Sorting Algorithms Report

The time differences between the various sorting algorithms were indeed more drastic than I expected. I generated 200,000 random doubles to test, and the BubbleSort algorithm took 5 minutes to sort, InsertionSort took 2 minutes, SelectionSort took 1 minute, and QuickSort and MergeSort took less than a second. I have a slow computer so I expected BubbleSort to take a while, but I expected it to take maybe a minute at most. I was also surprised at how fast QuickSort and MergeSort actually are with large data sets.

The main tradeoff between choosing between algorithms with O(n^2) runtimes and algorithms with O(nlogn) runtimes is the complexity in implementation vs. the faster runtime. If you aren't working with large data sets, one of the O(n^2) algorithms would get the job done if you wanted to implement something quickly and easily. Between MergeSort and QuickSort, if space complexity is an issue, QuickSort would be a better option because MergeSort partitions data into multiple arrays.

Using c++ to implement these sorting algorithms affected my ability to use MergeSort and QuickSort. Some languages don't allow you to use recursion, so you would never be able to implement either algorithm.

The biggest shortcoming with empirical analysis is that implementing multiple algorithms takes a lot more time and effort than simply analyzing the Big-O runtimes. It was also not the most precise because my program only analyzes the runtime down to the second. Analyzing large datasets was also difficult because of how long it takes to generate random data. I attempted to generate a million random numbers to test, but my computer was too slow for me to run the test in a reasonable amount of time.