# National University of Computer & Emerging Sciences

**Karachi Campus**



**Sorting Algorithms Visualizer**

**Design and Analysis of algorithms**

**Department: Software Engineering**

**Section: A**

|  |  |
| --- | --- |
| *Course Code* | *CS-302* |
| *Instructor* | ***Dr. Waheed Ahmed Abro*** |
| *Project Team Members* | *Maryam Raheem (20K-1700)*  *Samia Azeem (20K-1685)* |
| *Submission Date* | *12/12/22* |

**SORTING ALGORITHM VISUALIZER**

# **ABSTRACT :**

This project tested the benefits of animated sorting algorithms for teaching. To visualize multiple sorting algorithms, a web-based animation application was constructed. A visualization of data is implemented as a bar graph, after which a data sorting and algorithm may be applied. The resulting animation is then performed, who then sets their own speed. This is research on the computer science curriculum's approach to learning algorithms. The experiment featured a presentation and a survey, both of which asked students questions which may illustrate improvements in algorithm comprehension. These findings and reactions are cataloged in this project and compared to earlier investigations.

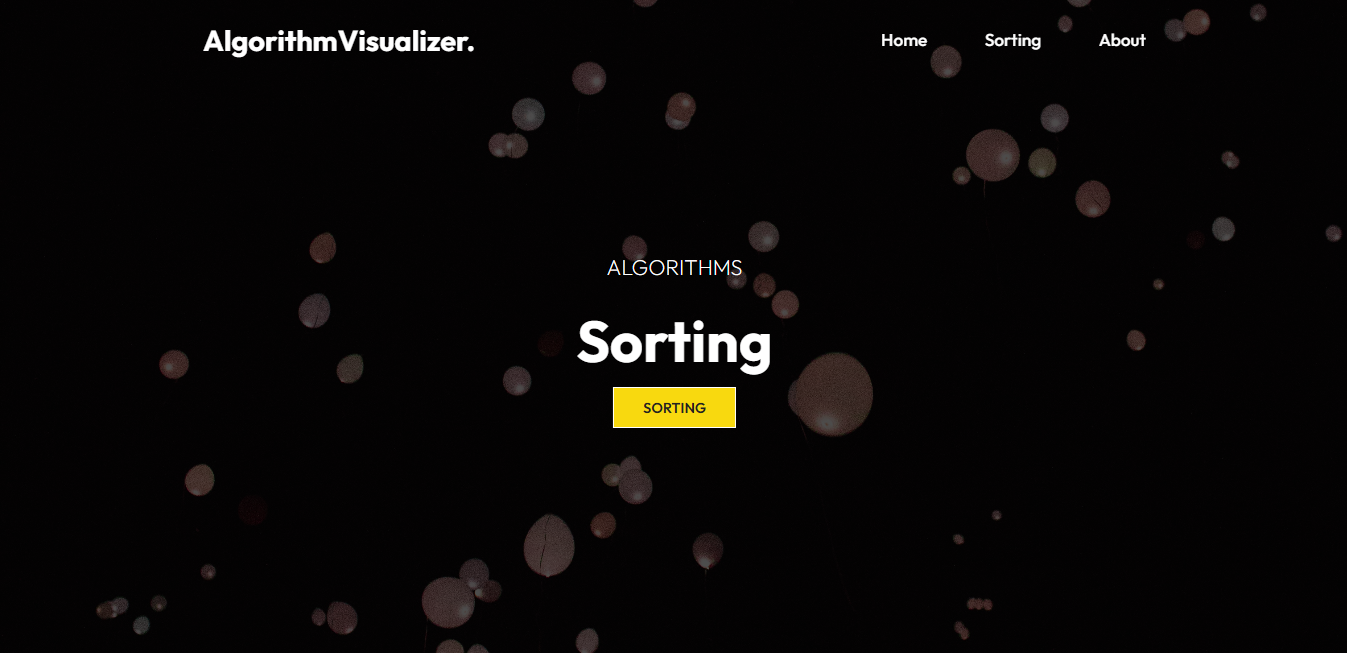
# **INTRODUCTION**

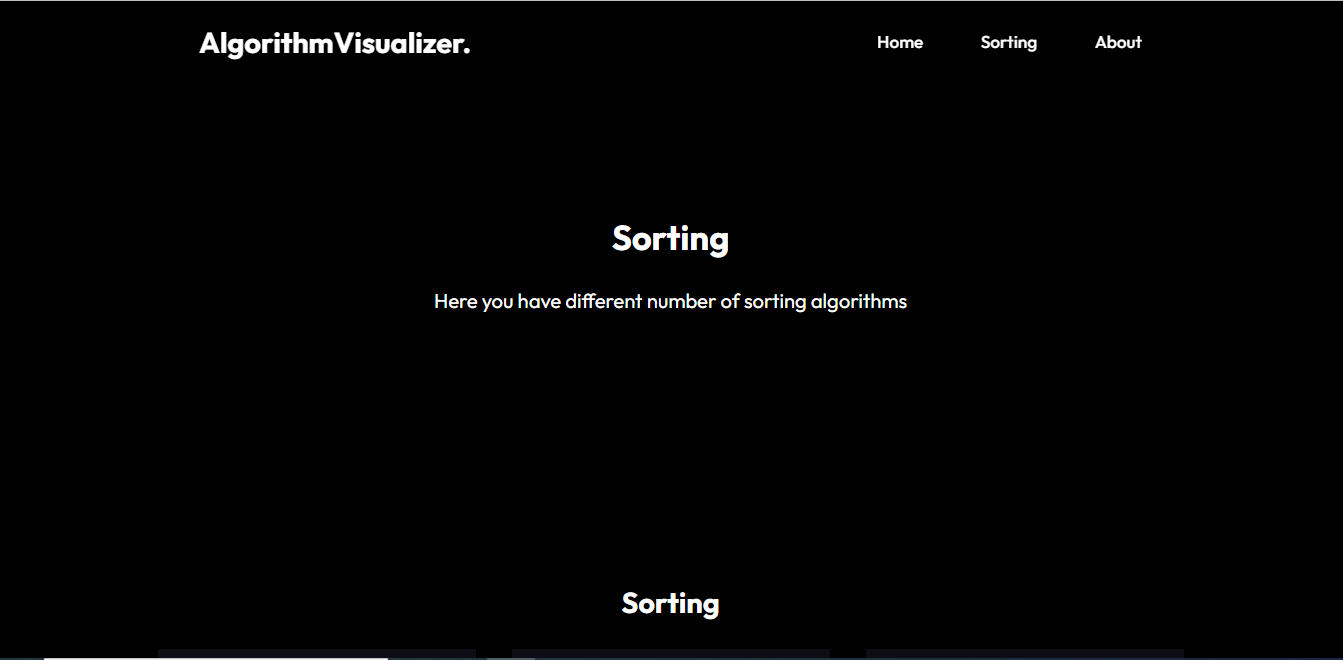
I built a project that explains how sorting algorithms transform and organize sets of data. It is possible to organize a list of people, for example, by their age in ascending order using different methods. To aid my visualization, I created a histogram of numerical data to represent well-known examples. Each number is depicted as a bar, and the height of each bar represents the value of that number. It is being shifted by the algorithm from its original, unordered location to its final ordered place, making it distinct from the rest of the data making it color pink. Radix Sort, Bubble Sort, Insertion Sort, and Merge Sort are some of the sorting algorithms. 2 Let's imagine that you have printed each person's age on a separate index card. Bring the youngest card to the front and then sort the cards by age. To discover the next smallest item, identify the age that has already been ordered and position it behind the already ordered age. Index cards full of ages will be at the end of the pile. Insertion Sort works in the same way as this. In this case, to sort a set of data, you select the smallest first, and then the next smallest, and so on until you've sorted all of the data. This technique is quite simple to explain to someone in conversation, but more advanced sorting algorithms, such as Quick Sort, which requires the data to be moved around a pivot point, are not easy to grasp using text alone. I wanted the animation to appeal to a wide spectrum of individuals utilizing various technology media, and so I had it made in a web-based format. Instead of requiring the user to install extra software or attempt to organize setups to use the tool, this helps to remove this source of anxiety. It uses HTML5 (Hypertext Markup Text Language) JavaScript using framework reactJS, and CSS for the website's layout (Cascading Style Sheets).

