e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:03/Issue:07/July-2021 Impact Factor- 5.354 www.irjmets.com

SORTING ALGORITHM VISUALIZER

Shubham Nath[*1], Jatin Gupta[*2], Abhinav Gupta[*3], Teena Verma[*4]
[*1,2,3]Student, Computer Science And Engineering, HMRITM, New Delhi, Delhi, India.
[*4]Assistant Professor, Computer Science And Engineering, HMRITM, New Delhi, Delhi, India.

ABSTRACT

This paper outlines a study that tested the benefits of animated sorting algorithms for teaching. To visualize four 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 either automatically or by the user, who then sets their own pace. This is a 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 catalogued in this document and compared to earlier investigations.

Keywords: Sorting Algorithms, React Visualizer, Selection Sort, Merge Sort, Bubble Sort, Insertion Sort, Heap Sort.

I. INTRODUCTION

How do you get something done? You don't have to be extremely complex in solving the problem, for example, if your car's headlight is broken (although nowadays, manufacturers are trying the patience of the community with their increasingly abstract, space-age designs). The main issue is figuring out the best way to go about it. To locate step-by-step directions in your car's handbook, you conduct research, or do you use instinct to find someone who knows how to do it? In short, my instinct tells me that I am a visual learner and hence more suited to acquire topics by watching them than by reading about them. In this case, I found that seeing the data move to its rightful spot as the result of an algorithm is MUCH easier to follow than looking at the source code and trying to figure out where the data was supposed to go. My project was born out of my curiosity about sorting algorithms, which inspired the idea for this paper, which details an online tool I built 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 four of the 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. Selection Sort, Bubble Sort, Insertion Sort, and Merge Sort are the four 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. Selection 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, and CSS for the website's layout (Cascading Style Sheets).

II. DESIGN

a) The User Interface

The design and structure of the user interface components has remained unchanged even if the underlying back-end code was refactored midway through the construction. Each component has its own feature: The canvas has twelve features; 10 control buttons, and a volume toggle button. The canvas area is where the four

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:03/Issue:07/July-2021 Impact Factor- 5.354 www.irjmets.com

sorting algorithms are visualized, and that area will be the location where the sorting algorithms' output is edited in. The first row of four blue-bordered buttons at the bottom of the canvas are the selectable algorithms: Selection Sort, Bubble Sort, Insertion Sort, and Merge/Insertion Sort. 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 (as shown in Figure 1 on the previous page) 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). Sorted order is the default. The sorting algorithm is picked once the input and sorting method have been selected. Following, the “Start” button in the next row of buttons is clicked to perform the sort from the beginning to the end. The user can click the yellow-orange-bordered “Step” button to watch the algorithm step-by-step. Once the process is already underway, you can simply stop it by pressing the “Stop” button. In order to reduce misunderstanding in user-friendliness, I've tried to make the interface basic and placed buttons related to one another for easy access. The algorithm buttons are on separate tiers and have a blue coloration associated with them. Additional buttons are arranged in a grouped fashion, with a gray background. The "oddballs" are the animation controls. Despite being grouped together, each controller has a distinct color to denote the kind of animation it does. The colors are modeled on a traffic light with green being the go signal, red being the stop signal, and yellow being the signal to slow down (or in this case, yellow-orange means pace yourself). Additionally, each button also provides visual feedback to the user by changing color as the cursor hovers over it. The volume toggle button is the final function that is available on the Web page. The button appears to the left of a speaker picture on the lower left-hand side of the web page. While conducting some study on sound effects for animations, I realized the tool could be more participatory by both hearing and seeing the animation, rather than merely observing. To determine the sound each bar makes, use the following rule: each bar has a sound allocated to it based on its height. To hear four octaves on a piano, from low to high, the bars need to be in order. With the sound enabled, each bar in the sound animation plays a different tone from left to right when it appears. You'll only be able to hear the full four octaves in order if the bars are out of order. The bar's presently being played color will change to green, and then, when the sound animation is finished, it will change back to blue. However, because the animation is hungry with memory, the animation may stop momentarily and then resume. This means that the sound animation is turned off by default, but the user has the option to toggle it. You'll get the best results if you use the sound animation with Selection Sort.

