A

Project Report

On

"ALGOLIZER – Basic Algorithm Visualization"

Submitted in partial fulfillment of the requirement of the University of Mumbai for the Degree of **Bachelor of Engineering**

Computer Engineering (Sem V)

By

Raj Jitendra Mhatre (41)

Shreya Krishna Mhatre (42)

Sahil Laxman Nakti (44)

Aqsa Sarnaik Rauf Khan (62)

Under the guidance of

Prof. Harishchandra Maurya

Department of Computer Engineering

Wilfred's Education Society
Chhatrapati Shivaji Maharaj Institute of Technology
Shedung, Panvel Dist:Raigad-410206

University of Mumbai

Academic Year 2022-23

Chhatrapati Shivaji Maharaj Institute of Technology

Department of Computer Engineering

Academic Year 2022-23

CERTIFICATE

This is to certify that

Mr. Raj Jitendra Mhatre , Ms. Shreya Krishna Mhatre , Mr. Sahil Laxman Nakti, Ms. Aqsa Sarnaik Rauf Khan Sem.V, TE Computer, Roll No: 41, 42, 44, 62 *.has*

satisfactorily completed the requirements of the Mini Project-2A entitled

" ALGOLIZER – Basic Algorithm Visualization "

As prescribed by the University of Mumbai Under the guidance of

Prof. Harishchandra Maurya

Guide	HOD	Principal
(Prof. Harishchandra Maurya)	**(Dr. Ankush Pawar)**	**(Dr. Dharmendra Dubey)**

Internal Examiner	External Examiner
Prof.	**Prof.**

ABSTRACT

ALGOLIZER comprises of two words mainly the “Algorithm” & “Visualization”. Algorithm visualization illustrates how algorithms work in a graphical way. It mainly aims to simplify and deepen the understanding of algorithms operation. Within the paper we discuss the possibility of enriching the standard methods of teaching algorithms, with the algorithm visualizations.

Algorithm invariant visualizing leads to codes that are computationally very demanding, and powerful software tools require downloading/installing compilers and/or runtime machines, which limits the scope of users. Nowadays, online learning environments have become very popular for teaching algorithms. To improve the learning process these environments often include various visualizations of the algorithms.

Using algorithm visualization, we can depict the execution of an algorithm as a sequence of graphical snapshots, the viewing of which is controlled by the user. Such visualizations can be a tremendous help to students, but their use is limited by the constraints of platform dependency. This constraint can now be overcome by making algorithm visualizations conveniently accessible on the World Wide Web.

ACKNOWLEDGEMENT

We would like to thank all those people who helped us in successful completion of the project **"ALGOLIZER".**

We would like to thank **Dr. Dharmendra Dubey**, Principal of **Chhatrapati Shivaji Maharaj Institute of Technology, Shedung, Panvel** for his immense support and motivation.

We would like to thank **Dr. Ankush Pawar**, IC Head of the Department, Computer Engineering of **Chhatrapati Shivaji Maharaj Institute of Technology, Shedung, Panvel** for his guidance. We are whole-heartedly thankful to him for giving us his valuable time & attention and for providing us a systematic way for completing our project in time. We have learnt so many things from him. Throughout the project work, his useful suggestions, constant encouragement has given usa right direction and shape to our learning.

We would like to thank **Mr. Harischandra Maurya**, our project guide for her guidance. We have learnt so many things from her and she motivated us and strengthened our confidence in doing the thesis. We express our deepest gratitude for her valuable suggestions and constant motivation that greatly helped the project work to successfully complete. Throughout the project work, her useful suggestions, constantencouragement has given us a right direction and shape to our learning.

We would also thank all the faculty members who have been a constant source and cooperation during the entire course of our study in this college.

The contribution and support received from all team members including **Raj Jitendra Mhatre, Shreya Krishna Mhatre, Sahil Laxman Nakti, Aqsa Sarnaik Rauf Khan** is vital. The team spirit is shown by all has made a project report work successful.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

ALGOLIZER – Basic Algorithm Visualization

CHAPTER 1

INTRODUCTION

1.1 Background:

The word Algorithm means "A set of rules to be followed in calculations or other problem-solving operations" Or "A procedure for solving a mathematical problem in a finite number of steps that frequently involves recursive operations". Visualization is a process of representing data graphically and interacting with representation

- Therefore Algorithm refers to a sequence of finite steps to solve a particular problem.
- Algorithms can be simple and complex depending on what you want to achieve.

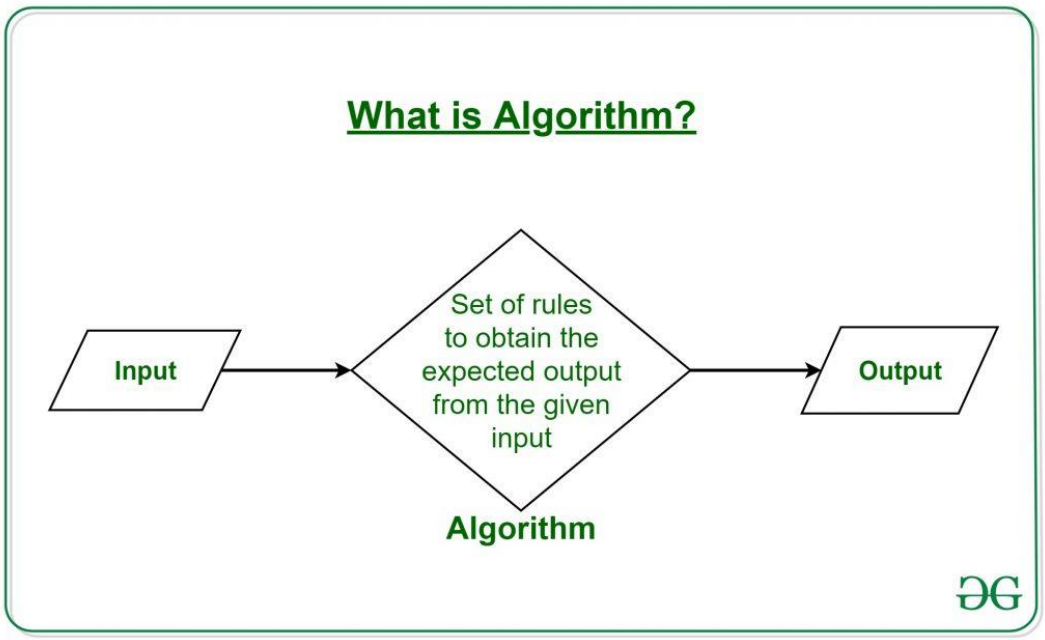

Figure 1 : Algorithm Defination

1.2 Relevance of an Algorithm:

- ✓ **Clear and Unambiguous:** The algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- ✓ **Well-Defined Inputs:** If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.

- ✓ **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should take at least 1 output.
- ✓ **Finite-ness:** The algorithm must be finite, i.e. it should terminate after a finite time.
- ✓ **Feasible:** The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything.
- ✓ **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

1.3 Properties of Algorithm:

- ✓ It should terminate after a finite time.
- ✓ It should produce at least one output.
- ✓ It should take zero or more input.
- ✓ It should be deterministic means giving the same output for the same input case.
- ✓ Every step in the algorithm must be effective i.e. every step should do some work.

1.4 Types of Algorithms:

There are several types of algorithms available. Some important algorithms are:

- ✓ **Selection Sort :** The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.
- ✓ **Insertion Sort :** It 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.
- ✓ **Bubble Sort :** It 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.
- ✓ **Merge Sort** : The **Merge Sort** algorithm is a sorting algorithm that is based on the **Divide and Conquer** paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

1.5 Advantages of Algorithms:

- ✓ It is easy to understand.
- ✓ An algorithm is a step-wise representation of a solution to a given problem.
- ✓ In Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

1.6 Disadvantages of Algorithms:

- ✓ Writing an algorithm takes a long time so it is time-consuming.
- ✓ Understanding complex logic through algorithms can be very difficult.
- ✓ Branching and Looping statements are difficult to show in Algorithms**(imp)**.

