

SORTING ALGORITHM VISUALIZER

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

KISHORE KUMAR K (37110345)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND
TECHNOLOGY(DEEMED TO BE
UNIVERSITY)

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by
AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

March 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC I 12B Status by UGC I Approved by
AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **VAIBHAV KONDABATHINI (37110345)** who carried out the project entitled "SORTING ALGORITHM VISUALIZER" undermy supervision from November 2020 to March 2021.

Internal Guide

Dr. B. BHARATHI, M.E., Ph.D.,

Head of the Department

Dr. S.Vigneshwari M.E., Ph.D.,

Head of the Department

Dr. L.Lakshmanan M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I VAIBHAV KONDABATHINI (37110345) hereby declare that the project entitled **“SORTING ALGORITHM VISUALIZER”** is done by me under the guidance of **Dr. B. BHARATHI, M.E., Ph.D.**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D., and Dr.S.Vigneshwari M.E., Ph.D.,** Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. B. BHARATHI, M.E., Ph.D.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

With the recent surge in interest in computational thought, an important question has arisen: what are the best methods for teaching students basic computing concepts? Visualization is considered as one way to support student learning. With an aim to help and motivate students, number of researchers have come up with various tools. Although many tools are available this problem seems to be still persistent. This paper proposes a web application used to visualize seven commonly used sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and Tim Sort. The web application represents data in the form of a bar graph and user will be able to select the algorithm, control and speed of visualization and control the size of the bar graph. This paper also shows the Architectural design and technical structure of the application as well as its practical use and educational benefits are presented and discussed.

TABLE OF CONTENTS

S.NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF FIGURES	viii
1.	CHAPTER 1: INTRODUCTION	
	1.1 INTRODUCTION	1
	1.2 RESEARCH AND SIGNIFICANCE	2
2.	CHAPTER 2: LITERATURE SURVEY	3
3.	CHAPTER 3: AIM AND SCOPE	
	3.1 AIM	5
	3.2 SCOPE	5
	3.3 VISUAL LEARNING	5
4.	CHAPTER 4: SYSTEM DESIGN AND METHODOLOGY	
	4.1 PROPOSED SYSTEM	6
	4.2 REQUIREMENT SPECIFICATION	6
	4.2.1 SOFTWARE REQUIREMENTS	6
	4.2.2 HARDWARE REQUIREMENTS	7
	4.3 SYSTEM FUNCTIONALITIES	7
	4.4 SORTING ALGORITHMS	9
	4.4.1 BUBBLE SORT	9
	4.4.2 SELECTION SORT	10
	4.4.3 INSERTION SORT	10
	4.4.4 QUICK SORT	11
	4.4.5 MERGE SORT	12
	4.4.6 HEAP SORT	12
	4.4.7 TIM SORT	13

5.	CHAPTER 5: SYSTEM DEVELOPMENT METHODOLOGY	
	SYSTEM ARCHITECTURE	14
	MODEL	14
	VIEW	17
	CONTROLLER	17
	INTERNAL STRUCTURE	17
6.	CHAPTER 6: RESULTS AND DISCUSSION	20
7.	CHAPTER 7: CONCLUSION	23
	 REFERENCES	 24
	 APPENDIX	 25
	1. PAPER ACCEPTANCE MAIL	
	2. PLAGIARISM REPORT	
	3. JOURNAL PAPER	
	4. SOURCE CODE	

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
4.1	DEFAULT BUTTON	9
4.2	SELECTED BUTTON	9
4.3	FROZEN BUTTON	9
4.4	BUBBLE SORT PSEUDO CODE	10
4.5	SELECTION SORT PSEUDO CODE	10
4.6	INSERTION SORT PSEUDO CODE	11
4.7	QUICK SORT PSEUDO CODE	11
4.8	MERGE SORT PSEUDO CODE	12
4.9	HEAP SORT PSEUDO CODE	13
4.10	TIM SORT ALGORITHM	13
5.1	MVC ARCHITECTURE	14
5.2	INTERNAL CODE STRUCTURE	19
6.1	HOME PAGE	20
6.2	ARRAY CONTROLS	21
6.3	ALGORITHM CONTROLS	21
6.4	COLOR PALETTE	21
6.5	TIME COMPLEXITIES	22

CHAPTER 1

INTRODUCTION

INTRODUCTION

Visualization is proposed as one of the ways for supporting student learning. Me and my partner are visual learners, and we are keener to picking up concepts by seeing it get done, rather than reading about it. For example, when we were learning about sorting algorithms during the second semester Computer Science degree, we found it hard to grasp the working of a sorting algorithm just by seeing it explained on board or reading it as there is loads of recursion that takes place, we thought that maybe if these algorithms were taught visually and professors show how the data moves to its position, we would understand the concept better. For years, many educators have depended on lecturing, storytelling, and blackboard teaching to deliver information. Standardized exams, in written format, highlight verbal learning. But in a world entirely filled with laptops, smartphones, tablets, and VR machines it becomes predominant to dynamically teach students to read and produce visual texts and to espouse this instinct to risk falling behind. As such, many professors find themselves without the assets needed to give 21st century students the dexterity they'll need to succeed in a progressively visual world. The advantages, difficulties, and possibilities of integrating such visualization, on the other hand, need to be clarified. Teaching staff are becoming more interested in integrating visual representation into their methods to generate stimulating learning experiences for students in face-to-face, blended, and online contexts. Visualization can be incorporated into presentation software such as PowerPoint, which is a widely used slide ware, but critics argue that they are mostly used for decoration rather than just in innovative ways to promote learning. Visual representation has a lot of potential to improve learning and teaching at all stages, from pedagogical practice research to scholarship, connecting research and teaching, planning and curriculum development, and presentation and assessment, to name a few. To keep current, curricula can include research, which may include the teacher's own disciplinary and pedagogical studies.

While visualization can help to visually reinforce text-heavy forms of communication, they can also express meaning and affect The Significance of Visual Learning and Teaching are:

- help students engage with the topics
- increase retention by 30-44%
- develop higher-order reasoning skills
- sharpen fundamental abilities that permit students to see and visualize data clearly
- Layout new opportunities to students with learning differences and challenge students who are twice exceptional.

RESEARCH AND SIGNIFICANCE

The main aim of this application I.e. SAV(Sorting Algorithm Visualizer) is to assist students who are primarily visual learners and understand topic better when they are exposed to content visually. Thus, we have developed a web-based application that animates how sorting algorithms work and it visualizes the process for us. If you want to organize a group of students in their height order, there are many algorithms that can perform this task. We visualized six of the commonly used sorting algorithms by representing the data as a bar graph. Each number is represented as a bar of different heights based on the value and even redundant or repeated data is also sorted out. The seven algorithms are: Bubble sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and Tim Sort.

The main goal for us was for the animation to be web based so that a whole lot of people using different technologies can use it without having the trouble of installing special software or an application. The web application is coded with HTML5, CSS3 and JavaScript. The physical elements of the web application are coded via HTML5. The appearance I.e., the interactive layout of the web application is entirely done by CSS. Finally, the remaining is done by JavaScript, which is used for the histogram representation, movement of bars, algorithm design.

CHAPTER 2

LITERATURE SURVEY

There have been a lot of studies and animation tools on visualizing sorting algorithms. Some are based on how to make such an application and others on different technologies which were aimed for increasing the understanding of the concept. The work that we looked up and which also helped us throughout this project was Clement Mihailescu Algorithm Visualizer. Other visualization tools include ViSA, Selection sorting Algorithm Visualization Using Flash to name a few. These applications had a few drawbacks therefore, the approach we focused on was making the visualizer more interactive and fix the drawbacks that the other visualizing tools had.

The paper “Algorithm Animation” by A. Kerren et al [1], is a bit-by-bit guide to show the means and necessary coding methods to use sorting visualization. The paper mentioned about a web application-based tool which is outdated now, and the rest of the tools had us wanting to use other software's or applications.

The paper “SELECTION SORTING ALGORITHM VISUALIZATION USING FLASH”
by

Hadi Sutopo, The International Journal of Multimedia & Its Applications (IJMA) Vol.3, No.1, February 2011[2], discussed about only Selection Sort algorithm and visualizing it. The project was built using Adobe Flash and ActionScript programming tools and as of December 2020 Adobe Flash has been discontinued due to the rise security risk of using it.

In the paper “ViSA: Visualization of Sorting Algorithms” by I. Reif and T. Orehovacki , 2012[3] they provide an in-depth view of how sorting visually takes place on the data, but the limitation was that the data should be entered manually by the user which isn't effective if the user wants to enter large numbers of data.