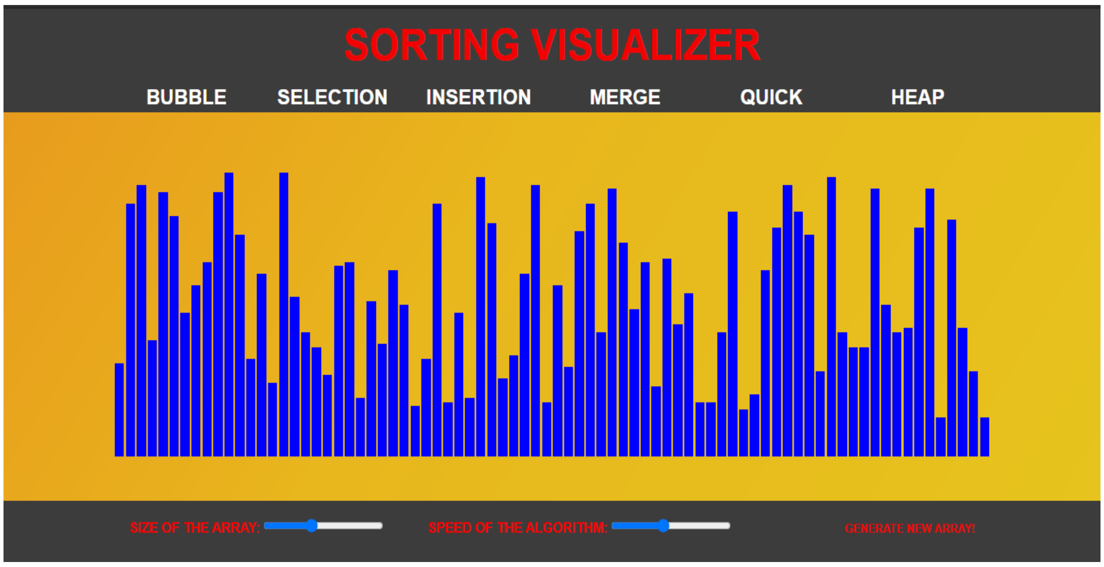

Figure 1: Main Window of Application

Finally, there is one feature that you will discover only after making a selection and attempting to sort: you may modify the sorting algorithm on the fly. In other words, you can pause an animation while using a sorting algorithm like Selection Sort, then choose another algorithm, like Bubble sort, while the animation is paused. To continue sorting the bars, start the animation by clicking "Start" or "Stop." How the movement of data changes based on what algorithms are already affecting it is remarkable to behold. Also, sorting semi-sorted data using the provided options provides a unique view on how the algorithms function.

b) System Architecture

HTML5, CSS, and JavaScript make up the back-end code. There are three varieties of code in one .html file and all three can be executed from this file alone. Including different types of web languages in a single page is one of the shortcomings of HTML 5. Since, therefore, there were three different types, each had been segregated, producing three different files (plus the miscellaneous sound and image files). Readability and keeping relevant code together are benefits of excellent programming practices. However, in the end, I opted not to break the code into two separate sections because of these two reasons. By just having to worry about one project file instead of three, the project may be more easily transported and sent. And because the changes to the coding languages are identified unambiguously in the project file, they do not reduce readability. An RIA can have more than one programming language in a single file (Rich Internet Application).

As you can see, the three coding languages are the only important components. However, since JavaScript 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.

III. IMPLEMENTATION

The use of HTML5 (Hypertext Markup Language 5), JavaScript, 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. The only additional piece of code added to the main HTML file is the .m4a sound files to support the sound animation functionality (which are saved as .m4a files). As of now, I only did extensive testing using the Mozilla Firefox browser, and it's the browser of choice in this context. However, tests done quickly revealed the possibility of Google Chrome and Safari integration.

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 (blue rectangle with dynamically changing height and position), and a final one to represent the positions that each bar moved to, or "pos" objects. 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, but this led to a greatly integrated mass of code that was difficult to maintain. Several big refactorings later, the code has now taken on the form of a Model-View-Controller Architecture. Although, because of its functional character, it possesses a multitude of individualized functions that alter the instance variables and Boolean values, which means it has a multitude of functions that directly alter the View and Controller. 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.

a) The View

There are three items on the view: the sortArea, the bar, and the position. These objects operate within the container defined by the <script> and </script> tags in the .html file. This function's space is sometimes referred to as the "main" function, the first function invoked in a program.