1.7 Organisation of Project Report:

The material presented in the Project Report is organised into five chapters:

Chapter 1: This is the introductory chapter.

Chapter 2: Chapter 2 describes the Literature Survey and the Research we did before and after finalizing the project.

Chapter 3: Chapter 3 presents an overview of the proposed system which contains the Algorithms, Design details & Methodology of the system.

Chapter 4: Chapter 4 describes the software and hardware used, details of the input received by the system.

Chapter 5: Finally, Chapter 5 presents us with the implementations of the project and the Timeline chart for Term I and Term II

CHAPTER 2

LITERATURE SURVEY

2.1 Existing System Survey:

- Concentrating on the sorting algorithms the existing proposed system considered more of theory of complex algorithm.
- So due to that difficulty to understand what the proposed system is about as well as harder to memorize.

2.2 Comparative Analysis of Existing System:

Author	Objective	Method	Merits	Demerits
Christopher D. Hundhausen, Sarah A Douglas, John T Stasko	The main Aim is to make easier to understand to the new learner	Generates Graphical Representation	It is a step-wise representation of a solution to a given problem, which makes it easy to understand.	Algorithms is Time consuming
Slavomir Simonak	Its aim is to generate algorithm in graphical format with animation	It Displays GUI Form	Like this software can be used in PC	The software cannot run on mobile
Euripides Vrachnos, Athanassios Jimoyiannis	Its aim is to display different types of visualizing algorithm	It visualizes different algorithm	Development platform for project was selected JAVA, ensuring high probability & very good support by available tools, libraries	The selection of topic within the scope of subject is quite wide

Table 1 : Existing System

2.3 Limitations of Existing System/Research Gap:

- Concentrating on the sorting algorithms the existing proposed system considered more of theory of complex algorithm so due to that it was difficult to understand what the proposed system is about as well as harder to memorize.
- Our project comprises more of visualization rather than just mugging this up and visualization is more effective than our traditional documentation.
- The UI is easy to understand due to the graphical representation.
- It is very basic and consist of basic 4 algorithm.

2.4 Problem Statement:

- An algorithms are hard to understand theoretically.
- We can understand easily by visualizing each kind of algorithms.
- Create an application or Website that show the working of Algorithms in Graphical Form.

2.5 Objectives:

- Easy to understand different types of algorithm.
- Graphically represent all algorithm so easy to understand and memorize.

CHAPTER 3

PROPOSED SYSTEM

3.1 Background of Proposed Methodology:

The proposed system involves the simulation of the different type of sorting algorithms codes. The scope has its limitations. Only 6 types of sorting algorithms codes are created which are bubble sort, insertion sort, selection sort, heap sort, merge sort and quick sort. Only the software application development began with desktop applications used in this project, it can be used on standalone personal computer only. Once the synthesize and simulation of the codes for the software application have been run, the following animations will show how successfully data sets from distinct unsorted data can be sorted using distinct algorithms.

3.2 Algorithm for Algolizer:

Step 1 : Start

Step 2 : Generate Homepage

Step 3 : It will Generate New Array / Random Array in Graphical Format

Step 4 : Select the any one Sorting Algorithm at a time from the given list.

Step 5 : It displays visualization graph sorting.

Step 6 : Stop

3.3 Flowchart for Algolizer:

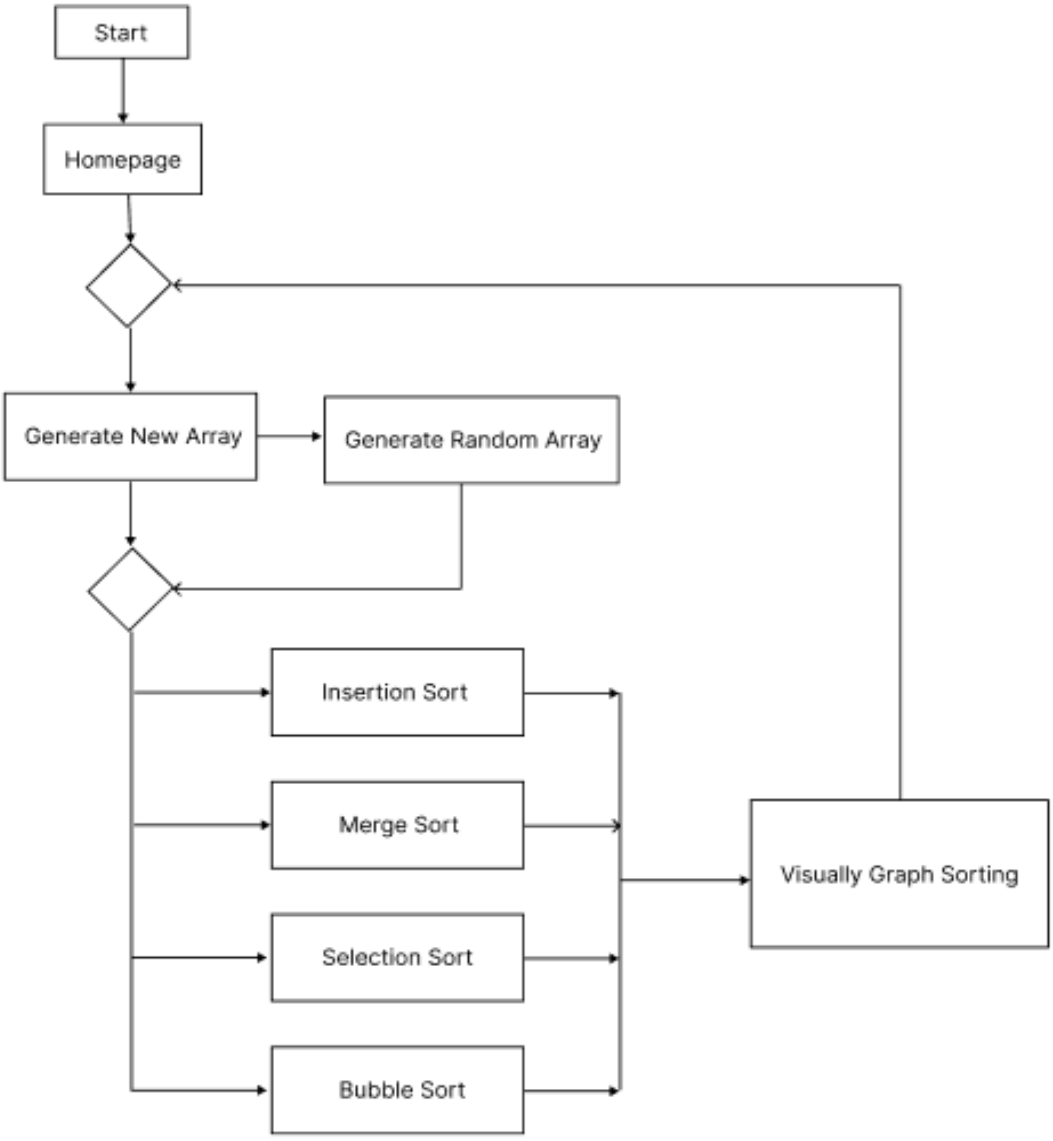

Figure 2 : Flowchart for Algolizer

3.4 Different Types of Soring:

Sorting algorithm is an effective algorithm in computer science. It carries out a vital undertaking that puts data of a list in a specific request or organizes a compilation of items into a specific request. Sorting data has been produced to orchestrate the array values in different routes for a database. For example, sorting will dictate an array of numbers in ascending or descending order. Usually, it sorts an array into ascending or descending order. Most basic sorting algorithms include two stages which are compare two items and swap two items or copy one item. It will keep on executing until the data has been fully arranged.