In Clement Mihailescu's work, 2016 [4] not all the commonly used algorithms were implemented, the project was only half done as the user interface was not interactive and there wasn't any description about the sorting algorithms.

In the paper “Visualizing Sorting Algorithms” by Brian Faria, 2017 [5] though the work only had four commonly used sorting algorithms the implementation by him on using different techniques like using sound to show the swapping of data was different from others, but there wasn't any way for the user to control the speed at which the visualization takes place nor the user was able to control the size of the array which in my opinion was needed for better understanding of the algorithms when the data size is huge.

In the paper “Increasing the Engagement Level in Algorithms and Data Structures Course by Driving Algorithm Visualizations” by Slavomir Simonak, 2020 [6], here he uses Algomaster platform, and the project is developed by C programming language, thus the user-interface isn't as interactive as a one uses HTML, CSS and JavaScript because these tools are more into web application development thus, making the UI more interactive.

We have seen quite a few sorting visualizers online in YouTube. There were quite fascinating ways of visually showing how the data was sorted, the algorithm visualization showed in these videos were via dance. These videos are on posted on the YouTube channel known as AlgoRythmics. The algorithms aren't animated using any tools or graphics, a group of performers represent the numbers by wearing them on as a dress. A Bulgarian folk music will be played in the background and the performers would stand in a line (or a number of lines depending upon the sorting algorithm) and then they dance to show the comparisons as well as the swapping of data. The comparisons are depicted by two dancers from the group coming forward and standing adjacent to each other temporarily as their values were being compared. If there was a need to swap after comparing them, they would perform a quick dance move and swap to each other's position. One of the features we found interesting was that they sometimes used a hat to show which index in the data is the pivot or the one being compared to others. This gave is the idea of representing different elements in different colors instead of just one-color throughout the entire sorting visualization.

CHAPTER 3

AIM AND

SCOPE

AIM

The primary aim of this proposed system is to help students who are naturally visual learners and tend to grasp concepts and remember it if they see it visually that is where SVA (Sorting Algorithm Visualizer) comes which helps people understand basic sorting algorithms by seeing them visually take place. This is not only for new learners but also for people who want to brush up on their knowledge of sorting algorithms helping them remember the topic faster rather than reading through it all over again.

SCOPE

Since SVA is a web application a large number of people can use it with ease without having the need of downloading any external software or application and this can be very helpful for students.

VISUAL LEARNING

People have natural tendencies to communicate visually, and visuals play a huge part in how the information is processed. Primitive cave drawings show an original desire to use pictures in an attempt to convey thoughts and meaning. Even today, most toddlers seek to make meaningful marks, drawing rudimentary pictures to show what's going on in their heads. Generative Art is one of the ways to represent computational processes. Transforming the data generated by an autonomous system into the features of an artwork can lead to unexpected results. SAV was born to create visual representations of sorting algorithms with the hope of finding visual patterns. It turned out that the visual footprints of algorithms are unique and differ from each other and helps people who are new to the concept of sorting understand these algorithms quicker and the way they work.

CHAPTER 4

SYSTEM DESIGN AND METHODOLOGY

PROPOSED SYSTEM

The proposed system I.e. SAV consists of various features which makes this web application efficient for the user. The entire user interface is entirely done by front end technologies (HTML, CSS, Bootstrap and JavaScript). The user interface has 14 components: The main navbar allows the user to set the size of the array and to control the speed of visualization, the buttons to generate a new random array, to stop the visualization and the sorting algorithm buttons. Below it is the secondary navbar containing the color palette which shows the colors used during the visualization. Below it is the canvas which is divided into 3 parts: The algorithms description, the bar graph and the text box fields which show the time complexities, number of swaps, number of comparisons and the time taken for each algorithm whenever a particular algorithm is clicked.

REQUIREMENT SPECIFICATION

This proposed software runs effectively on a computing system that has the minimum requirements. Undertaking all the equipment necessities are not satisfied but rather exist in their systems administration between the user's machines already. So, the main need is to introduce appropriate equipment for the product. The requirements are split into two categories, namely:

SOFTWARE REQUIREMENTS

The basic software requirements to run the program are:

1. Windows 10
2. Visual Studio Code
3. HTML, CSS, JavaScript, Bootstrap
4. Browser example Mozilla Firefox, Safari, chrome, etc.

HARDWARE REQUIREMENTS

The basic hardware required to run the program are:

1. Hard disk of 5 GB
2. System memory (RAM) of 512 MB
3. I3 processor-based computer or higher.

SYSTEM FUNCTIONALITIES

The first row of the Navbar section has the range unit to set the size of the array with a minimum size of 10 and a maximum of 150 elements the wide range is mainly to show how these algorithms work for larger data visually as it's impossible to show students how an array with over 100 elements gets sorted but visually seeing it people can understand how they work.

The second row has the range controller to set the speed of visualization with the slowest speed at showing every swap in detail so that students can grasp the idea of what's going on and the fastest to show how different algorithms quickly sort an array at larger data, for example trying to sort an array of 100 elements by bubble sort can take a really long time as the slowest setting and can be time consuming but merge sort can do it in couple of seconds by this even students can get an idea of how algorithms differ between each other in speed. The third row has the buttons to generate new array and to stop the sorting midway.

The fourth row contains all buttons for the selectable algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort and Tim Sort by default no algorithm is selected, user needs to select the desired algorithm to see it visually. The fifth row is the color palette. It represents the colors used on the graph where blue represents the default array, yellow represents the current element in which the user is on, red represents the elements which are compared and to be swapped and green represents the element which is in it sorted position.

Below the navbar is the canvas section which is divided into 3 parts.

The first part is a carousel slider gives a small description of each algorithm which we use in the visualizer, the second part is the bar graph representation itself, the third part shows the time complexities, number of swaps, number of comparisons and the time taken for each algorithm whenever a particular algorithm is clicked.

The first column mainly gives detailed explanation on what going on each algorithm thus when a student wants to know what the algorithm is about, they can navigate through the carousel and read about the desired algorithm they want to know more about the idea behind it is that students can have read it and get an idea of what exactly is the sorting algorithm about so that when they visually see it, they get a better idea.

The second column contains the bar graph representation where the sorting takes places, initially the default bars are represented by blue color and when an algorithm is selected the sorting takes place and colors change accordingly. The third column is a table representing all the time complexities of the algorithms, number of swaps, comparisons as well as the time taken for each algorithm to execute which can be helpful for a someone who needs to know and compare which algorithm performs well in what conditions. which can be helpful for a someone who need to know and compare which algorithm performs well in what conditions. We have kept user interface as simple as we could and grouped similar things together. All the buttons do show certain features when the cursor is hovered over them. An example is shown below where Figure 4.1 shows the default button, Figure 4.2 shows the button when hovered over it and

when an algorithm is clicked all the elements except for the “stop sorting” button is frozen and can't be clicked until and unless the “stop sorting” button is clicked which is shown in Figure 4.3.

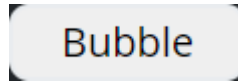


Figure 4.1: DEFAULT BUTTON

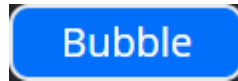


Figure 4.2: SELECTED BUTTON

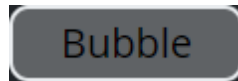


Figure 4.3: FROZEN BUTTON

SORTING ALGORITHMS

A Sorting Algorithms main purpose is to reposition a given array or list of elements in ascending or descending order. Now taking about the algorithms itself we are using 7 sorting algorithms in this visualizer namely Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and Tim Sort. These algorithms are the commonly used algorithms for sorting.

BUBBLE SORT

Bubble sort also known as sinking sort from time to time, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. But the algorithm is not suitable for amounts of data as its average complexity is of $O(n^2)$, Where n is the number of elements. The pseudocode of bubble sort is shown below in Figure 4.4.

```

begin BubbleSort(list)
  for all elements of list
    if list[i] > list[i+1]
      swap(list[i],list[i+1])
    end if
  end for

  return list
end BubbleSort

```

Figure 4.4: BUBBLE SORT PSEUDO CODE

SELECTION SORT

Selection sort is an in-place comparison sorting algorithm. It has an $O(n^2)$ time complexity, which makes it unsuitable for an array size of large numbers, and in general executes worse than the similar insertion sort. Selection sort is well known for its simplicity and has performance advantages over more intricate algorithms in certain circumstances, particularly where additional memory is limited. The algorithm works by pushing the least element to the start of the array by swapping it with the first element of the array and the pointer pointing to the first element is moved to the next element and this process goes on till the last element of the array and eventually the array will be sorted. Figure 4.5 shows the pseudo code.

```

begin SelectionSort(list)
  for all elements of list
    min = list[i]
    for j = i+1 to n
      if list[j] < list[min]
        min = j;
      end if
    end for
    if indexMin != i then
      swap list[min] and list[i]
    end if
  end for
end SelectionSort

```

Figure 4.5: SELECTION SORT PSEUDO CODE

INSERTION SORT

Insertion Sort is an in-place comparison sorting algorithm. In this algorithm, a sub array is kept which is always sorted.

