CSE104 Final Project Report

Ha Duy Nguyen, Truong Dat Pham
May 09, 2023

1. Background

Sorting algorithms are some of the simplest, and yet still extremely important algorithms in computer science. With various applications in searching algorithms, organizing databases, and data processing and analysis, sorting algorithms are often some of the first algorithms that new programmers learn when they enter the industry. Furthermore, the relatively simplistic nature of the task allows the learner to more clearly follow the algorithm's process, making it an ideal choice for starters to get used to the concept of algorithms.

However, many sorting algorithms can still be quite complicated and difficult to understand without considerable previous experience in computer science. With that in mind, we created this website, a **Sorting Visualizer**, in order to visualize several sorting algorithms, with the goal that it might help future generations of students.

Within the website are relatively detailed descriptions and interactive illustrations of four sorting algorithms: bubble sort, heap sort, insertion sort, and quicksort. Bubble sort and insertion sort are chosen for their relative simplicity and as such ease of understanding, while heap sort and quicksort are widely used as the primary sorting algorithms in most languages nowadays, with quicksort being the first-case algorithm and heap sort a backup should quicksort becomes degenerate. As such, the illustrations would provide a gentle learning curve, as well as successfully teach the logic behind in-built sorting functions.

2. Functionalities

The main website contains a very pleasing introduction to the website: a string of random letters that slowly settle down into the words "SORTING_VISUALIZER". After it is done, a little line of text saying "scroll down" will slowly appear at the bottom of the screen, while the initial text glows with random colors and each letter is slightly rotated, creating an effect that makes it seem as if the words are on a flier.

Scrolling down will reveal several possible choices of sorting algorithms for the user to try out. Hovering above them will reveal the description text for the algorithm. A small, fun surprise that can be easy to miss is that the name of the algorithm at the top of the square will have the same rotating effect as the words "SORTING_VISUALIZER" on the white screen above.

Double-clicking on each of the algorithms will lead to a subpage where one can play with it. The algorithms are all demonstrated in a similar method: by bars of various heights, and the algorithms will visually sort them from lowest to highest, left to right on the screen. There are several settings one can choose: a slider to choose the number of bars (or the length of the list being sorted), the speed of the algorithm (with three choices Slow, Medium, Fast), to initiate a list by randomizing it, and to start sorting the list. We also have a page where one can visually compare algorithms with each other, furthermore distinctly highlighting the runtime difference between algorithms - for example, quicksort and heapsort will dominate the quadratic algorithms, while a comparison between those two will reveal why quicksort is favored.

During the sorting process, bars directly involved in each operation will be highlighted. For example: in bubble sort, the current bar being evaluated will be highlighted with red. In quick sort, the pivot will be highlighted with red, and the left and right pointers respectively highlighted with green and blue. To conclude a successful sorting process, all bars will be swept green.

Some sound effects are also included in the sorting progress. Swapping two bars will produce a note that is proportionately as high as the sum of the height of the bars involved.

3. Tools and methods

3.1. Project structure

The website is created in large part by JavaScript, with approximately 60% of the program being made in JavaScript. The rest are divided into 27.2% of HTML and 12.1% of CSS.

The front of the site is the index.html file. Each sorting algorithm is given one HTML file (bubble-sort.html and such). While comparatively large in size, due to the similar structure of those files, they were practically copy-pasted from one template, with only slight differences in slider bounds (due to time complexity constraints of less efficient algorithms) and the reference file.

The src folder contains the vast majority of the program's running code, including the sorting algorithms themselves (in files of the type (algorithm name).js, such as bubble_sort.js), an overseer req.js file that contains various utility functions such as generating a random list or rendering the numerical list into the visual height of bars.

Three other files are also included in the src folder: animation.js, mainStyle.css, and smallStyle.css. As the name suggests, animation.js takes care of the animation tasks in the front index.html page, while mainStyle.css styles the index.html file and smallStyle.css styles the subpages containing the sorting algorithms.

3.2. Logic of the code

The code logic of the subpages is relatively simple. Imitating existing illustrations, the sorting algorithms highlight bars involved in each step, with small pauses between each step. This will give time for the watcher to study, while also serving the dual purpose of demonstrably showing the relative runtime between such algorithms, especially with larger inputs.

An interesting feature is the involvement of sound effects in the program. As mentioned before, swapping two bards will produce a note with proportional height to the sum of the height of the swapped bars. This serves two purposes: first and simply, it will provide filler noise for the user while watching, but more importantly, due to how the notes correspond to the height of the bars, it will naturally produce discordant noises while the list is unsorted while growing increasingly more harmonical as it is ordered. This provides an audio representation of the list's structuring, one that can be intuitively felt.

To ensure that the tallest bars always reach the maximum possible height, a variable HeightFactor is introduced, defined as (Maximum possible height / Tallest bar's height). This way, by multiplying all bars' heights by HeightFactor, the heights of the bar can still be randomized to any degree and is still appropriately scaled.

The algorithms themselves are comparatively simple. There are detailed pseudocode implementations on Wikipedia, so it was not very difficult to write JavaScript code for them.

The animation.js file, on the other hand, requires a lot of deliberation. While the code logic is more or less explained by its effects (fading in the “scroll down” line, for example), those animation effects are quite intricate and require a lot of effort.

4. Results

While the initial topic was not as complex as some others, we believe that we did an excellent job in implementing a functional, interactive, and engaging Sorting Visualizer. It encompasses both famous examples of simple and introductory sorting algorithms as well as the widely accepted modern counterpart. Furthermore, the processes were clearly explained by the highlights, the illustrations are aesthetically pleasing, and they even have good audio. Furthermore, the introductory page was very beautiful and intricately designed, with various fun effects. Overall, even if the scope of the project was limited, the execution was excellent and we believe satisfactory.