Data Structures Assignment 6: Sorting

Stephen White

I. Introduction

This assignment was all about sorting algorithms. There are five sorting algorithms demonstrated here: Bubble, Selection, Insertion, Quick, and Merge. The purpose of this paper is to display the varying efficiencies of each sorting algorithm.

II. Run-time Differences

Before writing a single line of code, some basic analysis can be done to predict the run-time of each sorting algorithm. Using mathematical analysis on each of the sorting algorithms, we can confidently determine the Big-O run-times:

- Bubble : $O(n^2)$
- Selection : $O(n^2)$
- Insertion : best case $O(N)$, worst case $O(n^2)$
- Quick : $O(n * log(n))$
- Merge : $O(n * log(n))$

According to the mathematical analysis above, we would be able to infer that bubble and selection sort would have the same run-time, same with Quick and Merge. But these are just mathematical estimations, what do the real numbers say? I implemented all five algorithms and ran them on the same data set of 100,000 doubles and the following table shows their run-times:

Name	Run-time (seconds)
Bubble	105.3
Selection	43.4
Insertion	12.8
Quick	45.1
Merge	0.07

As we can see, this is far more variable than the mathematical analysis would entail. The Bubble sort took an enormous amount of time compared to all the others, while Merge sort is unbelievably fast. I'm not quite sure why quick sort was so slow considering it's supposed to have the same efficiency as merge sort.

III. Trade-offs

All of these sorting algorithms leave the same array, but why are there so many ways to accomplish the same goal? There are pros and cons to all of these algorithms, but if you've made it this far in life, that shouldn't be a new concept. The advantages and disadvantages of these algorithms can be seen through not only their run-time, but also how hard they are to program and the stress they put on certain parts of the machine.

Name	Complexity	Memory Usage	Run-time
Bubble	Simple	Low	Very Slow
Selection	Simple	Low	Slow
Insertion	Simple	Low	Can be fast
Quick	Complex	Low	Fast
Merge	Complex	Extremely High	Very Fast

As you can see, the first two algorithms are all relatively simple to implement, but very slow. These should only be used if you need to implement something quickly and it's only for prototyping. Insertion sort can be extremely useful and the fastest of the five if the data is already sorted. This would be used if you are inserting data into an already sorted data set. The last two are very complex and hard to debug because of their recursive nature. Merge sort is the king of sorting if the machine has the required memory capacity.

IV. Language Choice

I believe C++ was an excellent choice to get great results from these sorting methods. C++ gave us very low level control over the arrays, unlike a language like Python. Python would be a very poor choice for this kind of a task because there are no arrays in python, they are all lists. I think this program could be further optimized by using multi-threading / multi-processing in combination with the merge sort. Allowing the program to be sorting the two halves of the data set simultaneously would vastly improve performance for larger data sets.

V. Downfalls of this Analysis

Empirical analysis is relatively easy for this kind of a task, but for much larger, more complex systems it can be extremely difficult. Doing empirical analysis on these algorithms did reveal the fact that although Bubble and Selection sort have the same Big-O run-time, but Selection sort is significantly faster. Mathematical analysis would also not reveal how quick and merge sort leverage CPU and memory differently.