An element which needs to be inserted in this sub array, has to find its right position where it should be and then it will be inserted in that position. Hence, it's called Insertion Sort. The pseudo code of it is shown below in Figure 4.6.

```
begin InsertionSort(list)
  for i = 1 to length(list):
    key = list[i]
    j = i-1

    while j >= 0 and key > list[j]
      list[j+1] = list[j]
      j = j-1
    end while

    list[j+1] = key
  end for
end Insertion Sort
```

Figure 4.6: INSERTION SORT PSEUDO CODE

QUICK SORT

Quick Sort is very efficient sorting algorithm except for its worst-case complexity and it is based on partitioning the array into two halves by taking a pivot element which can either be the first or last element of the array, the element on the left side of the pivot will be smaller than it and elements on the right side will be larger than it. We do this partitioning recursively until the array is sorted. In the below Figure 4.7 the pseudocode is shown.

```
procedure quickSort(left, right)

  if right-left <= 0
    return
  else
    pivot = A[right]
    partition = partitionFunc(left, right, pivot)
    quickSort(left, partition-1)
    quickSort(partition+1, right)
  end if
end procedure
```

Figure 4.7: QUICK SORT PSEUDO CODE

MERGE SORT

Merge Sort is one the most efficient sorting algorithms among all the algorithms as even its worst-case time complexity being $O(n \log n)$. It follows the divide and conquer method as it divides the array into smaller sub arrays and compares these sub arrays and sorts them accordingly and merges these sorted sub arrays to get the final array. The pseudo code is shown below in Figure 4.8.

```
procedure MergeSort(list)
    if len(list) <= 1
        return
    mid = len(list)/2
    left = list[:mid]
    right = list[mid:]

    MergeSort(left)
    MergeSort(right)
    Merge_two_list(list, left, right)

procedure Merge_two_list(list, l, r)
    compares both array's l and r sorts
    them and merge's the array's
```

Figure 4.8: MERGE SORT PSEUDO CODE

HEAP SORT

Heap sort is a comparison based sorting algorithm. Heap sort is actually like a concept of an upgraded selection sort: like selection sort, heap sort divides the array into a sorted array and an unsorted array, and it recursively reduces the unsorted array by taking the largest element from the unsorted array and inserts it into the sorted array. Unlike selection sort, heap sort does not waste time with a linear time scan of the unsorted array. Rather, heap sort keeps the unsorted array in a heap data structure to more quickly find the largest element in each recursion of the unsorted array.

Below in figure 4.9 is the psuedo code of heap sort.

```

Heapify(A as array, n as int, i as int)
{
    max = i
    leftchild = 2i + 1
    rightchild = 2i + 2
    if (leftchild <= n) and (A[i] < A[leftchild])
        max = leftchild
    else
        max = i
    if (rightchild <= n) and (A[max] > A[rightchild])
        max = rightchild
    if (max != i)
        swap(A[i], A[max])
        Heapify(A, n, max)
}
Heapsort(A as array)
{
    n = length(A)
    for i = n/2 downto 1
        Heapify(A, n, i)

    for i = n downto 2
        exchange A[1] with A[i]
        A.heapsize = A.heapsize - 1
        Heapify(A, i, 0)
}

```

Figure 4.9: HEAP SORT PSEUDO CODE

TIM SORT

Tim sort is a stable mixed sorting algorithm, acquired from insertion sort and merge sort, designed to perform exceptionally well on many kinds of real world data. Time Peters was the one who created this hybrid algorithm and it's the commonly used algorithm in the inbuilt functions for sorting in programming languages like Python, Java and JavaScript. The algorithm finds subsets of the data that are already sorted and uses them to sort the remaining array more efficiently. This is done by merging runs until certain criteria are fulfilled.

- Initialize a RUN with the size of 32.
- Implement Insertion sort for RUN size chunks.
- A function merge(int arr[], int l, int m, int r) takes an array, left elements, middle of the array and right elements of the array as input. The function returns the merged sorted chunks of size 32.
- Initialize length of the array having all the left elements and the length of the array having all the right elements.
- After filling the left array and right array, Iterate over the left array as well as right array.
- If the element in the left array is less than the element in the right array, push the element into a larger array.
- Otherwise push the element into a smaller array accordingly.
- Copy the remaining element in the left array and right array into a larger array.
- A function timSortAlgo(int arr[], int n) takes an array and its size as input. Which calls the insertion Sort initially and merges the array elements.
- Return the output as the final elements of the array using Tim Sort.

FIGURE 4.10 TIM SORT ALGORITHM

CHAPTER 5

SOFTWARE DEVELOPMENT METHODOLOGY

SYSTEM ARCHITECTURE

The project is implemented by a combination of HTML5, CSS3, Bootstrap (CSS framework) and JavaScript. It contains 3 files “index.html” which consists of the html code along with the scripts which links to JavaScript files, “style.css” which contains the user interface design and finally the “scripts” file which contains majority of the code that is the visualization and all the sorting algorithms. The code follows Model-View- Controller architecture. Below in Figure 5.1 the architecture diagram is shown.

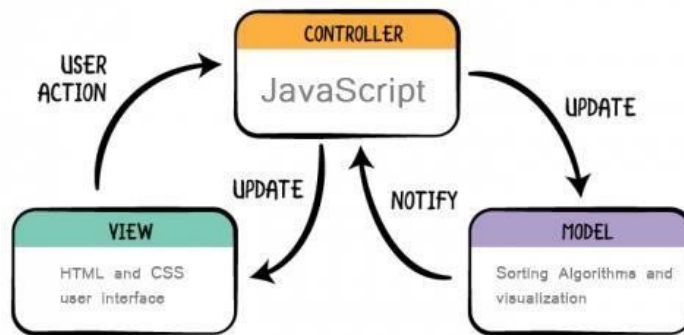


Figure 5.1: MVC ARCHITECTURE

MODEL

The Model is incorporated of one object, called sorter. This object holds all the sorting algorithm code arranged as individual methods. A centralized algorithm buttons when clicked goes to a switch case of sorting algorithm methods, and then calls the appropriate one. The seven buttons labeled as the seven sorting algorithms (Bubble sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and Tim Sort) on the user interface directly control this object. When the user selects a sorting algorithm, the corresponding sorting algorithm method is called, and the time complexities, swaps, comparisons and time taken for the algorithm to take place are calculated. All of this is stored as a variable named as best, avg, worst, swaps, comp and two other variables named t0 ad t1 to calculate the time taken to execute the algorithm, this is public and attainable for the methods connected to the buttons on the web application user interface.

The divs array is the kind of interface through which the user interface connects with the back end code. The result is a visualization of the bars comparing with each other and moving to its correct position thus giving us the sorted bar graph.

The View iterates over the array and update the bars in the bar graph to show the steps that the algorithm took. One thing to keep in mind about the steps is that they are only documented if the algorithm changes the position of a piece of data. Selection Sort, for example, completes quickly because it always moves one piece of data to its correct and final position after each step, whereas other algorithms may require multiple moves before a piece of data reaches its final position. This is deceiving because, while Selection Sort appears to finish the fastest visually, it performs the most work when comparing data and is the least preferred alternative when sorting data. As a consequence, the visualization does not efficiently express data comparisons, which is a key feature of sorting algorithms. The memory specifications of a two-dimensional array are one of its drawbacks. Because the div stores each step required to sort the data, the array's size is determined by the number of divs. We should look at the time it takes to analyze the algorithm's space requirements. Big-Oh analysis is a common technique in Computer Science for analyzing time requirements. The notation consists of a capital letter O followed by a constraint in parentheses that denotes the algorithm's worst-case performance. Selection Sort and Insertion Sort, for example, have a time complexity of $O(n^2)$, pronounced Big-Oh of n squared, since they complete at a rate proportionally bounded by n^2 , or the number of items in the set of data squared. This isn't always the same as the amount of space required for the array. Because there are (10-150) pieces of data in each algorithm, the array is random cells wide and the number of rows needed differs. The size of the divs array is determined by several factors, including the sorting algorithm used, the points in the algorithm that are displayed, and whether the entire array is stored or only one row is computed at a time. For example, points could be displayed each time an item is moved or each time an item is returned to its proper location. The size of the divs array is determined by the algorithm selected. Even if we display every time a value's position changes and store all the rows of the trace in the array at once, we only need n rows with Selection Sort.

