems by combining IoT technology, data science, and subject matter expertise.
We are a mature eclectic group of people who share a magical vision of the future, working towards bridging the technological gap in the current practices by developing innovative products and solutions.

Company Mission

While Every Industry Evolved with Tech, Buildings Remained Conservative. The last biggest disruption in buildings took place 20 years ago. Automation was first crudely introduced; while hardware progressed, not much was seen on the software side. Socially Responsible Buildings.

Feel good about your investment, both environmentally and financially. We’ve designed our platform to be socially-responsible that prioritizes low-carbon emissions, advances green energy practises, and makes sustainability a habit.

1.2 Abstract

Web based Application development were the main objective of this internship. To develop a GUI based application there are several programming languages that are in use. For example- React native ,React ,NodeJS ,Angular etc. There are also some other programming languages that are used to develop the dynamic functions of the application. For example , Java , Nodejs etc. Nowadays there are also some framework's that use vastly. Frameworks are basically structured programming by using Model, View, and Controller.It is also called as MVC. If we develop web based application that is very useful for us because we can access it from anywhere of the world. It is very helpful for our daily life. That is why I choose subject of my report is "Web based application on Sorting algorithm visualizer ". Working in **Fountlab Pvt Ltd** added huge experiences in my upcoming career. Solving real life problems was another key issue. This report takes us through all the details of Web based application on Sorting algorithm visualizer experience gathered during this internship period.

1.3 Existing System and Need for System

Existing System :

This project Sorting algorithm visualizer is a project which allows the users to select the sort algorithm, select the array size, and speed of the visualization of the algorithm . Sorting algorithm visualization can be defined as the use of images to convey how the array of data gets sorted. Bubble , Merge , Selection , Quick, Heap , Selection Sorting algorithms are there in the application .

Need for System :

- A Sorting algorithm visualizer helps students to understand the sorting algorithm visually .
- There are few sorting algorithm options there - Bubble sort , Merge sort , Selection sort , Quick Sort , ,Selection Sort ,Heap Sort .
- Users can understand which algorithm works faster and which one is quick between other algorithms as per the length of the Array.
- Student / User can adjust the speed of the sorting algorithm so that he goes by each step by understanding .

1.4 Scope of System

- A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.
- This Application lets everyone understand how these algorithms work and through this application you also will get a deep understanding of such sorting algorithms.
- Increase the core knowledge of sorting algorithms .
- Users will understand the time complexity of the algorithm and figure out which algorithm is fastest.
- This application will help to explain and implement sorting algorithms and be able to explain and implement selection sort, bubble sort, merge sort, quick sort.

1.5 Operating Environment - Hardware and Software

Software:-

IDE used : Visual Studio Code

Backend Technologies :

- Python
- JavaScript
- HTML
- CSS

UI :Web Page

Hardware:

Hardware: AMD RYZEN 5

RAM : 1 GB (minimum)

Soft Requirements:

- HTML
- CSS
- JAVASCRIPT,
- PYTHON
- DOM
- Flask

1.6 Brief Description of Technology Used

1.6.1 Operating System used - Windows

- Windows is a graphical operating system developed by Microsoft. It allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet. It was released for both home computing and professional work.

- **Windows 11** is the latest major release of Microsoft's Windows NT operating system, released in October 2021. It is a free upgrade to its predecessor, Windows 10 (2015), available for any Windows 10 devices that meet the new Windows 11 system requirements.

- Windows 11 features major changes to the Windows shell influenced by the canceled Windows 10X, including a redesigned Start menu, the replacement of its "live tiles" with a separate "Widgets" panel on the taskbar, the ability to create tiled sets of windows that can be minimized and restored from the taskbar as a group, and new gaming technologies inherited from Xbox Series X and Series S such as Auto HDR and DirectStorage on compatible hardware. Internet Explorer (IE) has been replaced by the Chromium-based Microsoft Edge as the default web browser like its predecessor, Windows 10, and Microsoft Teams is integrated into the Windows shell. Microsoft also announced plans to allow more flexibility in software that can be distributed via Microsoft Store, and to support Android apps on Windows 11.

1.6.2 Used DataBase - Dynamic data

- In data management, the time scale of the data determines how it is processed and stored. Dynamic data or transactional data is information that is periodically updated, meaning it changes asynchronously over time as new information becomes available.

- Data that is not dynamic is considered either static (unchanging) or persistent, which is data that is infrequently accessed and not likely to be modified. Dynamic data is also different from streaming data, which is a constant flow of information. Dynamic data may be updated at any time, with periods of inactivity in between.

- Mutable collections frequently experience additions, updates, and removals (among other changes). Dynamic Data provides two collection implementations, an observable list and an observable cache that expose changes to the collection via an observable change set. The resulting observable change sets can be manipulated and transformed using Dynamic Data's robust and powerful array of change set operators. These operators receive change notifications, apply some logic, and subsequently provide their own change notifications. Because of this, operators are fully composable and can be chained together to perform powerful and very complicated operations while maintaining simple, fluent code.

- Using Dynamic Data's collections and change set operators make in-memory data management extremely easy and can reduce the size and complexity of your code base by abstracting complicated and often repetitive collection based operations.

1.6.3 Python

- Python is a dynamic, interpreted (bytecode-compiled) language. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible, and you lose the compile-time type checking of the source code. Python tracks the types of all values at runtime and flags code that does not make sense as it runs.

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

- It is used for:
    - web development (server-side),
    - software development,
    - mathematics,
    - system scripting.

<u>1.6.4 Flask</u>

- Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

- Flask is an API of Python that allows us to build up web-applications. It was developed by Armin Ronacher. Flask's framework is more explicit than Django's framework and is also easier to learn because it has less base code to implement a simple web-Application. A Web-Application Framework or Web Framework is the collection of modules and libraries that helps the developer to write applications without writing the low-level codes such as protocols, thread management, etc. Flask is based on WSGI(Web Server Gateway Interface) toolkit and Jinja2 template engine.

1.6.5 JavaScript

- JavaScript is a lightweight, cross-platform, and interpreted compiled programming language which is also known as the scripting language for webpages. It is well-known for the development of web pages, many non-browser environments also use it. JavaScript can be used for Client-side developments as well as Server-side developments. Javascript is both imperative and declarative type of language. JavaScript contains a standard library of objects, like Array, Date, and Math, and a core set of language elements like operators, control structures, and statements.

- JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:

  - Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.

  - Server-side JavaScript extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

1.6.6 Cascading Style Sheets(CSS)

- CSS stands for Cascading Style Sheets

- CSS describes how HTML elements are to be displayed on screen, paper, or in other media

- CSS saves a lot of work. It can control the layout of multiple web pages all at once

- External stylesheets are stored in CSS file

- Cascading Style Sheets, fondly referred to as CSS, is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page. It describes how a webpage should look: it prescribes colors, fonts, spacing, and much more. In short, you can make your website look however you want. CSS lets developers and designers define how it behaves, including how elements are positioned in the browser.

1.6.7 HTML

- HTML stands for HyperText Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between web pages. A markup language is used to define the text document within the tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

- At its heart, HTML is a language made up of elements, which can be applied to pieces of text to give them different meaning in a document (Is it a paragraph? Is it a bulleted list? Is it part of a table?), structure a document into logical sections (Does it have a header? Three columns of content? A navigation menu?), and embed content such as images and videos into a page. This module will introduce the first two of these and introduce fundamental concepts and syntax you need to know to understand HTML.

2 .Proposed System (Research paper here)

2.1 Study of Similar Systems (If required research paper can be included)

Visualization application play an important role in the learning, visualizing complex information, study , data in various fieldsSoftware visualization is the practice of creating visual tools to map software elements or otherwise display aspects of source code. This can be done with all kinds of programming languages in different ways with different criteria and tools..This project Sorting algorithm visualizer is a project which allows the users to select the sort algorithm, select the array size, and speed of the visualization of the algorithm .Sorting algorithm visualization can be defined as the use of images to convey how the array of data gets sorted. Bubble , Merge , Selection , Quick, Heap , Selection Sorting algorithms are there in the application .

In 1644, Michael Florent Van Langren, a Flemish astronomer, is believed to have provided the first visual representation of statistical data. The one-dimensional line graph below shows the twelve known estimates at the time of the difference in longitude between Toledo and Rome as well as the name of each astronomer who provided the estimate. What is notable here is that while Van Langren could have provided this information in a table, it is the use of the graph that really visually displays the wide variations in estimates.

The 18th century saw the beginning of thematic mapping. Attempts at the thematic mapping of geologic, economic, and medical data were made near the end of the century. Abstract graphs of functions, measurement error, and collection of empirical data were introduced at this time.The latter half of the 19th century is what Friendly calls the Golden Age of statistical graphics. Two famous examples of data visualization from that era include John Snow’s map of cholera outbreaks in the London epidemic of 1854 and

Charles Minard’s 1869 chart showing the number of men in Napoleon’s 1812 infamous Russian campaign army, with army location indicated by the X-axis, and extreme cold temperatures indicated at points when frostbite took a fatal toll.

2.2 Feasibility Study

It is a general practice to analyze the scope of the project or the idea before taking any further steps and here is where feasibility study comes into the picture. The study includes estimating the level of expertise required, qualitative and quantitative assessment of the possible outcomes, the skill-set that is required, identification of the crucial factors, probable risk and operational gaps.

The thorough analysis of the practicality and sustainability of the business idea brings together all the elements of knowledge that helps in realizing if the project justifies the Study. A Sorting algorithm visualizer helps students to understand the sorting algorithm visually . There are few sorting algorithm options there - Bubble sort , Merge sort , Selection sort , Quick Sort , ,Selection Sort ,Heap Sort . Users can understand which algorithm works faster and which one is quick between other algorithms as per the length of the Array. Student / User can adjust the speed of the sorting algorithm so that he goes by each step by understanding .

2.3 Objectives of Proposed System

- The main aim of the ‘ Sorting algorithm Visualizer ’application is to provide an easy way to understand the process of sorting algorithms .

- This Application ‘Sorting algorithm visualizer’ is a very simple UI and it allows the users to select the sort algorithm, select the array size , and the user can control the speed of sorting so that the user can understand the core properly.

- Algorithm visualization uses computer graphics to show the actions of an algorithm step by step. Algorithm visualization is used to help students understand the concept of sorting algorithm to visualize it

- Increase Understanding of Sorting algorithms .
    - In computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order. 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.

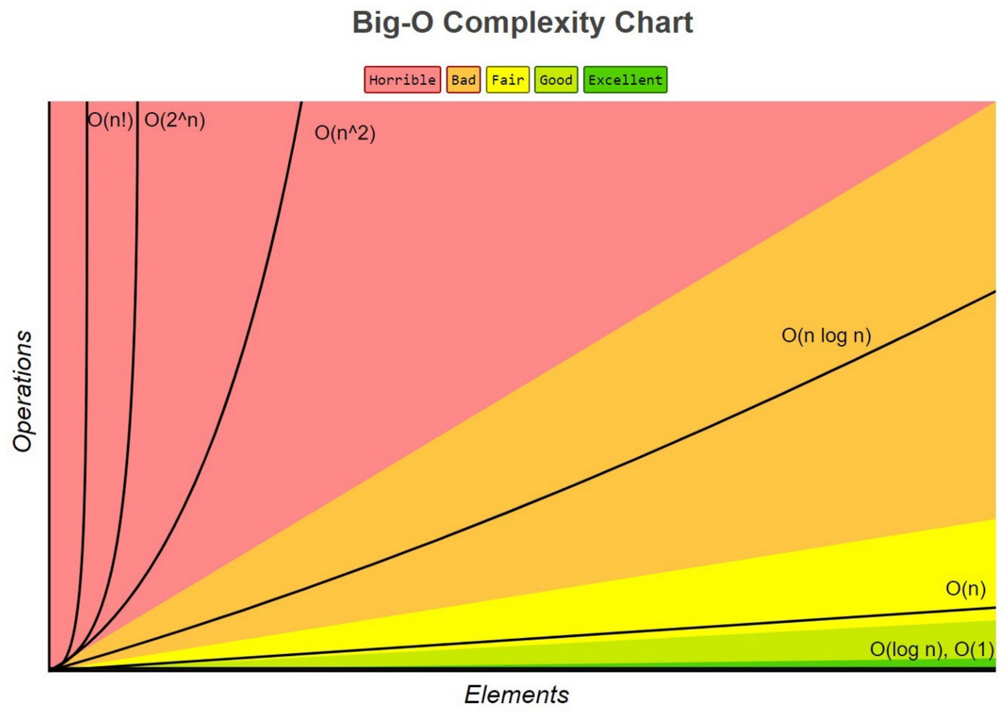
  - A sorting algorithm is a method for reorganizing a large number of items into a specific order, such as alphabetical, highest-to-lowest value or shortest-to-longest distance. Sorting algorithms take lists of items as input data, perform specific operations on those lists and deliver ordered arrays as output. The many applications of sorting algorithms include organizing items by price on a retail website and determining the order of sites on a search engine results page (SERP).

  - Since the dawn 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. For example, bubble sort was analyzed as early as 1956. Although many consider it a solved problem, useful new sorting algorithms are still being invented (for example, library sort was first published in 2004).

  - 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 algorithm.

- Increase Understanding of Time complexity .

  - Space complexity is an amount of memory used by the algorithm (including the input values of the algorithm), to execute it completely and produce the result.
  - We know that to execute an algorithm it must be loaded in the main memory. The memory can be used in different forms:
    - Variables (This includes the constant values and temporary values)
    - Program Instruction
    - Execution



- Increase Understanding of Space complexity
  - In computer science, the time complexity is the computational complexity that describes the amount of computer time it takes to run an algorithm. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform. Thus, the amount of time taken and the number of elementary operations performed by the algorithm are taken to be related by a constant factor.

  - Time complexity is simply a measure of the time it takes for a function or expression to complete its task, as well as the name of the process to measure that time. It can be applied to almost any algorithm or function but is more useful for recursive functions. There is little point in measuring time complexity for applications such as fetching the username and password from a database for comparison or simply saving data whether it is 20 ms or 5 ms; that would be more in the line of access time. It has nothing to do with caring about its execution time, but rather that the difference is negligible. However, if there is a recursive function that may be called multiple times, determining and understanding the source of its time complexity may help shorten the overall processing time from, say, 600 ms to 100 ms.

- Time complexity is expressed typically in the "big O notation," but there are other notations. This is a mathematical representation of the upper limit of the scaling factor for an algorithm and is written as O(Nn), with "N" being the number of inputs and "n" being the number of looping expressions. For example, we have the algorithm

<u>2.4 Users of System</u>

<u>2.4.1 Teacher</u>:-

One of the most common ways to teach students visualizing is to describe it as creating a picture or movie in our mind. We want students to constantly be adding to, changing, tweaking, and revising their mental images just like a movie is constantly evolving.

Visualization enhances attention because it keeps the audience focused on the subject matter. For example, when students visualize the content of a lesson or a passage that they are reading, comprehension increases dramatically. Many students report that they create a "movie in their head" and the visuals maintain their concentration and interest. In addition, when students visualize a scene before they write it, their ideas tend to be more cohesive, their thoughts remain fluid, and their stories integrate more descriptive language.

<u>2.4.1 Student</u>:-

Visualization refers to our ability to create pictures in our heads based on what we read or hear. When words are consciously used to create mental images, understanding is accelerated. Consequently, those who make use of visualization have an advanced ability to understand, learn, and remember.

Visualizations present a large number of relationships at a single time. Visual or textual clues can focus attention on meaningful items or guide the learner through the visualization in a particular order. Here are 5 importance of Studying concepts visually .

- Visualization decreases stress and reduces performance anxiety.
- Visualization boosts confidence.
- Visualization improves your abilities.
- Visualization alleviates pain & helps you heal faster. .
- Visualization enhances motivation.