3.4.1 Insertion Sort : Insertion sort continuously keeps up a sorted sub array in the small some portion of the list. It loops the input element by growing the sorted array and contrasts the present data of element with the biggest value in the sorted array. Every iteration moves an element from unsorted array to sorted array until all of the elements are sorted in the list.

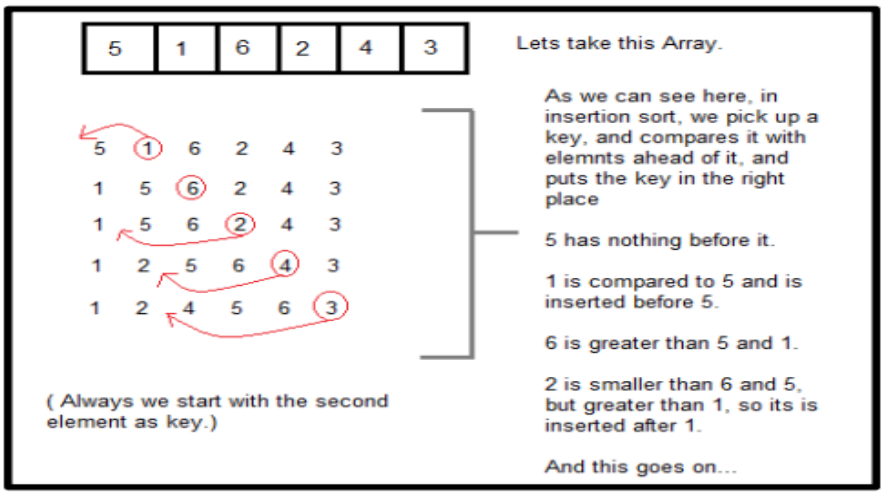

Figure 3 : Insertion Sort

3.4.2 Selection Sort : Selection sort making just a single trade for each go through the array that improves on the bubble sort. It is a set up correlation based algorithm in which the list is separated into two segments, the arranged segment at the left end and the unsorted part at the correct end. Because of the average and worst case time complexities are of O(n2), this algorithm is not suitable for large data sets. The student easily understands the concept of selection sort by looking at the visualization. The text of a learning material is more effective if it is provided with graphics, animation or video for the student to learn.

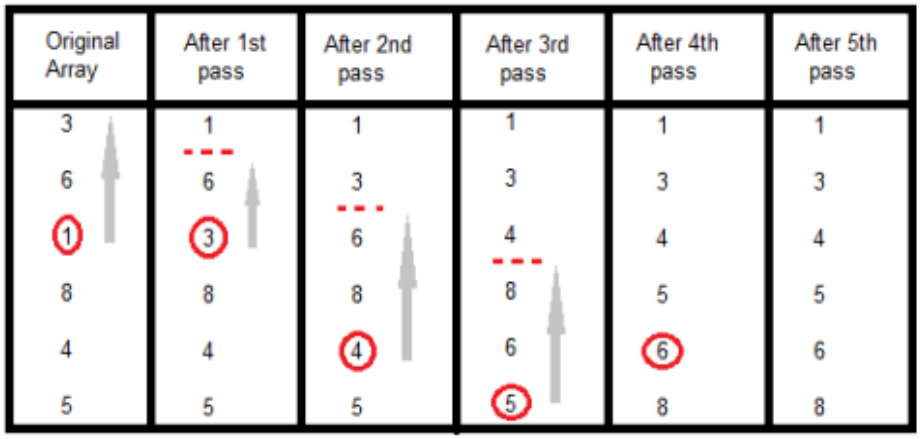

Figure 4 : Selection Sort

3.4.3 Comb Sort : Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all inversions are removed one by one. Comb Sort improves on Bubble Sort by using a gap of the size of more than 1. The gap starts with a large value and shrinks by a factor of 1.3 in every iteration until it reaches the value 1. Thus Comb Sort removes more than one inversion count with one swap and performs better than Bubble Sort. The shrink factor has been empirically found to be 1.3 (by testing Combsort on over 200,000 random lists). Although it works better than Bubble Sort on average, worst-case remains O(n2).

	0	1	2	3	4	5	6	7	
i = 0	2	11	24	22	29	27	49	44	2<11, no swap
i = 1	2	11	24	22	29	27	49	44	11<24, no swap
i = 2	2	11	24	22	29	27	49	44	24>22, swap
i = 3	2	11	22	24	29	27	49	44	24<29, no swap
i = 4	2	11	22	24	29	27	49	44	29>27, swap
i = 5	2	11	22	24	27	29	49	44	29<49, no swap
i = 6	2	11	22	24	27	29	49	44	49>44, swap

Figure 5 : Comb Sort

3.4.4 Shell Sort : Shell sort is mainly a variation of Insertion Sort. In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved. The idea of ShellSort is to allow the exchange of far items. In Shell sort, we make the array h - sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element are sorted.

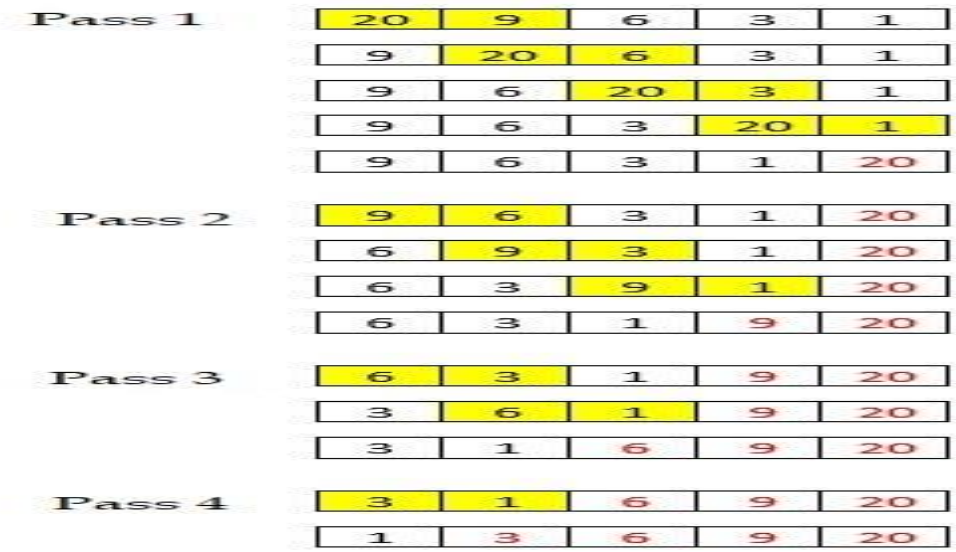

Figure 6 : Shell Sort

3.5 Criteria for Comparision:

Many algorithms that have the similar proficiency do not really have a similar speed and conduct on a similar information. The first and most important factor is that algorithms must be judged in light of their best-case, average-case, and worst-case efficiency. There are some algorithms that show different behavior of different combinations of inputs. Algorithms like quick sort perform particularly well for some sources of information, however awfully for others. Other algorithms, such as merge sort, are unaffected by the ordering of the input data.

T(n)	Name	Problems
O(1)	Constant	Easy-solved
O(log n)	Logarithmic	
O(n)	Linear	
O(n log)	Linear-log.	
O(n2)	Quadratic	
O(n3)	Cubic	
O(2n)	Exponential	Hard-solved
O(n!)	Factorial	

Table 2 : Complexity of Algorithm

From the above analysis it can be said that fairly straight forward sorting techniques are Bubble Sort, Insertion Sort and Selection Sort, but they are not better and efficient for small elements lists. Quick Sort and Merge Sort are more complicated as compare with Bubble Sort, Insertion Sort and Selection Sort but it is much faster for large number of elements lists. On average the Quick Sort is the faster Sorting Algorithm. Bubble Sort is slower Sorting Algorithm.

