Assignment 6

An observation on the time complexity of several sorting algorithms.

Jadyn Gonzalez

Chapman University

12/14/18

***Abstract*—In this assignment I implemented and tested the run-times of BubbleSort, InsertionSort, QuickSort, and CombSort. The language used was C++ and run-time was tested using the chrono functions in the std library. Time was measured in milliseconds from start to end of each algorithm.**

Keywords—BubbleSort, InsertionSort, QuickSort, CombSort, Time Complexity, Run-time

I. Were the time differences more drastic than you expected?

Yes, I tested each algorithm three times using three different data sets. One was a randomized set of 10,000 numbers, the other was a randomized set of 100,000 numbers and the last was a reverse sorted set of 10,000 numbers. Overall BubbleSort was the slowest algorithm having an average sorting time of 375ms across all sets. InsertionSort was faster having an average run-time of 88ms across all sets. CombSort was the quickest with an average of 1ms which was very surprising. QuickSort was the most interesting to observe as on the randomized sets it had an average run-time of 1ms but when sorting a reversed sorted set it took the longest of any algorithm averaging 486ms for the three tests.

II. What tradeoffs are involed in picking one algorithm over the other?

Things to consider when choosing an algorithm have to be complexity, efficiency, computing power, and memory available. Overall BubbleSort was the easiest to implement, but was fairly slow across all tests. Insertion sort was slightly more complex but still easy to implement; it also had faster run-time than BubbleSort and was faster in sorting a reverse sorted array. QuickSort was the most difficult to implement but was the fastest in sorting randomized sets, however, performance got very slow once a reverse sorted array was sorted. CombSort is an improvement on BubbleSort and showed during my testing. It was by far the fastest algorithm and was not that hard to implement.

III. How did your choice of programming language affect the results?

C++ made it easier for me to implement these algorithms since I am most comfortable with this language. The use of pointers also allows for ease of manipulating arrays. Classes in C++ made it easy for me to structure my code as well. However the only way I found to time my algorithms was using the chrono class in C++ which from my research I found it is not as accurate as other methods.

IV. What are some shortcomings of this empirical analysis.

This test took time and energy to do whereas in a mathematical analysis we could look at the different Big-O runtimes of each algorithms to determine what we want. I am using a fairly good computer to test these algorithms and when monitoring my performance while they were running I noticed no real spike in CPU, RAM, or Disk usage, but I could see how an empirical test could be beneficial in showing how an algorithm would run on a system that has limited resources.