Before I started this assignment, I had done research on the different mathematical analysis of the chosen sorting algorithms: itemize

Bubble Sort - $O(n^2)$

Insertion Sort - $O(n^2)$

Quick Sort - $O(nlog(n))$

Merge Sort - $O(nlog(n))$
I was defintely expecting Bubble Sort and Insertion sort to be slower than Quick Sort and Merge Sort. The results were defintely unexpectadly drastic. The output of an example run is: 32046 9224 425 647 Bubble Sort:

- Elapsed time: 32046 nanoseconds

Insertion Sort:

- Elapsed time: 9224 nanoseconds

Quick Sort:

- Elapsed time: 425 nanoseconds

Merge Sort:

- Elapsed time: 647 nanoseconds

This test result was from a list of 2451 in length. Bubble Sort clocked in at an impressively slow 32046 nanoseconds, Insertion at 9224 nanoseconds, Quick at 425 nanoseconds, and MergeSort at 647 nanoseconds. Thus Quick sort was a full 75.4 times faster! The gap becomes even more extreme if the list is increased to a length of 71319
Bubble Sort:

- Elapsed time: 31348700 nanoseconds

Insertion Sort:

- Elapsed time: 5882090 nanoseconds

Quick Sort:

- Elapsed time: 15072 nanoseconds

Merge Sort:

- Elapsed time: 19308 nanoseconds

This time around, QuickSort and MergeSort are very close in speed, while BubbleSort was running at an impressive 73761.6 times slower than QuickSort!!! * For BubbleSort, the tradeoff is extremley inefficient speed but has a space complexity of O(1) * For InsertionSort, the tradeoff is inefficient speed but also has a space complexity of O(1) * For MergeSort, the speed is very impressive but has a space compelxity of O(n) * For QuickSort, the speed if very impressive but has a space complexity of O(logn)

The differences in time are probably highlighted even more simply from the fact that the program is written in C++, thus not too much overhead code is under effect and the results are almost purely based off of the efficiency of the algorithms. Thus doing empirical analysis with C++ is more accurate than doing the same empirical analysis with higher level languages. Empirical Analysis though, no matter the language, has a lot of shortcomings. The biggest of all is the time issue. To get very accurate results, we would have to use a better random generator than srand() and a list of a much greater size. The much larger list would cause the time of the execution to be ridicuously long and would require a lot of computing power. If I would expand on this project I would examine sorting algorithms with O(n+k) such as Bucket, radix, and Counting Sort and see how drastic their time difference is compared to O(n^2)