3.Analysis and Design

3.1 System Requirements

3.1.1 Functional Requirements:

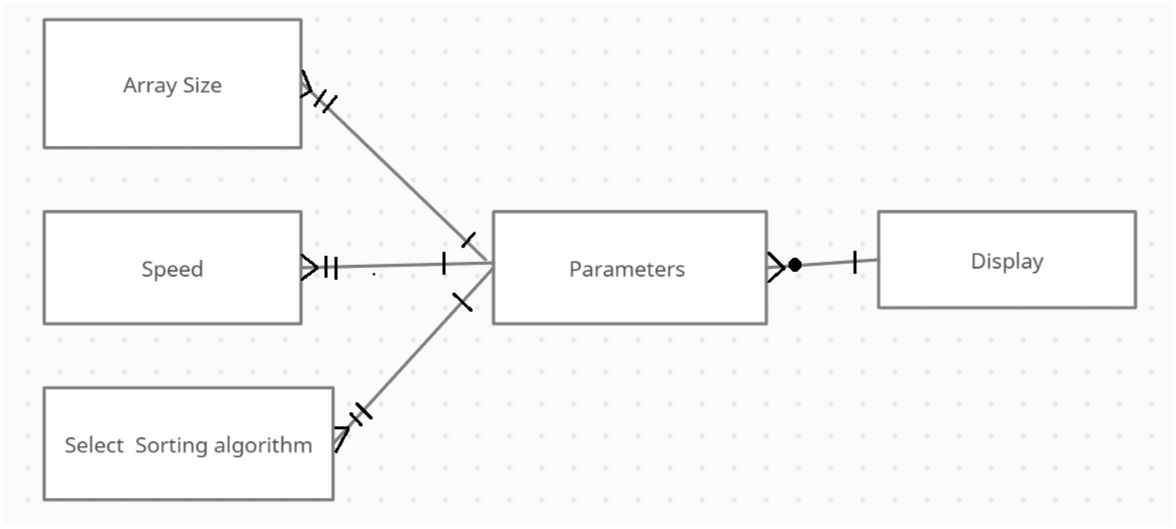
There are a lot of software requirements included in the functional requirements of the Sorting algorithm visualizer, which contains various processes like

- Array size:
  - User set the length of the array
- Speed:
  - Speed is used to set the working speed of the sorting algorithm.
- Select sorting algorithm:
  - There are many sorting algorithm , from that user select one from that

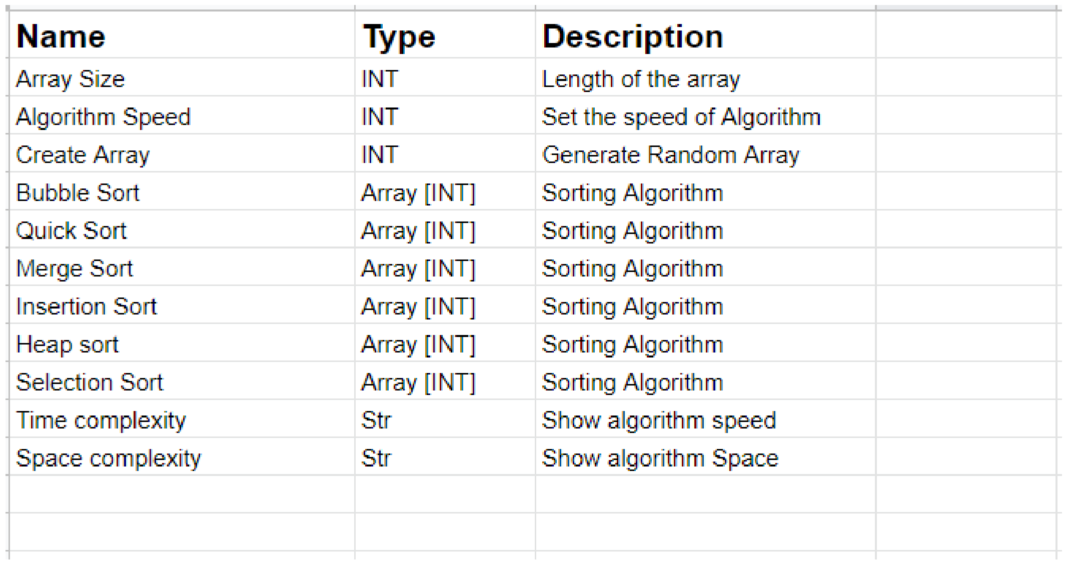
3.1.2 Non Functional Requirements:

There are few software requirements included in the non-functional requirements of the Sorting algorithm visualizer , which contains various processes like improving code quality , improving visualization quality and readability ,maintainability, performance ,etc .

3.2 Entity Relationship Diagram (ERD)

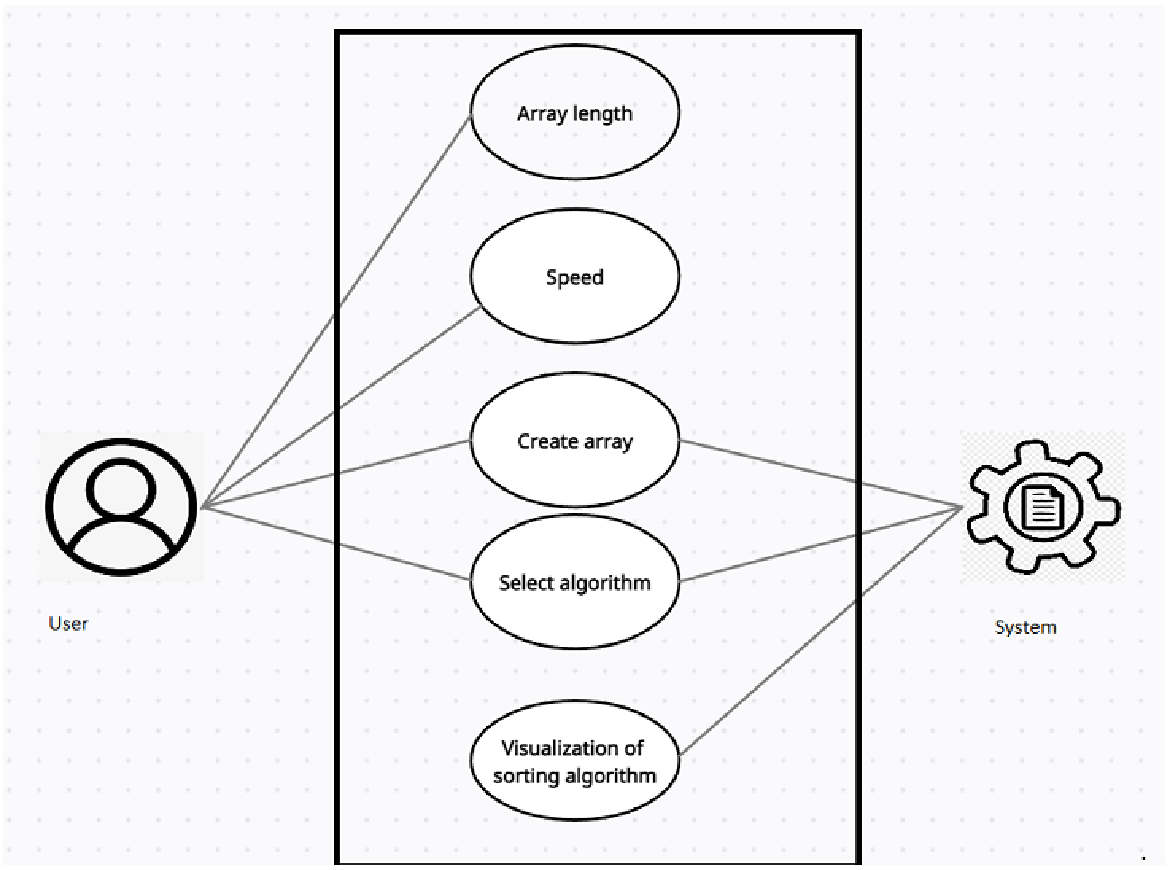


3.3 Table Structure

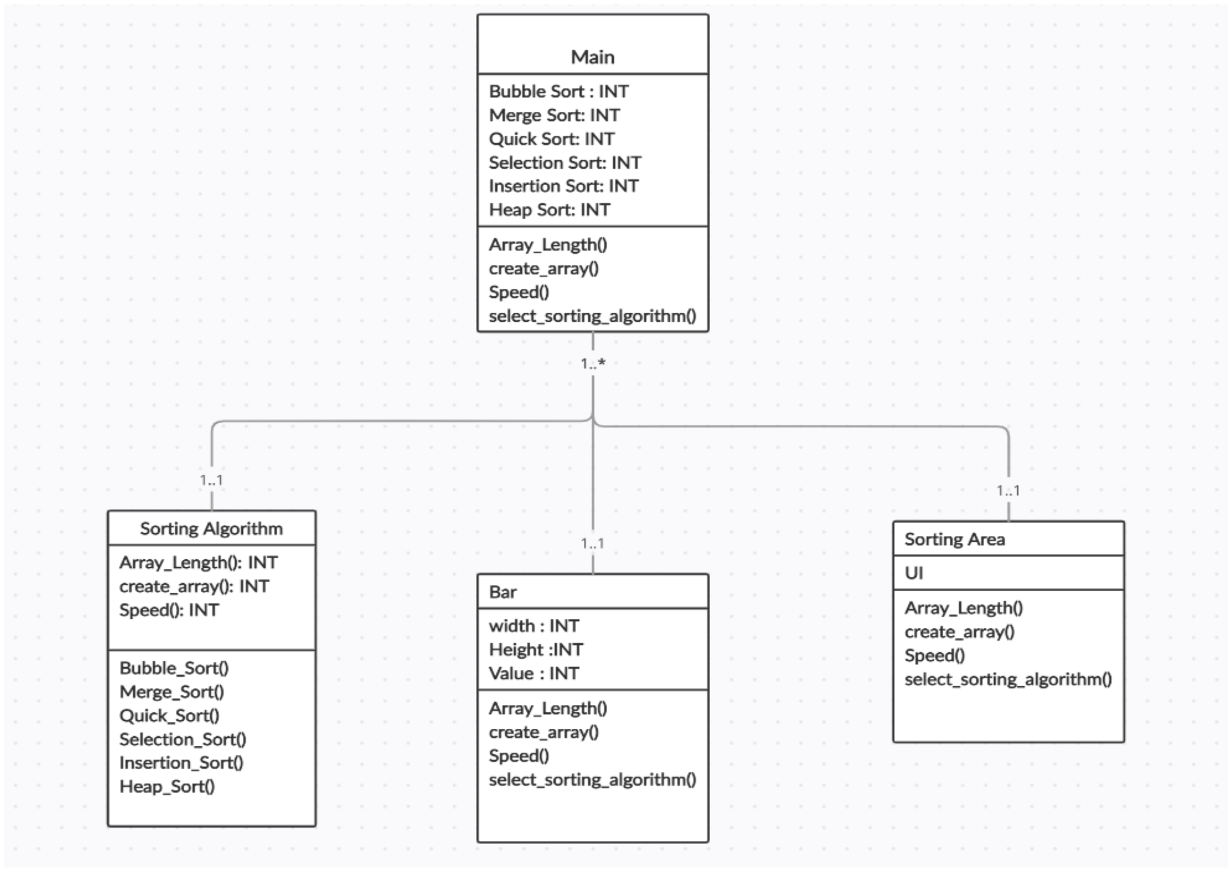


Name	Type	Description	
Array Size	INT	Length of the array	
Algorithm Speed	INT	Set the speed of Algorithm	
Create Array	INT	Generate Random Array	
Bubble Sort	Array [INT]	Sorting Algorithm	
Quick Sort	Array [INT]	Sorting Algorithm	
Merge Sort	Array [INT]	Sorting Algorithm	
Insertion Sort	Array [INT]	Sorting Algorithm	
Heap sort	Array [INT]	Sorting Algorithm	
Selection Sort	Array [INT]	Sorting Algorithm	
Time complexity	Str	Show algorithm speed	
Space complexity	Str	Show algorithm Space	