3.6 Research Methodology:

The main goal of this project is to design a system for sorting algorithm visualization as well as investigating and visualizing the best and worst case for each implemented sorting algorithm.

Design Steps : The software which has been used in this system is Eclipse. Java language is used to develop the algorithm visualization. For the Java code here, Java methods is used because of method can be used many times in the program by call the method. It is reusable and the program will become more readable. After the sorting algorithms codes have been written and designed, they have been synthesized and simulated in the Eclipse software. Once the synthesize and simulation of the Java codes have been run, the user can choose the types of the sorting algorithm and searching algorithm to find the time complexity for the each of the sorting algorithms. The flow chart of designing this sorting algorithm visualization via Eclipse could be seen below, as in Figure.

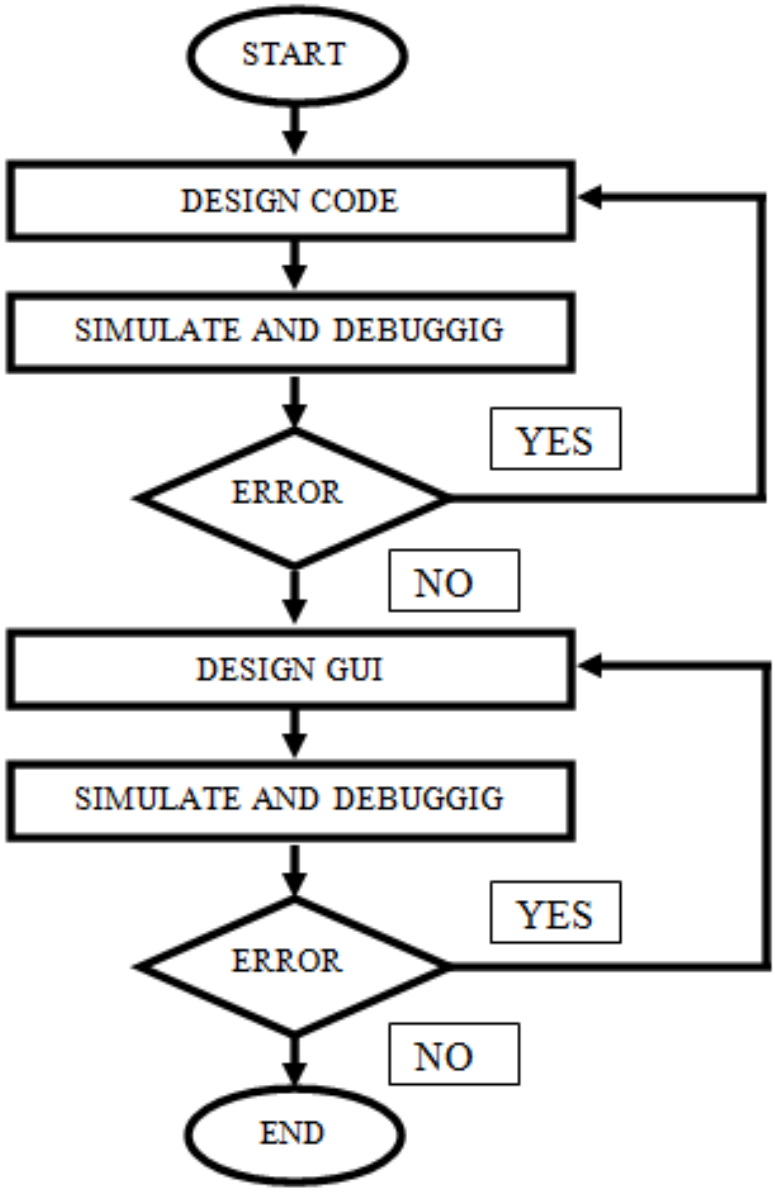

Figure 7 : Flowchart of System for Algorithm Visualization

3.7 Method of Sorting:

While there are a large number of sorting algorithms, in practical implementations some algorithms predominate. Insertion sort is widely used for small datasets, whereas for large datasets, an asymptotically efficient sort is used, primarily heap sort, merge sort, or quick sort. Each algorithm uses a different mechanism to sort the data shown in the following pseudocode:

```
Bubble Sort(A)
for i ← 1 to length[A]
        do for j ← length[A] down to i + 1
                do if A[j] < A[j - 1]
                        then exchange A[j] & A[j - 1]
Insertion Sort(A)
for j ← 2 to length[A]
        do key ← A[j]
        Insert A[j] into the sorted sequence A[1 : j - 1].
        i ← j - 1
        while i > 0 and A[i] > key
                do A[i + 1] ← A[i] i ← i - 1
                A[i + 1] ← key

Selection Sort (A)
for i ← 1 to length[A] – 1
        do min ← i
                for j ← i + 1 to length[A]
                do if A[j] < A[min]
                        then min ← j
exchange A[i] & A[min]

Mergesort(A, P, R)
if p < r
        then q ← (p + r)/2
                MERGE-SORT(A, p, q)
                MERGE-SORT(A, q + 1, r)
                MERGE(A, p, q, r)
```

```
Heapsort(A)
BUILD-MAX-HEAP(A)
for i ← length[A] downto 2
        do exchange A[1] ↔ A[i]
                heap-size[A] ← heap-size[A] - 1
                MAX-HEAPIFY(A, 1)
```

```
Quicksort(A, P, R)
if p < r
        then q ← PARTITION(A, p, r)
                QUICKSORT(A, p, q - 1)
                QUICKSORT(A, q + 1, r)
```

The strategy of bubble sort is quite simple, however can be quite inefficient. The algorithm iterates through the array, contrast each pair of side by side elements, and if the elements are in the wrong order swaps them.

Insertion sort is another straightforward approach to sorting an array. Insertion sort starts at the beginning of the array and checks each element with the next and swaps them if necessary.

Selection sort is very intuitive and incredibly inefficient. The algorithm starts by creating an empty list to store values as they are sorted from the unsorted array. It then iterates through the entire unsorted array and adds the smallest (or largest) value to the sorted array.

Merge sort is one of the more effective sorting algorithms out there with a time complexity of only O(n log n). The algorithm starts by creating n lists with single values in each list and then proceeds to combine the lists into an array, sorting the values as the lists are combined.

The algorithms that based on comparison in sorting algorithm. Albeit to some degree slower by and by on most machines than a decent execution of quick sort. The time complexity for all case is O(n log n). Heap sort joint the effective of time for merge sort and effective of storage for quick sort.

Quick sort uses divide and conquer to pick up an indistinguishable focal points from the merge sort, while not utilizing extra memory. As an exchange off, nonetheless, it is conceivable that the list may not be partitioned into equal parts. At the points when this happens, the execution is reduced.

ALGOLIZER – Basic Algorithm Visualization

Solutions to sorting problems have pulled in a lot of research in the current year and in this procedure many sorting algorithm have started enhanced effectiveness. Throughout the year analysts have been contrasting and analyzing the sorting algorithm with decide their applicability to applications.

CHAPTER 4

EXPERIMENTAL SETUP

4.1 SYSTEM SPECIFICATION:

4.1.1 Hardware Requirements:

(Minimum of the hardware required)

- Hardware - Pentium
- Speed - 1.1 GHz
- RAM - 1GB
- Hard Disk - 20 GB
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - SVGA

4.1.2 Software Requirements:

(Any higher version will do)

- Operating System - Windows / Linux / Mac
- Technology - Web 2.0
- Web Technologies - Html, JavaScript, CSS
- IDE - VS Code
- Client - Google Chrome, Internet Explorer

4.2 TECHNOLOGY USED:

4.2.1 HTML:

What is HTML?