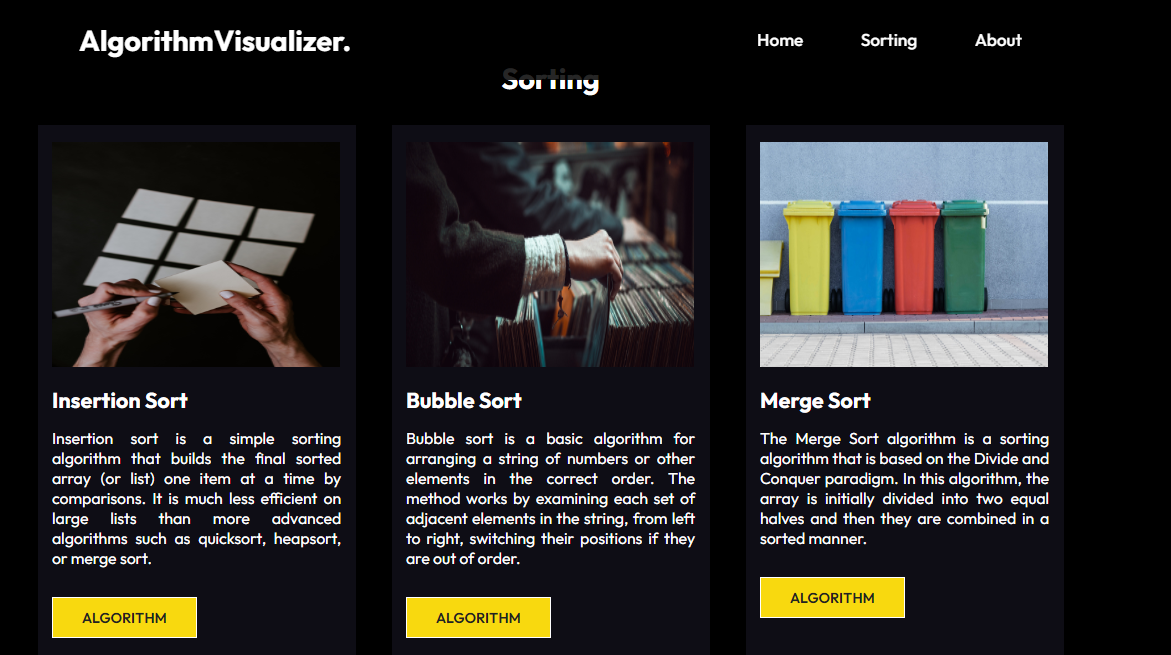
# **DESIGN**

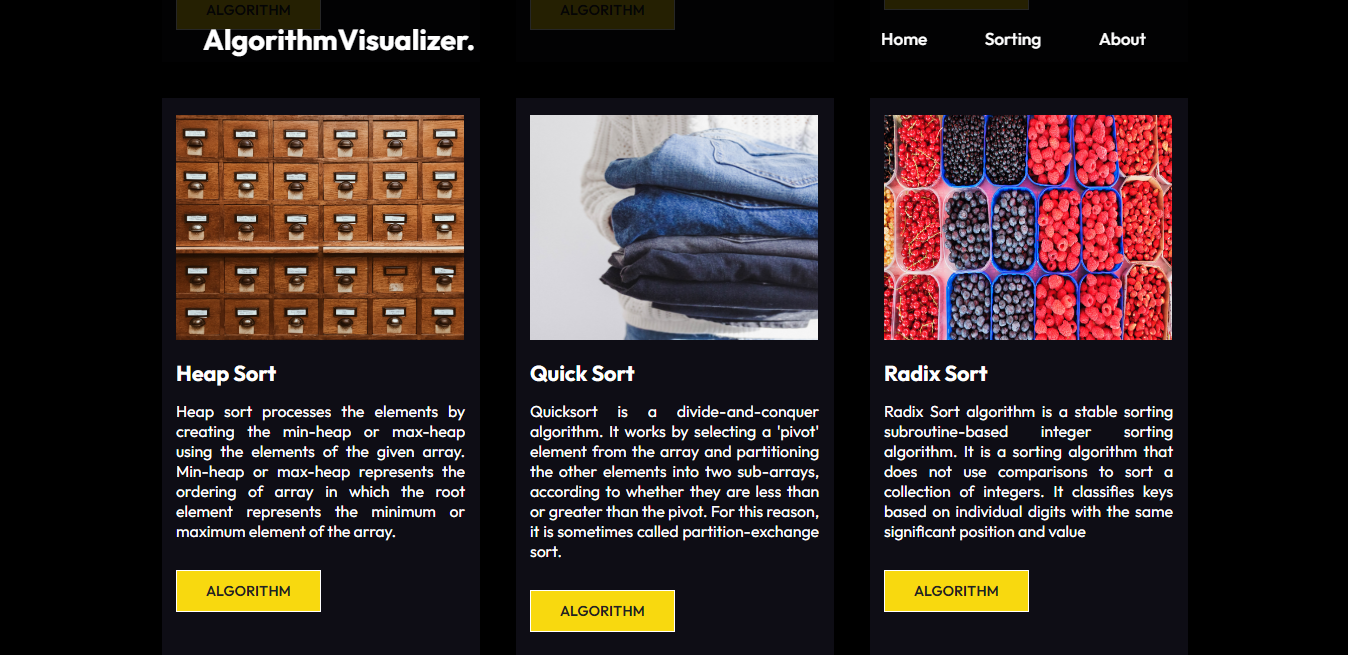
## **a) The User Interface**

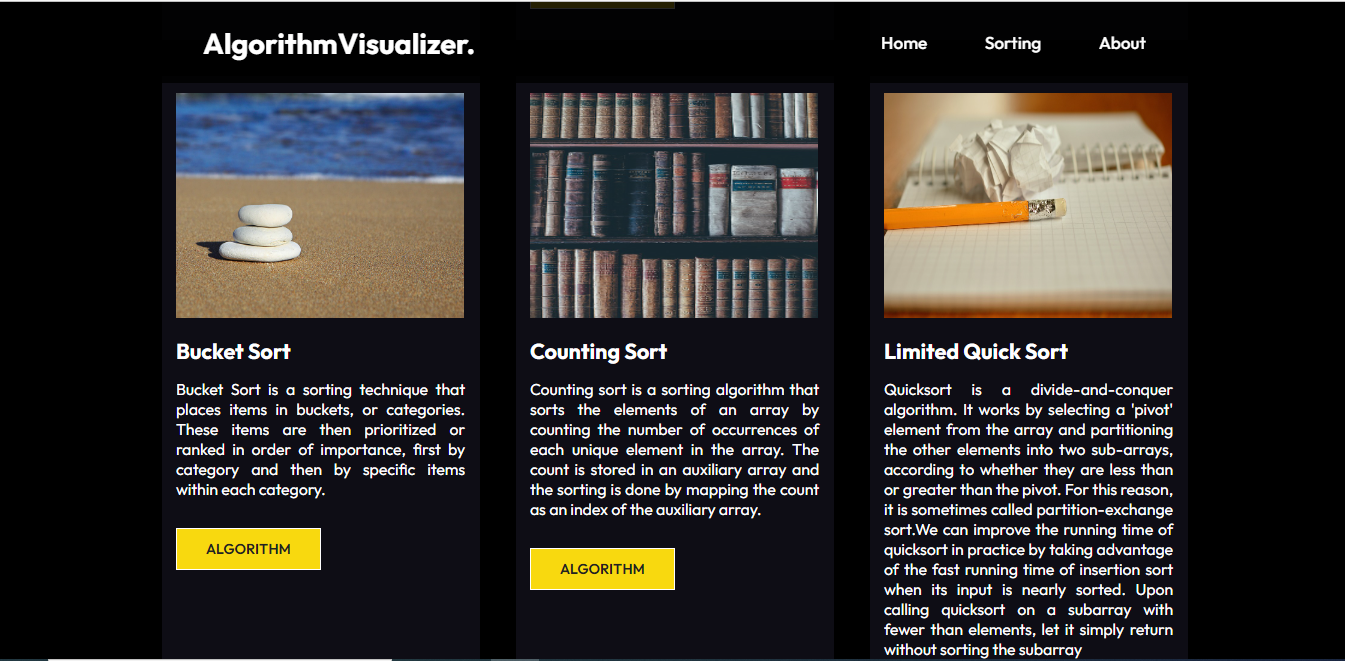
Each component has its own feature: The canvas has twelve features; control buttons, and a speed toggle bar. The canvas area is where the sorting algorithms are visualized, and that area will be the location where the sorting algorithms' output is edited. The top of the canvas are the selectable algorithms: Radix Sort, Bubble Sort, Insertion Sort, and Merge/Insertion Sort via drop down. This type of visualization is offered to users to select an algorithm of their choice, and to observe how that algorithm functions. Before launching the animation, the user will need to select an algorithm. The sorting algorithm must be selected before the input data type is specified. To choose between sorting input data that is already in order or to reverse and randomize the order, the three gray-bordered buttons on the left of the bottom row are available (shown in Figures 2 and 3 in the following paragraphs). The sorting algorithm is picked once the input and sorting method have been selected. Following, the “Sort” button in the bottom of the window is clicked to perform the sort from the beginning to the end.