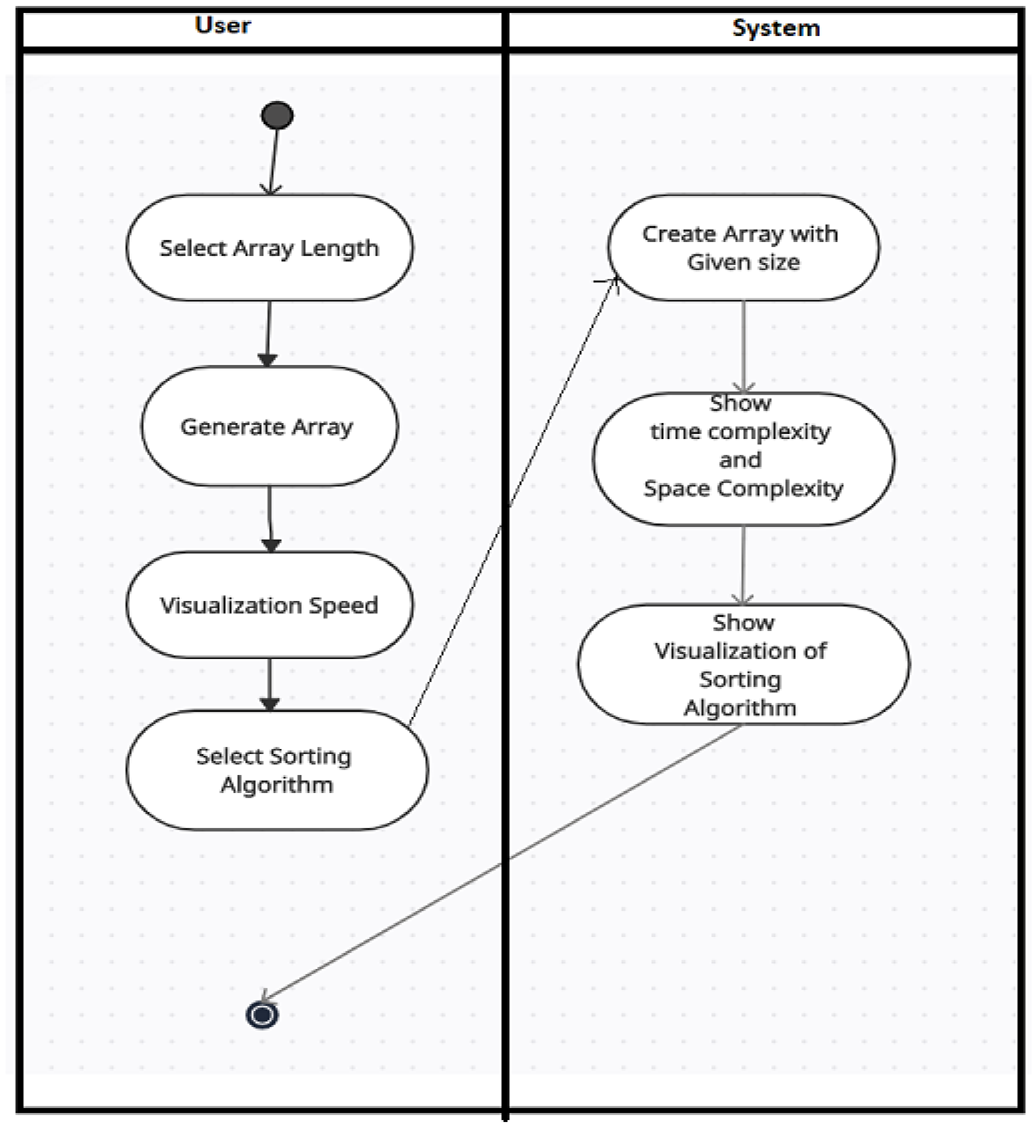
3.4 Use Case Diagram



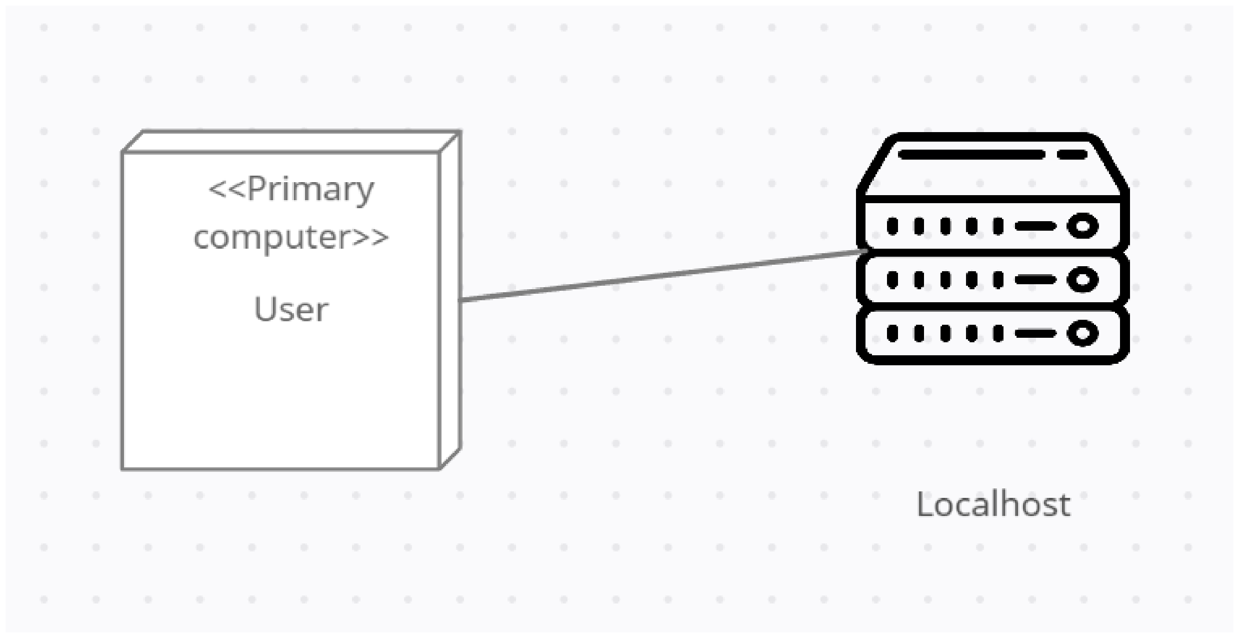
3.5 Class Diagram



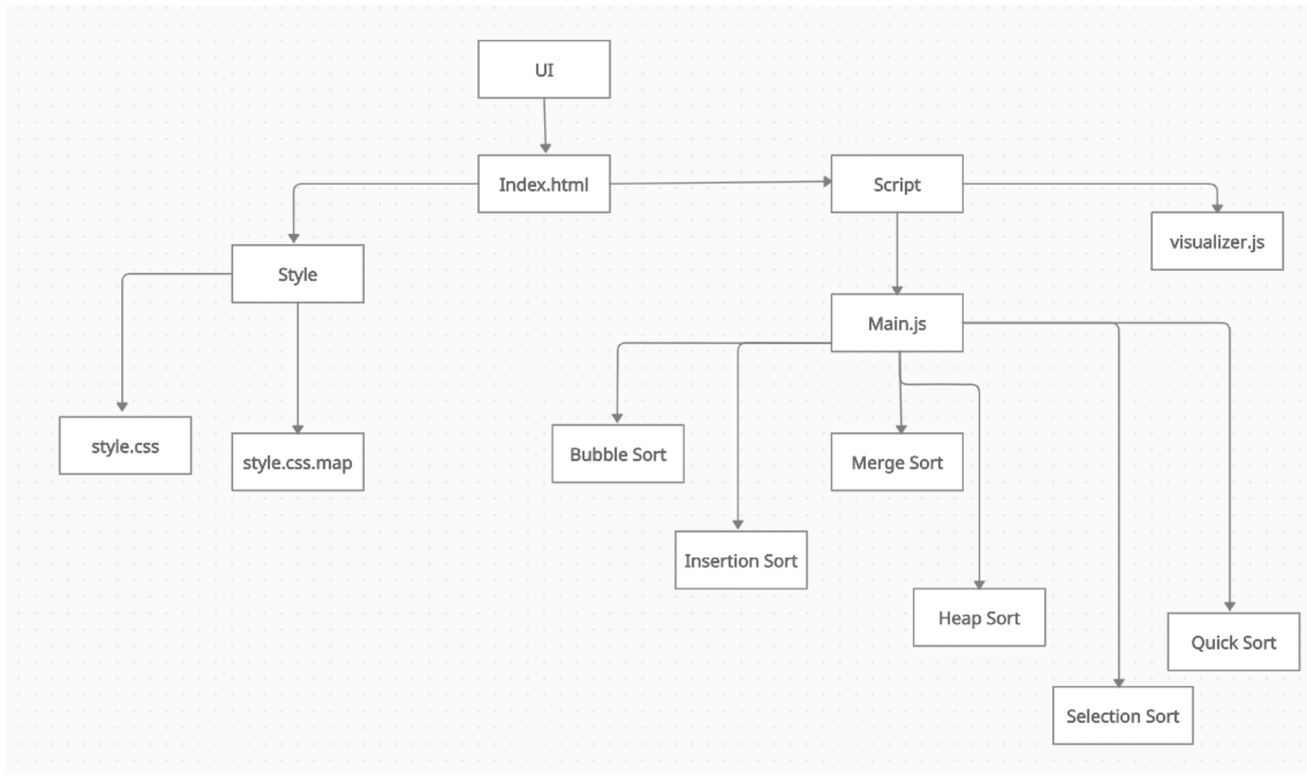
3.6 Activity Diagram


User
System
Select Array Length
Generate Array
Visualization Speed
Select Sorting Algorithm
Create Array with Given size
Show time complexity and Space Complexity
Show Visualization of Sorting Algorithm