HTML is a computer language devised to allow website creation. These websites can then be viewed by anyone else connected to the Internet. It is relatively easy to learn, with the basics being accessible to most people in one sitting; and quite powerful in what it allows you to create. The definition of HTML is HyperText Markup Language

HyperText is the method by which you move around on the web by clicking on special text called hyperlinks which bring you to the next page. The fact that it is hyper just means it is not linear i.e. you can go to any place on the Internet whenever you want by clicking on links

Markup is what HTML tags do to the text inside them. They mark it as a certain type of text (italicized text, for example).

HTML is a Language, as it has code-words and syntax like any other language.

How does it work?

HTML consists of a series of short codes typed into a text-file by the site author — these are the tags. The text is then saved as a html file, and viewed through a browser, like Internet Explorer or Netscape Navigator. This browser reads the file and translates the text into a visible form, hopefully rendering the page as the author had intended. Writing your own HTML entails using tags correctly to create your vision. You can use anything from a rudimentary text-editor to a powerful graphical editor to create HTML pages.

Is there anything HTML can’t do?

Of course, but since making websites became more popular and needs increased many other supporting languages have been created to allow new stuff to happen, plus HTML is modified every few years to make way for improvements. Cascading Stylesheets are used to control how your pages are presented.

4.2.2 CSS:

What is CSS?

A cascading style sheet (CSS) is a Web page derived from multiple sources with a defined order of precedence where the definitions of any style element conflict. CSS gives more control over the appearance of a Web page to the page creator than to the browser designer or the viewer.

Where is CSS Used?

CSS is used to style Web pages. But there is more to it than that. CSS is used to style HTML, XHTML and XML markup. This means that anywhere you have XML markup (including XHTML) you can use CSS to define how it will look.

Why is CSS Important?

CSS is one of the most powerful tools a Web designer can learn because with it you can affect the entire mood and tone of a Web site. Well written style sheets can be updated quickly and allow sites to change what is prioritized or valued without any changes to the underlying HTML.

Compared with the number of HTML tags and attributes to learn, that can feel like a cake walk. But because CSS can cascade, and combine and browsers interpret the directives differently, CSS is more difficult than plain HTML. But once you start using it, you'll see that harnessing the power of CSS will give you more options and allow you to do more and more things with your Web sites.

If you want to be a professional Web designer, you need to learn Cascading Style Sheets. But luckily, they are fun to learn.

4.2.3 JavaScript:

What is JavaScript?

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area.

Where is JavaScript Used?

Javascript is used by programmers across the world to create dynamic and interactive web content like applications and browsers. JavaScript is so popular that it's the most used programming language in the world, used as a client-side programming language by 97.0% of all websites. Client-side languages are those whose action takes place on the user's computer, rather than on the server.

JavaScript is versatile enough to be used for a variety of different applications, like software, hardware controls, and servers. JavaScript is most known for being a web-based language, because it's native to the web browser. The web browser can naturally understand the language, like how a native English speaker can naturally understand English.

4.3 Performance Evaluation Parameters (for Validation):

4.3.1 Memory usage:

Memory usage is also an important application performance metric. High memory usage indicates high resource consumption in the server.

4.3.2 Requests per minute:

Tracking the number of requests your application's API receives per minute can help determine how the server performs under different loads. It's equally important to track the amount of data the application handles during every request. You might find that the application receives more requests than it can manage or that the amount of data it is forced to handle is hurting performance.

4.3.3 Latency and uptime:

Latency usually measured in millisecond refers to the delay between a user's action on an application and the response of the application to that action. Higher latency has a direct effect on the load time of an application.

4.3.4 Average response time:

The average response time is calculated by averaging the response times for all requests over a specified period of time. A low average response time implies better performance, as the application or server has taken less time to respond to requests or inputs.

4.3.5 Request rates:

Request rate is an essential metric that provides insights into the increase and decrease in traffic experienced by your application. In other words, it provides insights into the inactivity and spikes in traffic that your application receives. You can correlate request rates with other application performance metrics to determine how your application can scale. You should also keep an eye on the number of concurrent users in your application.

