SORTING ALGORITHM VISUALIZER

Arya Kumar Johary, Abhay Prasad, Sagarika Ghosh and Hriday Banerjee
University Of Engineering and Management, Jaipur.

ABSTRACT

The study that examined the educational value of animated sorting algorithms is described in this publication. An animation application on the web was created to show four sorting algorithms. A bar graph is used to visualize data, and then an algorithm or sorting process may be used. The user then controls the speed at which the generated animation plays, either automatically or manually. This study examines the method used in computer science curricula to teach algorithms. Both the presentation and the survey used in the experiment asked questions of the students that might have shown progress in their understanding of the algorithms. This document contains a catalog of these findings and responses along with a comparison to previous studies.

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

INTRODUCTION

The understanding of sorting algorithms forms a fundamental part of computer science education. However, these algorithms, especially the more complex ones, can be challenging to comprehend through text-based learning alone. Visualization tools can bridge this gap by providing a more intuitive understanding of these algorithms. This tool, built using React.js, serves to animate the process of various sorting algorithms, thereby transforming abstract concepts into tangible visuals. The visualizer operates on an array of data, representing each element as a bar in a histogram. As the chosen sorting algorithm progresses, the bars rearrange themselves in real-time, providing a clear depiction of how data is manipulated during the sorting process. Algorithms such as Bubble Sort, Selection Sort, Insertion Sort, and Merge Sort are included, catering to a range of complexity levels. The usage of React.js, a popular JavaScript library, ensures efficient updates and rendering of the components, enhancing the user experience. The tool is web-based, eliminating the need for additional software installation and making it widely accessible. This Sorting Algorithm Visualizer thus serves as a valuable resource for learners, educators, and enthusiasts alike, offering a unique perspective on the inner workings of sorting algorithms.

LITERATURE REVIEW

Sorting algorithms are fundamental building blocks in computer science, yet their theoretical descriptions can be challenging for students to grasp intuitively. Sorting algorithm visualizers (SAVs) have emerged as valuable tools to bridge this gap by offering dynamic, interactive representations of these algorithms. This literature review investigates previous research on SAVs, highlighting their contributions and exploring potential avenues for future development. One of the pioneering works in SAVs is "Algorithm Animation: A Strategy for Teaching Algorithms" by Baecker (1990). It emphasized the role of animation in enhancing algorithm understanding and presented an early framework for interactive visualization design. Shneier (1998) further explored the pedagogical value of SAVs, highlighting their ability to promote active learning and improve problem-solving skills. These early works laid the foundation for subsequent research by demonstrating the potential of SAVs in computer science education. Several studies have focused on optimizing the visual design of SAVs for effective learning. Faria et al. (2017) compared different visualization techniques like bar charts and color variations, concluding that the choice of representation impacts learners' understanding. Hundhausen et al. (2002) investigated the effectiveness of interactive features like speed control and data size manipulation, finding that active engagement with the visualization enhances learning outcomes. These studies suggest that carefully designed visuals and interactive elements play a crucial role in maximizing the educational impact of SAVs. The effectiveness of SAVs in promoting learning has been the subject of numerous studies. Moreno et al. (2001) conducted

a controlled experiment comparing static diagrams, text descriptions, and interactive visualizations for teaching sorting algorithms. Their results showed that students who used the interactive visualization performed significantly better on knowledge tests. Similarly, Gomes et al. (2012) found that using a SAV improved students' understanding of sorting algorithms compared to traditional lecture-based instruction. These studies provide empirical evidence for the positive impact of SAVs on learning outcomes. Research has also explored the use of SAVs for teaching more complex algorithms and data structures. Walker and Richards (2012) developed a SAV for the Quicksort algorithm, incorporating interactive features like code tracing and visualization of data structures. Palit et al. (2016) designed a SAV for teaching graph algorithms, demonstrating its effectiveness in improving student understanding of complex concepts. These studies showcase the potential of SAVs to go beyond basic sorting algorithms and address a wider range of computer science topics.

Recent research focuses on expanding the functionalities and applications of SAVs. Basca et al. (2020) proposed a gamified SAV, incorporating game mechanics to increase user engagement and motivation. Ioannou et al. (2021) explored the use of virtual reality (VR) for visualizing sorting algorithms, offering a more immersive and interactive learning experience. These innovative approaches suggest that the future of SAVs lies in leveraging advances in technology to create even more engaging and effective learning tools.

