Sorting Algorithms and their Efficiency

Travis Mayer
Professor German
CPSC 350
December 13, 2019

***Abstract*—This document is about my experience using the sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort. It will cover what I learned from using them, and discovering how Big O translates to the time complexity of the code in practice.**

I. Time Differences

The time differences between the different sorting algorithms were a lot more drastic than I expected. I coded Bubble Sort, Selection Sort, and Insertion Sort first. I used an array of elements n = 21,000. I measured the time complexity of the sorting methods using microseconds. Bubble Sort took on average 1.4 million microseconds, Selection Sort 320k, and Insertion Sort 250k. When it came to coding Merge Sort and Quick Sort I actually thought I coded them incorrectly, and that they weren't actually sorting anything when I noticed that the time stamp said that it only took them about 3-4k microseconds. When I actually realized that this was correct, I was shocked. It has helped me to understand the actual differences between a Big-Oh of (n^2), and ($nlog(n)$).

II. Trade-offs

The main trade offs between sorting algorithms are time complexity and ease of coding. I would group Bubble Sort, Selection Sort, and Insertion Sort into a group of sorting algorithms easiest to code, and Merge Sort, and Quick Sort into a group of sorting algorithms with the best time complexity. Among the simple to code group, I would choose Insertion Sort as a preference as it has the best average performance, and a best case performance that beats even the fastest recursive sorting algorithms. Among the faster sorting algorithm group, I would choose Quick Sort as a preference as it is a bit easier to code than Merge Sort, is just as fast, and has a much better space complexity at O($log(n)$) rather than O(n).

III. Effect of Programming Languages

The effect of programming languages on the results would be fairly large. C++ is significantly faster than code in an interpreted language. This could make the run-time of these algorithms in C++ faster than it may otherwise be in interpreted languages such as Java.

IV. Shortcomings of Empirical Analysis

The major shortcoming of empirical analysis is that it is significantly more time consuming than mathematical analysis. In this case, it requires actually knowing and coding all of the sorting algorithms rather than just knowing their Big-Oh. While empirical analysis allows for much greater accuracy than mathematical analysis, both have their use cases. In some cases accuracy is more important, and sometimes time is more important.