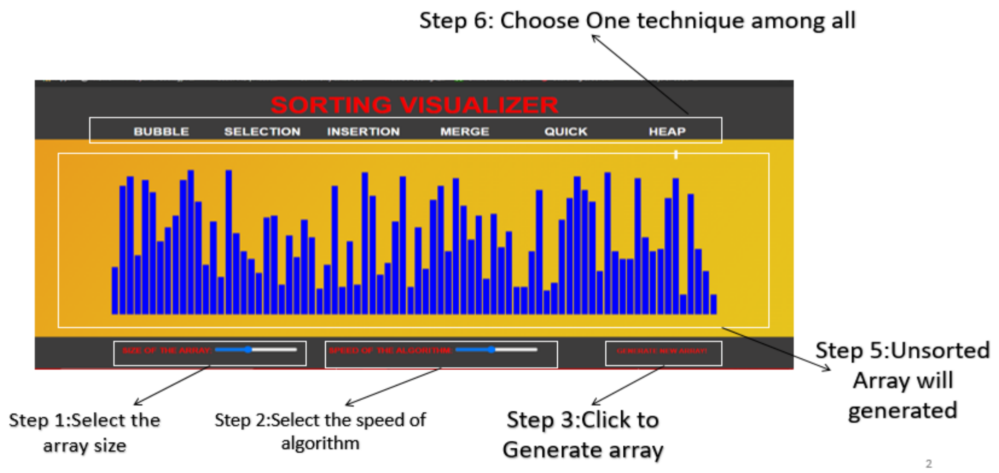
Step 6: Choose One technique among all
SORTING VISUALIZER
BUBBLE
SELECTION
INSERTION
MERGE
QUICK
HEAP
Step 5:Unsorted Array will generated
Step 1:Select the array size
Step 2:Select the speed of algorithm
Step 3:Click to Generate array

Figure 2: View and explanation of the application

It is the sortArea that keeps the bars up to date using a timer, while at the same time generating the bar graph. As a result, whenever "Step" is invoked, the bar values are updated depending on the steps array (discussed later). In the sortArea, after every second iteration of the timer, the rectangles will be redrawn with varied heights that represent the new values. The bars change sixty times per second, so when the “Step” button is selected, the change is instantaneous. In the sortArea, the bar object represents each piece of data. The statement encompasses all of the aspects of color, value, location, height, and sound. While having a distinct array named bars for the current bars in the bar graph helps preserve attributes such as the total number of bars (total value) independent of other characteristics, it is simple to update any or all of the attributes by iterating over the array as necessary. To update the bars to a fresh data configuration, the In Order, Reverse, and Random buttons iterate fast across the array. The bar object is related to the pos object (which is short for position). The region on the canvas that is updated when the SortArea event fires is an x-y pixel coordinate grid. This item was made to make arranging the bars a little bit simpler (1-32). Thus, if I wanted to move a bar, I would supply merely the number of the bar's location. In order for the bar to move, it will first determine the exact coordinates and then go to that location. Another way to say this is to say that, position one is defined by the two-dimensional coordinate pair (9, 135), which is the bottom left-hand corner of the bar. As long as each bar has a rectangular object that is associated with it, as well as a top left-hand corner point that defines the rectangle's height, the bar must be relocated to its right location.