CHAPTER 5

IMPLEMENTATION WORK

5.1 Source Code:

5.1.1 Index.js:

```
const container =
document.querySelector(".data-container");
// Function to generate bars
function generatebars(num = 20) {

  //For loop to generate 20 bars
 for (let i = 0; i < num; i += 1) {

   // To generate random values from 1 to 100
   const value = Math.floor(Math.random() * 100) + 1;

    // To create element "div"
   const bar = document.createElement("div");

   // To add class "bar" to "div"
   bar.classList.add("bar");

    // Provide height to the bar
   bar.style.height = `${value * 3}px`;
   // Translate the bar towards positive X axis
   bar.style.transform = `translateX(${i * 30}px)`;

   // To create element "label"
   const barLabel = document.createElement("label");

   // To add class "bar_id" to "label"
   barLabel.classList.add("bar__id");
```

```
    // Assign value to "label"
    barLabel.innerHTML = value;

    // Append "Label" to "div"
    bar.appendChild(barLabel);

    // Append "div" to "data-container div"
    container.appendChild(bar);
  }
}
// Asynchronous function to perform "Insertion Sort"
  async function InsertionSort(delay = 600) {
  let bars = document.querySelectorAll(".bar");

  // Provide lightgreen colour to 0th bar
  bars[0].style.backgroundColor = " rgb(49, 226, 13)";
  for (var i = 1; i < bars.length; i += 1) {
    // Assign i-1 to j
    var j = i - 1;

    // To store the integer value of ith bar to key
    var key =
    parseInt(bars[i].childNodes[0].innerHTML);

    // To store the ith bar height to height
    var height = bars[i].style.height;

    // For selecting section having id "ele"
    var barval=document.getElementById("ele")

    // For dynamically Updating the selected element
      barval.innerHTML=
      `<h3>Element Selected is :${key}</h3>`;
```

```
  //Provide darkblue color to the ith bar
  bars[i].style.backgroundColor = "darkblue";

  // To pause the execution of code for 600 milliseconds
  await new Promise((resolve) =>
  setTimeout(() => {
   resolve();
  }, 600)
);
  // For placing selected element at its correct position
  while (j >= 0 && parseInt(bars[j].childNodes[0].innerHTML) > key) {

   // Provide darkblue color to the jth bar
   bars[j].style.backgroundColor = "darkblue";

   // For placing jth element over (j+1)th element
   bars[j + 1].style.height = bars[j].style.height;
   bars[j + 1].childNodes[0].innerText =
   bars[j].childNodes[0].innerText;
   // Assign j-1 to j
   j = j - 1;
   // To pause the execution of code for 600 milliseconds
   await new Promise((resolve) =>
    setTimeout(() => {
     resolve();
    }, 600)
   );
   // Provide lightgreen color to the sorted part
   for(var k=i;k>=0;k--){
    bars[k].style.backgroundColor = " rgb(49, 226, 13)";
   }
  }
  // Placing the selected element to its correct position
  bars[j + 1].style.height = height;
```

```
    bars[j + 1].childNodes[0].innerHTML = key;
    // To pause the execution of code for 600 milliseconds
    await new Promise((resolve) =>
      setTimeout(() => {
        resolve();
      }, 600)
    );
    // Provide light green color to the ith bar
    bars[i].style.backgroundColor = " rgb(49, 226, 13)";
  }
  barval.innerHTML="<h3>Sorted!!!</h3>";
  // To enable the button
  // "Generate New Array" after final(sorted)
  document.getElementById("Button1")
  .disabled = false;
  document.getElementById("Button1")
  .style.backgroundColor = "#6f459e";

  // To enable the button
  // "Insertion Sort" after final(sorted)
  document.getElementById("Button2")
  .disabled = false;
  document.getElementById("Button2")
  .style.backgroundColor = "#6f459e";
}
// Asynchronous function to perform "Selection Sort"
async function SelectionSort(delay = 300) {
  console.log("clicked");
    let bars = document.querySelectorAll(".bar");
    // Assign 0 to min_idx
     var min_idx = 0;
     for (var i = 0; i < bars.length; i += 1) {
      var key =
    parseInt(bars[i].childNodes[0].innerHTML);
```

```
var barval=document.getElementById("ele")
barval.innerHTML=
    `<h3>Element Selected is :${key}</h3>`;
    // Assign i to min_idx
    min_idx = i;
    // Provide darkblue color to the ith bar
    bars[i].style.backgroundColor = "darkblue";
    for (var j = i + 1; j < bars.length; j += 1) {

      // Provide red color to the jth bar
      bars[j].style.backgroundColor = "red";

      // To pause the execution of code for 300 milliseconds
      await new Promise((resolve) =>
        setTimeout(() => {
          resolve();
        }, 300)
      );
      // To store the integer value of jth bar to var1
      var val1 = parseInt(bars[j].childNodes[0].innerHTML);

      // To store the integer value of (min_idx)th bar to var2
      var val2 = parseInt(bars[min_idx].childNodes[0].innerHTML);

      // Compare val1 & val2
      if (val1 < val2) {
        if (min_idx !== i) {
          // Provide skyblue color to the (min-idx)th bar
          bars[min_idx].style.backgroundColor = "  rgb(24, 190, 255)";
        }
        min_idx = j;
      } else {
        // Provide skyblue color to the jth bar
        bars[j].style.backgroundColor = "  rgb(24, 190, 255)";
```

```
      }
    }
    // To swap ith and (min_idx)th bar
    var temp1 = bars[min_idx].style.height;
    var temp2 = bars[min_idx].childNodes[0].innerText;
    bars[min_idx].style.height = bars[i].style.height;
    bars[i].style.height = temp1;
    bars[min_idx].childNodes[0].innerText = bars[i].childNodes[0].innerText;
    bars[i].childNodes[0].innerText = temp2;

    // To pause the execution of code for 300 milliseconds
    await new Promise((resolve) =>
      setTimeout(() => {
        resolve();
      }, 300)
    );
    // Provide skyblue color to the (min-idx)th bar
    bars[min_idx].style.backgroundColor = "  rgb(24, 190, 255)";

    // Provide lightgreen color to the ith bar
    bars[i].style.backgroundColor = " rgb(49, 226, 13)";
  }
  // To enable the button "Generate New Array" after final(sorted)
  document.getElementById("Button1").disabled = false;
  document.getElementById("Button1").style.backgroundColor = "#6f459e";

  // To enable the button "Selection Sort" after final(sorted)
  document.getElementById("Button3").disabled = false;
  document.getElementById("Button3").style.backgroundColor = "#6f459e";
}

// Function calculate_gap
function calculate_gap(gap) {
```

```
  gap = parseInt((gap * 10) / 13, 10);
  if (gap < 1) return 1;
  return gap;
}

// Asynchronous function to perform "Comb Sort"
async function CombSort(delay = 600) {
  console.log("clicked")
  let bars = document.querySelectorAll(".bar");
  var gap = 20;
  let swapped = true;

  while (gap != 1 || swapped == true) {

    // Calling calculate_gap function
    gap = calculate_gap(gap);

    // Initializing swapped with false
    swapped = false;

    for (var i = 0; i < 20 - gap; i++) {

      // Assigning value of ith bar into value1
      var value1 = parseInt(bars[i].childNodes[0].innerHTML);

      // Assigning value of i+gapth bar into value2
      var value2 = parseInt(bars[i + gap].childNodes[0].innerHTML);
      if (value1 > value2) {

        // Provide red color to the ith bar
        bars[i].style.backgroundColor = "red";

        // Provide red color to the i+gapth bar
        bars[i + gap].style.backgroundColor = "red";
```

```
  // Swap ith bar with (i+gap)th bar
  var temp1 = bars[i].style.height;
  var temp2 = bars[i].childNodes[0].innerText;
  // To pause the execution of code for 300 milliseconds
  await new Promise((resolve) =>
    setTimeout(() => {
      resolve();
    }, 300)
  );
  // Swap ith bar with (i+gap)th bar
  bars[i].style.height = bars[i + gap].style.height;
  bars[i].childNodes[0].innerText = bars[i + gap].childNodes[0].innerText;
  bars[i + gap].style.height = temp1;
  bars[i + gap].childNodes[0].innerText = temp2;
  // Set swapped
  swapped = true;

  // To pause the execution of code for 300 milliseconds
  await new Promise((resolve) =>
    setTimeout(() => {
      resolve();
    }, 300)
  );
}
// Provide skyblue color to the ith bar
bars[i].style.backgroundColor = "rgb(0, 183, 255)";

// Provide skyblue color to the i+gapth bar
bars[i + gap].style.backgroundColor = "rgb(0, 183, 255)";

// To pause the execution of code for 300 milliseconds
await new Promise((resolve) =>
  setTimeout(() => {
    resolve();
```

```
        }, 300)
      );
    }
  }
  for (var x = 0; x < 20; x++) {
    bars[x].style.backgroundColor = "rgb(49, 226, 13)";
  }

  // To enable the button "Generate New Array" after final(sorted)
  document.getElementById("Button1").disabled = false;
  document.getElementById("Button1").style.backgroundColor = "#6f459e";

  // To enable the button "Comb Sort" after final(sorted)
  document.getElementById("Button4").disabled = false;
  document.getElementById("Button4").style.backgroundColor = "#6f459e";
}
// Asynchronous function to perform "Shell Sort"
async function ShellSort(delay = 600) {
  let bars = document.querySelectorAll(".bar");

  for (var i = 10; i > 0; i = Math.floor(i / 2)) {

      // To pause the execution of code
      // for 300 milliseconds
      await new Promise((resolve) =>
          setTimeout(() => {
              resolve();
          }, 300)
      );
      for (var j = i; j < 20; j++) {
          var temp = parseInt(
              bars[j].childNodes[0].innerHTML);
          var k;
          var temp1 = bars[j].style.height;
```

```
var temp2 = bars[j].childNodes[0].innerText;
for (
    k = j;
    k >= i && parseInt(bars[k - i]
        .childNodes[0].innerHTML) > temp;
    k -= i
) {
    bars[k].style.height
            = bars[k - i].style.height;

    bars[k].childNodes[0].innerText =
        bars[k - i].childNodes[0].innerText;

    // To pause the execution of code
    // for 300 milliseconds
    await new Promise((resolve) =>
        setTimeout(() => {
            resolve();
        }, 300)
    );
}
// Provide darkblue color to the jth bar
bars[j].style.backgroundColor = "darkblue";

// Provide darkblue color to the kth bar
bars[k].style.backgroundColor = "darkblue";
bars[k].style.height = temp1;
bars[k].childNodes[0].innerText = temp2;

// To pause the execution of code for
// 300 milliseconds
await new Promise((resolve) =>
    setTimeout(() => {
        resolve();
```

```
            }, 600)
        );
        // Provide skyblue color to the jth bar
        bars[j].style.backgroundColor = "rgb(0, 183, 255)";

        // Provide skyblue color to the kth bar
        bars[k].style.backgroundColor = "rgb(0, 183, 255)";

        // To pause the execution of code for
        // 300 milliseconds
        await new Promise((resolve) =>
            setTimeout(() => {
                resolve();
            }, 300)
        );
    }
}
for (var x = 0; x < 20; x++) {
    bars[x].style.backgroundColor
        = "rgb(49, 226, 13)";
}
// To enable the button "Generate New Array"
// after final(sorted)
document.getElementById("Button1")
        .disabled = false;

document.getElementById("Button1")
    .style.backgroundColor = "#6f459e";

// To enable the button "Shell Sort"
// after final(sorted)
document.getElementById("Button5").disabled = false;
document.getElementById("Button5")
    .style.backgroundColor = "#6f459e";
```

```
}
// Call "generatebars()" function
generatebars();
// Function to generate new random array
function generate()
{
  window.location.reload();
}
// Function to disable the button
function disable()
{
  // To disable the button "Generate New Array"
  document.getElementById("Button1")
  .disabled = true;
  document.getElementById("Button1")
  .style.backgroundColor = "#F1F1F1";

  // To disable the button "Insertion Sort"
  document.getElementById("Button2")
  .disabled = true;
  document.getElementById("Button2")
  .style.backgroundColor = "#F1F1F1";

  document.getElementById("Button3").disabled = true;
  document.getElementById("Button3").style.backgroundColor = "#F1F1F1";

  document.getElementById("Button4").disabled = true;
  document.getElementById("Button4").style.backgroundColor = "#F1F1F1";

  document.getElementById("Button5").disabled = true;
  document.getElementById("Button5").style.backgroundColor = "#F1F1F1";
}
```