3.7 Deployment Diagram

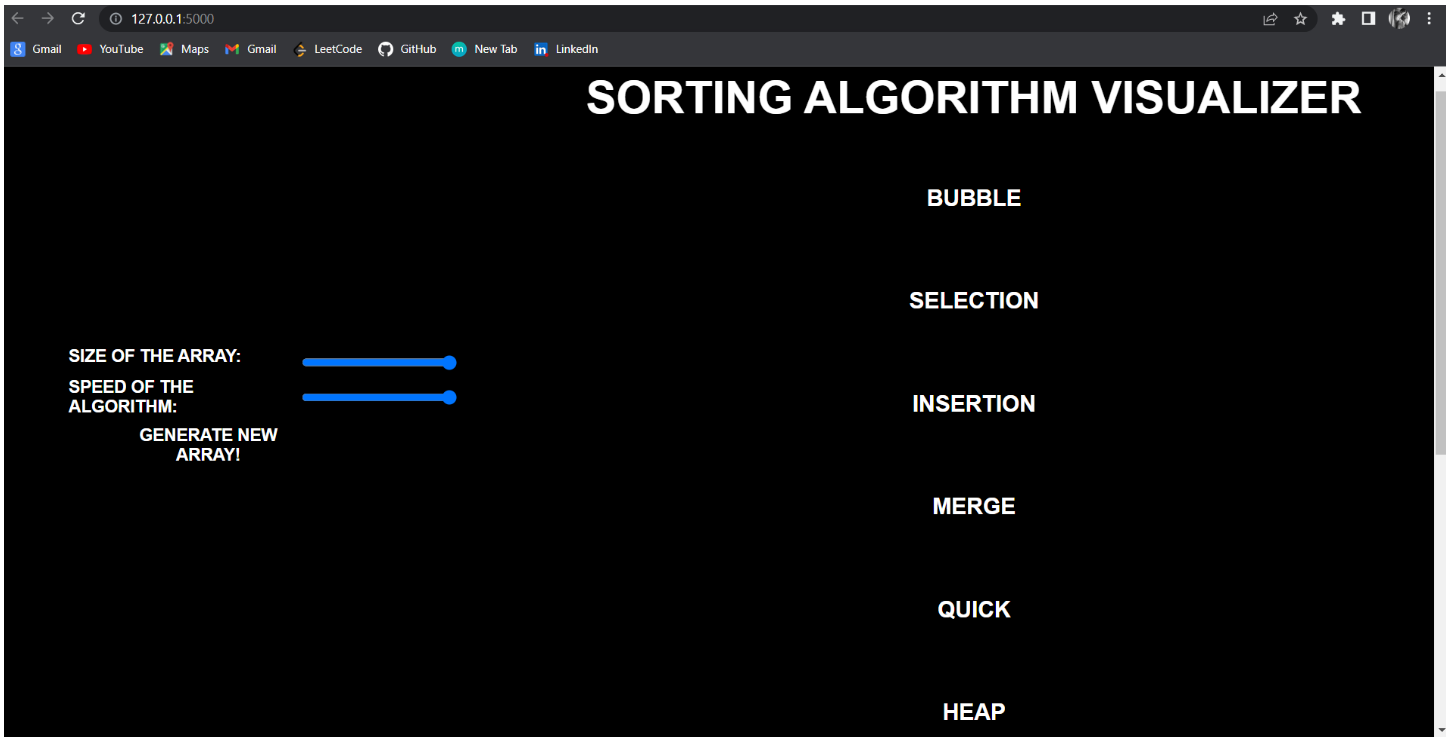


3.8 Module Diagram

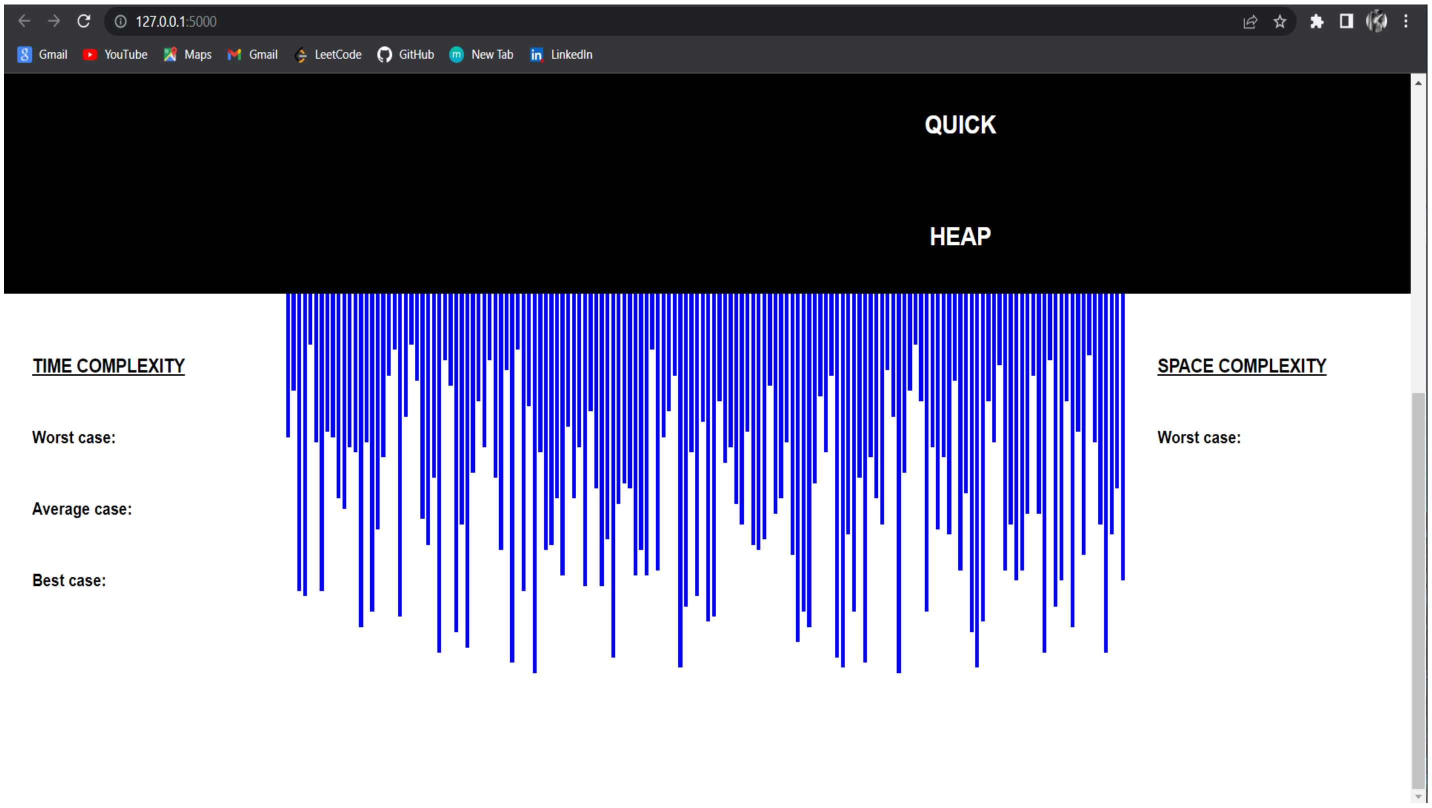


3.9 Sample Input and Output Screens

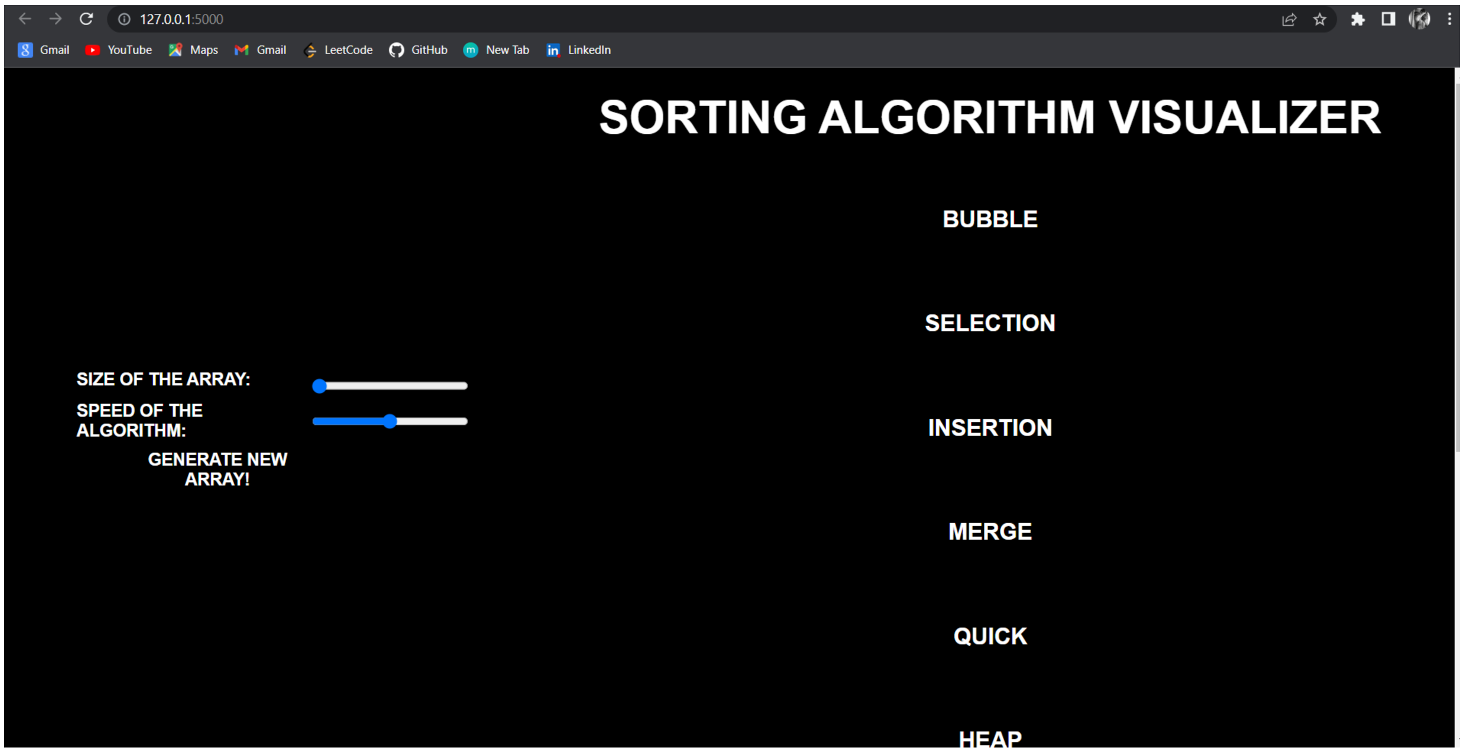
Sample Input 1 :-



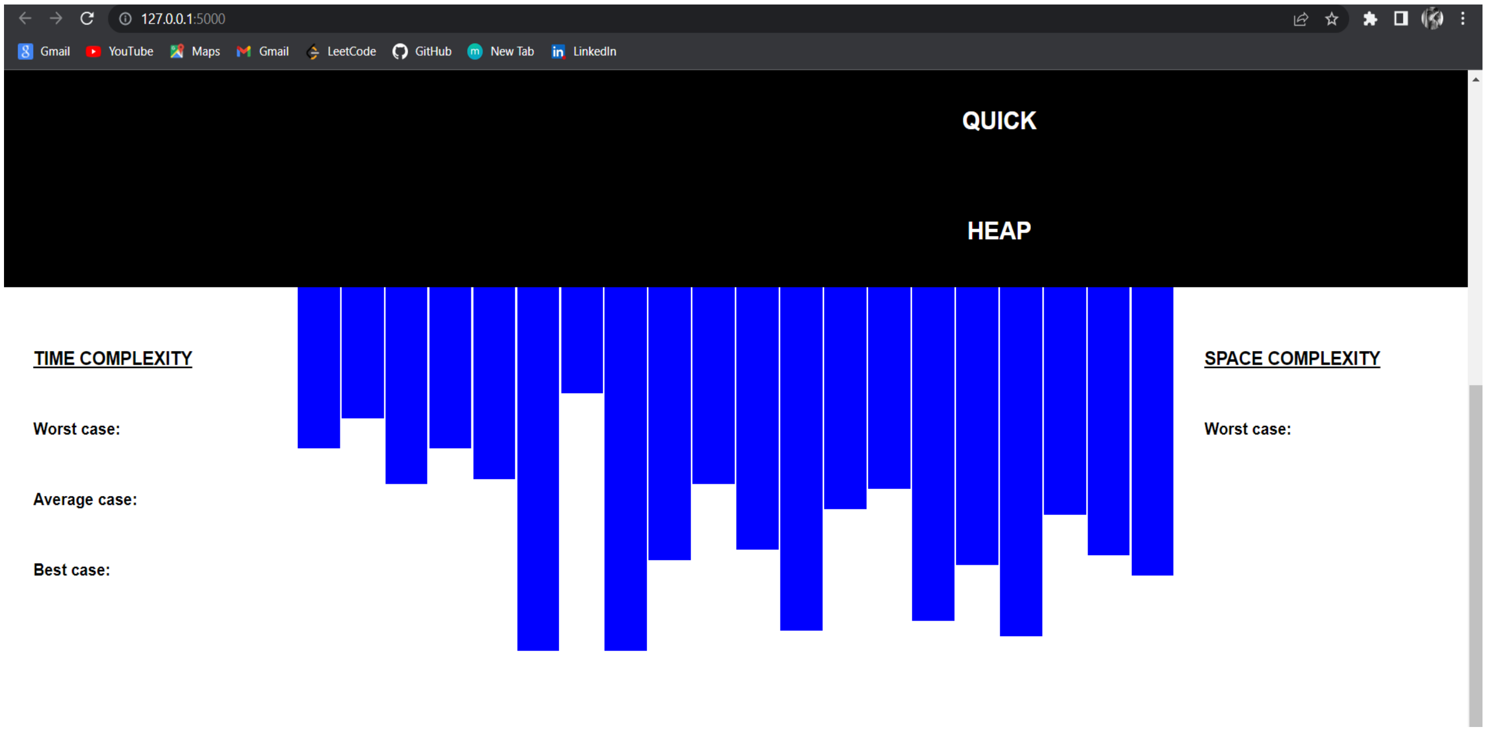
Output



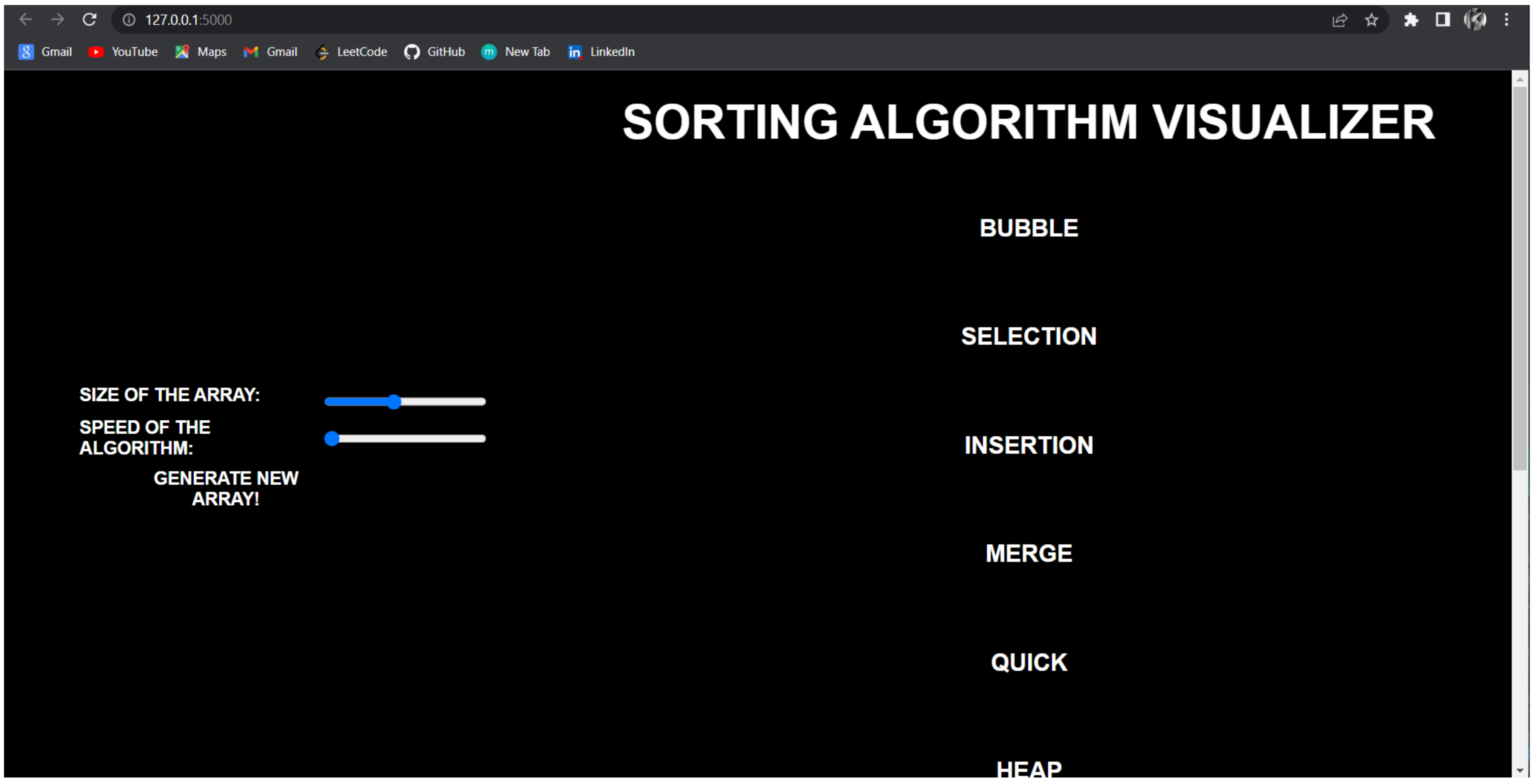
Sample Input 2 :-



Output



Sample Input 3 :-



SORTING ALGORITHM VISUALIZER

BUBBLE

SELECTION

INSERTION

MERGE

QUICK

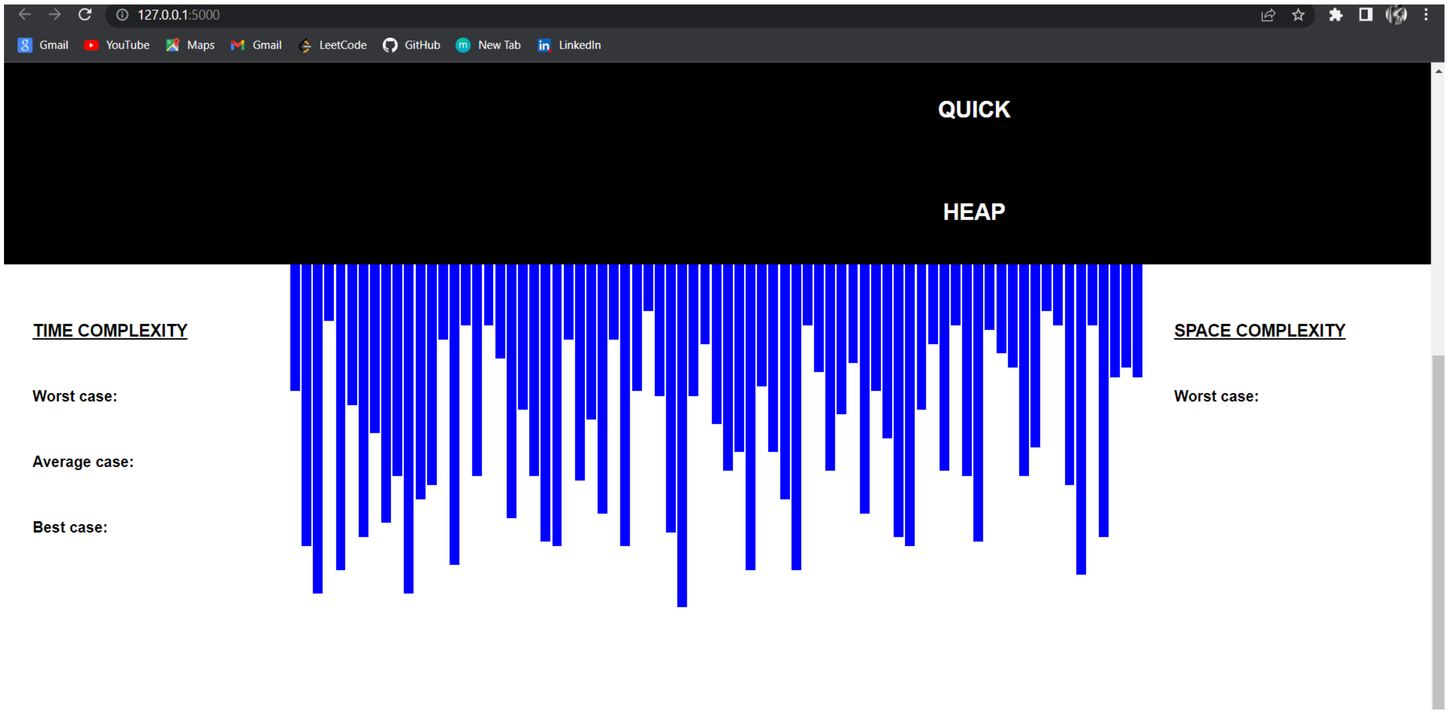
HEAP

SIZE OF THE ARRAY:

SPEED OF THE ALGORITHM:

GENERATE NEW ARRAY!

Output



QUICK

HEAP

TIME COMPLEXITY

Worst case:

Average case:

Best case:

SPACE COMPLEXITY

Worst case:

4 .Coding

4.1 Algorithms

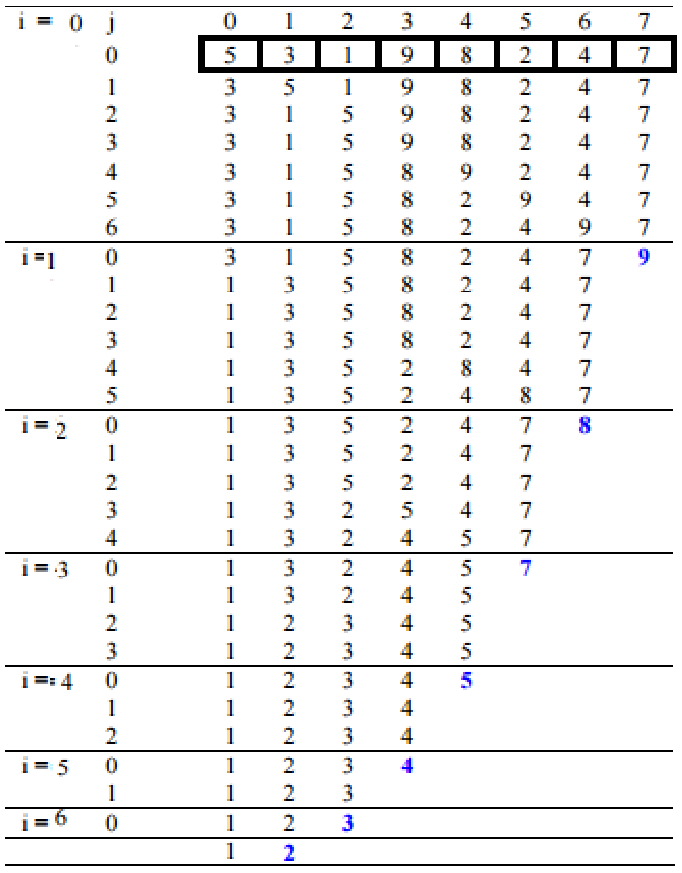
A programming algorithm is a procedure or formula used for solving a problem. It is based on conducting a sequence of specified actions in which these actions describe how to do something, and your computer will do it exactly that way every time. An algorithm works by following a procedure, made up of inputs. Once it has followed all the inputs, it will see a result, also known as output.

Characteristics of an algorithm:

- Precision – the steps are precisely stated.
- Uniqueness – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- Finiteness – the algorithm stops after a finite number of instructions are executed.
- Input – the algorithm receives input.
- Output – the algorithm produces output.
- Generality – the algorithm applies to a set of inputs.

4.1.1 Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst case time complexity is quite high.

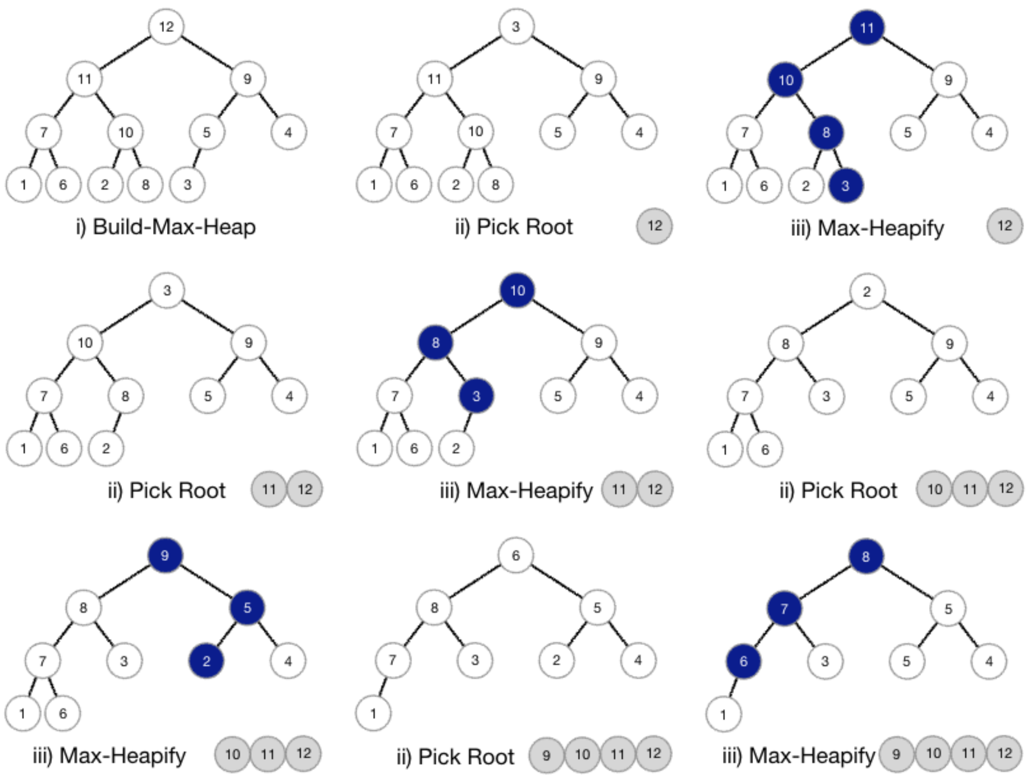


i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	4				
	1	1	2	3					
i = 6	0	1	2	3					
		1	2						

```
function Bubble() {
  //Setting Time complexities
  document.getElementById("Time_Worst").innerText = "O(N^2)";
  document.getElementById("Time_Average").innerText = "Θ(N^2)";
  document.getElementById("Time_Best").innerText = "Ω(N)";

  //Setting Space complexity
  document.getElementById("Space_Worst").innerText = "O(1)";

  c_delay = 0;

  for (var i = 0; i < array_size - 1; i++) {
    for (var j = 0; j < array_size - i - 1; j++) {
      div_update(divs[j], div_sizes[j], "yellow"); //Color update

      if (div_sizes[j] > div_sizes[j + 1]) {
        div_update(divs[j], div_sizes[j], "red"); //Color update
        div_update(divs[j + 1], div_sizes[j + 1], "red"); //Color update

        var temp = div_sizes[j];
        div_sizes[j] = div_sizes[j + 1];
        div_sizes[j + 1] = temp;

        div_update(divs[j], div_sizes[j], "red"); //Height update
        div_update(divs[j + 1], div_sizes[j + 1], "red"); //Height update
      }
      div_update(divs[j], div_sizes[j], "blue"); //Color updat
    }
    div_update(divs[j], div_sizes[j], "green"); //Color update
  }
  div_update(divs[0], div_sizes[0], "green"); //Color update

  enable_buttons();
}
```

