Data Structures: Assignment 6 Report

Noah Estrada-Rand
Folwer School of Engineering
Chapman University
Orange, CA
estra146@mail.chapman.edu

***Abstract*—This document is to report on the findings from the time trials for Assignment 6 in Data Structures, CPSC-350-01, concerning the different implementations of sorting algorithms.**

***Index Terms*—data, sorting, merge, insertion, bubble**

I. Introduction

For Assignment 6, we were tasked with researching and implementing variuos sorting algorithms. We were told to implement bubble, insertion, and selection sort and given the option to implement either quick or merge sort. I decided to implement merge sort and tested all algorithms on 150,000 double values.

II. Sorting Test Results

A. Bubble Sort

- Sort start time: 04:52:44 pm
- Sort end time: 04:54:41pm
- Time Elapsed: 117.0528 seconds

B. Insertion Sort

- Sort start time: 04:55:11 pm
- Sort end time: 07:15:38 pm
- Time Elapsed: 26.6934 seconds

C. Selection Sort

- Sort start time: 04:55:38 pm
- Sort end time: 04:56:05 pm
- Time Elapsed: 26.56494 seconds

D. Merge Sort

- Sort start time: 04:56:09 pm
- Sort end time: 04:56:09 pm
- Time Elapsed: 0.05486 seconds

III. Conclusion

Overall algorithms performed as expected. As expected, merge sort was the fastest algorithm, clocking in at under a second for complete sorting. Selection and insertion sort were the next fastest, both performing at about 26 seconds of sorting time. Bubble sort came in dead last, taking a total of 117 seconds to complete its sorting.

When observing how my computer handled all four algorithms, it became clear that insertion, selection, and bubble sort required the largest amount of CPU while merge sort barely required any. However, using merge sort saw a large increase in memory usage as a result of the extra arrays needed to complete the algorithm.

A. Choice of Programming Language

After observing the effects of the different algorithms on the rate of sorting, I believe that using C++ allowed the process to take place much faster than it would have in perhaps an interpreted language. By compiling the algorithms in C++ to machine code, I believe the rate of computing to be much faster than if, perhaps, Python had been used instead.

B. Algorithm Tradeoffs

When picking one algorithm over another it is important to consider the memory and computing power necessary to carry out the algorithm at scale. While two algorithms may get the job done just as equally as the other, the time, memory, and computing resources spent on each weight heavily on which will be used. This clearly is displayed in this assignment as there was a clear cut winner for sorting algorithms when looking at time spent. While merge sort barely broke a sweat on larger dataset sizes, bubble, insertion, and selection sort showed their weaknessses as they were unable to come close to merge sort. However, it is important to remember that the creation of new arrays in merge sort led to large memory consumption. While quick sort would have compensated for this, it would use increasing amounts of CPU.

Ultimately, choosing one algorithm over another depends on a number of different variables, including time, memory, and computing power; all of which must be taken into account at each implementation. Thus, new perspectives must be taken in each situation to find the optimal solution.

C. Emprical Shortcomings

Though this empirical analysis proved helpful in observing tangible differences between different sorting algorithms, it ultimately was very time consuming. From the implimentation to the actual testing and observing of results, the time needed to observe differences between different algorithms was very costly. Had we instead been allowed to use analytical, mathematical analyses, it would have been meant much less time spent coding all of the algorithms and isntead only spent quickly considering the Big-O runtimes of each algorithm and more time implementing and perfecting the final choice, merge sort.