Nikita Shetty

Analysis of Sorting Algorithms

The time differences between the bubble, selection, insertion, merge, and quick sorting methods was actually much more drastic than I expected, especially with a large set of numbers to sort. When I sorted 40,000 generated numbers, bubble sort took 7.54 seconds, selection sort took 3.83 seconds, insertion sort took 3.17 seconds, merge sort took 0.04 seconds, and quick sort took 0.02 seconds. That is a huge difference between bubble and quick sorting, so it's really interesting to actually see the different runtimes we've been talking about in a theoretical sense in class. Even though merge and quick sorting algorithms are slightly trickier to implement, it makes much more sense to use those algorithms especially if you have a lot of data to sort. However, if there are very few numbers, the difference in runtime is very small and bubble or selection sort would be the better option to code. Bubble sort's runtime for large data sets is poor, with a runtime of O(n^2), but it's pretty quick and simple to code. Selection and insertion sort both also have runtime O(n^2) and require less swaps than bubble, but insertion is runtime O(n) if the data is already partially sorted. Merge and quick sort are both much faster with O(nlogn) on average. However, merge sort requires O(n) additional memory elements, so if space is a constraint, it is not the best sorting method to use. Quick sort's runtime is its tradeoff because in the worst case, it can have runtime O(n^2) if the pivot point selected for partitioning is the smallest or largest number. Lower level programming languages will produce slightly faster results for sorting due the code they create when compiling. For example, C++ creates optimized cpu code while Java creates byte code and Python creates interpreted code, so it takes slightly less time for the code to run as the computer does not have to translate the code as much for C++. One major shortcoming for this analysis is the generation of the numbers themselves. As you try to generate more and more numbers, it takes much more time for the program to create them. Especially going into the tens of thousands of numbers, the program can take up to a minute to generate numbers. With this analysis, it would be very impractical to near impossible to test sorting times of data sets with millions or billions of numbers.