4.1.2 Heap Sort

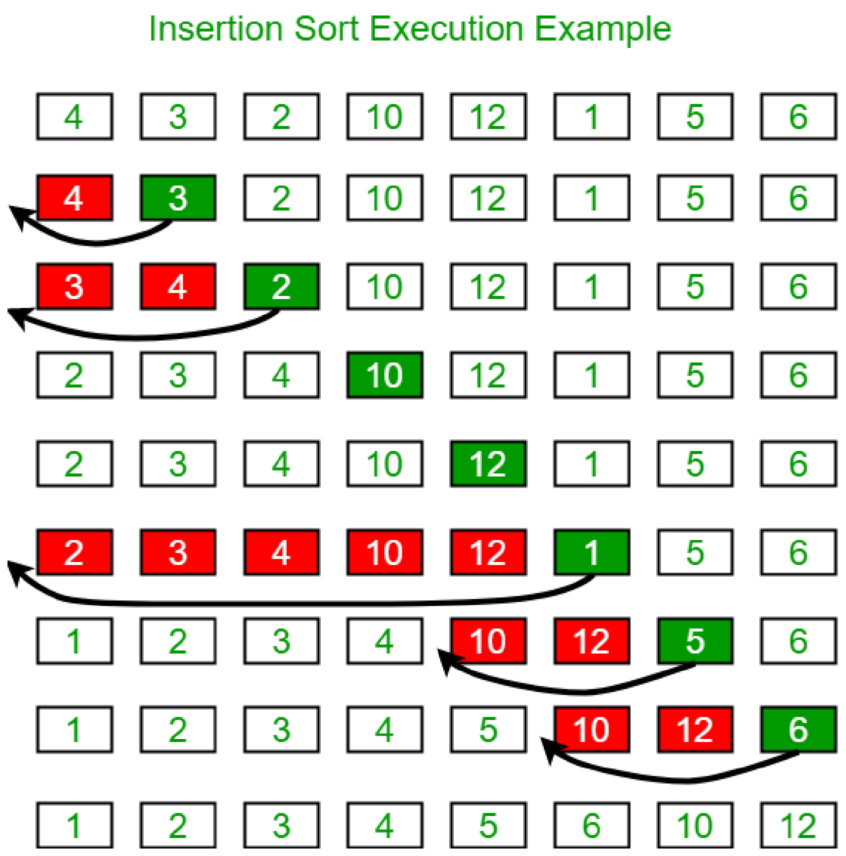
Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.



```
function Heap() {
  //Setting Time complexities
  document.getElementById("Time_Worst").innerText = "O(N log N)";
  document.getElementById("Time_Average").innerText = "Θ(N log N)";
  document.getElementById("Time_Best").innerText = "Ω(N log N)";

  //Setting Space complexity
  document.getElementById("Space_Worst").innerText = "O(1)";
  c_delay = 0;
  heap_sort();
  enable_buttons();}

function swap(i, j) {
  div_update(divs[i], div_sizes[i], "red"); //Color update
  div_update(divs[j], div_sizes[j], "red"); //Color update

  var temp = div_sizes[i];
  div_sizes[i] = div_sizes[j];
  div_sizes[j] = temp;

  div_update(divs[i], div_sizes[i], "red"); //Height update
  div_update(divs[j], div_sizes[j], "red"); //Height update

  div_update(divs[i], div_sizes[i], "blue"); //Color update
  div_update(divs[j], div_sizes[j], "blue"); //Color update
}

function max_heapify(n, i) {
  var largest = i;
  var l = 2 * i + 1;
  var r = 2 * i + 2;

  if (l < n && div_sizes[l] > div_sizes[largest]) {
    if (largest != i) {
      div_update(divs[largest], div_sizes[largest], "blue"); //Color update
    }

    largest = l;

    div_update(divs[largest], div_sizes[largest], "red"); //Color update
  }
  if (r < n && div_sizes[r] > div_sizes[largest]) {
    if (largest != i) {
      div_update(divs[largest], div_sizes[largest], "blue"); //Color update
    }

    largest = r;

    div_update(divs[largest], div_sizes[largest], "red"); //Color update
  }

  if (largest != i) {
    swap(i, largest);

    max_heapify(n, largest);
  }
}

function heap_sort() {
  for (var i = Math.floor(array_size / 2) - 1; i >= 0; i--) {
    max_heapify(array_size, i);
  }

  for (var i = array_size - 1; i > 0; i--) {
    swap(0, i);
    div_update(divs[i], div_sizes[i], "green"); //Color update
    div_update(divs[i], div_sizes[i], "yellow"); //Color update

    max_heapify(i, 0);

    div_update(divs[i], div_sizes[i], "blue"); //Color update
    div_update(divs[i], div_sizes[i], "green"); //Color update
  }
  div_update(divs[i], div_sizes[i], "green"); //Color update
}
```

4.1.3 Insertion Sort

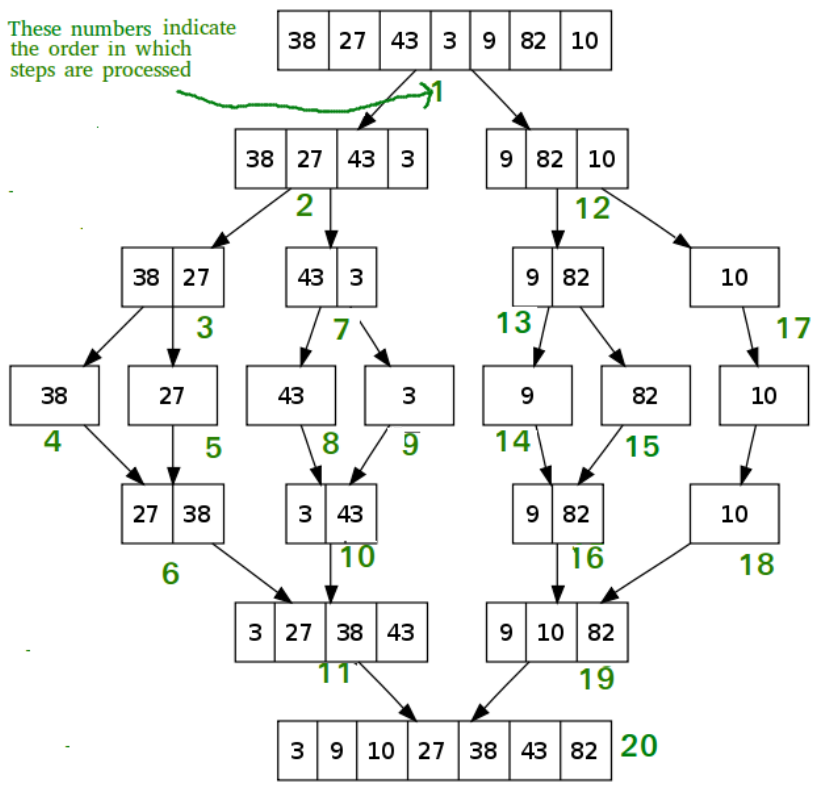
Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.



```
function Insertion() {
 //Setting Time complexities
 document.getElementById("Time_Worst").innerText = "O(N^2)";
 document.getElementById("Time_Average").innerText = "Θ(N^2)";
 document.getElementById("Time_Best").innerText = "Ω(N)";

 //Setting Space complexity
 document.getElementById("Space_Worst").innerText = "O(1)";
 c_delay = 0;
 for (var j = 0; j < array_size; j++) {
  div_update(divs[j], div_sizes[j], "yellow"); //Color update

  var key = div_sizes[j];
  var i = j - 1;
  while (i >= 0 && div_sizes[i] > key) {
   div_update(divs[i], div_sizes[i], "red"); //Color update
   div_update(divs[i + 1], div_sizes[i + 1], "red"); //Color update

   div_sizes[i + 1] = div_sizes[i];

   div_update(divs[i], div_sizes[i], "red"); //Height update
   div_update(divs[i + 1], div_sizes[i + 1], "red"); //Height update
   div_update(divs[i], div_sizes[i], "blue"); //Color update
   if (i == j - 1) {
    div_update(divs[i + 1], div_sizes[i + 1], "yellow"); //Color update
   } else {
    div_update(divs[i + 1], div_sizes[i + 1], "blue"); //Color update}
   i -= 1; }
  div_sizes[i + 1] = key;
  for (var t = 0; t < j; t++) {
   div_update(divs[t], div_sizes[t], "green"); //Color update
  }
 }
 div_update(divs[j - 1], div_sizes[j - 1], "green"); //Color update

 enable_buttons();}
```

4.1.4 MergeSort

The Merge Sort algorithm is a sorting algorithm that is considered as an example of the divide and conquer strategy. So, in this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner. We can think of it as a recursive algorithm that continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, we split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both the halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.



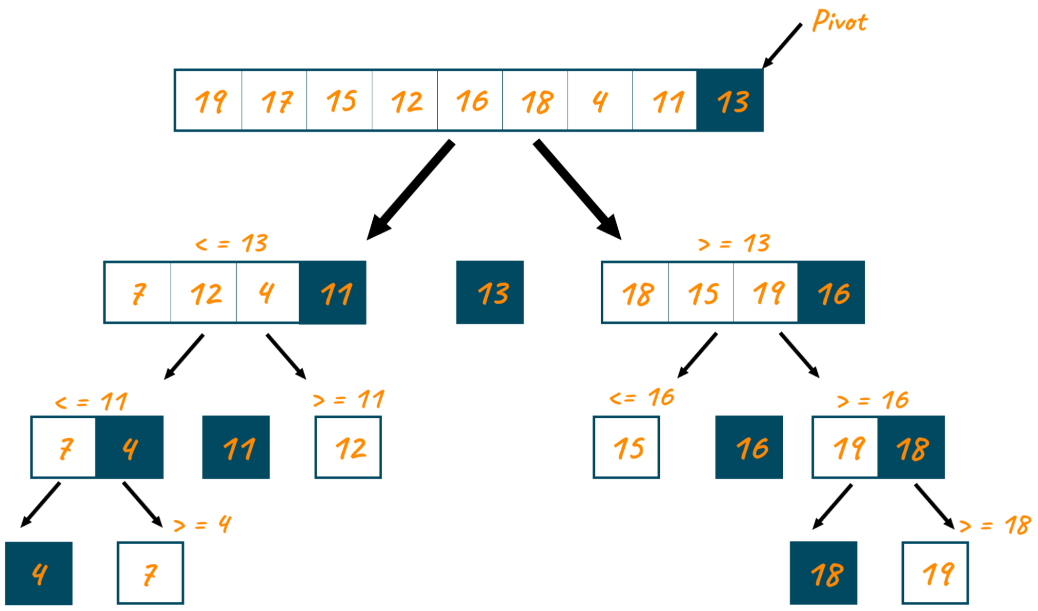
```
function Merge() {
  //Setting Time complexities
  document.getElementById("Time_Worst").innerText = "O(N log N)";
  document.getElementById("Time_Average").innerText = "Θ(N log N)";
  document.getElementById("Time_Best").innerText = "Ω(N log N)";

  //Setting Space complexity
  document.getElementById("Space_Worst").innerText = "O(N)";

  c_delay = 0;

  merge_partition(0, array_size - 1);

  enable_buttons();
}

function merge_sort(start, mid, end) {
  var p = start,
    q = mid + 1;

  var Arr = [],
    k = 0;

  for (var i = start; i <= end; i++) {
    if (p > mid) {
      Arr[k++] = div_sizes[q++];
      div_update(divs[q - 1], div_sizes[q - 1], "red"); //Color update
    } else if (q > end) {
      Arr[k++] = div_sizes[p++];
      div_update(divs[p - 1], div_sizes[p - 1], "red"); //Color update
    } else if (div_sizes[p] < div_sizes[q]) {
      Arr[k++] = div_sizes[p++];
      div_update(divs[p - 1], div_sizes[p - 1], "red"); //Color update
    } else {
      Arr[k++] = div_sizes[q++];
      div_update(divs[q - 1], div_sizes[q - 1], "red"); //Color update
    }
  }
  for (var t = 0; t < k; t++) {
    div_sizes[start++] = Arr[t];
    div_update(divs[start - 1], div_sizes[start - 1], "green"); //Color update
  }
}

function merge_partition(start, end) {
  if (start < end) {
    var mid = Math.floor((start + end) / 2);
    div_update(divs[mid], div_sizes[mid], "yellow"); //Color update
    merge_partition(start, mid);
    merge_partition(mid + 1, end);

    merge_sort(start, mid, end);
  }
}
```

4.1.5 QuickSort

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- Always pick first element as pivot.
- Always pick last element as pivot (implemented below)
- Pick a random element as pivot.
- Pick median as pivot.

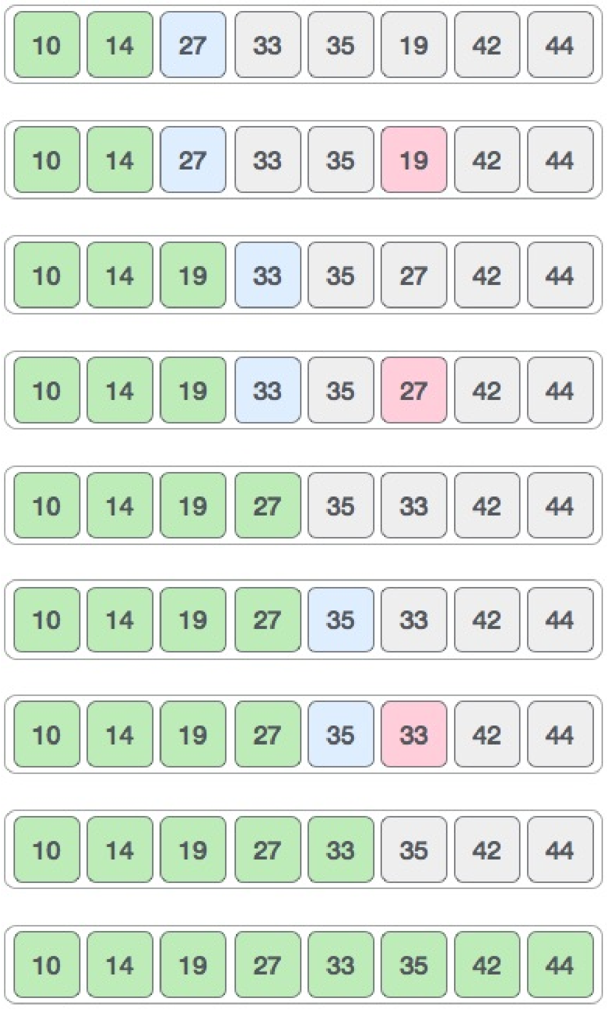
The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.



```
function Quick() {
  //Setting Time complexities
  document.getElementById("Time_Worst").innerText = "O(N^2)";
  document.getElementById("Time_Average").innerText = "Θ(N log N)";
  document.getElementById("Time_Best").innerText = "Ω(N log N)";

  //Setting Space complexity
  document.getElementById("Space_Worst").innerText = "O(log N)";

  c_delay = 0;

  quick_sort(0, array_size - 1);

  enable_buttons();
}

function quick_partition(start, end) {
  var i = start + 1;
  var piv = div_sizes[start]; //make the first element as pivot element.
  div_update(divs[start], div_sizes[start], "yellow"); //Color update

  for (var j = start + 1; j <= end; j++) {
    //re-arrange the array by putting elements which are less than pivot on
one side and which are greater that on other.
    if (div_sizes[j] < piv) {
      div_update(divs[j], div_sizes[j], "yellow"); //Color update

      div_update(divs[i], div_sizes[i], "red"); //Color update
      div_update(divs[j], div_sizes[j], "red"); //Color update

      var temp = div_sizes[i];
      div_sizes[i] = div_sizes[j];
      div_sizes[j] = temp;

      div_update(divs[i], div_sizes[i], "red"); //Height update
      div_update(divs[j], div_sizes[j], "red"); //Height update
      div_update(divs[i], div_sizes[i], "blue"); //Height update
      div_update(divs[j], div_sizes[j], "blue"); //Height update

      i += 1;
    }
  }
  div_update(divs[start], div_sizes[start], "red"); //Color update
  div_update(divs[i - 1], div_sizes[i - 1], "red"); //Color update

  var temp = div_sizes[start]; //put the pivot element in its proper place.
  div_sizes[start] = div_sizes[i - 1];
  div_sizes[i - 1] = temp;

  div_update(divs[start], div_sizes[start], "red"); //Height update
  div_update(divs[i - 1], div_sizes[i - 1], "red"); //Height update

  for (var t = start; t <= i; t++) {
    div_update(divs[t], div_sizes[t], "green"); //Color update
  }

  return i - 1; //return the position of the pivot
}

function quick_sort(start, end) {
  if (start < end) {
    //stores the position of pivot element
    var piv_pos = quick_partition(start, end);
    quick_sort(start, piv_pos - 1); //sorts the left side of pivot.
    quick_sort(piv_pos + 1, end); //sorts the right side of pivot.
  }
}
```