b) The Model

The model is made up of one item, known as the sorter. This object houses the algorithm's code divided into methods. Start method centralizes on an integer constant and uses it to order the algorithm's possible algorithms. This object is directly controlled by the four sorting algorithms shown on the user interface as "Selection Sort, Bubble Sort, Insertion Sort, and Merge/Insertion Sort." The sort algorithm method is invoked as the user selects a sorting technique and clicks on one of the sort algorithm buttons. Then, when the algorithm sorts the data, a trace is created. The steps array, which contains all the movements in the animation, is a two-dimensional integer array that is available to methods on the web user interface. A typical back-end code interface is implemented using the steps array. The “Start,” “Stop,” and “Step” buttons function as controllers for which sub-array will be displayed on the canvas, after the computational back-end has completed the tracing. If this loads before the user has selected "Start" or "Step", then this represents a user action. When you click the “Step” button, the next step on the canvas loads and animates. An animated sequence of a single bar moving across the others is produced because the timer continually redraws the bars. There's no “Start” button, only a “Step” button that is set to go off on a timer. Using a two-dimensional array gives you the option to view the sorting algorithm's stages within the View. The process of adding an algorithm is similar to writing down the trace of the new algorithm, which is then saved in the same location. To complete the algorithm's walkthrough, the View will cycle through the data and update the bars in the bar graph to show how the algorithm calculated the steps it took. It's important to note that if the algorithm generated a change in the position of a piece of data, the steps are merely recorded.

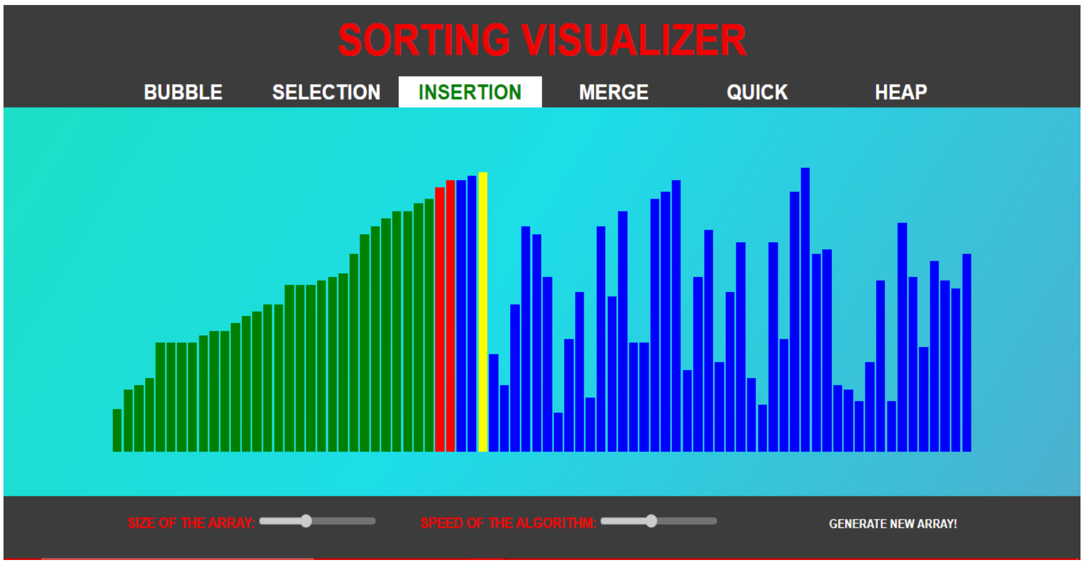

Figure 3: View of Insertion Sorting

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 at 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.