There are no extra data movements at each step because one of the values is switched into its proper location.

In contrast, insertion sort would necessitate $O(n^2)$ rows. If we show each time a value shifts, the other values would shift as well.

Every time the data is divided in half, you'll get an equal number of values on each side, all the way down to one. Merge/Insertion Sort will still work on different quantities of data; the only difference is that one half may have more values than the other.

We also chose number of elements in the array to be divisible by 2 because it gives us a lot of data to look at visually, which makes it easier to see the symmetry in the array in small as well as in larger arrays. Also, we can minimize the number of rows required to $O(n)$ by only displaying the state of the array after an entire pass through the data (when the sorted left-hand array of data set includes one more value), but this may not be as helpful to someone watching the animation. Insertion/Merge Sort's divide and conquer strategy necessitates swapping values in small sub arrays on a regular basis, resulting in different steps for each sub array merging. Bubble Sort is similar to Selection Sort in that it moves different pieces of data in the same direction to their final positions, requiring several steps to imagine a single outer-loop execution. The comparisons also decide how long an algorithm executes, so this is just half the fight. The slow $O(n^2)$ runtime of Selection Sort, for example, is due to the number of comparisons it performs, not the amount of memory it takes. In order to sort all objects in the data set, the algorithm only performs $n-1$ swaps, limiting memory use and access. Merge Sort, on the other hand, takes twice the memory to make a second array as a sorting ground, but when the data is divided into bits, the number of comparisons made when combining the already sorted sub arrays is reduced. We already know that a div must be created for each movement of the data to be recorded. The steps array would not be very large if the algorithm in question is Selection Sort. If Insertion, Bubble, Merge Sort or Heap Sort is selected, the memory requirement increases in order to save the various steps required for each outer-loop execution or recursive call. Instead, each step can be calculated as the algorithm runs in parallel in the animation, which uses a one-dimensional array to temporarily store the current step before updating to the next. This would keep the memory requirement low and constant, making the animation tool more stable as it runs.

VIEW

Div_update, divs, and div_sizes are the three objects that make up the View. These objects do not belong to any unique object called "View," but rather operate within the space defined by the tags in the .html file. This area can be thought of as the "major" function, which is the first one called in a program, div_update is the object that sorts the bar graph and keeps the individual bars updated using a timer. As a result, when "Step" is named, the existing values of the bars are updated to the new values based on the divs array. At the next iteration of the timer, the div_update will indicate the changes by redrawing the rectangles with various heights that represent the new values.

The bar object is used to represent each piece of data in the generate_array function. It includes the attributes color, value, position and height. By keeping a separate array called bars for the current bars in the bar graph, it is then easy to change any or all the attributes by iterating over the array as necessary.

CONTROLLER

Controllers serve as a link between the Model and View components, processing all business logic and incoming requests, manipulating data with the Model, and interacting with Views to render the final output. The Customer controller, for example, can manage all interactions and inputs from the Customer View and use the Customer Model to update the database. The Customer data can be viewed using the same controller.

INTERNAL STRUCTURE

The View is composed of main and visualization. These objects aren't connected to a particular object known as view, but they are JavaScript files which are under the scripts tags in the .html file. It acts as the "main" function where the entire code starts taking shape.

The main.js consists of HTML dom which is an object model for HTML and also acts as an API for JavaScript to add, change or remove HTML attributes,

here we get to generate the random array, the array size and the speed of visualization from the .html file by the ID. A function generate_array() generates a random array including duplicate elements for the sorting to be more flexible. It is generated by the following snippet

```
div_sizes[i]=Math.floor(Math.random()*0.5*(inp_as.max-  
inp_as.min))+10
```

when a particular algorithm out of the seven is clicked in the main.js a dom element addEventListener() gets the algorithm which the user selected and calls the function disable_buttons() as the name suggests freezes all the buttons except the button to stop the sorting and the function runalgo() which contains a switch case and depending on the algorithm selected it calls the respective .js files which contains the sorting algorithm code. The Model consists of all the sorting algorithm code.

This object contains the codes organized in methods. The seven buttons of the sorting algorithm on the user interface of the home page directly controls this object. When the user clicks on one of the sorting algorithms the correlated sorting algorithm method is called and bar graph starts sorting.

The bar graph is attained from the id "array_container" from the .html file and stored in a variable "cont" via HTML DOM. Inside each of these sorting algorithm codes a variable is declared to assign the time complexities, swaps and comparisons are stored and is displayed in the homepage by the DOM.

In visualization.js a function vis_speed() contains a switch case for the visual speed of the algorithm the lowest being speed=1 and the fastest being speed = 100 we get the value from the id "array_speed" from the .html file via DOM. The next function is the div_update() this is where the bar graphs change in heights when they need to get swapped **"cont.style=" margin:0% "+ margin_size + "%; width:" + (100/array_size-(2*margin_size)) + "%; height:" + height + "%; background-color:" + color + ";; "**. Here the heights of both the bar that need to swapped are passed as parameters in the function call of each algorithm and they are swapped accordingly by taking the difference in their heights and assigning them to one another. After each algorithm is done visualizing the enable_button() function is called

which enables all the disabled buttons since when an algorithm is clicked all the buttons except for the “stop algorithm” button is disabled.

The time taken for each algorithm to take place is calculated by an inbuilt API in JavaScript. called `performance.now()`. The controller incorporates of all the interactive buttons on the web page. These buttons are coded in HTML and are called directly by a JavaScript method when clicked. CSS is used for styling the homepage, buttons, layout, colors etc.

These methods control the state of the Model and update the interactive elements of the web page whenever necessary. Below in Figure 5.2 the entire code structure is shown.

