Data Structures Assignment 6 Paper

Matthew Parnham
2287511
parnham@chapman.edu

December 14, 2018

1 Sorting Algorithms

Each sorting algorithm was far different than I expected in execution time. Knowing that bubble sort, selection sort, and insertion sort have a runtime of $O(n^2)$, I expected them to perform similarly. They did not, however, and I now see why. Bubble sort took on average 3000 microseconds to completely sort my test data set of 1000 numbers. Selection sort on the other hand only took around 2500 microseconds to complete. This is probably due to the fact that bubble sort performs exactly n^2 iterations while selection sort performs $n^2 - n$ iterations. This will also lead to less swaps.

Insertion sort was the algorithm that surprised me the most. Even though the runtime of insertion sort is also $O(n^2)$, the algorithm completed sorting in 650 microseconds on average. That's almost 5 times faster. I looked into why this occurs and found that insertion sort is considerably faster on the average case than bubble and selection sort. It only reaches $O(n^2)$ on its very worst case, which is when the data set is completely reversed, as this will result in the maximum number of shifts in the "while loop" portion of the sort, or simply n^2 iterations.

The obvious fastest algorithm was quick sort. With an average sorting time of about 80 microseconds, this algorithm really took the cake, as expected with its $O(n \log n)$ runtime. When picking a sorting algorithm, the decision really comes down to time versus space. If hardware is not an issue, quick sort would be the definite choice. On some sort of embedded system, however, insertion sort is a great option. It isn't significantly slower than quick sort, but its a much simpler algorithm to implement. The only times I could really see myself using bubble or selection sort are when I need to sort some arbitrary collection of numbers for my own purposes, and time is not an issue.

Since this is empirical and not mathematical analysis, things like language choice and hardware choice are of value. Using a compiled language like C++ that runs natively on my operating system (Windows), I likely had shorter runtimes than using an interpreted language like Python. I also have far more powerful hardware than necessary to sort 1000 numbers, so I likely found faster runtimes than one might on a lower-end system. These are all shortcomings of empirical analysis however. All of these issues are naught when comparing the raw asymptotic analysis of each algorithm. I also wouldnd't have needed to implement and create a test environment to collect runtime data. It was still a fun experience though!