

Algorithm Visualizer: Engineering College Prospective

Rani Dubey-Faculty at Gandhi Institute For Technology, CSE Dept. (Affiliated to Biju Patnaik University of Technology)

Bhagyashree Sahoo – Student at Gandhi Institute For Technology, CSE Dept. (Affiliated to Biju Patnaik University of Technology)

Siprarani Acharya– Student at Gandhi Institute For Technology, CSE Dept. (Affiliated to Biju Patnaik University of Technology)

Vishal Kumar – Student at Gandhi Institute For Technology, CSE Dept. (Affiliated to Biju Patnaik University of Technology)

Abstract

This paper discusses a study performed on animating sorting algorithms as a learning aid for classroom instruction. A web-based animation tool was created to visualize four common sorting algorithms: Selection Sort, Bubble Sort, Insertion Sort, and Merge Sort. The animation tool would represent data as a bar-graph and after selecting a data-ordering and algorithm, the user can run an automated animation or step through it at their own pace. Afterwards, a study was conducted with a voluntary student population at Gandhi Institute For Technology, Bhubaneswar who were in the process of learning algorithms in their Computer Science curriculum. The study consisted of a demonstration and survey that asked the students questions that may show improvement when understanding algorithms. The results and responses are recorded and analyzed in this paper with respect to previous studies.

Keywords: — *Algo – Visualizer*

Introduction

How do you work out a problem? The problem itself doesn't need to be anything overly complex, such as trying to replace a broken headlight in your car (although nowadays, manufacturers are trying the patience of the community with their increasingly abstract, space-age designs). The point is how to *attack* the problem. Do you perform research, such as looking through your car's manual for step-by-step instructions, or is your first instinct to find someone who knows how to do it (whether they are right next to you or in an online video)? My instinct is the latter, as I am a visual learner and am adept to picking up concepts by *seeing* it done, rather than *reading* about it. For example, when I was learning about sorting algorithms while pursuing my Computer Science degree, I found that *seeing* the data move to its correct position under the constraints of an algorithm was much easier to follow than tracing the code by hand.

of the webpage (buttons and layout) are coded with very minimal HTML5 code. The next biggest contributor would be the CSS code, which is responsible for the appearance and behavior of the buttons and text. Finally, the rest is devoted to JavaScript, responsible for the histogram generation, movement, algorithm design, and sound. All the buttons refer to designated parts of the JavaScript code to perform the task.

I did have some experience creating web pages from previous internships, but not nearly enough to handle this task effectively. Most of the time, I was coding in a simple text editor and was hunting for syntactical errors by hand. Toward the end of the project was when I

found out that browsers have a built-in developer tab that highlights syntactical errors. Before this, when my code broke, the histogram would just disappear and I had no idea why.

Chapter Two of this paper discusses work and conclusions made by other similar studies, and also provides references to types of more recent video animation. Chapter Three is a top-level view of how the animation works and how a user would perform the sorts. Chapter Four is an in-depth view of the underlying structure of the code and how it relates to the final animation. Chapter Five describes my methods and results of user testing and Chapter Six concludes my findings with a tangent of how this would relate to future work. All the code is in the appendix and can be made into a single HTML file, so if you want to use it, go for it.

1. The USER-INTERFACE

Even though the underlying back-end code went through a drastic refactor midway through the implementation, the overall design and layout of the user-interface components has remained the same. The interface has twelve components: a canvas area, ten control buttons, and a volume on/off toggle button. Below in Figure 1 is a picture of the web page after it loads.



The top section that shows the blue bars is the canvas area and is what updates to visualize the four sorting algorithms. Below the canvas, the first row of four blue-bordered buttons are the selectable algorithms: Selection Sort, Bubble Sort, Insertion Sort, and Merge/Insertion Sort (the interface does not show the title as “Merge/Insertion Sort” because this was a design change after the refactor). The user can select any of these algorithms to see the visualization of how that algorithm works. There is no algorithm selected by default, so the user will need to select one before starting the animation.

Before selecting an algorithm, the user must select the type of input data to be sorted. The three gray-bordered buttons on the left of the bottom row allow the user to choose between sorting input data that is already in order (as shown in Figure 1 on the previous page), or in reverse and random orders (shown in Figures 2 and 3 in the following paragraphs). The default is in sorted order.

Once the input and the sorting algorithm have been selected, the user can click the green-bordered “Start” button in the next row of buttons to see the sort run from beginning to end. To see the algorithm execution slowly step-by-step, the user can click the yellow-orange-bordered “Step” button.

The “Stop” button simply halts the auto-animating process if in progress.

2. SYSTEM ARCHITETURE

The back-end code is comprised of HTML5, CSS, and JavaScript. All three types of code are contained in one .html file and can be run solely from this file. One of the advantages of HTML 5 is that it is not necessary to include different types of web languages in a single file. Therefore, each type could have been separated, making a total of three files (plus the miscellaneous sound and image files). This is good practice for readability and keeping related code together. However, I decided not to separate the code for two reasons: 1) to increase the portability of the project by only needing to worry about one project file instead of three, and 2) where in the project file, the change in coding languages is distinctly marked and therefore does not significantly reduce readability. Also, the ability to put more than one web language in a single file is an example of an RIA (Rich Internet Application).