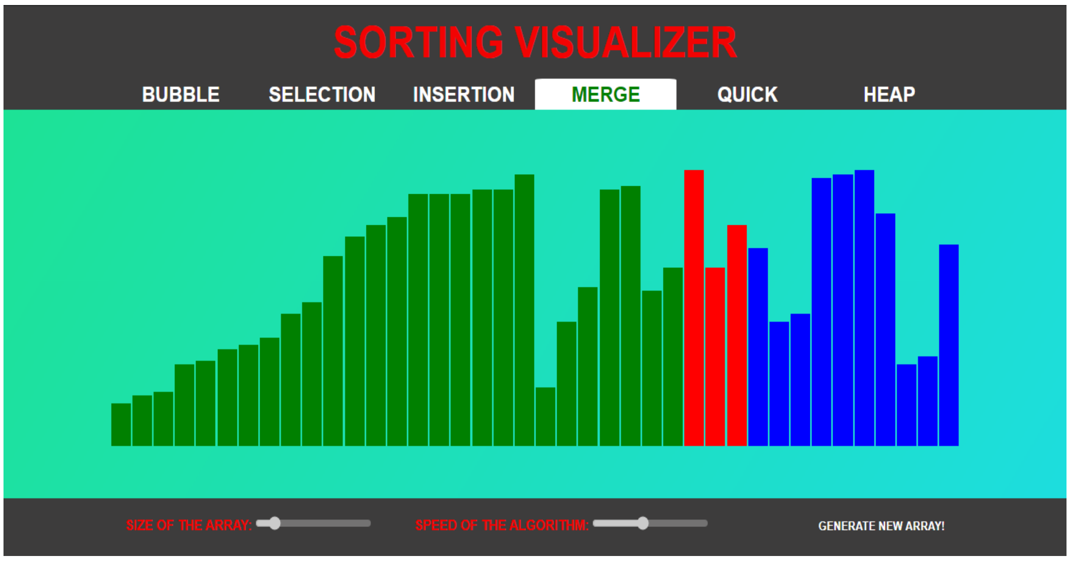

Figure 4: View of Merge Sorting

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.

c) The Controller

Every web page on the web has at least one Controller. The buttons have been programmed in HTML and lead the browser to execute a JavaScript procedure upon clicking. The CSS is used for making the buttons look attractive, arranging them, choosing their colors, and applying visual effects. The methods alter the Model's state, prepare it for the update, and then apply the modifications to the View. Four groups of buttons are distinguished. The blue-bordered buttons at the top of the interface are the sorting algorithm buttons, and they interact directly with the sorter object. The Model calls functions as needed to construct an array of steps for animation. In this animation, the bottom-right buttons allow you to activate the animation, pause it, and advance a single frame. This pulls fresh values from the step array to keep the bar graph fresh. These bar changes are seen in the View, which causes the bars to shift position dependent on the methodology used. The lone button in this category is the Step button. When using the Start and Stop buttons, you may either set a timer to call the Step button every quarter of a second, or you may halt the timer. To the left of the gray-bordered buttons, the bars are controlled directly by the bottom-left buttons. To arrange the bars in the correct sequence, in reverse, or in random order, they move the beginning location of the bars. The bars do not physically move. Each button instead loads an array of data that represents the sequence the user wants. This alteration appears on the canvas instantly as soon as it is made. This last group is a single button, and it's an image. A speaker icon toggles between “mute” and “audio” in the bottom-left corner of the website. To enable or disable the sound animation, just select the “Sound Animations On/Off” option. When the web page loads, the sound will be muted by default (for reasons of memory and performance issues).

IV. UNIT TESTING

In order to perform an experiment to evaluate my animation tool, I recruited coworkers to act as test subjects and survey respondents. To that end, the plan was to expose her Computer Science Data Structures class to the tool and get them to take part in a survey comprising questions that gave the students the ability to write down what they observed and learnt, even if it was little. The questions contained in Appendix C are the questions which were utilized for the survey. Despite the class size being of twenty-one students, only thirteen were present; fortunately, all of the students who had completed the survey were there. The set-up here was ideal since it addressed the audience for whom I was designing this course: a group of college students who were taking their first course in computer science and who needed assistance with algorithms.

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. There was considerable confusion caused by the way the ordering buttons and algorithm buttons were shown in the UI after the surveys were completed. When beginning the sorting process, the student noted that she was having problems starting because she believed that she was hitting the buttons in the wrong order. This then led to her failing to execute the animation. The answers can be found in Appendix D. Overall, my animation tool did not aid with the understanding of sorting algorithms. Among those who answered question 3, which questioned if their knowledge of a particular algorithm changed after using the tool, 5 of the 13 students (38%) stated that they had in some way altered their previous knowledge of the algorithm. Many thought the tool was a good concept, while the other 7 did not find it useful at all. It was said that one student stated a false positive about the instrument (whom I did not include in the 5 that said it was helpful).

