Jessie Pao

CPSC 350-02

Write a 1 page report on your experience. Were the time differences more drastic than you expected? What tradeoffs are involved in picking one algorithm over another? How did your choice of programming language affect the results? What are some shortcomings of this empirical analysis? Include this report as a pdf with your deliverables.

For my program I imported a text file with 2400 values. When I ran the program there was no difference in the time. The sorting algorithms finished within a second and the time wasn't registered on the clock I had printed out. In fact the whole program ran within seconds and there wasn't much difference to in runtime speed for the different algorithms. I was expecting there so be a least a second difference with 2400 input items to sort but I guess the processing power of my mac is just really fast. I was expecting there to be a small difference to occur. I tested my code with a smaller number of inputs first to make sure there weren't any errors with my code. I expected there to be no difference in time for the sorting algorithms to run and my prediction was right. I increased my number of inputs to 2400 input values but even at that size the algorithm still ran really fast and the sorting algorithms were still able to finish within a second. I know that selection sort, insertion sort and bubble sort have a big-oh runtime of $O(n^2)$ so they have a slower runtime once the number of inputs grows increasingly large. Merge sort and quick sort are recursive algorithms that both have a big-oh runtime of O(nlogn). When the number of inputs grows increasingly large the runtime will still be relatively fast compared to the other three sorting algorithms. The only drawback for merge sort and quick sort is that they need extra memory to perform the sorting algorithm. For selection sort, bubble sort, and selection sort, they do not need extra space because the sorting algorithms are performed in place. C++ is a compiled language so it is directly converted to machine code so it is faster and more efficient compare to an interpreted language, such as Python, where the code is not directly converted into machine code but instead it there is an interpreter that reads and executes the code which can take longer. Some shortcomings of this empirical analysis is that the number of data values needed to show a real difference for the sorting algorithms is a lot and the data set that I used isn't a good representation of real world data. In the real world it wouldn't be worth it for a company to hire someone just to test the runtime of an algorithm. It would be a waste of money, time, and resources. Some other shortcomings is that when these algorithms are run on different devices with different processing powers the times that are logged for each sorting algorithm will be different for each machine. This empirical analysis shows the importance of knowing big-oh: Given two functions f(n) and g(n), f(n) = O(g(n)) if there exists constants c and n0 such that f(n) <= cg(n) for all n > n0.