Empirical Analysis of Sorting Algorithms

Chris Galdi

Assignment 6 was to create and test four different sorting methods to sort the same data and record the run times for each method. I chose Insertion sort, Bubble sort, Selection sort, and Quick sort. For testing, a list of 10,000 random numbers were chosen. Here are the processing times I received when each algorithm sorted the numbers.

- Insertion Sort - .0001604 seconds
- Bubble Sort - .0014928 seconds
- Selection Sort - .00019287 seconds
- Quick Sort - 8.49 e -5 seconds

As seen here, the time differences between each method were more drastic than initially expected. The Bubble sort algorithm took almost 18x longer than Quick sort to sort the same amount of data. Despite learning that there exists a time gap between those two specifically, I did not expect the time gap to be drastic when compared to each other. While programming each algorithm, there was a clear correlation between the ease of programming versus the speed of the method. The easiest ones to program were the slowest to run, which demonstrates the trade-off of using each algorithm. Quick sort would be easy to write but was more difficult to get it running. Based on that, there is no reason not to use a fast method. The speed difference between different programming languages was not tested, due to only writing the program in C++. Based on my knowledge, i would assume C++ would provide the fastest results because of the language's simplicity when compared to Java or other OOP languages. Also, the relative speeds between each sorting algorithm would scale with each language, so the ratios would be the same. I calculated the time taken for each method at runtime with empirical analysis, because it gives an actual time that the program takes. A mathematical analysis can only provide a very rough estimate of it. Unfortunately for empirical analysis, the time taken will change at each runtime due to several other factors such as other programs running on the same machine, which means there is no true value for an empirical analysis. Also, these differences can compromise the data we collect on the run-times of each sorting method, but the differences between each method was great enough for this to not be an issue. In terms of CPU being affected, the Bubble Sort function was the sorting algorithm that spiked my CPU, and it spiked it by about 3x the lowest one, Quick Sort did. Selection and Insertion sort both were around a 1.5x spike in the CPU when being compared to Quick Sort. I found it interesting that the correlation between processing time and CPU spike correlate with each other.