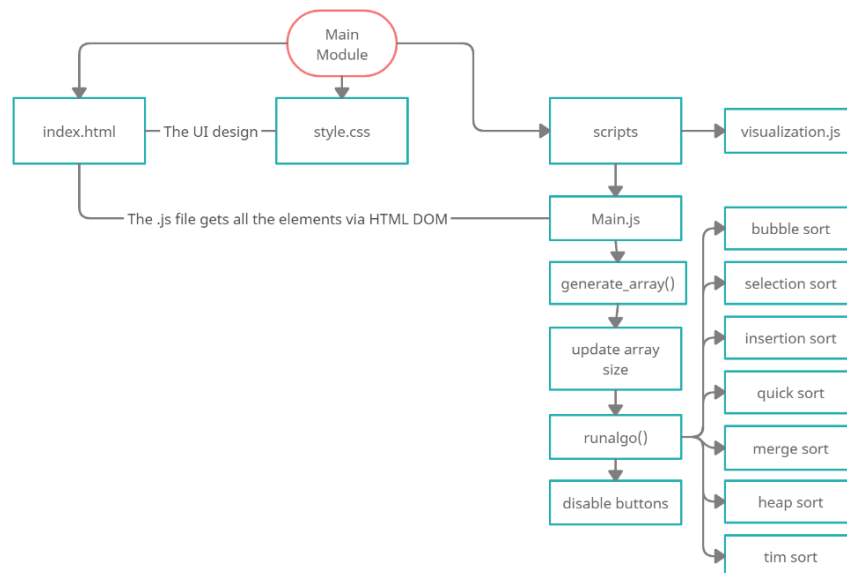


FIGURE 5.2 INTERNAL CODE STRUCTURE

CHAPTER 6

RESULTS AND DISCUSSION

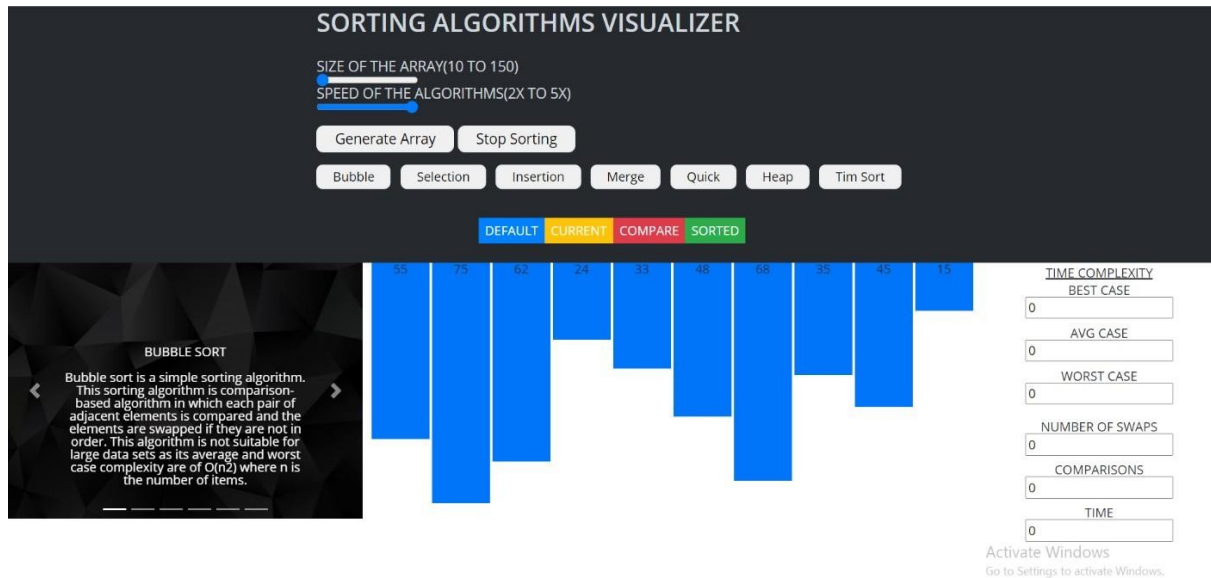


FIGURE 6.1 HOME PAGE

The main goal was to keep the web page and user interface as simple as possible so that it's easy for the user to navigate through the page. All the similar elements are coupled together. In this system the user interface build is professional. It is the front end of the project or it can also be termed as user interface. Here the user gets the multiple options to execute or access their task as per need. The user interface has 14 components: The main navbar allows the user to set the size of the array and to control the speed of visualization, the buttons to generate a new random array, to stop the visualization and the sorting algorithm buttons. Below it is the secondary navbar containing the color palette which shows the colors used during the visualization. Below it is the canvas which is divided into 3 parts: The algorithms description, the bar graph and the text box fields which show the time complexities, number of swaps, number of comparisons and the time taken for each algorithm whenever a particular algorithm is clicked.

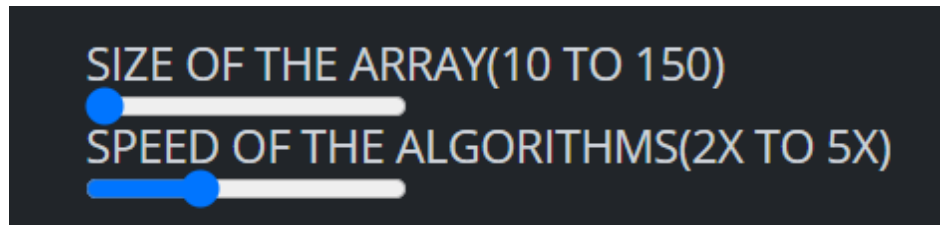


FIGURE 6.2 ARRAY CONTROLS

In the above figure you can see that the bar graph can be controlled via the size of array range option and the speed of the algorithm range option allows us to control the speed of visualization.

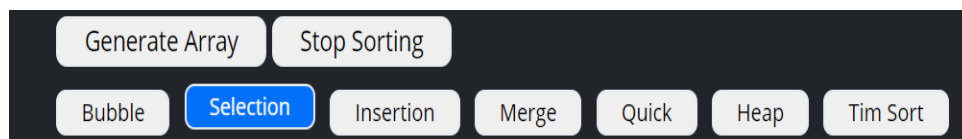


FIGURE 6.3 ALGORITHM CONTROLS

In the above figure we can see all the clickable buttons and how a button looks when the cursor is hovered it. These buttons are primarily designed by CSS and Bootstrap, these are all the interactive buttons in this web application.

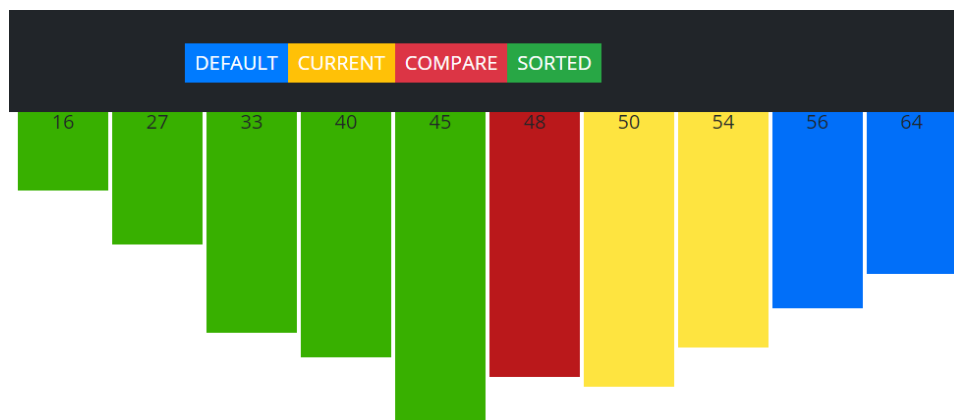


FIGURE 6.4 COLOR PALETTE

The above diagram shows the different colors used to highlight different elements during sorting so that the user can differentiate on what's going during the sorting algorithm. It represents the colors used on the graph where blue represents the default array, yellow represents the current element in which the user is on, red represents the elements which are compared and to be swapped and green represents the element which is in its sorted position.

<u>TIME COMPLEXITY</u>
BEST CASE
0
AVG CASE
0
WORST CASE
0
NUMBER OF SWAPS
0
COMPARISONS
0
TIME
0

FIGURE 6.4 TIME COMPLEXITIES

In the above diagram when an algorithm is clicked the time complexities along with the number of swaps, comparisons and time taken for the selected algorithm is displayed. "0" is the default value and when an algorithm is clicked the time complexities along with the number of swaps, number of comparisons and the time taken for an algorithm is displayed depending of the algorithm which we select.

CHAPTER 7

CONCLUSION

Relationship between concepts and their theory can be more easily and quickly understood by students using diagrams and visualization rather than in textual form. Lecturers can use more of visual tools such as this project to teach students by providing them with visual stimuli helping them grasp the concept just by watching the how the process takes place. The result presented here doesn't indicate that it outperforms black board learning but shows a different way of engaging students to learn not just by reading and writing but also by engaging them visually. This project mainly shows the visual representation of sorting algorithms but other topics on data structures such as trees, queues, linked lists, graphs etc. can be shown and taught visually for better understanding the topic.

REFERENCES

- [1] A. Kerren and J. T. Stasko. (2002) Chapter 1 Algorithm Animation. In: Diehl S. (eds) Software Visualization. Lecture Notes in Computer Science, vol 2269. Springer, Berlin, Heidelberg.
- [2] Hadi Sutopo, "SELECTION SORTING ALGORITHM VISUALIZATION USING FLASH", The International Journal of Multimedia & Its Applications (IJMA) Vol.3, No.1, February 2011
- [3] I. Reif and T. Orehovacki, "ViSA: Visualization of Sorting Algorithms", Conference: 35th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2012), Opatija, Croatia, May 21-25, 2012, Volume: Computers in Education
- [4] Clement Mihailescu, "Sorting visualizer", 2016, <https://www.youtube.com/watch?v=pFXYym4Wbkc&t=903s>
- [5] Faria, Brian, "Visualizing Sorting Algorithms" (2017). Honors Projects Overview. 127.
- [6] Slavomir Simonak, "Increasing the Engagement Level in Algorithms and Data Structures Course by Driving Algorithm Visualizations" September 2020, [Informatica](#) 44(3)
- [7] Hungarian (Küküllőmenti legényes) folk dance showing sorting by dance, <https://www.youtube.com/user/AlgoRythmics>

APPENDIX

1.PAPER ACCEPTANCE

2. PLAGIARISM REPORT

RESEARCH PAPER

ORIGINALITY REPORT

11 %	8 %	3 %	6 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	docplayer.net Internet Source	3 %
2	eprints.uthm.edu.my Internet Source	1 %
3	Submitted to American Intercontinental University Online Student Paper	1 %
4	Submitted to Bilkent University Student Paper	1 %
5	c-madeeasy.blogspot.com Internet Source	1 %
6	ru.scribd.com Internet Source	1 %
7	Submitted to Aston University Student Paper	<1 %
8	arxiv.org Internet Source	<1 %
9	Submitted to Derby College	

3. RESEARCH PAPER

SORTING ALGORITHM VISUALIZER

Kishore Kumar K, Vaibhav Kondabathini, Dr. B. Bharathi

Department of Computer Science, Sathyabama IST. kraisedtothree15@gmail.com

Department of Computer Science, Sathyabama IST. vaibhavkondabathini@gmail.com

Professor, School of Computing, Sathyabama IST. bharathi.cse@sathyabama.ac.in

ABSTRACT

With the recent surge in interest in computational thought, an important question has arisen: what are the best methods for teaching students basic computing concepts? Visualization is considered as one way to support student learning. With an aim to help and motivate students, number of researchers have come up with various tools. Although many tools are available this problem seems to be still persistent. This paper proposes a web application used to visualize seven commonly used sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and Tim Sort. The web application represents data in the form of a bar graph and user will be able to select the algorithm, control and speed of visualization and control the size of the bar graph. This paper also shows the Architectural design and technical structure of the application as well as its practical use and educational benefits are presented and discussed.