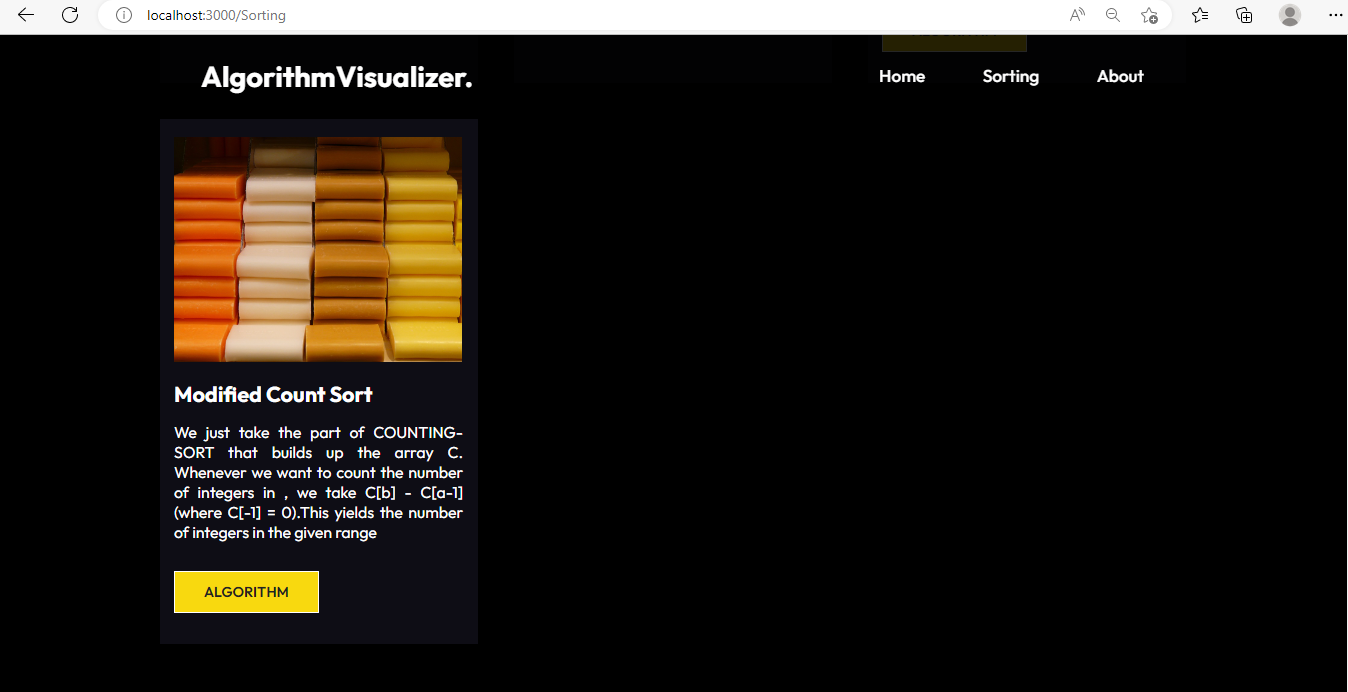


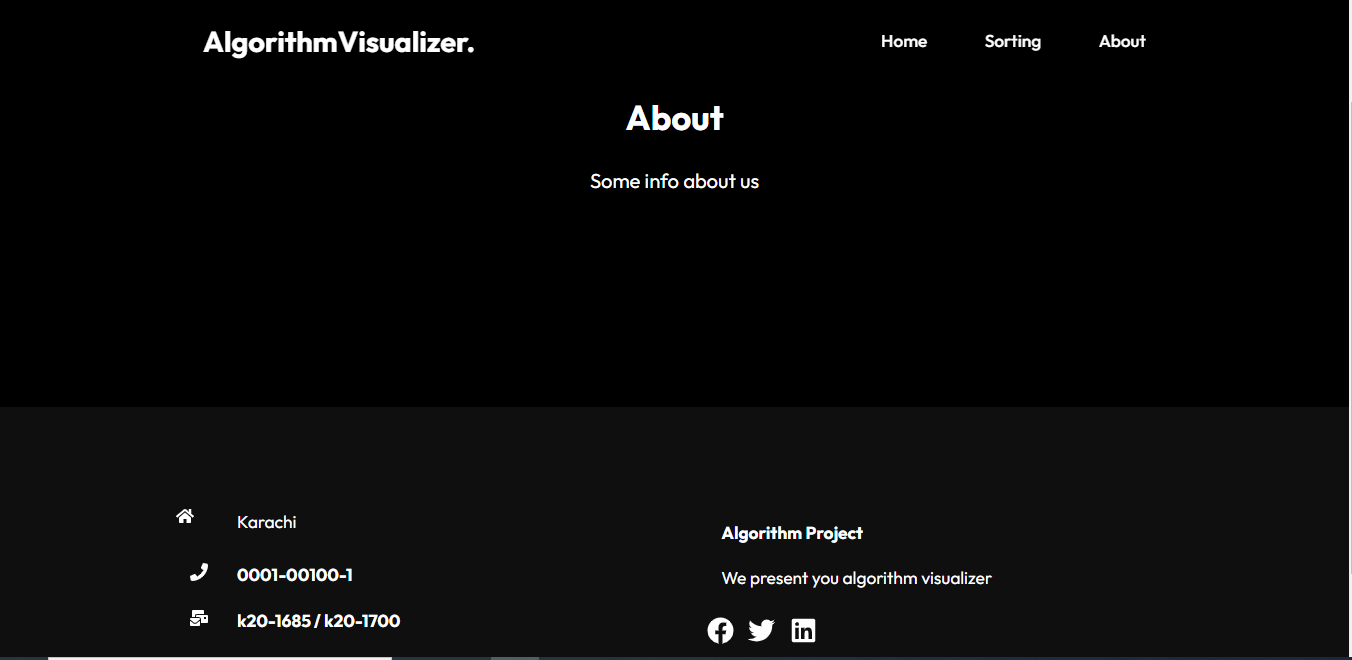












**Figure 1:** Main Window of Application

**b) System Architecture**

HTML, CSS, and JavaScript make up the back-end code. We use React JS as Framework that helps us to make different files for each algorithm that makes the space friendly.

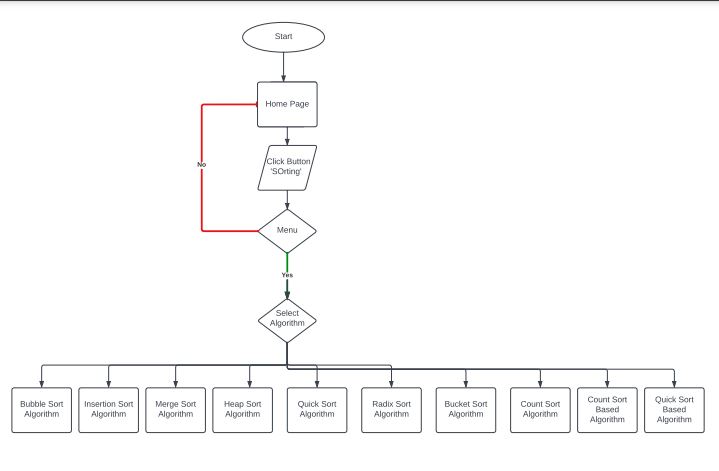
As you can see, the three coding languages are the only important components. However, since React runs immediately in the browser, it is unnecessary to employ a server on the back-end (like PHP). HTML5 and CSS are employed in web development. As illustrated with a single, bidirectional arrow, the HTML5 and JavaScript communicate to run the relevant algorithms and update the interface. The code for HTML5 and CSS did not change significantly throughout the project. The parts of HTML5 that were updated were the function calls for each button, since they were altered from a functional programming mindset to an object-oriented one. We've abstracted away all of the back-end code behind all of the different algorithms and animation selectors.