<u>4.1.6 Selection Sort</u>

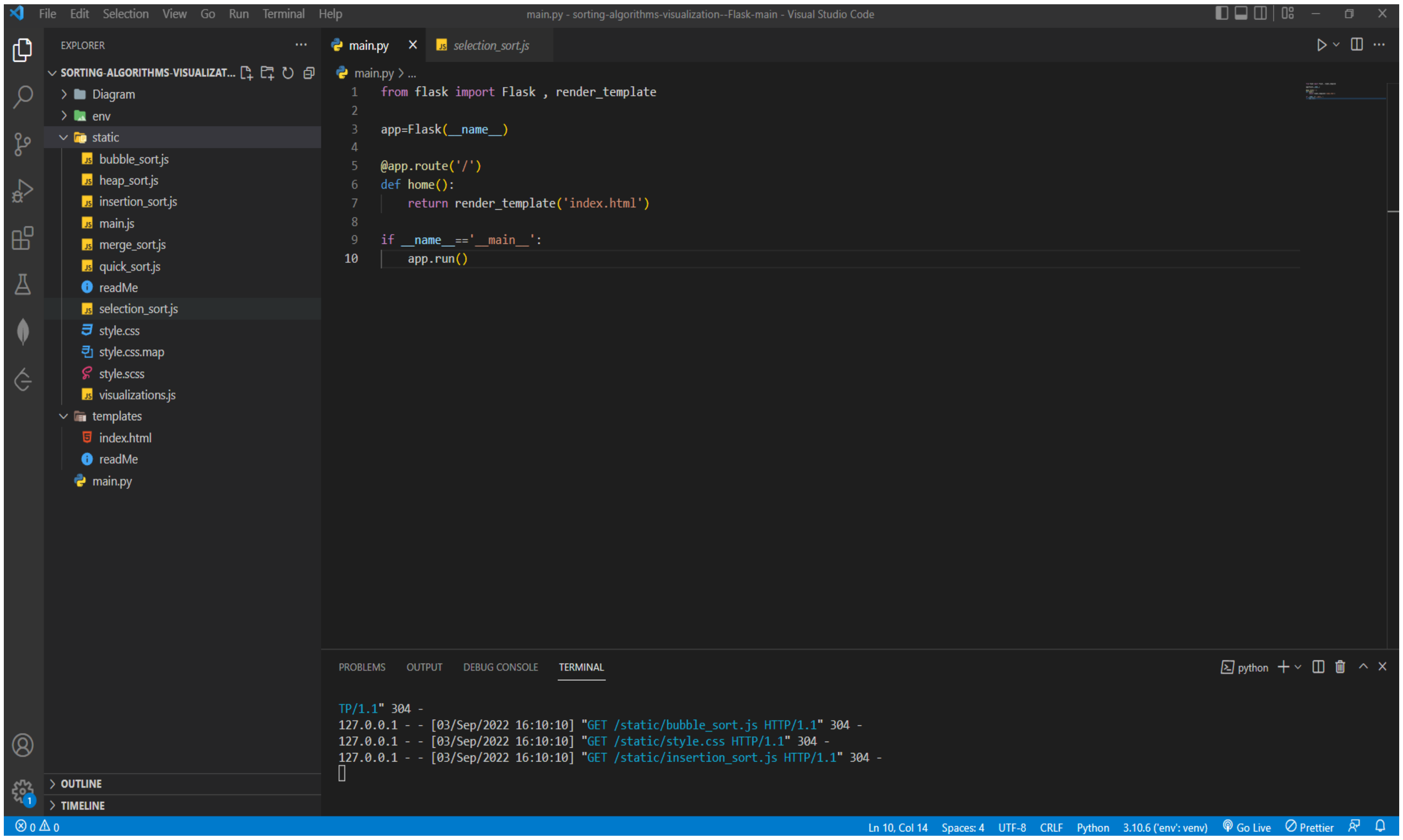
Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.



```
function Selection_sort() {
  //Setting Time complexities
  document.getElementById("Time_Worst").innerText = "O(N^2)";
  document.getElementById("Time_Average").innerText = "Θ(N^2)";
  document.getElementById("Time_Best").innerText = "Ω(N^2)";

  //Setting Space complexity
  document.getElementById("Space_Worst").innerText = "O(1)";

  c_delay = 0;

  for (var i = 0; i < array_size - 1; i++) {
    div_update(divs[i], div_sizes[i], "red"); //Color update

    index_min = i;

    for (var j = i + 1; j < array_size; j++) {
      div_update(divs[j], div_sizes[j], "yellow"); //Color update

      if (div_sizes[j] < div_sizes[index_min]) {
        if (index_min != i) {
          div_update(divs[index_min], div_sizes[index_min], "blue"); //Color
update
        }
        index_min = j;
        div_update(divs[index_min], div_sizes[index_min], "red"); //Color
update
      } else {
        div_update(divs[j], div_sizes[j], "blue"); //Color update
      }
    }

    if (index_min != i) {
      var temp = div_sizes[index_min];
      div_sizes[index_min] = div_sizes[i];
      div_sizes[i] = temp;

      div_update(divs[index_min], div_sizes[index_min], "red"); //Height
update
      div_update(divs[i], div_sizes[i], "red"); //Height update
      div_update(divs[index_min], div_sizes[index_min], "blue"); //Color
update
    }
    div_update(divs[i], div_sizes[i], "green"); //Color update
  }
  div_update(divs[i], div_sizes[i], "green"); //Color update

  enable_buttons();
}
```

4.2 Code Snippet

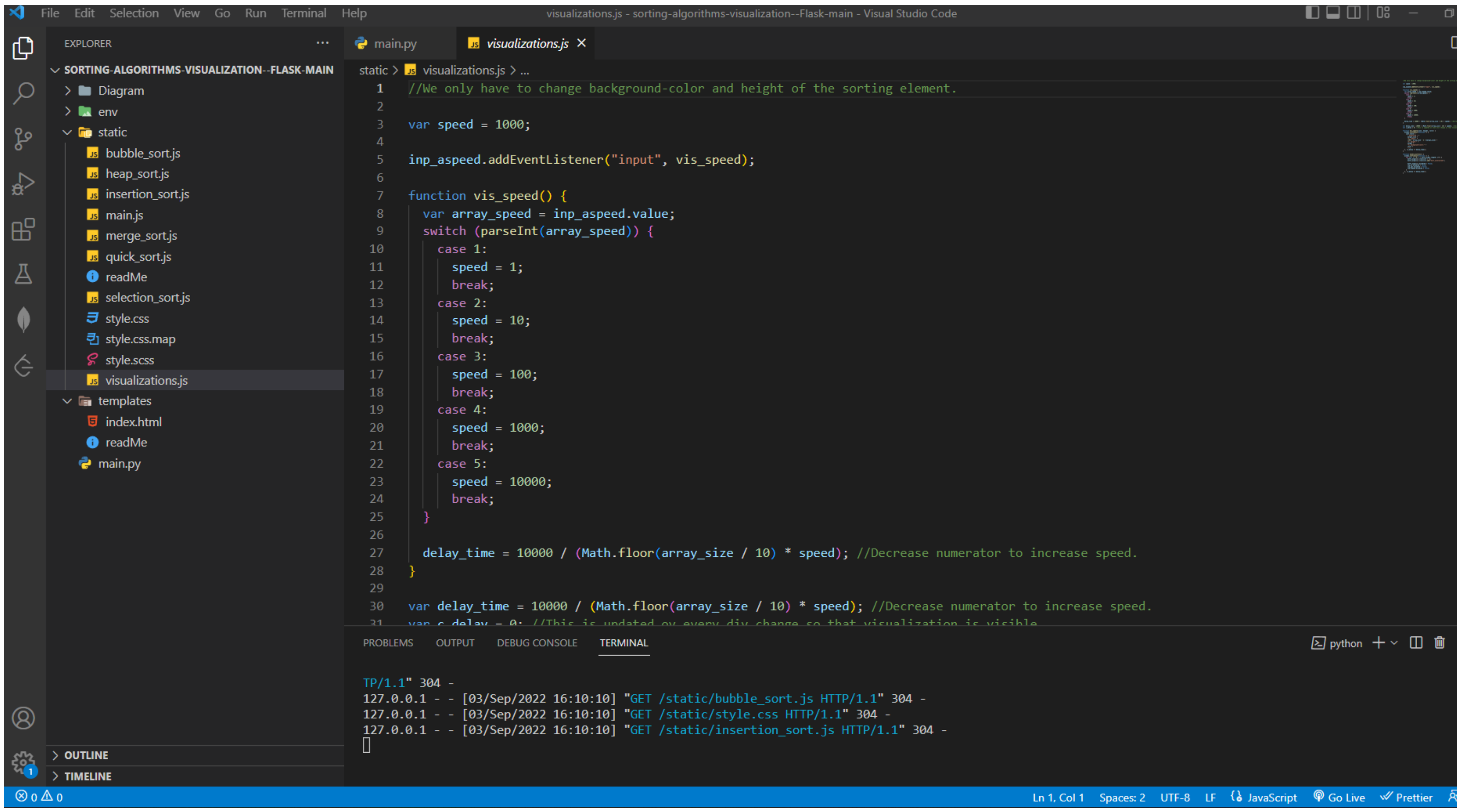
4.2.1 Framework - Flask


File Edit Selection View Go Run Terminal Help
main.py - sorting-algorithms-visualization--Flask-main - Visual Studio Code
EXPLORER
SORTING-ALGORITHMS-VISUALIZAT...
Diagram
env
static
bubble_sort.js
heap_sort.js
insertion_sort.js
main.js
merge_sort.js
quick_sort.js
readMe
selection_sort.js
style.css
style.css.map
style.scss
visualizations.js
templates
index.html
readMe
main.py
OUTLINE
TIMELINE
main.py
selection_sort.js
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
python
Ln 10, Col 14 Spaces: 4 UTF-8 CRLF Python 3.10.6 ('env': venv) Go Live Prettier

```python
from flask import Flask , render_template

app=Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

if __name__=='__main__':
    app.run()
```

```
TP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/bubble_sort.js HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/insertion_sort.js HTTP/1.1" 304 -
```

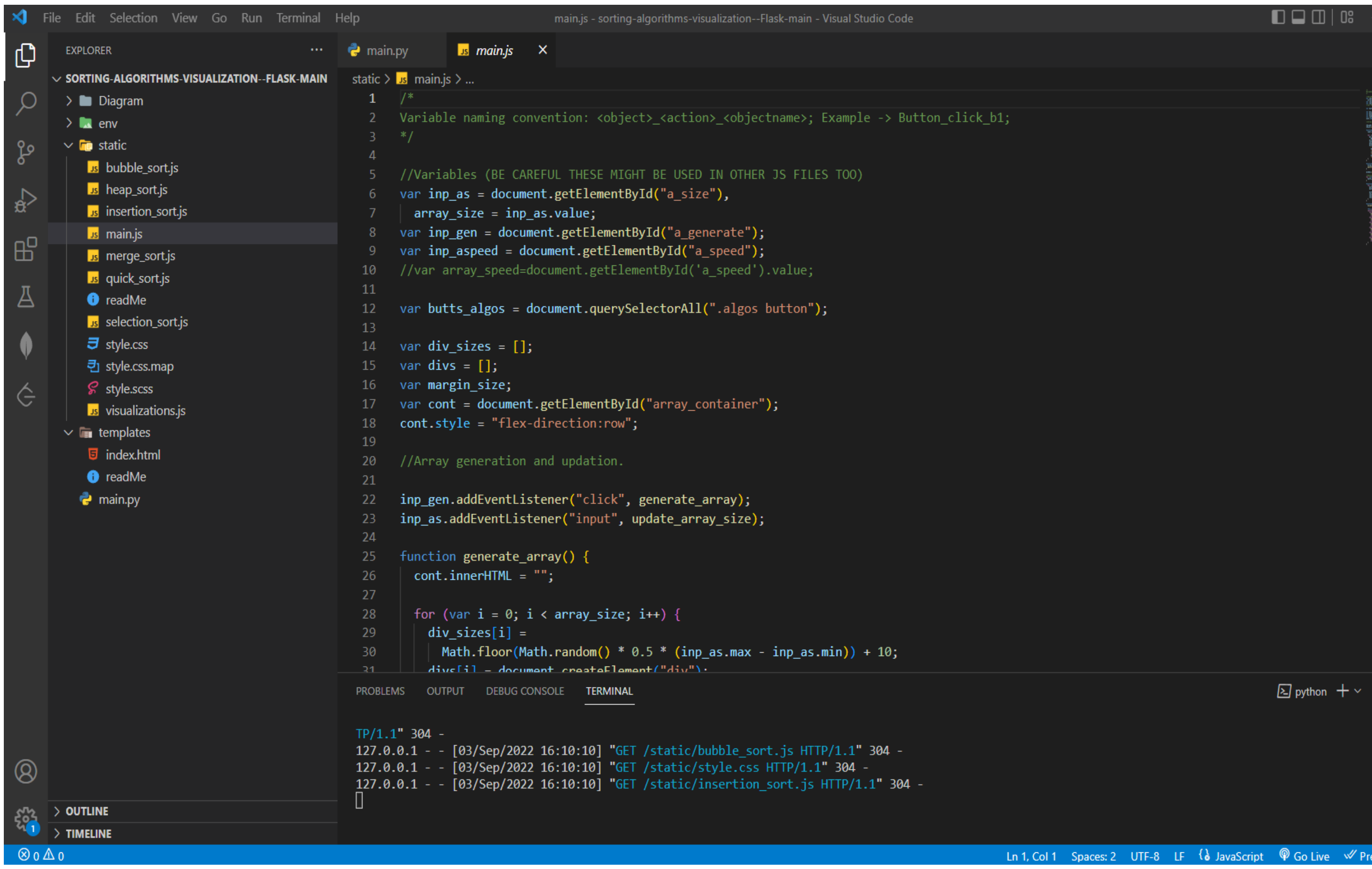
4.2.2 Visualization.js



```
//We only have to change background-color and height of the sorting element.

var speed = 1000;

inp_aspeed.addEventListener("input", vis_speed);

function vis_speed() {
  var array_speed = inp_aspeed.value;
  switch (parseInt(array_speed)) {
    case 1:
      speed = 1;
      break;
    case 2:
      speed = 10;
      break;
    case 3:
      speed = 100;
      break;
    case 4:
      speed = 1000;
      break;
    case 5:
      speed = 10000;
      break;
  }

  delay_time = 10000 / (Math.floor(array_size / 10) * speed); //Decrease numerator to increase speed.
}

var delay_time = 10000 / (Math.floor(array_size / 10) * speed); //Decrease numerator to increase speed.
var c_delay = 0; //This is updated ov every div change so that visualization is visible
```

```
TP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/bubble_sort.js HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/insertion_sort.js HTTP/1.1" 304 -
```