KEYWORDS: Visualization, Sorting, Algorithm.

I. INTRODUCTION

Visualization is proposed as one of the ways to contribute to student learning. Me and my partner are visual learners, and we are keener to learning concepts by seeing it get done, rather than sitting for hours reading about it and trying to understand the concept behind it. For example, when we were learning about sorting algorithms during the second semester Computer Science degree, we found it hard to grasp the working of a sorting algorithm just by seeing it explained on board or reading it as there is loads of recursion that takes place, we thought that maybe if these algorithms were taught visually and professors show how the data moves to its position, we would understand the concept better. That is what inspired us to come up with this project, we have developed a web-based application that animates how sorting algorithms work and it visualizes the process for us. If you want to organize a group of students in their height order, there are many algorithms that can perform this task. We visualized six of the commonly used sorting algorithms by representing the data as a bar graph. Each number is represented as a bar of different heights based on the value and even redundant or repeated data is also sorted out. The seven algorithms are: Bubble sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and Tim Sort. The main goal for us was for the animation to be web based so that a whole lot of people using different technologies can use it without having the trouble of installing special software or an application. The web application is coded with HTML5, CSS3 and JavaScript. The physical elements of the web application are coded via HTML5. The appearance I.e., the interactive layout of the web application is entirely done by CSS. Finally, the remaining is done by JavaScript, which is used for the histogram representation, movement

of bars, algorithm design.

II. RELATED WORK

There have been a lot of studies and animation tools on visualizing sorting algorithms. Some are based on how to make such an application and others on different technologies which were aimed for increasing the understanding of the concept. The work that we looked up and which also helped us throughout this project was Clement Mihailescu Algorithm Visualizer. Other visualization tools include ViSA, Selection sorting Algorithm Visualization Using Flash to name a few. These applications had a few drawbacks therefore, the approach we focused on was making the visualizer more interactive and fix the drawbacks that the other visualizing tools had.

The paper “Algorithm Animation” by A. Kerren et al [1], is a bit-by-bit guide to show the means and necessary coding methods to use sorting visualization. The paper mentioned about a web application-based tool which is outdated now, and the rest of the tools had us wanting to use other software's or applications.

The paper “SELECTION SORTING ALGORITHM VISUALIZATION USING FLASH” by Hadi Sutopo, The International Journal of Multimedia & Its Applications (IJMA) Vol.3, No.1, February 2011[2], discussed about only Selection Sort algorithm and visualizing it. The project was built using Adobe Flash and ActionScript programming tools and as of December 2020 Adobe Flash has been discontinued due to the rise security risk of using it.

In the paper “ViSA: Visualization of Sorting Algorithms” by I. Reif and T. Orehovacki , 2012[3] they provide an in-depth view of how sorting visually takes place on the data, but the limitation was that the data should be entered manually by the user which isn't effective if the user wants to enter large numbers of data.

In Clement Mihailescu's work, 2016 [4] not all the commonly used algorithms were implemented, the project was only half done as the user interface was not interactive and there wasn't any description about the sorting algorithms. In the paper “Visualizing Sorting Algorithms” by Brian Faria, 2017 [5] though the work only had four commonly used sorting algorithms the implementation by him on using different techniques like using sound to show the swapping of data was different from others, but there wasn't any way for the user to control the speed at which the visualization takes place nor the user was able to control the size of the array which in my opinion was needed for better understanding of the algorithms when the data size is huge.

In the paper “Increasing the Engagement Level in Algorithms and Data Structures Course by Driving Algorithm Visualizations” by Slavomir Simonak, 2020[6], here he uses Algomaster platform, and the project is developed by C programming language, thus the user-interface isn't as interactive as a one uses HTML, CSS and JavaScript because these tools are more into web application development thus, making the UI more interactive.

We've come across a number of sorting visualizer videos on YouTube. The algorithm visualization shown in these videos was done through dancing, which was a fascinating way of visually demonstrating how the data was sorted. These videos can be found on the AlgoRhythms YouTube channel. The algorithms aren't animated with any tools or graphics; instead, a group of artists dress up as the numbers. Bulgarian folk music will be played in the background, and the actors will form a line (or several lines, depending on the sorting algorithm) and then dance to demonstrate the comparisons and data swaps. The contrasts are represented by two dancers from the group stepping forward and momentarily standing next to each other as their values are compared. They would perform a fast dance move and swap to each other's position if they needed to swap after contrasting them. One of the features we liked was how they used a hat to indicate which index in the data was the pivot or was being compared to others. This inspired the concept of using different colors to represent different elements during the sorting visualization rather than just one.

III. PROPOSED SYSYTEM

The proposed system I.e. SAV consists of various features which makes this web application efficient for the user. The entire user interface is entirely done by front end technologies (HTML, CSS, Bootstrap and JavaScript). The user interface has 14 components: The main navbar allows the user to set the size of the array and to control the speed of visualization, the buttons to generate a new random array, to stop the visualization and the sorting algorithm buttons. Below it is the secondary navbar containing the color palette which shows the colors used during the visualization. Below it is the canvas which is divided into 3 parts: The algorithms description, the bar graph and the text box fields which show the time complexities, number of swaps, number of comparisons and the time taken for each algorithm whenever a particular algorithm is clicked. In the below figure 1 shows the home page of the web application.



Figure 1: Application Home page

The first row of the Navbar section has the range unit to set the size of the array with a minimum size of 10 and a maximum of 150 elements the wide range is mainly to show how these algorithms work for larger data visually as it's impossible to show students how an array with over 100 elements gets sorted but visually seeing it people can understand how they work.

The second row has the range controller to set the speed of visualization with the slowest speed at showing every swap in detail so that students can grasp the idea of what's going on and the fastest to show how different algorithms quickly sort an array at larger data, for example trying to sort an array of 100 elements by bubble sort can take a really long time as the slowest setting and can be time consuming but merge sort can do it in couple of seconds by this even students can get an idea of how algorithms differ between each other in speed. The third row has the buttons to generate new array and to stop the sorting midway.

The fourth row contains all buttons for the selectable algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort and Tim Sort by default no algorithm is selected, user needs to select the desired algorithm to see it visually. The fifth row is the color palette. It represents the colors used on the graph where blue represents the default array, yellow represents the current element in which the user is on, red represents the elements which are compared and to be swapped and green represents the element which is in its sorted position. The color palette is shown below in figure 2.

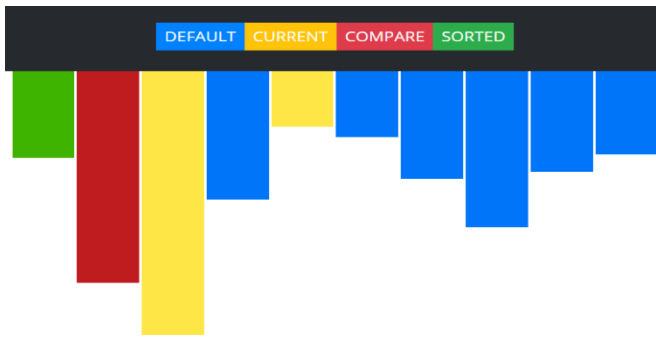


Figure 2: Color palette used to represent the elements

Below the navbar is the canvas section which is divided into 3 parts. The first part is a carousel slider gives a small description of each algorithm which we use in the visualizer, the second part is the bar graph representation itself, the third part shows the time complexities, number of swaps, number of comparisons and the time taken for each algorithm whenever a particular algorithm is clicked. The first column mainly gives detailed explanation on what going on each algorithm thus when a student wants to know what the algorithm is about, they can navigate through the carousel and read about the desired algorithm they want to know more about the idea behind it is that students can have read it and get an idea of what exactly is the sorting algorithm about so that when they visually see it, they get a better idea.

