Assignment 6

An observation on the time complexity of four sorting algorithms.

Matthew Tonks

Chapman University

12/12/18

***Abstract*—In this assignment I implemented and recorded the run-times of Bubble Sort, Insertion Sort, Quick Sort, and Heap Sort. The language the methods where implemented in C++ and run-time was tested using the standard libraries chrono functions. Time was measured in milliseconds from start to finish of each algorithm.**

Keywords—Bubble Sort, Insertion Sort, Quick Sort, Heap Sort, Time Complexity, Time Complexity, Space Complexity

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

No, I tested each algorithm four times using two different data sets. Both sets where randomized 10,000 doubles. Overall Bubble Sort was consistently the slowest sorting method getting well over 500 milliseconds each time. But the implementation of the Bubble Sort allows for some leniency in its time to operate. Insertion sort was marginally faster taking on average ~360 milliseconds to sort. The implementation for insertion sort is marginally complex. Quick sort was faster and broke into single digit milliseconds every time. The implementation was difficult, but the performance benefits made it worth it. Heap Sort also worked on par and with Quick sort and it was difficult to differentiate which was faster.

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

The things to consider when choosing a sorting algorithm is time complexity, efficiency, computing power and memory allocation. Overall Bubble Sort was the worst in time complexity and computationally dependent but memory allocation outside of the array being sorted is constant. A similar story is told with Quick sort Insertion sort and, Heap Sort being heavy on the computation but has a constant space complexity.

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

C++ made implementation of each different sorting method simple since this is the programming language, I have used the most during the last few months. Writing methods was simple to create and use. To find the time I implemented the standard library's chrono class in C++ to find the time of the start and the end to find the time it took.

IV. What are some shortcomings of this empirical analysis.

The testing I undertook took time and energy to do instead of a mathematical analysis. We could use the time complexity and the space complexity of each algorithm to choose which sorting method is best based on the scenario you are going to need one. An aspect that needs to be taken into account is that I ran these tests on a good computer with a great processor and more than enough RAM and an NVME solid state drive, so there were no spikes in any of the mentioned aspects of my computer when monitored. I also see that true empirical analysis could be beneficial in understanding what each algorithm costs so that it could work on a system that has very limited resources.