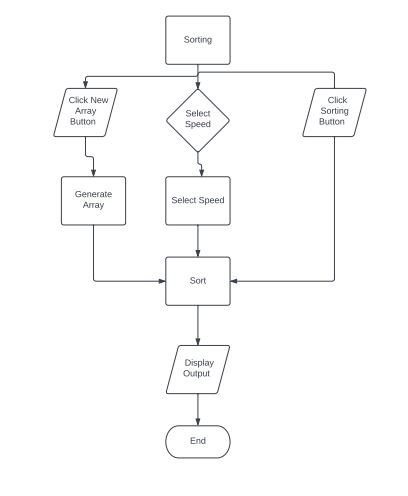
# **IMPLEMENTATION**

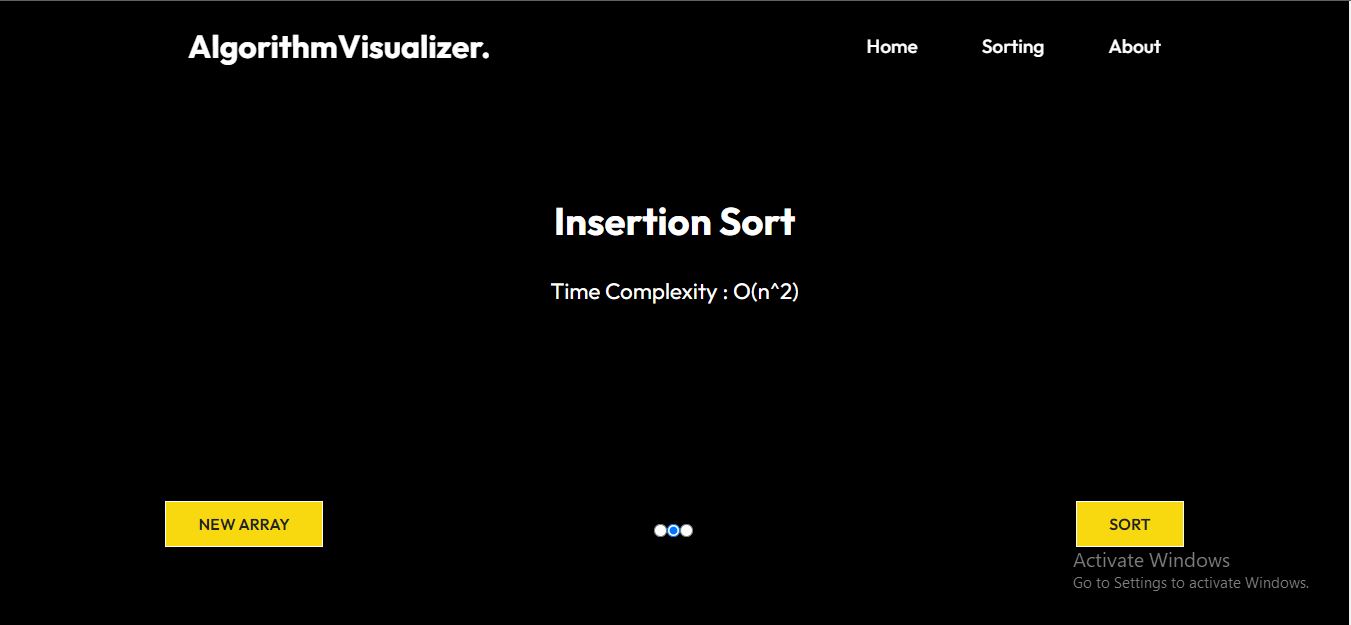
The use of HTML5 (Hypertext Markup Language 5), JavaScript, React and CSS combine to form this project's implementation (Cascading Style Sheets). There is only one project file which is an HTML file and contains the code.

This software uses both object-oriented and functional programming paradigms in how it organizes the code. Before the final phase of development, the design was almost completely functional, where only three objects were used: one to control the canvas that displayed the animation, another to represent a piece of data, or “bar” object (gray rectangle with dynamically changing height and position), and a final one to represent the positions that each bar moved. Although this incorporated several function calls, some instance variables and Boolean values were utilized to keep track of the algorithm picked and when to animation. The major module in the HTML code between the <script> and </script> tags is known as the global scope. Everything within the framework is able to access the aforementioned variables and methods.

