Empirical Analysis of Sorting Algorithms

Andre Perkins
Chapman University
5/12/17

I. Introduction

In this assignment I was able to implement Quick sort, bubble sort, merge sort, and Insertion sort. I timed the algorithms using the clock() which is a built in library to time starting and endpoints in milliseconds. I was able to run these algorithms in different conditions (different sizes text file input).

II. Results

Bubble Sort	0(n^2)
Insertion Sort	O(n^2)
Merge Sort	O(n log n)
Quick Sort	O (n log n)

What I had found was that Bubble Sort on average had a time of about 9 -10 seconds with a data set of 100K. Insertion Sort had a run time of 3-5 seconds. Quick and Merge sort were both under 1 second. The time differences to me were pretty drastic seeing how efficient these algorithms are. It was a great presentation of the power of Merge and Quick.

III. Conclusion

Completing this assignment, I was able to understand the huge tradeoffs in picking certain algorithms based on the situation. Bubble Sort and Insertion sort both struggle to perform with large data sets while Merge and Quick are able to quickly sort them with no problem. It was a great visualization to understand when and where to use these algorithms.

Using C++ for implementing these algorithms was interesting. I noticed that in C++ you are unable to directly get the size of an array like in Java ray.size();. Instead I would have to pass in the size through the function so that it understands the size of the array given and the data inside of it. In any other cases I would prefer to use Java to implement these algorithms.

The shortcomings of this analysis were the various data sets that I had used. These data sets (text files) were much different in size and the type of double inside of the file. Smallest set was 90, largest was around 3 million.