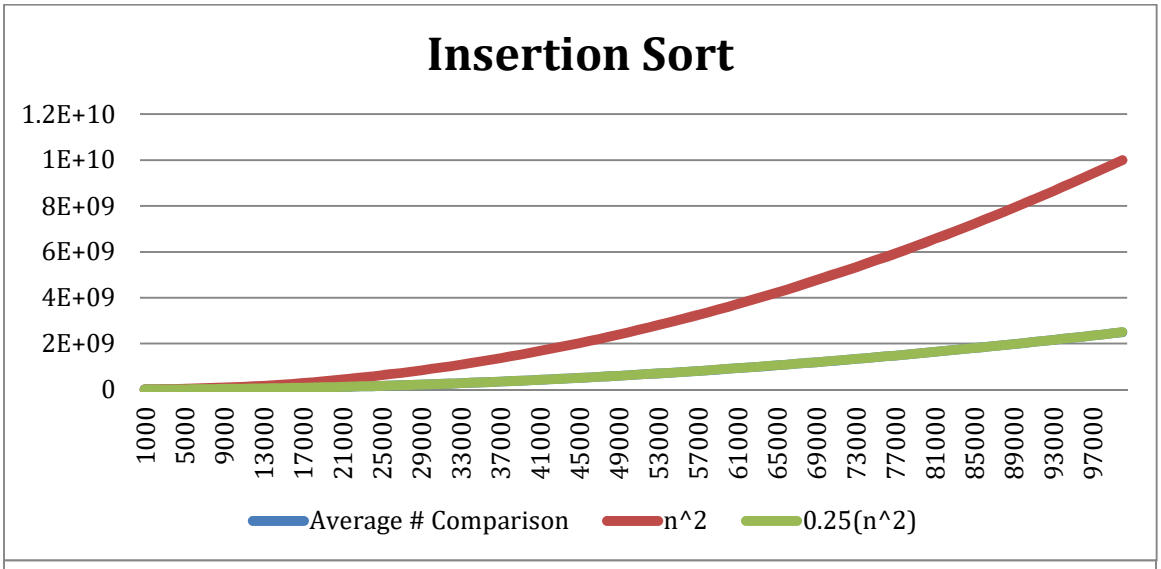
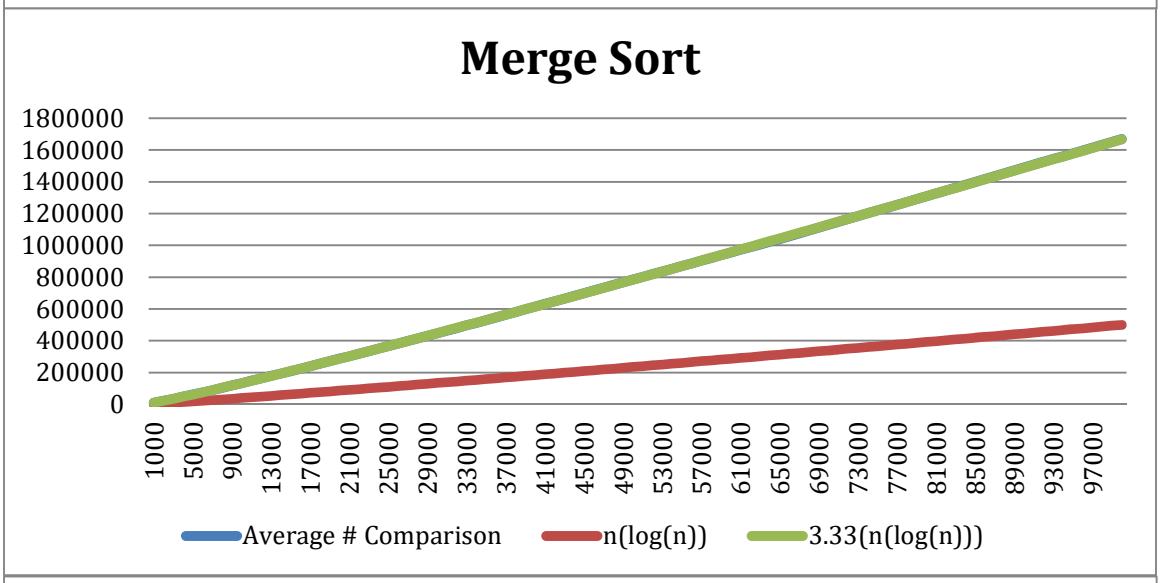
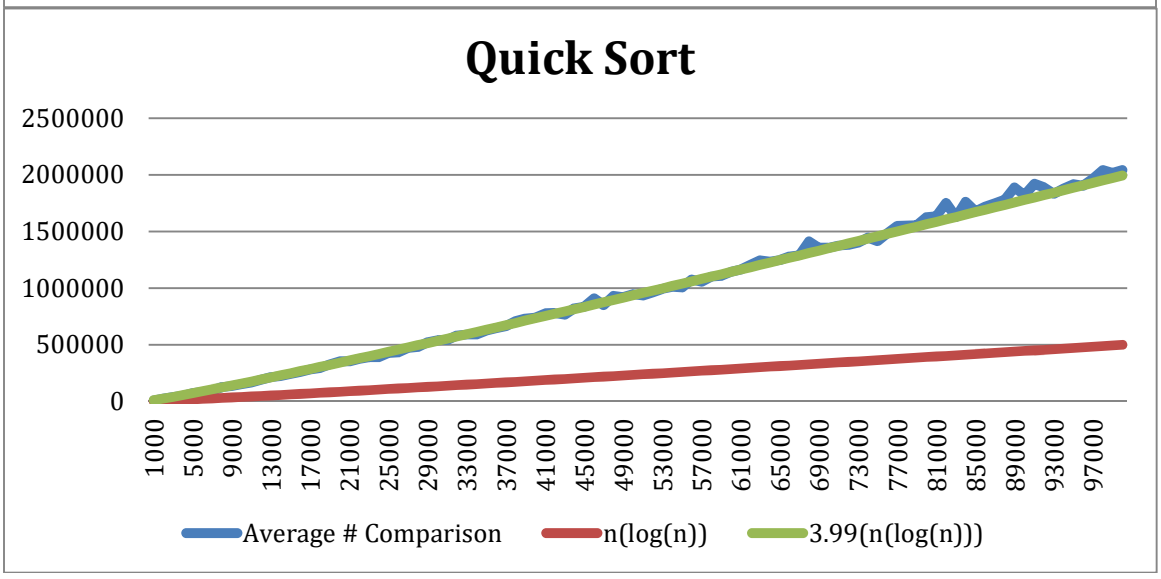
Rebecca Wilson & Setul Patel
Project 3 Report | Dec 5, 2015