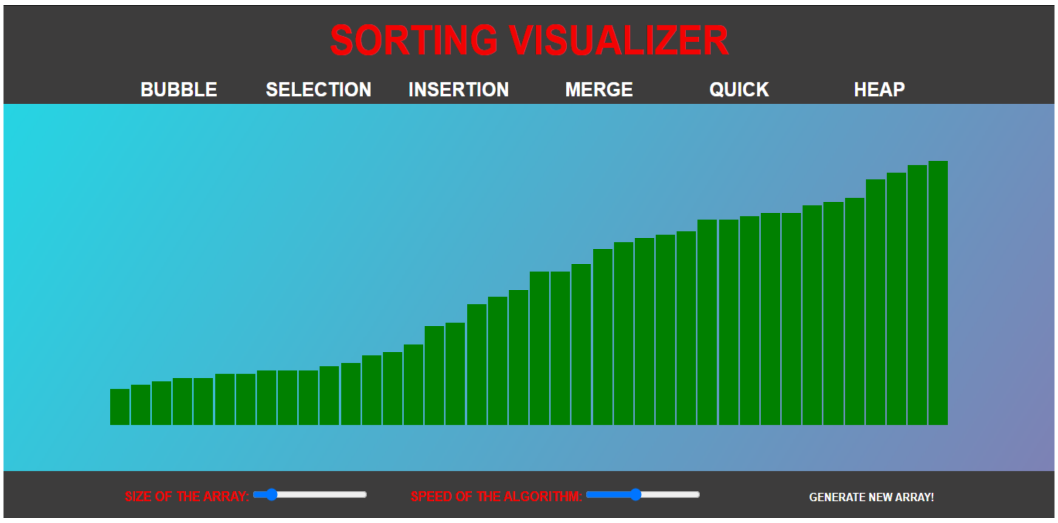

Figure 5: View after the array is sorted

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.

b) Feedback

Testing the algorithm's memory needs leads to endless stalemates and crashes. Students were notified of this issue before to the test, and three students responded to the question concerning it. After seeing the 3.9 average rating on question 4, in which the students were asked to evaluate the usefulness of the tool, I was still amazed. Also, to my amazement, I found features that I'd rejected when I first thought of them, which I have listed below:

- Ability to modify the animation's pace
- Using visual feedback while clicking the buttons on the user interface to keep them selected
- By creating a box-like split for Merge Sort, you'll be able to more clearly identify the split phases.
- Colors should be used to show whether or not anything is being compared.

While students regularly indicate that they would have like to view this animation earlier in the semester, two of them did not make that request. The students reasoned that doing so would help the topics to be more accessible. After giving this presentation, I saw that the students learned about Merge Sort during the presentation. This was the first thing they had thought of, which is why 7 of the 13 (54 percent) decided to include it and make use of the animation. As the author suggests, it may be beneficial to study each algorithm individually instead than mastering all of them at once. Alas, this question tasked the students to determine which sorting algorithm the teacher wanted them to concentrate on: they may choose between Selection Sort, Insertion Sort, or Merge/Insertion Sort. As Data Structures is not a part of the RIC curriculum, Bubble Sort was not included in the survey. I had demonstrated it together with the other algorithms, and it was pleasant to observe the faces of those who had seen it for the first time. This supports the hypothesis that creating animations has shown to be a feasible process. While my memory issues held me back, however, I was able to implement a new notion by utilizing the animation software.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:03/Issue:07/July-2021 Impact Factor- 5.354 www.irjmets.com

V. CONCLUSION AND FUTURE WORK

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. This is consistent with my prior research, which revealed that there was no substantial difference in learning the content. What I do agree with totally is the attitude that holds there is a great need to investigate and produce animated presentations to enhance education in the classroom. Overall, I am not concerned that a large rework to a different language will be required soon because JavaScript is still one of the most popular web languages. We all know about my laundry list of upcoming projects, but there is one elephant in the room that still has to be addressed: resolving the memory difficulties. Following this, we would implement Merge/Insertion Sort, which takes into account the Merge Sort. Then, I would start up Quick Sort so as to finish the job because the code is ready to be integrated.

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.

VI. 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.
[8] <http://dl.acm.org/citation.cfm?id=2449352>.
[9] J. Stasko. Using Student-built Algorithm Animations As Learning Aids. Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education. SIGCSE '97 (San Jose, California), pages 25-29. http://doi.acm.org/10.1145/268084.268091
[10] J. Stasko, A. Badre, and C. Lewis. Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis. Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems, CHI-93 (Amsterdam, the Netherlands), pages 61-6.<http://doi.acm.org/10.1145/169059.169078>.