**Flowchart:**

****

****



**Figure 3:** View of Insertion Sorting

Let's give an example: When sorting the pieces of data using Selection Sort, each piece of data is moved to its final and accurate location after one step, whilst the others require numerous steps to get to their final positions. While this sorting method appears to do the most effort compared to other sorting methods, it finishes sorting the most slowly. As a result, the visualization doesn't provide the correct visual impression of the data comparisons, which is one of the most important aspects of sorting algorithms. Two-dimensional arrays do demand more memory than a one-dimensional array. The size of the array is based on the number of steps that are required to sort the data. We may assess the algorithm's space needs by examining how long it takes. In Computer Science, using Big-Oh analysis is the standard way for determining how long something will take. The notation consists of a capital letter O, which represents the worst-case performance of the algorithm in question, followed by a constraint in parentheses that describes the worst-case performance of the algorithm.

When O(n2) is calculated, it is the selection sort and insertion sort's Big-Oh, meaning the time complexity grows at a rate proportional to n2, or the number of items in the collection of data squared. Space required for the steps array may or may not be the same as the space needed for the steps array. Because there are 32 pieces of data, the array is 32 cells wide, but the number of rows varies. The steps array's size relies on a number of factors, including the sorting method and which points from the algorithm are shown to the user. For example, points could be tracked whenever an item is moved, or when an item gets where it is supposed to be.

**UNIT TESTING**

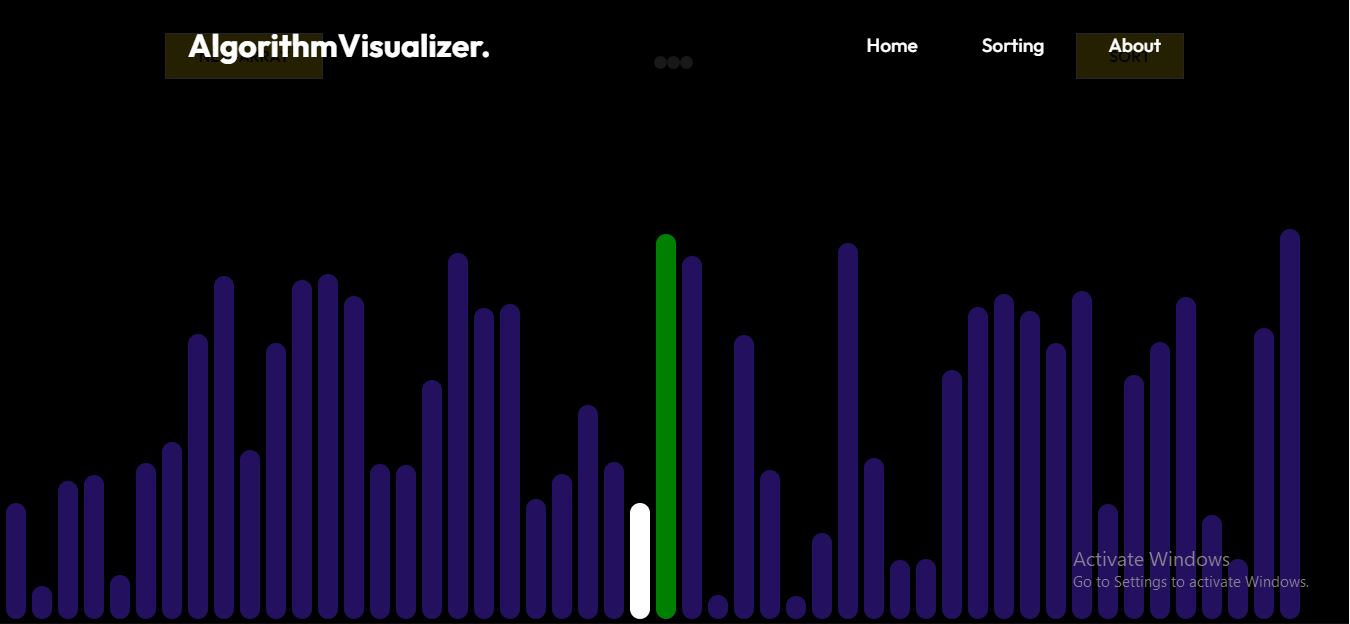
For testing Algorithms we use Random Array Generator every time you click the New Array button on any Sorting page.