This project focused on the implementation and time complexity of three major sorting algorithms: Insertion sort, Merge sort, and Quick sort. For this project, we implemented Quick sort by using the element in the last index as our pivot point and using the standard, non-randomized partition. To do our experimentation for this project, we used three C++ files: sort, DataGenerator, and getAverage. The sort file included methods to complete Insertion sort, Merge sort, and Quick sort, and a main method. The main method used the command line flag to determine which sorting algorithm to use and passed in the array of integer given as command line arguments. The DataGenerator file also included methods to complete Insertion sort, Merge sort, and Quick sort, and a main method. The main method of DataGenerator creates a list of a predetermined length of randomly generated numbers to be passed into the predetermined sorting algorithm. It also appends the count to a file called comparison.txt. Our makefile contains a loop that runs DataGenerator 50 times, then calls getAverage to read the number of comparisons for each trial from the file comparison.txt and average the number of comparisons for all 50 trials at that input size. These averages were entered into an excel sheet for input sizes 1000, 2000, 3000, … , 100000 for Insertion sort, Merge sort, and Quick sort.

Once we ran all of the trials, we plotted the points (input size, average number of comparison) in blue. These graphs are shown on page 1. For Insertion sort, the theoretical running time curve is $y = n^2$, shown in red. We discovered that the closest approximation to our experimental values was the curve $y = 0.25n^2$, shown in green and overlapping the blue points plotted. For both Merge sort and Quick sort, the theoretical running time curve is $y = n(\log(n))$, shown in red. For Merge sort, the closest match to our experimental data was the curve $y = 3.33n(\log(n))$, shown in green. For Quick sort, the closest match was $y = 3.99n(\log(n))$, shown in green. Quick sort varies the most from this matched curve due to some fluctuations in the number of comparisons among the larger data sets.

In conclusion, I believe that our experimental data strongly agrees with the Big-O theoretical running time expected for Insertion sort, Merge sort, and Quick sort. I came to this conclusion because we were able to find a constant $c = 0.25$ such that $y = cn^2$ matches our experimental data exceptionally well for Insertion sort and $c = 3.33$ and $c=3.99$ such that $y = cn(\log(n))$ matches the experimental data for Merge sort and Quick sort respectively. Even though Quick sort varies some from this line, inspection of the graph shows that this is due to inconsistencies in the data collection, not because $y = cn(\log(n))$ is not the best fit for that line. Based on our experimental results, I would say Merge sort is the best because it has the lowest number of comparison needed, even at data sets up to 100,000 elements. Insertion sort is clearly not the best because it has a time complexity of $O(n^2)$. Merge sort and Quick sort are in the same time complexity class of $O(n(\log n))$, but Merge sort has a smaller coefficient based on our experimental data, so it is the best.