Comparing Sorting Algorithms in C++

Jacob Anabi

***Abstract*— This paper attempts to empirically compare the time complexity of the four sorting algorithms - bubble sort, selection sort, insertion sort, and quick sort. This will be done rather crudely, using built-in C++ functions.**

I. INTRODUCTION

There are many ways to sort a set of numbers. One way is by using a comparative approach. Four sorting algorithms that exhibit this idea of comparative sorting are bubble sort, selection sort, insertion sort, and quick sort. These algorithms have a average time complexity of $\Theta(n^2)$, $\Theta(n^2)$, $\Theta(n^2)$, and $\Theta(n\log(n))$, respectively. The worst case run-time for quick sort, though rare, is $O(n^2)$, and this can caused by choosing a bad pivot point, or sorting a pre-sorted set of data. Additionally, the best time complexity a comparative sorting algorithm can achieve is $\Omega(n\log(n))$.

II. PROCEDURE

Empirical analysis was performed by using the C++ std::chronos library. The time was taken before (t_0) and after (t_f) each algorithm was run. The time each algorithm took to run was simply calculated by taking the difference of those two times ($\Delta t = t_f - t_0$).

III. RESULTS

The results that were achieved are recorded in the table 1. As we can see, for a set of 10 numbers (relatively small), insertion sort was the fasted, followed by selection sort, then bubble sort, with quick sort in last. However, when we increased the set of numbers to 1000 (relatively large), quick sort was the fastest, followed by selection sort, than insertion sort, with bubble sort in last.

A. Figures and Tables

TABLE I
EMPIRICAL RUNTIMES

Algorithm	Small Set	Large Set
Bubble Sort	0.002384 ms	8.41203 ms
Selection Sort	0.001729 ms	4.02211 ms
Insertion Sort	0.001255 ms	4.66054 ms
Quick Sort	0.002545 ms	0.375357 ms

IV. ANALYSIS

The time differences seen above are expected. For smaller data sets, quick sort doesn't seem to be the best option (given its relative performance and complexity compared to the other sorting algorithms). However, for larger data sets, it is very valuable. The difference between bubble sort and quick sort for the large data set is quite extreme. If we performed our tests in a higher-level language, like Python, the individual times for each algorithm would be different (most likely slower), but the relative times (time differences) between them should show the same expected behavior. That is, quick sort is better for larger data sets, but not for relatively small or presorted ones. Additionally, testing these algorithms empirically might yield different individual time results depending on the machine. That is one of the downfalls of empirical analysis; it is very hardware dependent and not very time and cost effective. By using mathematical analysis, we could clearly see that quick sort is indeed the fastest on the average case, without having to physically test each algorithm.

V. CONCLUSIONS

When deciding which algorithm to use, it is very important to consider the trade offs. That is, one should consider what machine they are programming for (embedded device, supercomputer, etc.), and what there data set actually is. A naive approach is to assume that quick sort is the best in every case, however, based on our results and analysis, we can clearly see that it is not.