Assignment 6 Analysis Report

Krishna Narayan
CPSC 350-03
12/19/20

The time differences between the 5 sorting algorithms were much more drastic than I expected. Quick sort and merge sort were consistently operating at much quicker speeds -- their times were always a full power of 10 faster than insert, selection, and bubble sort. I've never timed any programs before, as I've never written something so large where runtime is noticeable. As a result, seeing such a prevalent time difference was quite a shock.

While quick sort and merge sort were clearly much faster, they were much more difficult to implement. Both of them required two separate functions (quicksort required a quicksort function and partition function, mergesort required a mergesort function and a merge function) and both of them required a driver class. Furthermore, they both process arrays recursively and require the swapping of array elements using pointers. Between insertion, selection, and bubble sort, insertion sort was consistently faster when sorting 1000 elements. Bubble sort was consistently slower than the other two sorting methods. On smaller scales, however, the difference in time between the three was not noticeable. Furthermore, they all have relatively simple, self-contained methods of implementation.

The language chosen for this assignment was C++, possibly leading to faster runtimes given C++ only uses a compiler and not an interpreter. Having chosen language such as Java or Python would have led to slower runtimes, given they are both compiled and interpreted. Therefore, C++ is an optimal choice. That being said, one could argue that a language such as Fortran might be faster, as its compiler is arguably better optimized, given its much older and has undergone more development.

While providing promising insight into the drastic differences in runtimes of sorting algorithms, this empirical analysis took time to implement and run, and this implementation might not be completely optimal. Furthermore, conducting this experiment with extremely large datasets would mean extreme runtimes that one can't wait through. Mathematical analysis often bypasses these limitations, as it doesn't require implementation or the testing of extremely large datasets. Instead, it only analyzes the theoretical nature of input and the quality of the code.