## **a) Results**

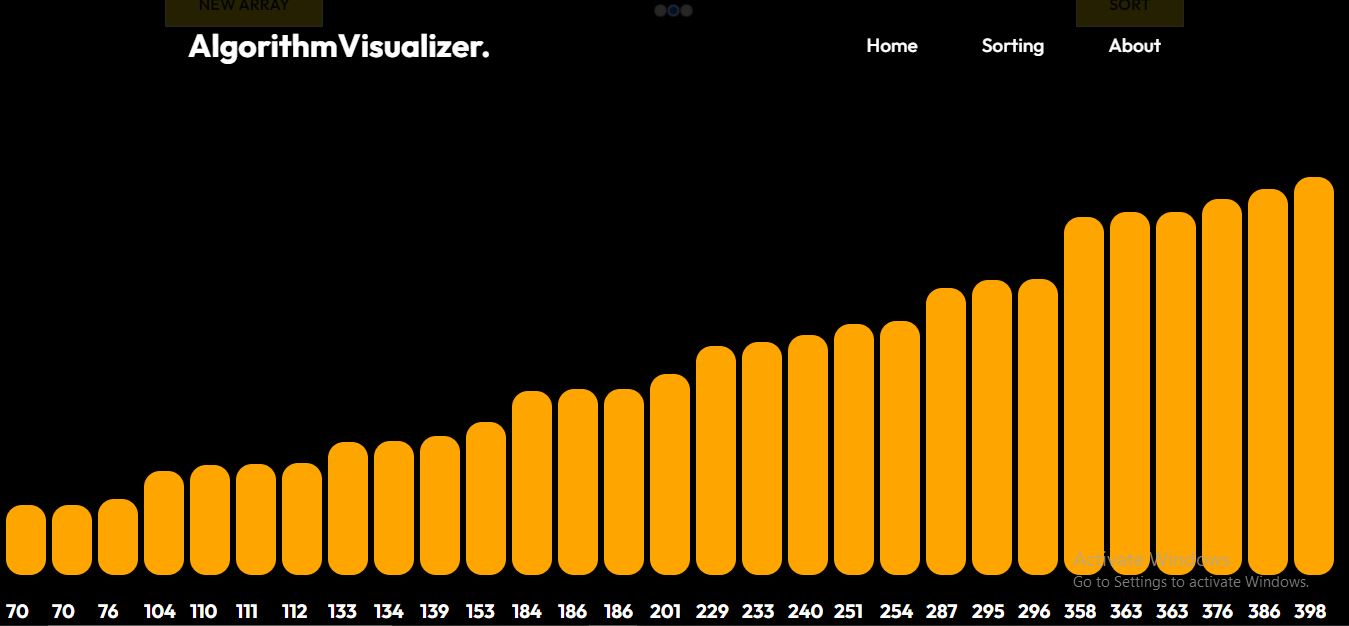
Start by arranging the data, and then pick the visualization algorithm to use. Algorithm buttons provide sorting of data as it arrives on the interface. Asking to specify the ordering of elements takes precedence because when the algorithm has completed running the initialization process, the interface is now showing a new ordering, while the code has already completed running the initialization with the prior data set. The animation shows how sorting work according to their specifics way and direct way for linear time complexities Algorithms.

****

**Figure 5.1:** Generate New Array

****

**Figure 5.2:** View of how Sorting Works

****

**Figure 5:** View after the array is sorted

A shortcoming of the animation is that it does not provide comparisons of the data's motions that result in such movements. Selection Sort's performance advantage over the other sorting algorithms is due to the fact that there are O(n) swaps, which eliminates superfluous computer movements. Comparing the data produces a runtime complexity of O(n squared) (the slowest overall). In response to question 5, where students were asked for input and thoughts, another student stated that Merge Sort is the best of the four kinds. The average runtime of Merge Sort is O(n log2 n), which is the best average runtime among all sorting algorithms. Integrating visualization of comparisons as well as motions would help fix this. A good technique to accomplish this is to use an algorithm that highlights the bars in red when it is examining data, requiring additional time in the animation. The following sorting algorithms, Selection Sort and Bubble Sort, would require a considerable amount of comparisons in order to finish.

# **CONCLUSION**

We can conclude that this algorithm visualizer contains all the necessities required by any person working on Algorithms that will help them do their work efficiently and effectively.

This web-based animation tool for viewing the following sorting algorithms functions in great part because of all the time and effort that I invested into it. In spite of its memory overhead, the feedback given to it was mostly good from the students that worked with it. Finally, I would make the online tool available to the public, with the feature I want most, which is to make it available to the public. This might be tough as well. The application that created the animation tool knows that it's available locally, but because of concurrency, it can serve numerous requests to the web site by separate users. As I try to figure out how to make the code as efficient as possible, I'd need to spend some time thinking about how to make it work with numerous people using it. This would be excellent, as it would enable a form of comparison study.

# **REFERENCES**

1. T. Bingmann. “The Sound of Sorting - ‘Audibilization’ and Visualization of Sorting Algorithms.” Panthemanet Weblog. Impressum, 22 May 2013. Web. 29 Mar. 2017.
2. <http://panthema.net/2013/sound-of-sorting/>.
3. Bubble-sort with Hungarian (“Cs´ang´o”) Folk Dance. Dir. K´atai Zolt´an and T´oth L´aszl´o. YouTube. Sapientia University, 29 Mar. 2011. Web. 29 Mar.2017.
4. <https://www.youtube.com/watch?v=lyZQPjUT 5B4> .
5. 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.
6. <http://homepage .lnu.se/staff/akemsi/pubs/22690001.pdf>.
7. A. Moreno, E. Sutinen, R. Bednarik, and N. Myller. Conflictive animations as engaging learning tools. Proceedings of the Koli Calling ’07 Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88, Koli ‘07 (Koli National Park, Finland), pages 203-206.