Lab 9 Report

Runtime for Sorting Algorithms

Derek Campbell
FA20 CPSC 1110
November 9, 2020

I am not sure if the reason this lab was easier than the previous labs is because a large portion of the code was provided for us, or if it is because I had a better time grasping and understanding the issues that I encountered during testing and was able to articulate through those roadblocks.

I elected to hand copy the code from the book as instructed as opposed to copy/pasting it. In doing so, I was able to walk myself through the code mentally and understand each step in the algorithms. This helped me to see each class as its own piece of the package and how it would be called upon in my tester. The benefit behind this practice greatly outweighs the convenience of copy/paste as re-articulating code helps me in understanding it.

That being said, the largest roadblock I encountered was in my tester class. After creating an array of the desired size and filling it with random values, it began interacting strangely with my sorting methods. I choose to sort in the order: Selection Sort, Merge Sort, and finally Array Sort. After running through the Selection Sort method, my tester class would produce the desired output indicating the time it took for the computer to sort the array. However, the following two sorts would be drastically shorter; often times a sort time of zero (0) milliseconds! While it is not uncommon for the runtime of the sorting algorithm to vary this much, it was strange to me that I rarely had a sort that took any time other than the Selection Sort. What I realized through debugging was that I was essentially running a sorting algorithm on an already sorted array! Of course it would take no time at all as the computer was not needing to do anything to further sort the array.

In order to solve this problem, I created duplicate arrays with different names so that they would be identical in content, but separate objects altogether. This way, I could get an accurate sorting time for an equally randomized array. After making this change to my code, I began seeing runtime numbers which were more indicative of the sorting methods being used; thus solving my issue.

The only other issue I ran into was with utilizing the stopwatch. It seemed that the results were constantly increasing at a nearly identical rate. This meant that I wasn't reseting the timer between each sort method and instead just adding on time. In order to fix this, I called the reset method in the Stopwatch class to set the runtime to zero (0) after running a sort algorithm in preparation of the next one. A much simpler solution than the duplicate arrays. I was also able to share this solution during the lab time on Thursday, November 5th with another student.

Writing the output of the test program to a .TXT file was fairly simple considering this was a topic we covered a few labs ago. The only issue was formatting the spaces, and from there I hand copied the values into the powerpoint in order to build my graphs.

A note on the graphs. The standard scale line graph had to be broken up into two separate graphs. This is because with the size of the time values on the larger arrays, it made it impossible to see any other pattern information regarding the runtime of the sorts. By serrating it into separate graphs, we can see the distinct patterns in the arrays as they grow in size and not be completely overshadowed.