The second column contains the bar graph representation where the sorting takes places, initially the default bars are represented by blue color and when an algorithm is selected the sorting takes place and colors change accordingly. The third column is a table representing all the time complexities of the algorithms, number of swaps, comparisons as well as the time taken for each algorithm to execute which can be helpful for a someone who needs to know and compare which algorithm performs well in what conditions. We have kept user interface as simple as we could and grouped similar things together. When the cursor is hovered over any of the buttons, some characteristics appear. Figure 3 depicts the default button, Figure 4 depicts the button when hovered over it, and when an algorithm is selected, all components except the "stop sorting" button are frozen and cannot be selected until and unless the "stop sorting" button is selected (Figure 5).

The algorithm works by pushing the least element to the start of the array by swapping it with the first element of the array and the pointer pointing to the first element is moved to the next element and this process goes on till the last element of the array and eventually the array will be sorted. Figure 7 shows the pseudo code.

Insertion Sort is an in-place comparison sorting algorithm. In this algorithm, a sub array is kept which is always sorted. An element which needs to be inserted in

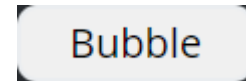


Figure 3: Default Button

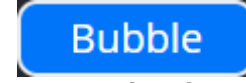


Figure 4: Selected Button

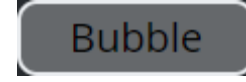


Figure 5: Frozen Button

A Sorting Algorithms main purpose is to reposition a given array or list of elements in ascending or descending order. Now taking about the algorithms itself we are using 7 sorting algorithms in this visualizer namely Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and Tim Sort. These algorithms are the commonly used algorithms for sorting. Bubble sort also known as sinking sort from time to time, is a straightforward sorting algorithm that iterates through the list, comparing adjacent elements and swapping them if they're out of order. The list is passed through again and again until it is sorted. But the algorithm is not suitable for amounts of data as its average complexity is of $O(n^2)$, Where n is the number of elements. The pseudocode of bubble sort is shown below in Figure 6

```
begin BubbleSort(list)
  for all elements of list
    if list[i] > list[i+1]
      swap(list[i],list[i+1])
    end if
  end for

  return list
end BubbleSort
```

Figure 6: BUBBLE SORT PSEUDO CODE

Selection sort is a sorting algorithm that uses in-place comparisons. It has an $O(n^2)$ time complexity, making it unsuitable for large arrays, and it performs worse in general than the equivalent insertion sort. Selection sort is well-known for its simplicity, and it outperforms more complex algorithms in some situations, especially when extra memory is limited.

```
begin SelectionSort(list)
  for all elements of list
    min = list[i]
    for j = i+1 to n
      if list[j] < list[min]
        min = j;
      end if
    end for
    if indexMin != i then
      swap list[min] and list[i]
    end if
  end for
end SelectionSort
```

Figure 7: SELECTION SORT PSEUDO CODE

this sub array, has to find its right position where it should be and then it will to be inserted in that

position. Hence, it's called Insertion Sort. The pseudo code of it is shown below in Figure 8

```
begin InsertionSort(list)
    for i = 1 to length(list):
        key = list[i]
        j = i-1

        while j >= 0 and key > list[j]
            list[j+1] = list[j]
            j = j-1
        end while

        list[j+1] = key
    end for
end Insertion Sort
```

Figure 8: INSERTION SORT PSEUDO CODE

Quick Sort is very efficient sorting algorithm except for its worst-case complexity and it is based on partitioning the array into two halves by taking a pivot element which can either be the first or last element of the array, the element on the left side of the pivot will be smaller than it and elements on the right side will be larger than it we do this partitioning recursively until the array is sorted. In the below Figure 9 the pseudocode is shown.

```
procedure quickSort(left, right)

    if right-left <= 0
        return
    else
        pivot = A[right]
        partition = partitionFunc(left, right, pivot)
        quickSort(left,partition-1)
        quickSort(partition+1,right)
    end if

end procedure
```

Figure 9: QUICK SORT PSEUDO CODE

Merge Sort is one the most efficient sorting algorithms among all the algorithms as even its worst-case time complexity being $O(n \log n)$. It follows the divide and conquer method as it divides the array into smaller sub arrays and compares these sub arrays and sorts them accordingly and merges these sorted sub arrays to get the final array. The pseudo code is shown below in Figure 10.

```
procedure MergeSort(list)
    if len(list) <= 1
        return
    mid = len(list)/2
    left = list[:mid]
    right = list[mid:]

    MergeSort(left)
    MergeSort(right)
    Merge_two_list(list,left,right)

procedure Merge_two_list(list,l,r)
    compares both array's l and r sorts
    them and merge's the array's
```

Figure 10: MERGE SORT PSEUDO CODE

Heap sort is a sorting algorithm that uses comparisons to sort data. Heap sort is similar to a more advanced version of selection sort in that it separates the array into a sorted and an unsorted array, then recursively reduces the unsorted array by inserting the highest element from the unsorted array into the sorted array. Unlike selection sort, heap sort does not waste time with a linear time scan of the unsorted array. Heap sort, on the other hand, preserves the unsorted array in a heap data structure so that the largest element in each recursion of the unsorted array can be found more easily. Below in figure 11 is the pseudo code of heap sort.

```
Heapify(A as array, n as int, i as int)
{
    max = i
    leftchild = 2i + 1
    rightchild = 2i + 2
    if (leftchild <= n) and (A[i] < A[leftchild])
        max = leftchild
    else
        max = i
    if (rightchild <= n) and (A[max] > A[rightchild])
        max = rightchild
    if (max != i)
        swap(A[i], A[max])
        Heapify(A, n, max)
}

Heapsort(A as array)
{
    n = length(A)
    for i = n/2 downto 1
        Heapify(A, n ,i)

    for i = n downto 2
        exchange A[1] with A[i]
        A.heapsize = A.heapsize - 1
        Heapify(A, i, 0)
}
```

Figure 11: HEAP SORT PSEUDO CODE

Tim sort is a stable mixed sorting algorithm that combines insertion and merge sort to perform extremely well on a wide range of real-world data. This hybrid algorithm was invented by Tim Peters, and it's the most commonly used algorithm in inbuilt sorting functions in programming languages like Python, Java and JavaScript. The algorithm detects pre sorted sub array of the data and uses them to sort the remaining array ore quickly. This is accomplished by combining runs until certain requirements are met.

- Initialize a RUN with the size of 32.
- Implement Insertion sort for RUN size chunks.
- A function `merge(int arr[], int l, int m, int r)` takes an array, left elements, middle of the array and right elements of the array as input. The function returns the merged sorted chunks size 32.
- Initialize length of the array having all the left elements and the length of the array having the right elements.
- After filling the left array and right array, Iterate over the left array as well as right array.
- If the element in the left array is less than the element in the right array, push the element into a larger array.
- Otherwise push the element into a smaller array accordingly.
- Copy the remaining element in the left array and right array into a larger array.
- A function `timSortAlgo(int arr[], int n)` takes an array and its size as input. Which calls insertion Sort initially and merges the array elements.
- Return the output as the final elements of the array using Tim Sort.

FIGURE 12: TIM SORT ALGORITHM

The project is implemented by a combination of HTML5, CSS3, Bootstrap (CSS framework) and JavaScript. It contains 3 files “index.html” which consists of the html code along with the scripts which links to JavaScript files, “style.css” which contains the user interface design and finally the “scripts” file which contains majority of the code that is the visualization and all the sorting algorithms. The code follows Model-View-Controller architecture. Below in Figure 13 the architecture diagram is shown.

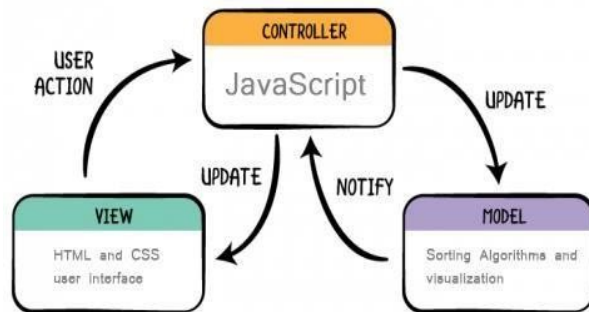


Figure 13: MVC ARCHITECTURE

The View is composed of main and visualization. These objects aren't connected to a particular object known as view, but they are JavaScript files which are under the scripts tags in the .html file. It acts as the “main” function where the entire code starts taking shape.