PROPOSED METHODOLOGY

In web applications, boxes are used to visualize all of the discussed algorithms. Every box with a number that matches its value denotes one of the array's elements. As soon as the animation begins, boxes are switched out in accordance with algorithm iterations, allowing the user to see how the algorithm operates. The bars that are being compared will shift to various colors. The positions of the boxes will change correspondingly if the numbers in comparison meet the requirement. The user has the ability to further customize the tools' animation speed. Every time the algorithm iterates, the color changes to green once the visualization is complete.

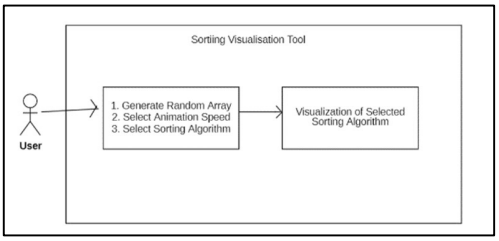

Figure 1: A block diagram illustrating how a user interacts with a visualization tool

Figure 2 illustrates the overall process of creating animation for various sorting algorithms. The user first provides the sorting algorithm with the randomly created array. The sorting algorithm's goal is to sort the array as quickly as possible and produce an animation array with the activities that the algorithms took while sorting the array. This log is an array that holds data, including swap and comparison indexes. Afterwards, the animation engine receives these animated arrays with logs as input. The animation engine then loops over the logs and displays the necessary animation for each log.

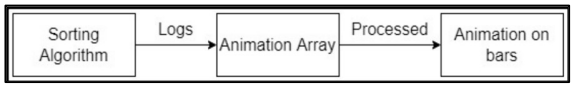

Fig. 2. The sorting visualization tool's internal operation

Internal working of sorting visualization tool This action array is referred to as the animation array below.

RESULT ANALYSIS

Algorithm Visualizer's main page has a neat and straightforward user interface. The website's big navigation bar makes it simple for visitors to reach the many sections. The algorithms are listed on the upper side of the page, and each one is followed by a short explanation of what it accomplishes. These

algorithms include more obscure ones like Merge Sort as well as more well-known ones like Bubble Sort, Quick Sort, and Heap Sort. The method is seen in action in the visualization panel. The visualization panel immediately refreshes to display the algorithm in action when users choose an algorithm from the list. Users can easily comprehend how the algorithm works and see its outcomes in real-time as a consequence. Algorithm Visualizer's main page offers a strong tool for learning and comprehending algorithms and is, overall, a well-designed and user-friendly interface.

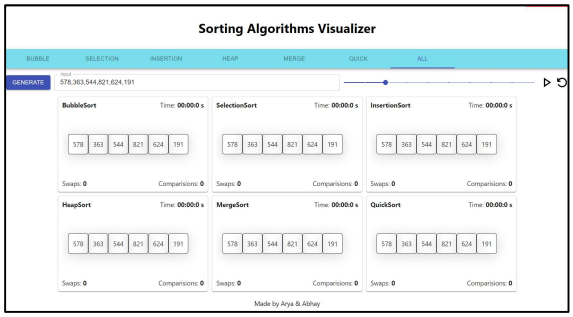

Figure 9: After selecting the "generate new array" button, random array boxes are created.

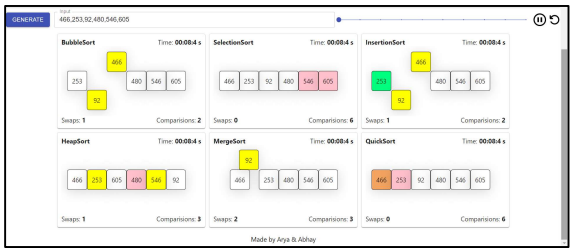

Fig. 10: The outcome of the merge sort visualization.

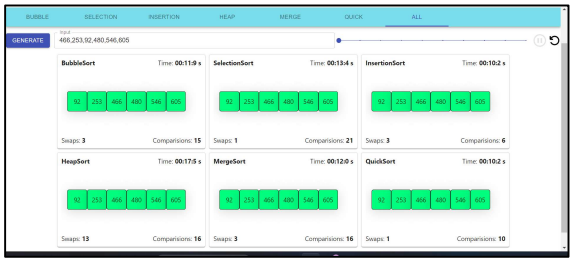

Fig. 11: The outcome obtained upon the completion of the merge sort visualization

This is important because it shows how different algorithms can have an impact on data movement and how that movement can change. It may also provide an alternative viewpoint on how the algorithms work with semi-sorted data in comparison to the available ordering choices. As demonstrated in Table 1, the Sorting Visualization tool can also be used to compare various algorithms and obtain insightful information. It displays the amount of time that each visualization algorithm takes in relation to the total number of array elements that need to be sorted.

