Mark Griffin

5/12/21

CPSC 350 – Data Structures and Algorithms

Rene German

Assignment 7 Experience Report

As forecasted, this assignment was the first assignment that turned on the fans in my laptop. I did not realize how taxing on computer hardware sorting could be, but this assignment was a good eye-opener that sorting could be expensive.

The run times shocked me when I used large input sizes of doubles. I knew that bubble sort, selection sort, and insertion sort were going to have slower times, but I did not realize how much slower they are compared to shell sort, merge sort, and quicksort. I now have a greater understanding of how drastic a change from $O(N^2)$ to O(n logn) is and how important it is to have as low a Big-Oh runtime as possible. I had my program sort 500,000 randomly generated doubles to gauge performance in speed and hardware usage. Bubble sort took 663.242 seconds to complete, and my CPU usage during the duration spiked up from around 10% to approximately 30%. Selection sort took 252.806 seconds to complete, and my CPU usage mirrored that of bubble sort. Insertion sort took 230.863 seconds to complete, and just like bubble sort and selection sort, my CPU usage peaked around 30% and had little to no effect on my memory usage. Shell sort took 0.199015 seconds to sort 500,000 doubles, and my CPU usage peaked around 30%, just like the $O(N^2)$ sorting algorithms. Merge sort took 0.105506 seconds to complete, with my CPU usage peaking around 25% and memory usage peaking around 56%, from the usual 53%. Quicksort was the fastest sorting algorithm for 500,000 randomly generated

doubles as it only took 0.069374 seconds, with my CPU usage peaking around 30%. After doing these tests, it was evident that shell sort, merge sort, and quick sort is much more efficient sorting algorithms than bubble sort, selection sort, and insertion sort for highly randomized data. Quicksort was an astonishing 9,560 times faster than bubble sort when sorting these doubles.

For picking an algorithm for sorting, it is clear that the more complex algorithms (shell, merge, quick) are better at sorting highly randomized data in a short amount of time. However, if the data is somewhat sorted, insertion sort could have a runtime of O(N) as it does not enter its second loop, giving it a better runtime than the more complex algorithms. If an algorithm for sorting is needed on the fly and not much knowledge of sorting algorithms is necessary, bubble sort, although slow, can be a good option.

I have not used sorting algorithms like these in other programming languages, so I do not have any personal experience with one language being better than another for sorting. Still, based on what is online, it seems as though C++ tends to be one of the faster programming languages for sorting.

From this assignment, it is clear that empirical analysis is not cost or time-efficient. It can take a long time to run an entire program with a large input size, and the hardware necessary to run that program can be costly. 500,000 doubles is relatively tiny compared to the millions upon millions of data stored in databases that need sorting all of the time. It would be highly inefficient to physically run these sorting algorithms on input sizes of millions to billions of data entries, as it could take several hours to even days.