5.1.2 Index.html:

```
<!DOCTYPE html>
<html lang="en">
  <!-- head -->
  <head>
    <!-- linking style.css -->
    <link href="style.css" rel="stylesheet" />
  </head>
  <!-- body -->
  <body>
    <nav>
      <span>Algolizer</span>
  </nav>
    <section class="data-container"></section>
    <section id="ele"></section>
  <div class="new__array">
     <!-- "Generate New Array" button -->
   <button  class="button-89 "
   onclick="generate()" id="Button1" >
Generate New Array</button>
  </div>
    <div class="sort__btn">
   <!-- "Insertion Sort" button -->
   <button class="button-89 btn2 insBtn"
          onclick="InsertionSort(),disable()"
          id="Button2" >
     Insertion Sort</button>
    <!-- </div>
    <div style="margin: auto; width: fit-content;"> -->
     <!-- "Generate New Array" button -->
   <!-- "Insertion Sort" button -->
   <button class="button-89 btn2" style="margin-top: 12px;"
          onclick="SelectionSort(),disable()"
          id="Button3" >
```

```
    Selection Sort</button>
   <button class="button-89 btn2" style="margin-top: 12px;"
        onclick="CombSort(),disable()"
        id="Button4" >
    Comb Sort</button>
   <button class="button-89 btn2" style="margin-top: 12px;"
        onclick="ShellSort(),disable()"
        id="Button5" >
    Shell Sort</button>
    <!-- <input type="button" value="Selection Sort" onclick="SelectionSort()"> -->
   </div>
  </body>
  <!-- linking index.js -->
  <script src="index.js"></script>
  <!-- <script src="selection.js"></script> -->
</html>
```