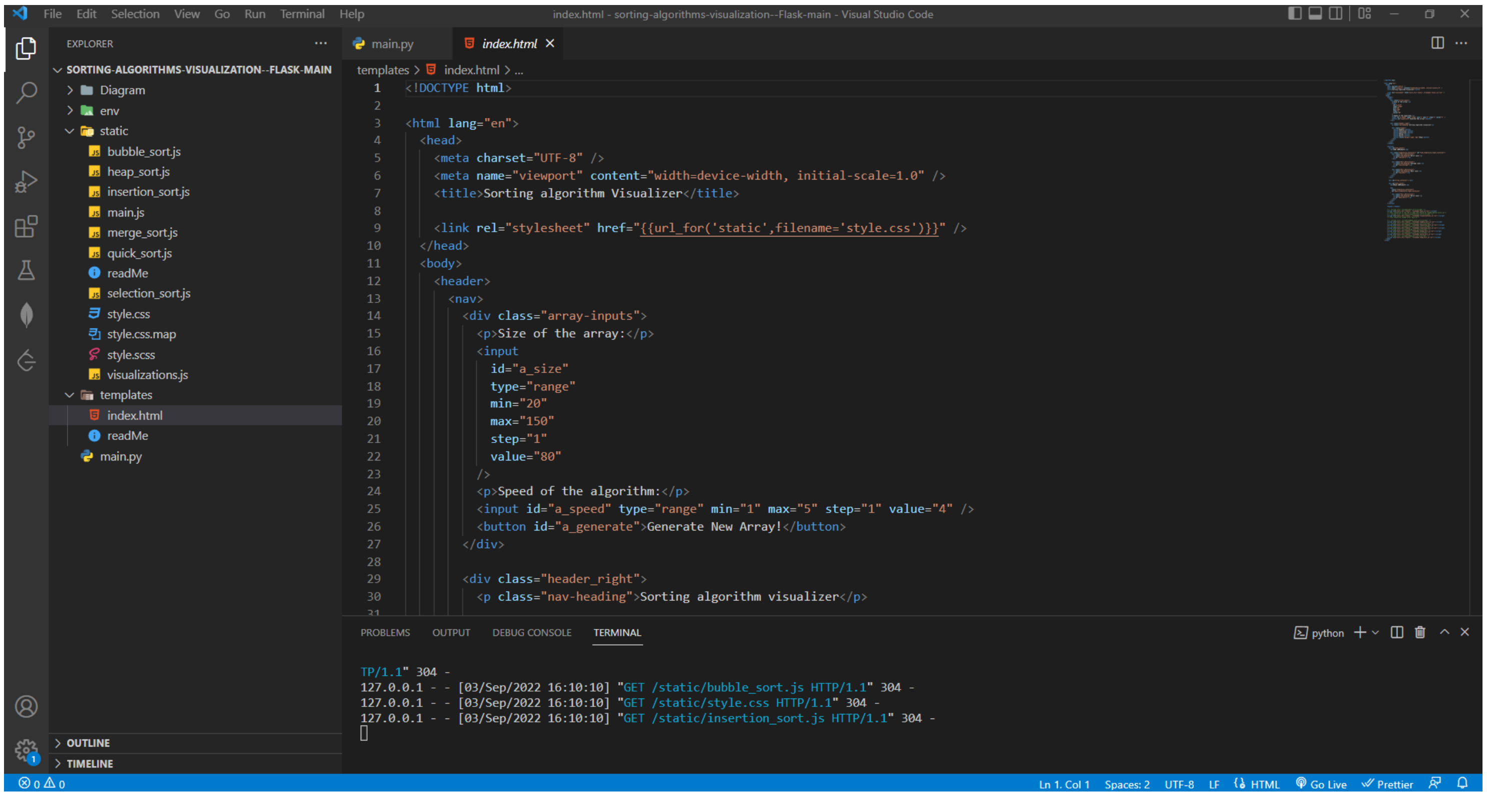
4.2.3 Algorithm Main.js



```javascript
/*
Variable naming convention: <object>_<action>_<objectname>; Example -> Button_click_b1;
*/

//Variables (BE CAREFUL THESE MIGHT BE USED IN OTHER JS FILES TOO)
var inp_as = document.getElementById("a_size"),
  array_size = inp_as.value;
var inp_gen = document.getElementById("a_generate");
var inp_aspeed = document.getElementById("a_speed");
//var array_speed=document.getElementById('a_speed').value;

var butts_algos = document.querySelectorAll(".algos button");

var div_sizes = [];
var divs = [];
var margin_size;
var cont = document.getElementById("array_container");
cont.style = "flex-direction:row";

//Array generation and updation.

inp_gen.addEventListener("click", generate_array);
inp_as.addEventListener("input", update_array_size);

function generate_array() {
  cont.innerHTML = "";

  for (var i = 0; i < array_size; i++) {
    div_sizes[i] =
      Math.floor(Math.random() * 0.5 * (inp_as.max - inp_as.min)) + 10;
    divs[i] = document.createElement("div");
```

```
TP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/bubble_sort.js HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/insertion_sort.js HTTP/1.1" 304 -
```

4.2.4 index.html



```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Sorting algorithm Visualizer</title>

    <link rel="stylesheet" href="{{url_for('static',filename='style.css')}}" />
  </head>
  <body>
    <header>
      <nav>
        <div class="array-inputs">
          <p>Size of the array:</p>
          <input
            id="a_size"
            type="range"
            min="20"
            max="150"
            step="1"
            value="80"
          />
          <p>Speed of the algorithm:</p>
          <input id="a_speed" type="range" min="1" max="5" step="1" value="4" />
          <button id="a_generate">Generate New Array!</button>
        </div>

        <div class="header_right">
          <p class="nav-heading">Sorting algorithm visualizer</p>
```

```
TP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/bubble_sort.js HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [03/Sep/2022 16:10:10] "GET /static/insertion_sort.js HTTP/1.1" 304 -
```

5 Testing:

5.1 Test Strategy

The "in scope" components to be evaluated (hardware, software, middleware, and so on) are specified. The device components that will not be checked must also be explicitly described as "out of scope."

5.1.1(A) Functional Testing:

It will be performed to test the functionalities of each feature on the Sorting Algorithm Visualizer .It may include internal file links that are dependent on each other that will go from one function to another. We can take one example like if a user wants to increase size of the array or increase speed of visualizing.

5.1.2 (B) DOM API:

The Document Object Model is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree.

5.2 Unit Test Plan

Unit testing is essential for the verification of the code produced during the coding phase and hence the goal is to test the internal logic of the modules. Using the detailed design description as a guide, important paths are tested to uncover errors within the boundary of the modules. These tests were carried out during the programming stage itself. All units of setting data and increasing speed and selecting are successfully tested.

5.3 Acceptance Test Plan

This testing is done to verify the readiness of the system for the implementation. Acceptance testing begins when the application is complete. Its purpose is to provide the user with the confidence that the application is ready for use. It involves planning and execution of functional tests, performance tests and stress tests in order to demonstrate that the implemented application satisfies its requirements.

Tools to special importance during acceptance testing include:

Timing Analyzer – also called a time complexity, reports the time spent in various regions of the code are areas to concentrate on to improve application performance.

Coding standards – static analyzers and standard checkers are used to inspect code for deviations from standards and guidelines.

5.4 Test Cae / Test Script

We are going to share the possible scenario for sorting algorithm visualizer how the algorithm will perform and how much space it consume while sorting the algorithm

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	Ω(N)	Θ(N^2)	O(N^2)	O(1)
Selection Sort	Ω(N^2)	Θ(N^2)	O(N^2)	O(1)
Insertion Sort	Ω(N)	Θ(N^2)	O(N^2)	O(1)
Quick Sort	Ω(N log N)	Θ(N log N)	O(N^2)	O(N)
Merge Sort	Ω(N log N)	Θ(N log N)	O(N log N)	O(N)
Heap Sort	Ω(N log N)	Θ(N log N)	O(N log N)	O(1)

Sorting Algorithms	Special Input Condition	Time Complexity			Space Complexity
		Best Case	Average Case	Worst Case	Worst Case
Counting Sort	Each input element is an integer in the range 0- K	Ω(N + K)	Θ(N + K)	O(N + K)	O(K)
Radix Sort	Given n digit number in which each digit can take on up to K possible values	Ω(NK)	Θ(NK)	O(NK)	O(N + K)
Bucket Sort	Input is generated by the random process that distributes elements uniformly and independently over the interval [0, 1)	Ω(N + K)	Θ(N + K)	O(N^2)	O(N)

5.5 Defect report / Test Log

The test case log documents different test cases for a particular test type to be executed during testing. It also records the results of the tests, which provides the detailed evidence for the test log summary report and enables us to reconstruct the test, if necessary.

<u>6 . Limitations of Proposed System</u>

1) The cold-start problem

2) Maintenance

3) Need dependency

4) Need environment

5) Basic understand of algorithm to understand

6) Need desktop to visualize nicely

<u>7 . Proposed Enhancement</u>

The proposed system is Sorting algorithm visualizer. We can enhance this system by including more facilities like new branch for the problem solving and coding site and competitive coding etc.

Providing such features enable the users to this features will drive more user to this application .

8. Conclusion

In computer science sorting algorithms, their performance in many different languages. Sorting is the act of rearranging data so that it is in some pre-specified order, such as ascending or descending order. The exact ordering does not really matter, as long as it is a linear, or total ordering, which means that any two elements can be ordered.

The cache-oblivious Distribution Sort based suffix sorting incurs memory management overheads. • Factor of 3 to 4 slower than qsort based approach.

Our Divide and conquer algorithm is cache-efficient and requires no memory parameters to be set. • O(N+IgN) time and 8N extra space

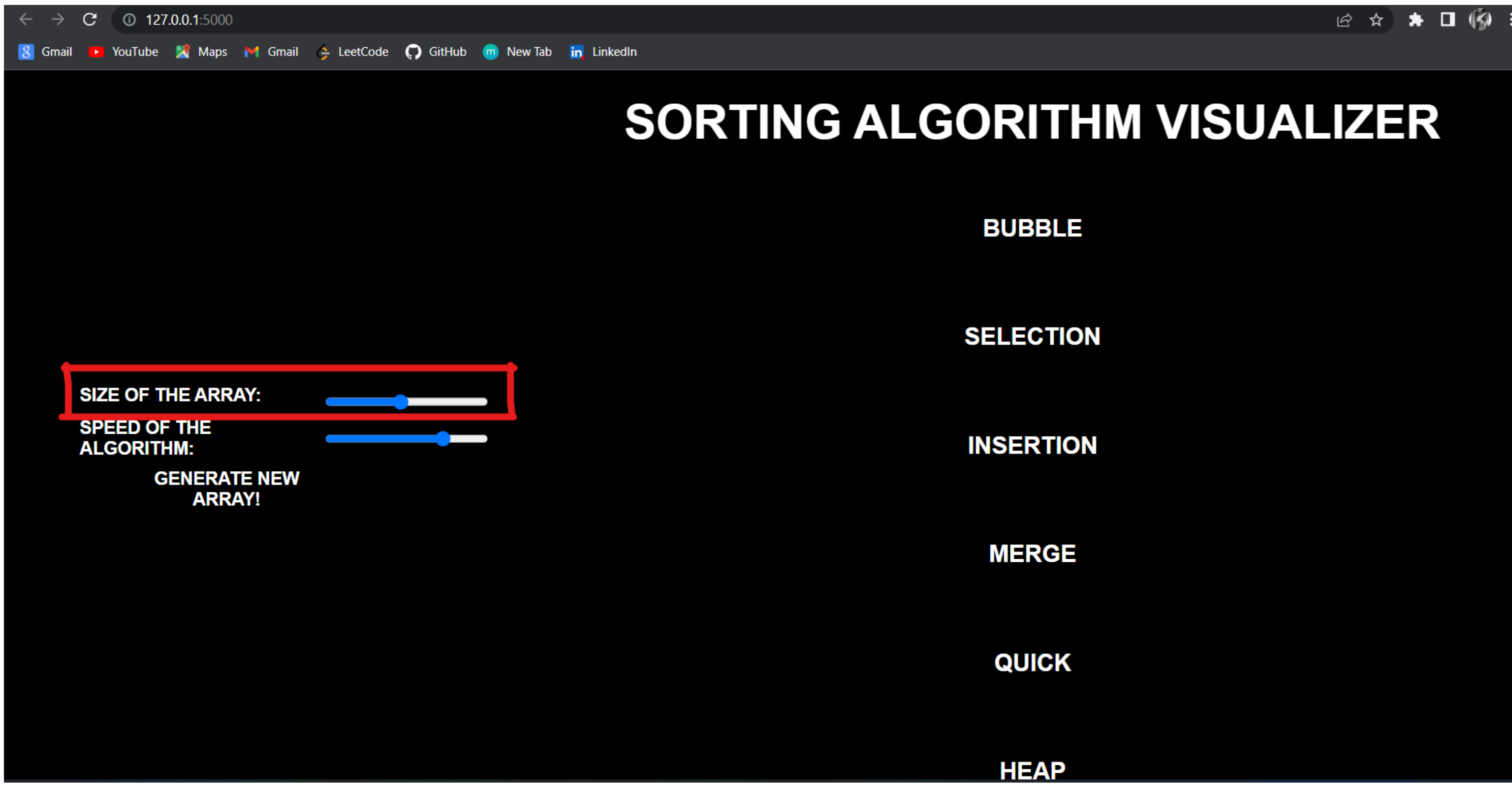
Linear time suffix sorting algorithm's performance can be improved further. Requires more space. . Factor of 2 slower than qsufsort.

9. Bibliography

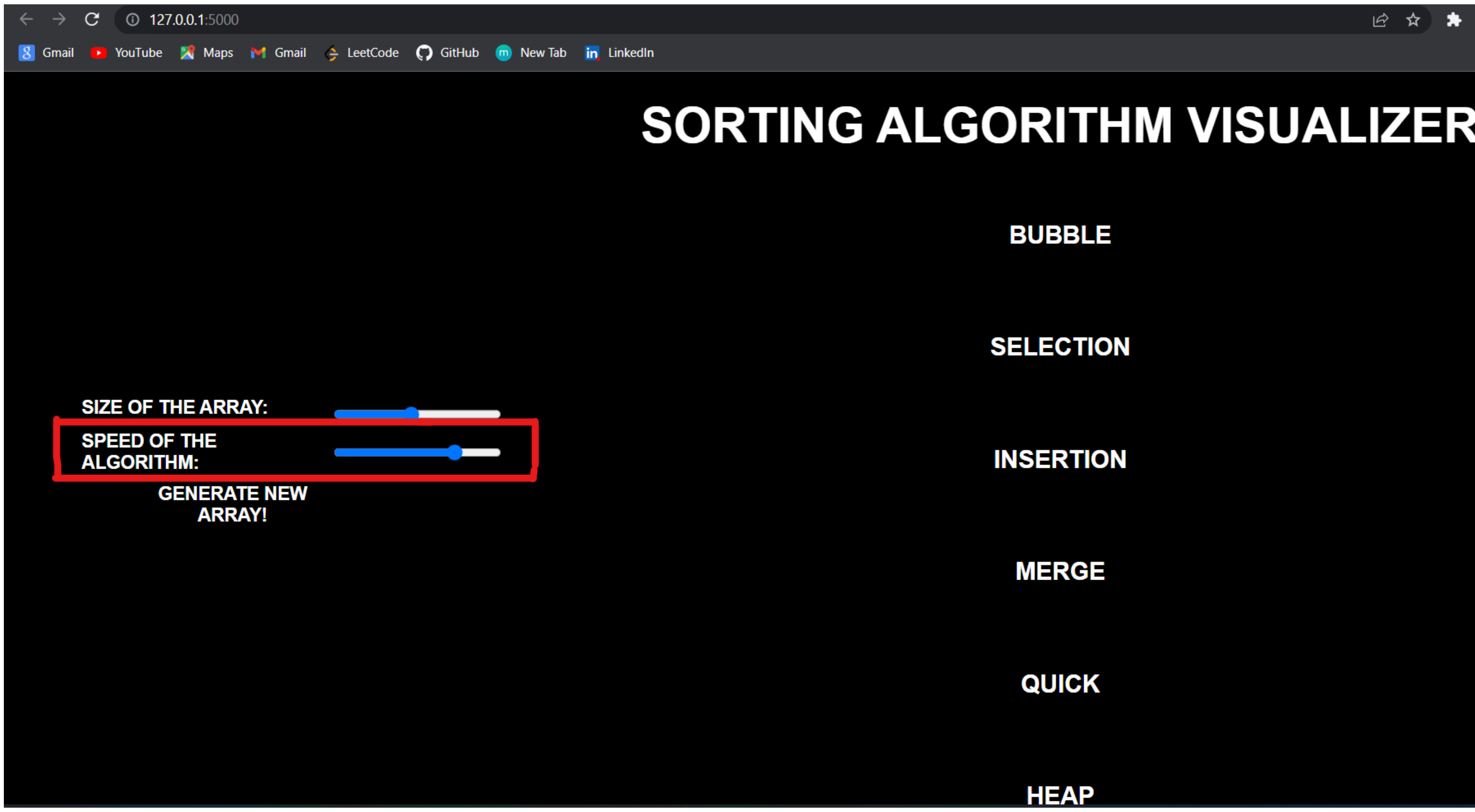
- Python.org
- https://stackoverflow.com
- https://www.geeksforgeeks.org/sorting-algorithms/
- https://flask.palletsprojects.com/en/2.2.x/
- https://stackoverflow.com/questions/2514841/comparison-of-sorting-algorithms
- https://github.com/diptangsu/Sorting-Algorithms
- https://docs.python.org/3/howto/sorting.html
- https://en.wikipedia.org/wiki/Sorting_algorithm