Comparing algorithms might be more difficult when it comes to theoretical time complexity. Users can thus gain a better idea of each algorithm's performance by comparing the visualization time taken by the algorithms.

No. of elements in array	Algorithm visualization time (sec)					
	Bubble sort	Selection sort	Insertion sort	Quick sort	Merge sort	Heap sort
25	4.5	3.7	3	3.5	5	1.3
50	18.5	13.2	10	6.5	7.7	5.4
75	42.5	28.9	23.5	11	12.5	9.3
100	75	50	41	17	17.5	13.5

Table 1. Table for Time-Taken to Complete Sort

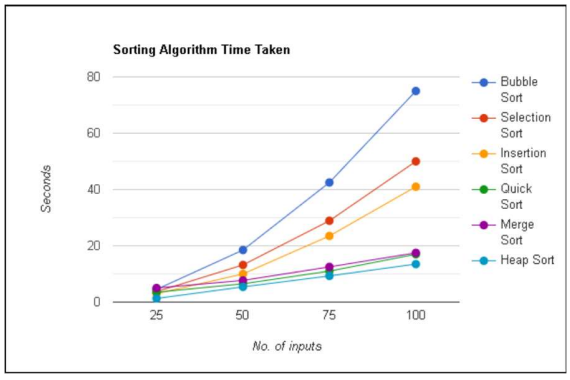

Fig. 12: Time-taken for visualization

Furthermore, Fig. 12 provides a visual representation of this data, making it much simpler to draw the conclusion that, of the sorting algorithms taken into consideration for this experiment, Heap sort performed the best.

CONCLUSION

The functionality of this web-based animation tool for viewing various sorting algorithms is largely due to the time and effort that we invested into it. Despite its memory overhead, the feedback received was predominantly positive from the students who interacted with it. This aligns with our previous research, which indicated no significant difference in content comprehension. We fully agree with the perspective that emphasizes the need to explore and develop animated presentations to enhance classroom education. Overall, we are not concerned about the necessity of a major overhaul to a different language in the near future, given the continued popularity of JavaScript. While we are aware of our extensive list of upcoming projects, there is one major issue that still needs to be addressed: resolving the memory issues. Following this, the implementation of Merge/Insertion Sort, which takes into account the Merge Sort, would be pursued by us. Then, we would initiate Quick Sort to complete the task, as the code is prepared for integration.

FUTURE SCOPE

1. The tool could be expanded to include more sorting algorithms, such as Heap Sort, Shell Sort, Radix Sort, and Bucket Sort. This would provide users with a more comprehensive understanding of sorting algorithms. [1]
2. A feature could be added that allows users to modify existing algorithms or create their own. This would make the tool more interactive and could help users better understand the impact of different operations on the sorting process.
3. The tool could provide real-time performance [2] metrics, such as the number of comparisons or swaps made, the time taken, and the memory used. This would help users understand the efficiency of [3] different algorithms.
4. The visualizer could show pseudocode or actual code in multiple programming languages. This would make the tool useful to a wider audience.

Interactive Tutorials: The tool could include interactive tutorials or challenges that guide users through the workings of different algorithms. This would provide a more engaging learning experience.

5. The tool could be optimized for use on mobile devices, making it accessible to users on-the-go.
6. Features such as a discussion forum or user-generated content could be added. This would foster a community of learners and enthusiasts.

ACKNOWLEDGMENT

The author expresses profound gratitude to the Lord Almighty for the blessings that enabled the completion of their project. They acknowledge the support of extraordinary individuals who played a crucial role in making the thesis possible, creating a web of encouragement around them.

Special thanks are extended to research mentors for their guidance, insightful suggestions, and encouragement throughout the academic journey, turning the project into a valuable and treasured experience.

REFERENCES

[1] T. Bingmann. “The Sound of Sorting - ‘Audibilization’ and Visualization of Sorting Algorithms.” Panthemanet Weblog. Impressum, 22 May 2013. Web. 29 Mar. 2017.
<http://panthema.net/2013/sound-of-sorting/>.

[2] 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.
<https://www.youtube.com/watch?v=lyZQPjUT5B4> .

[3] 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.

[4] JHAVE: supporting algorithm visualization

https://ieeexplore.ieee.org/abstract/document/1510539

[5] Algorithm Visualizer: Its features and working https://ieeexplore.ieee.org/abstract/document/9667586

[6] Enhanced Bubble Sorting Visualizer with Sound by Shubham Tiwari, Neha Gupta, Devendra Chouhan, Ishwarlal Rathod & Harsh Vaja https://link.springer.com/chapter/10.1007/978-3-031-50518-8_6