5.2 Implementation Details (Output):

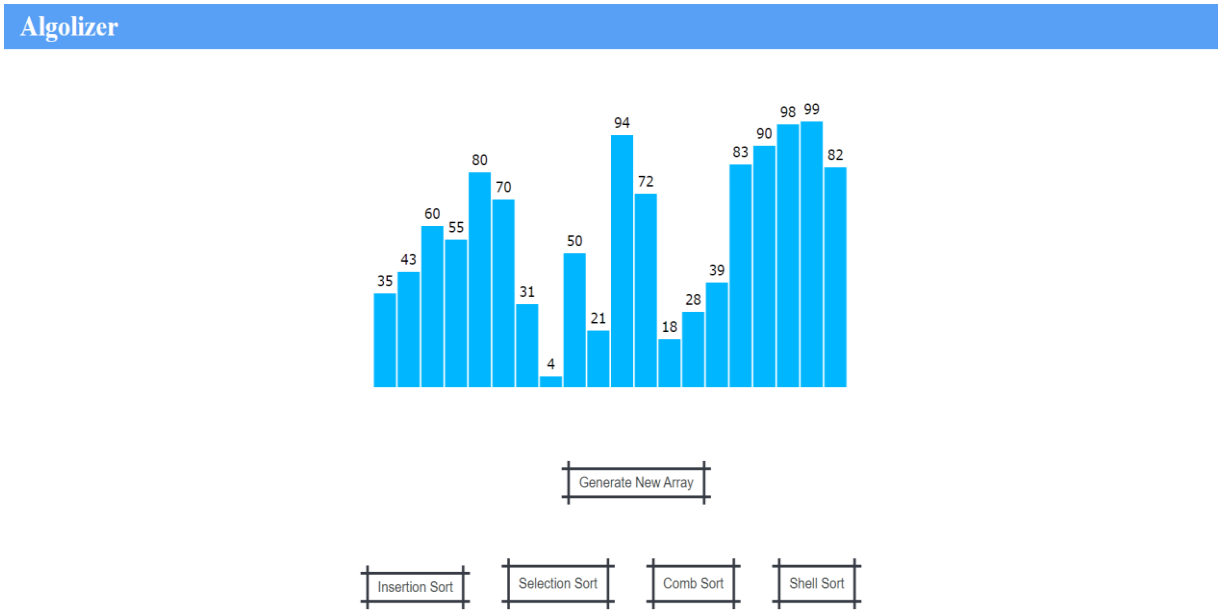

Figure 8 : Output After Running Program

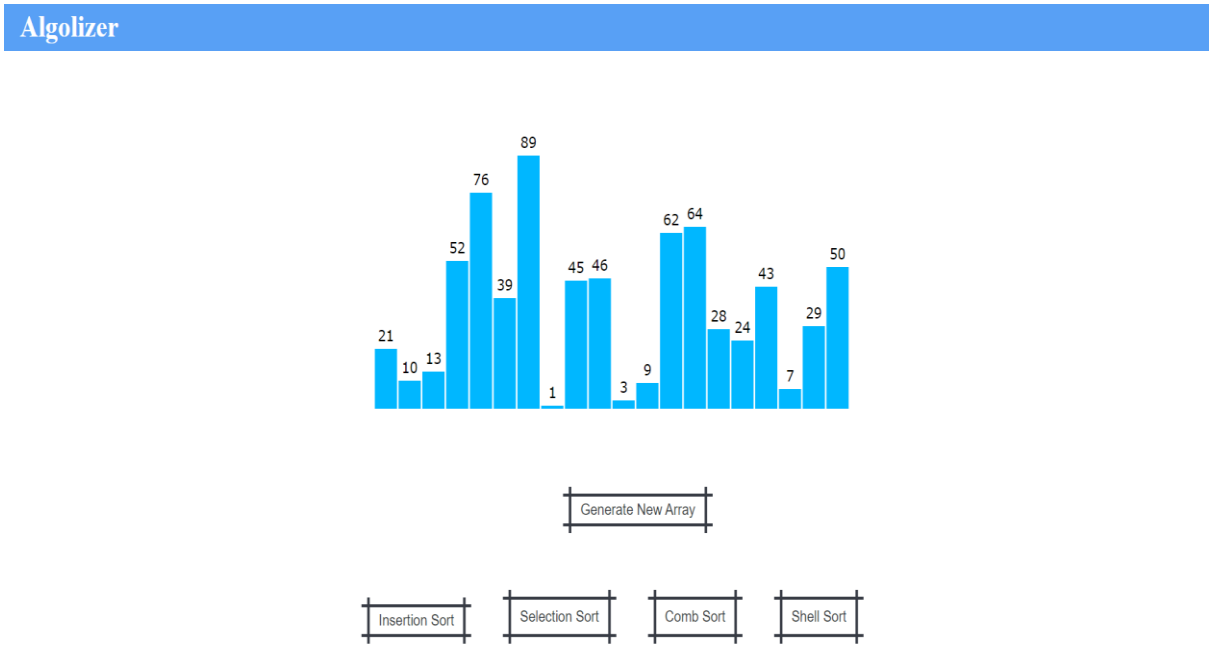

Figure 9 : Output after Click on Generate New Array

ALGOLIZER – Basic Algorithm Visualization

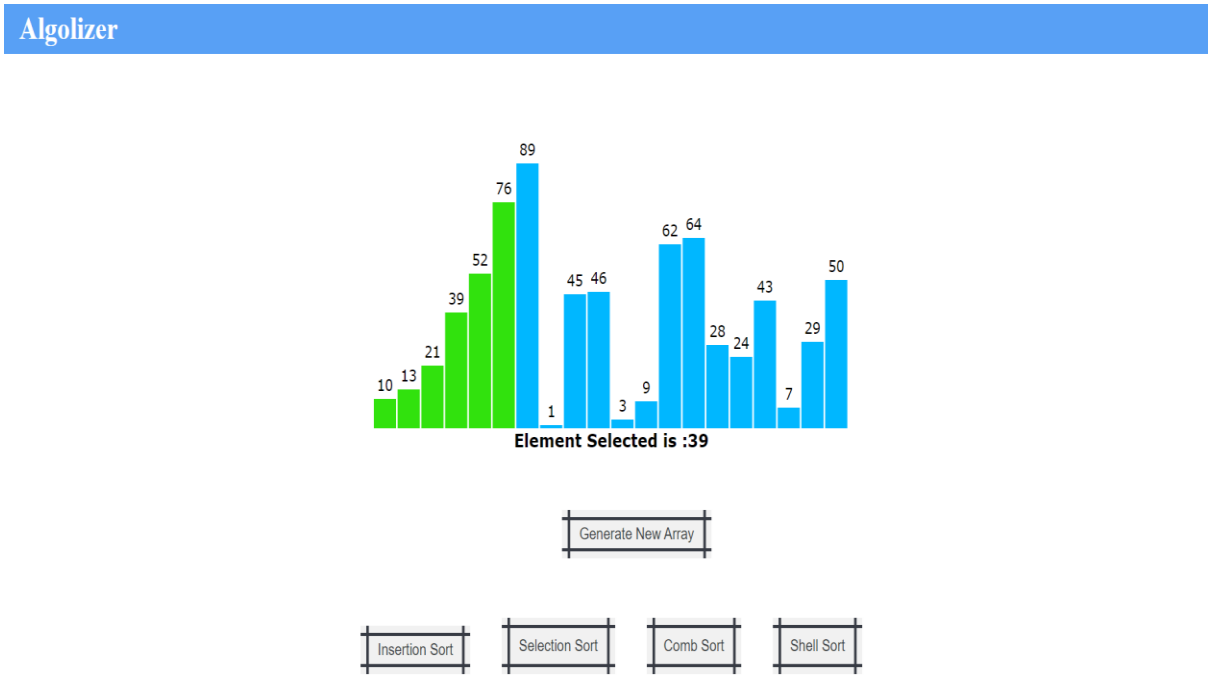

Figure 10 : Algorithm Visualization Ongoing Process

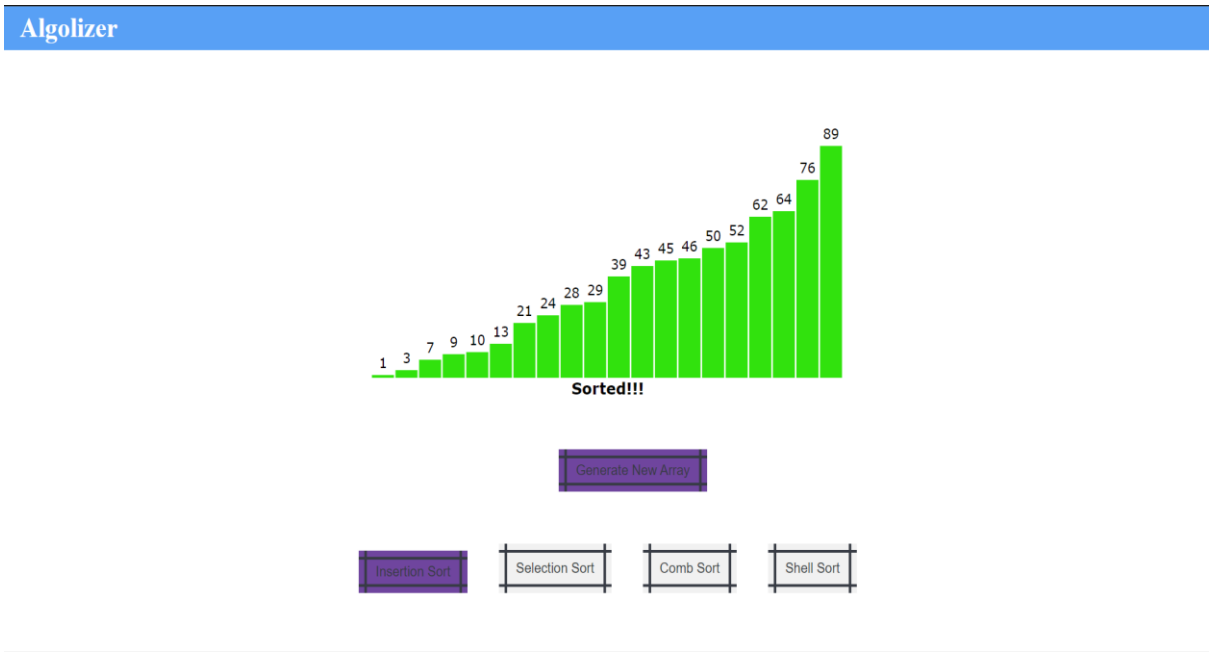

Figure 11 : Output After Sorting Algorithm

CHAPTER 6
CONCLUSION

According to our findings, algorithm visualization can be seen as a valuable supporting tool, used in addition to standard ways of education in the field of computer science. Within the paper we provided an overview of the Algolizer algorithm visualization platform as well as our practical experiences with the system. We believe (and the results of questionnaire support our belief) it helps to improve the quality of education in the field and contribute to the solution for some of the problems in higher education mentioned at the beginning of the paper.

I tried to create high-quality software with a user-friendly and easy-to-use interface, which could be used by lecturers, tutors, and students. Possible next improvement of the applications is extension it by other algorithms. The first part of the thesis text is more theoretical. It tells about algorithms in general and the algorithms represented in the application. The second part is focused on the application itself.

REFERENCES :

1. CORMEN, T. H.; LEISERSON, C. E.; RIVEST, D. L.; STEIN, C. Introduction to algorithms. Second Edition. 2001. ISBN 0-262-03293-7.
2. KNUTH, D. The Art of Computer Programming: Fundamental Algorithms. Third Edition. 2004. ISBN 0-201-89683-4.
3. SIPSER, M. Introduction to the Theory of Computation. Boston, MA: PWS Publishing Company, 1997. ISBN 0-534-94728-X.
4. BĚLOHLÁVEK, R. Algoritmická matematika 1 : část 2. Available also from: hhttp://belohlavek.inf.upol.cz/vyuka/algoritmicka-matematika-1-2.pdfi.
5. KNUTH, D. The Art of Computer Programming: Sorting and Searching. Second Edition. 2004. ISBN 0-201-89685-0.
6. SEDGEWIK, R. Algorithms in C : Fundamentals, data structures, sorting, searching. Third Edition. 2007. ISBN 0-201-31452-5.
7. GeeksforGeeks. Available from: hhttps://www.geeksforgeeks.org/i.
8. BĚLOHLÁVEK, R. Algoritmická matematika 1 : část 1. Available also from: hhttp://belohlavek.inf.upol.cz/vyuka/algoritmicka-matematika-1-1.pdfi.
9. Stackoverflow. Available from: hhttps://stackoverflow.com/i.
10. Java documentation. Available from: hhttps://docs.oracle.com/javase/8/i.