10 . User manual

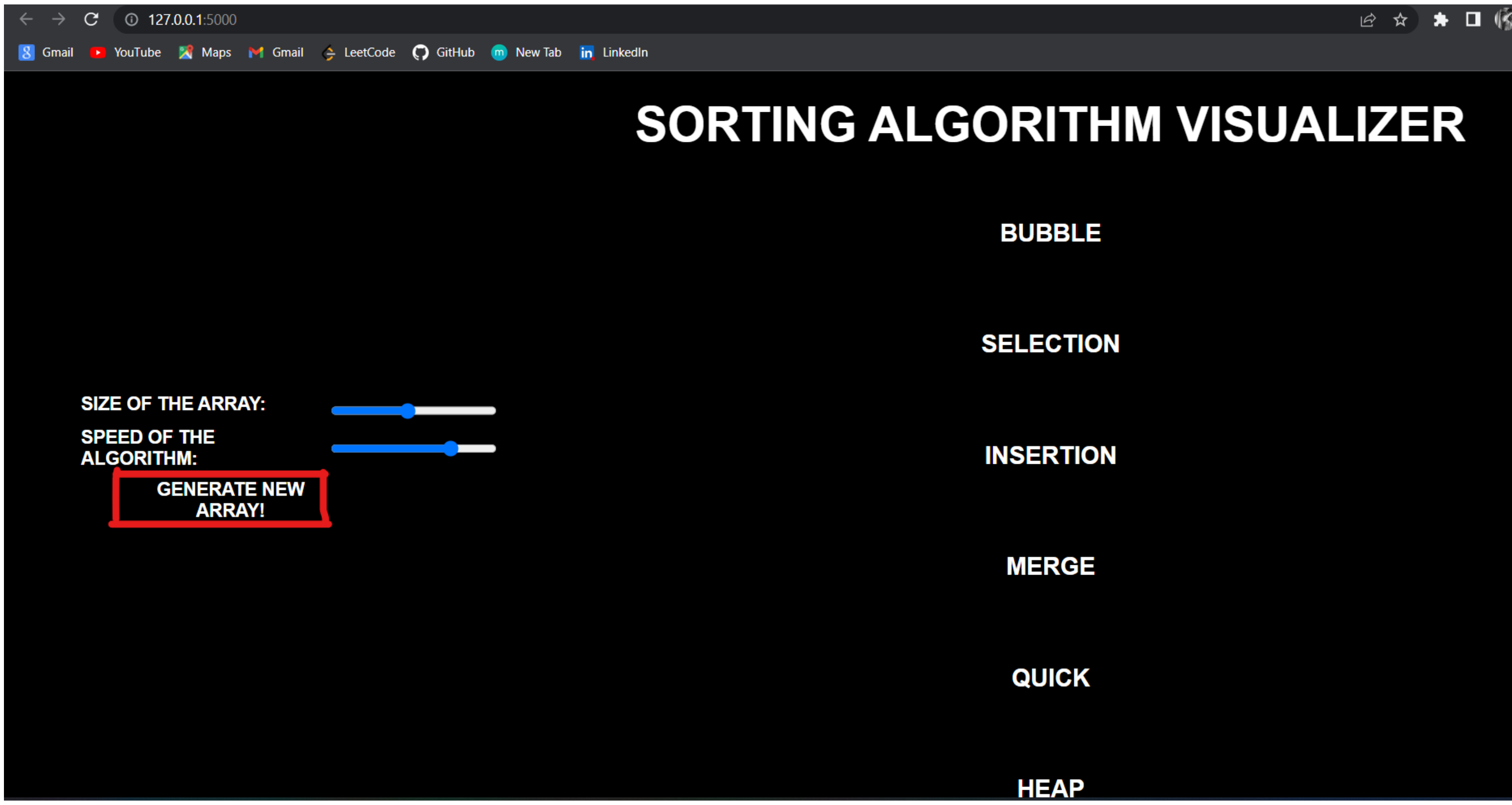
10.1 Increase Size of array :



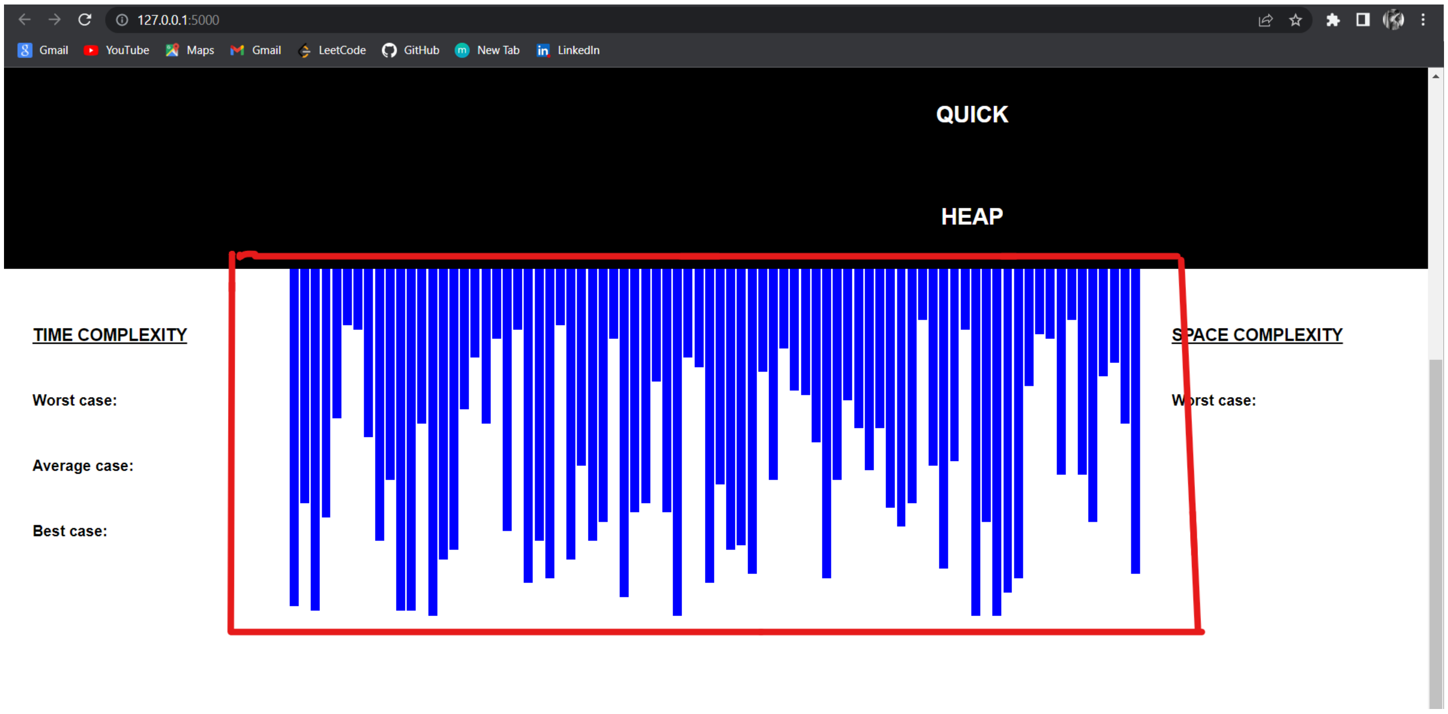
10.2 Increase Speed of sorting algorithm:



<u>10.3 Generating the array :</u>



10.4 After generating the array



TIME COMPLEXITY

Worst case:

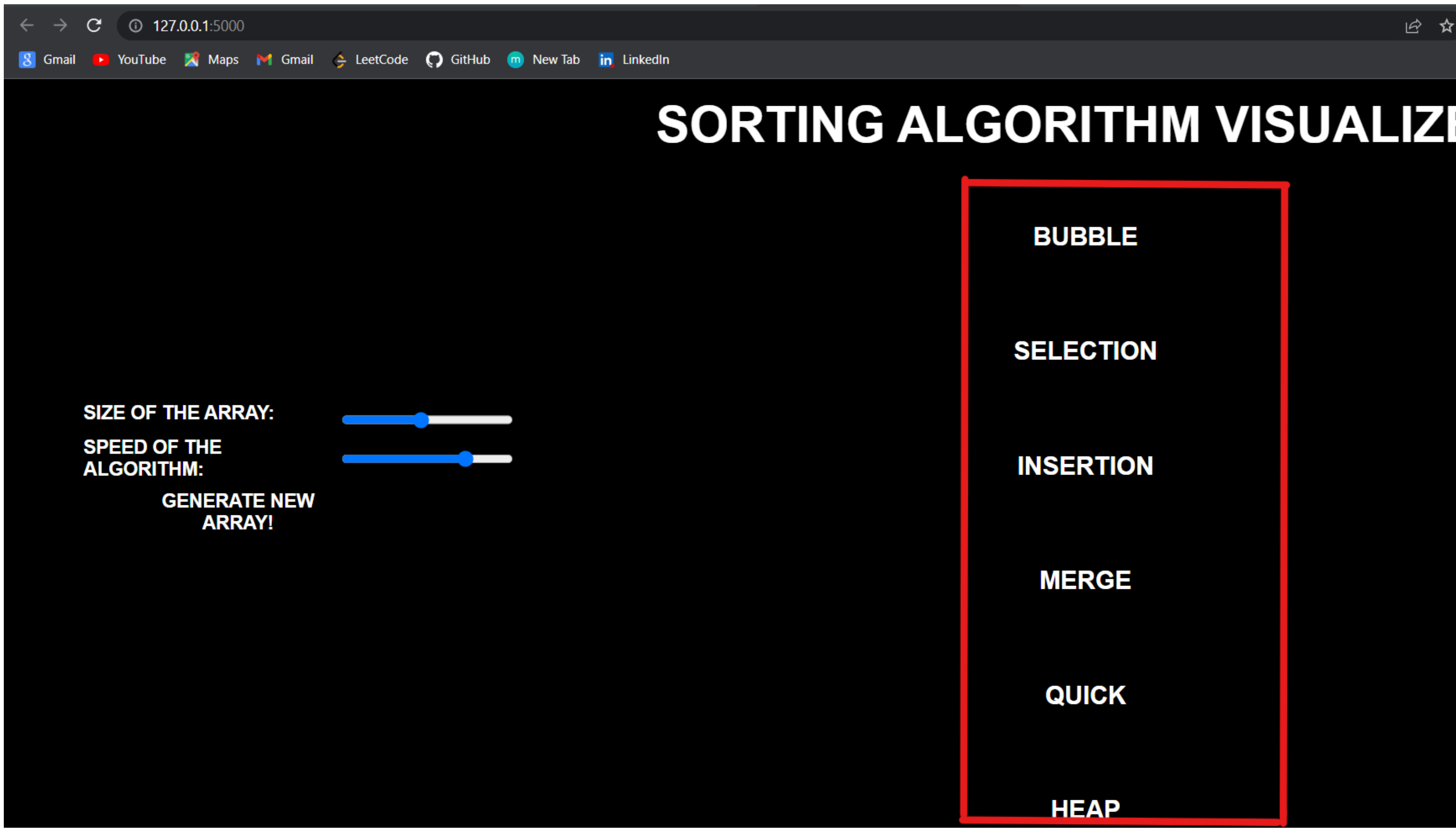
Average case:

Best case:

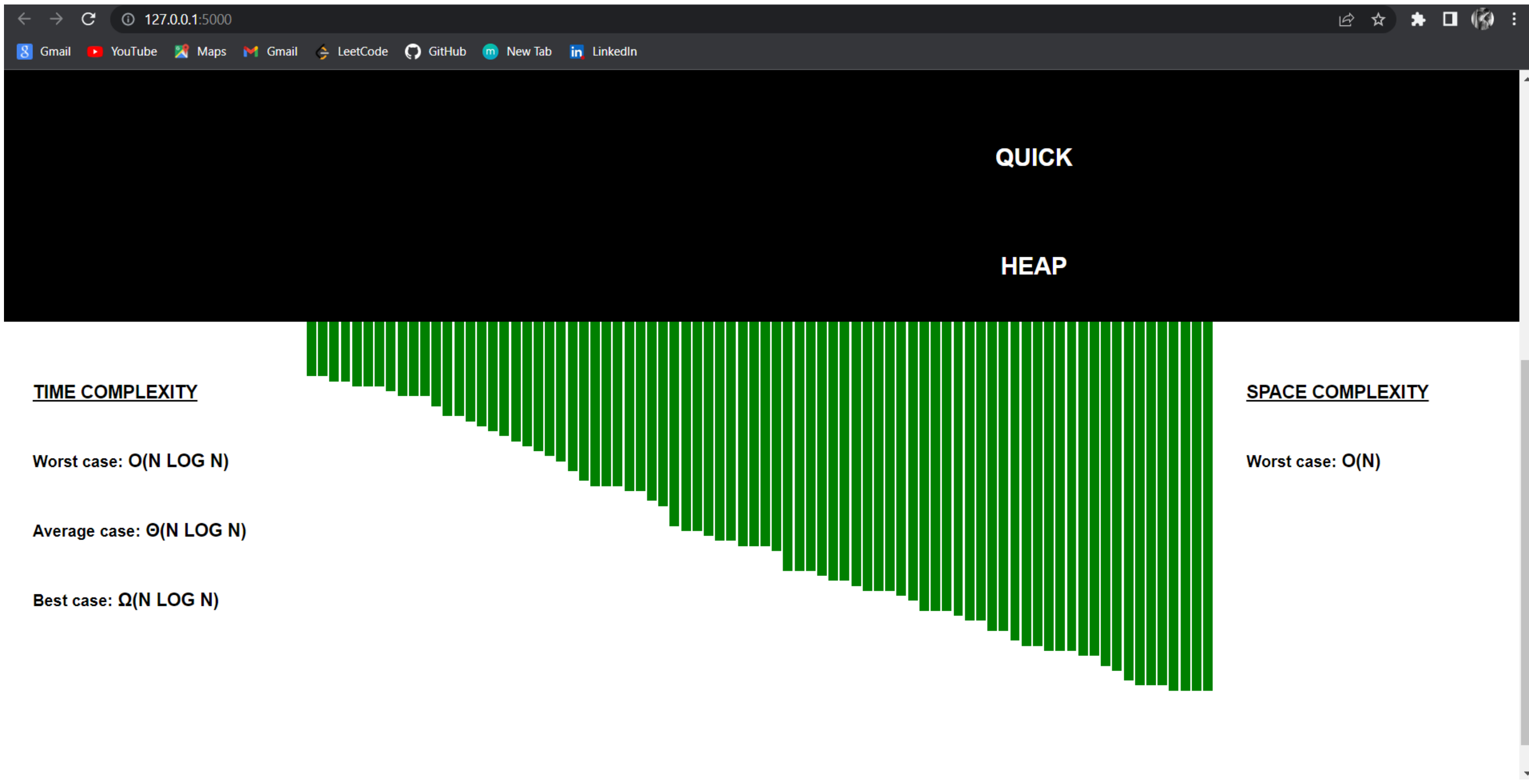
SPACE COMPLEXITY

Worst case:

10.5 Select Sorting algorithm



10.6 After Visualization



TIME COMPLEXITY

Worst case: O(N LOG N)

Average case: Θ(N LOG N)

Best case: Ω(N LOG N)

SPACE COMPLEXITY

Worst case: O(N)