The main.js consists of HTML dom which is an object model for HTML and also acts as an API for JavaScript to add, change or remove HTML attributes, here we get to generate the random array, the array size and the speed of visualization from the .html file by the ID. A function `generate_array()` generates a random array including the interactive components of the web page and control the state of the Model as needed. Below in Figure

duplicate elements for the sorting to be more flexible. It is generated by the following snippet `“div_sizes[i]=Math.floor(Math.random()*0.5*(inp_as.max- inp_as.min))+10”` when a particular algorithm out of the seven is clicked in the main.js a dom element `addEventListener()` gets the algorithm which the user selected and calls the function `disable_buttons()` as the name suggests freezes all the buttons except the button to stop the sorting and the function `runalgo()` which contains a switch case and depending on the algorithm selected it calls the respective .js files which contains the sorting algorithm code. The Model consists of all the sorting algorithm code.

This object contains the codes organized in methods. The seven buttons of the sorting algorithm on the user interface of the home page directly controls this object. When the user clicks on one of the sorting algorithms the correlated sorting algorithm method is called and bar graph starts sorting.

The bar graph is attained from the id “array_container” from the .html file and stored in a variable “cont” via HTML DOM. Inside each of these sorting algorithm codes a variable is declared to assign the time complexities, swaps and comparisons are stored and is displayed in the homepage by the DOM.

In visualization.js a function `vis_speed()` contains a switch case for the visual speed of the algorithm the lowest being `speed=1` and the fastest being `speed = 100` we get the value from the id “array_speed” from the .html file via DOM. The next function is the `div_update()` this is where the bar graphs change in heights when they need to get swapped `“cont.style=“margin:0% “+ margin_size`

`+ “%; width:“ + (100/array_size-(2*margin_size)) + “%; height:“ + height + “%; background-color:“ + color + “;”;`. Here the heights of both the bar that need to swapped are passed as parameters in the function call of each algorithm and they are swapped accordingly by taking the difference in their heights and assigning them to one another. After each algorithm is done visualizing the `enable_button()` function is called which enables all the disabled buttons since when an algorithm is clicked all the buttons except for the “stop algorithm” button is disabled. An inbuilt API in JavaScript

called `performance.now()` calculates the time taken for each algorithm to run. All the interactive buttons on the web page are included in the controller.

When you click one of these buttons, a JavaScript method is called directly. The homepage, buttons, layout, colours, and other elements are

all styled using CSS. These methods

update 14 the entire code structure is shown.

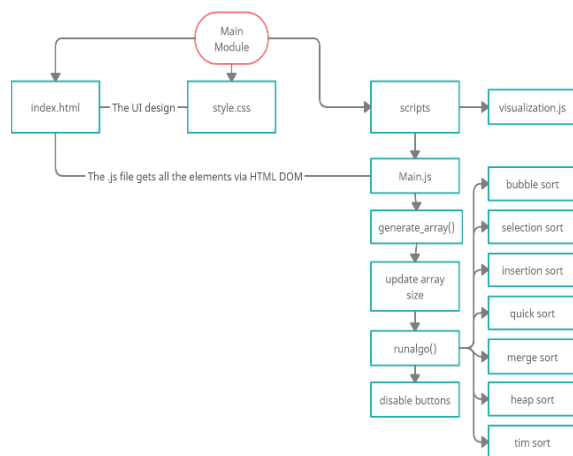


Figure 14: Internal Code Structure
IV. RESULTS

The main objective in doing this project was to help students who are new to the field of data structures and by making this sorting algorithms visualizer it further increased our own knowledge in the field of web development. The most challenging part of this project was trying to make the interface interactive with the user and making it easier to access as it is web based and doesn't require any external software or application to be installed, therefore a wider range of people can use it if they have an internet connection.

V. CONCLUSION

Relationship between concepts and their theory can be more easily and quickly understood by students using diagrams and visualization rather than in textual form. Lecturers can use more of visual tools such as this project to teach students by providing them with visual stimuli helping them grasp the concept just by watching the how the process takes place. The result presented here doesn't indicate that it outperforms black board learning but shows

a different way of engaging students to learn not just by reading and writing but also by engaging them visually. This project mainly shows the visual representation of sorting algorithms but other topics on data structures such as trees, queues, linked lists, graphs etc. can be shown and taught visually for better understanding the topic

REFERENCES

- [1] A. Kerren and J. T. Stasko. (2002) Chapter 1 Algorithm Animation. In: Diehl S.(eds) Software Visualization. Lecture Notes in Computer Science, vol 2269. Springer, Berlin, Heidelberg.
- [2] Hadi Sutopo, "SELECTION SORTING ALGORITHM VISUALIZATION USING FLASH", The International Journal of Multimedia & Its Applications (IJMA) Vol.3, No.1, February 2011
- [3] I. Reif and T. Orehovacki, "ViSA: Visualization of Sorting Algorithms", Conference: 35th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2012), Opatija, Croatia, May 21-25, 2012, Volume: Computers in Education
- [4] Clement Mihailescu, "Sorting visualizer", 2016, <https://www.youtube.com/watch?v=pFXYym4Wbkc&t=903s>
- [5] Faria, Brian, "Visualizing Sorting Algorithms" (2017). Honors Projects Overview. 127.
- [6] Slavomir Simonak, "Increasing the Engagement Level in Algorithms and Data Structures Course by Driving Algorithm Visualizations" September 2020, *Informatica* 44(3)
- [7] Hungarian (Küküllőmenti legényes) folk dance showing sorting by dance, <https://www.youtube.com/user/AlgoRythmics>

4.SOURCE CODE

main.js

```
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()
{
    let best = "0";
    let avg = "0";
    let worst = "0";
    var swaps = "0";
    var comp = "0";
    var time = "0";
    document.getElementById("best").value = best;
    document.getElementById("avg").value = avg;
    document.getElementById("worst").value = worst;
    document.getElementById("swap").value = swaps;
    document.getElementById("comp").value = comp;
    document.getElementById("time").value = time;
    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;
        if(array_size<40){
            divs[i]=document.createElement("article");
            /* divs[i].className = "i"; */
            /* divs[i].id = 'abc-'+i; */
            /* var text = document.getElementById('abc'-i); */
            divs[i].innerHTML = div_sizes[i];
        }
        else{
            divs[i]=document.createElement("div");
        }

        cont.appendChild(divs[i]);
    }
}
```

```

margin_size=0.2;

    /* to change color of the bars */
    divs[i].style=" margin:0% " + margin_size + "%; background-
color:#016ff9; width:" + (100/array_size-
(2*margin_size)) + "%; height:" + (div_sizes[i]) + "%;";
    }
}
function update_array_size()
{
    array_size=inp_as.value;
    generate_array();
}
window.onload=update_array_size();
//Running the appropriate
algorithm. for(var
i=0;i<butts_algos.length;i++)
{
    butts_algos[i].addEventListener("click",runalgo);
}
function disable_buttons()
{
    for(var i=0;i<butts_algos.length;i++)
    {
        butts_algos[i].classList=[];
        butts_algos[i].classList.add("butt_locked");

        butts_algos[i].disabled=true;
        inp_as.disabled=true;
        inp_gen.disabled=true;
        inp_aspeed.disabled=true;
    }
}
function runalgo()
{
    disable_buttons();

    this.classList.add("butt_selected");
    switch(this.innerHTML)
    {
        case "Bubble":Bubble();
            break;
        case "Selection":Selection_sort();
            break;
        case "Insertion":Insertion();
            break;
        case "Merge":Merge();
            break;
    }
}

```

```

        case "Quick":Quick();
            break;
        case "Heap":Heap();
            break;
        case "Tim Sort":tim();
            break;
    }
}

```

visualization.js

```

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.
function div_update(cont,height,color)
{
    window.setTimeout(function(){
        cont.style=" margin:0% " + margin_size + "%; width:" + (100/array_size-
(2*margin_size)) + "%; height:" + height + "%; background-color:" + color + ";";
        },c_delay+=delay_time);
    }
function enable_buttons()
{
    window.setTimeout(function(){

```

```
for(var i=0;i<butts_algos.length;i++)
{
    butts_algos[i].classList=[];
    butts_algos[i].classList.add("butt_unselected");
    butts_algos[i].disabled=false;
    inp_as.disabled=false;
    inp_gen.disabled=false;
    inp_aspeed.disabled=false;
}
},c_delay+=delay_time);
}
```