Below in Figure 7 is an illustration of how the three coding languages relate and communicate With each other.

As you can see, there are no major components besides the three coding languages. Most websites have tools or scripts that require a server on the back-end (like PHP), but it is not necessary in this case since JavaScript runs right in the user's browser. HTML5 and CSS are used for the interface. The HTML5 communicates with the JavaScript code and vice versa to launch the appropriate algorithms and update the interface accordingly, as seen with a single, bidirectional arrow.

Throughout the project, the code for the HTML5 and CSS did not change much. As the JavaScript was modified from a functional programming focus to a more object-oriented one, the parts of the HTML5 that did change were the function calls for each button. All of the back-end interaction is abstracted to the various buttons for selecting algorithms and running the animation.

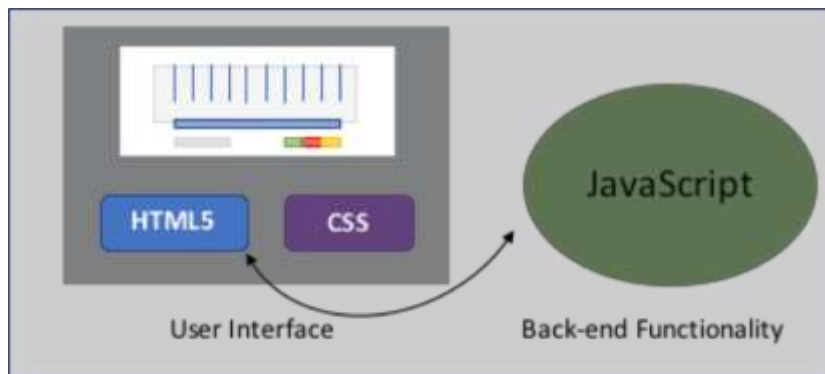
3. DESIGNING AND EXPERIMENT

As this tool is designed to be a learning aid in the classroom, this has presented ethical issues in experiment design. I briefly mentioned this earlier in Chapter 3 as being a part of the paper "Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis" [6]. The students in Stasko's study were randomly separated into two groups. In one, the students were taught the material using conventional methods (i.e. textbook). In the other, the students had both the textbook and an animation tool to assist them when learning the material. This separation is necessary for more reliable results that will hopefully help improve teaching and learning for many students, but this comes at the cost of giving the participants in the study different experiences. Although Stasko's results proved a non-significant difference in the post-tests of both groups, it still created a noticeable learning gap, giving the students that used the animation tool an advantage over the others.

Given that Stasko had already reported these results and I am fond of fair practices, I decided not to follow his approach and made my post-test into a short survey that asked questions about what they recently learned in class about algorithms and how that changed when exploring my animation tool. An included 5-point rating scale gave the opportunity to judge if they believed the tool helped their current understanding at all. The survey was also voluntary, so it was up to the students' discretion to participate. The animation tool, however, was integrated into their class lecture via my presentation and time dedicated afterward to use

the tool. Therefore, all the students were exposed to the tool instead of a select few, which avoided any academic disadvantage.

4. ARCHITECTURE DIAGRAM



5. CONCLUSION

- Performance comparison in AV may enhance student knowledge about algorithm. This statement is concluded from the result of performance comparison in questionnaire which yields fairly good result
- Based on controlled experiment and observational study, it can be concluded that our implementation AV, It is easy to use and quite effective to improve student knowledge about algorithm

6. REFERENCES

- T. L. Naps, G. Röbling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger and J. A. Velázquez-Iturbide, "Exploring the role of visualization and engagement in computer science education," in ITiCSE-WGR '02 Working group reports from ITiCSE on Innovation and technology in computer science education, New York, 2003.
- C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce and S. H. Edwards, "Algorithm Visualization: The State of the Field," ACM Transactions on Computing Education (TOCE), vol. 10, no. 3, 2010

- E. Fouh, M. Akbar and C. A. Shaffer, "The role of visualization in computer science education," *Computers in the Schools: Interdisciplinary Journal of Practice, Theory, and Applied Research*, vol. 29, no. 1-2, pp. 95-117, 2012.
- S. Halim, Z. C. Koh, V. B. H. Loh and F. Halim, "Learning Algorithms with Unified and Interactive Web-Based Visualization," *Olympiads in Informatics*, vol. 6, pp. 53-68, 2012.
- J. Á. Velázquez-Iturbide and A. Pérez-Carrasco, "Active learning of greedy algorithms by means of interactive experimentation," in *ITiCSE '09 Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, New York, 2009.
- "Curriculum Guidelines for Undergraduate Degree Programs in Computer Science," ACM and IEEE Computer Society. The Joint Task Force on Computing Curricula: Computer Science Curricula 2013, 2013. [Online]. Available: <http://www.acm.org/education/CS2013-final-report-pdf>.
- S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach* (3rd Edition), Prentice Hall, 2009.