Jayson Werth

47631094

CS 3352

**Lab 1 Report – Sorting**

Execution Stats (Avg. Timing in Seconds):

 

 

Graphs of Timing Performance:

Analysis of Results:

The results that I got were along the lines of my predictions going into the lab; I knew Bubble sort would take much longer times to sort than both Insertion and Merge, but my predictions varied a little when it came to the longer times of Insertion (which I had predicted to be lower at 100000 elements) and then the speed of Merge (which was much faster than I thought it was going to do, especially at 100000).

My Bubble Sort was still a little slower than most when asking around even after optimizing it by lowering the iteration after each successive one. This may be due to my use of vectors versus an array or even a doubly-linked list. My Merge Sort was much faster than I expected. After fleshing out the idea from class with the inclusion of a temporary vector to later copy back from, the merging portion of the sort became extremely fast. My Insertion Sort was slower at larger data set size than expected/predicted, but I did end up using a vector for this method as well rather than a linked-list which is easier to rearrange compared to the expensive re-copying that occurred during many iterations of my sort.

For the smallest data set of 10 elements, the Merge Sort, on average, was actually the slowest algorithm for each data set type, with the Bubble Sort closely followed by the Insertion Sort in terms of slowest to fastest times. However, as expected, with the set of 1000 elements, the Merge and Insertion Sorts averaged out to be around the same time as the Bubble Sort started to become the outlier of the three. By the 10000 element list, Merge was running, on average, running 10x as fast as Insertion Sort and 40x that of Bubble Sort and with the 100000 set, Merge was running 100x times faster for Insertion and 400x for Bubble. This timing data is consistent with the Big O run times for these sorts, nonetheless. With Merge with O(n log n) and Insertion and Bubble with O(n­2) for average and worst case scenarios, the differences in times shown especially in the last two sets start to become more reasonable.

In terms of the type of data set, Bubble Sort and Insertion Sort both seemed to struggle the most with the reversed data set, and Merge was a little more consistent but the purely random seemed to be the worst of the four. For Bubble and Insertion, this result isn’t too surprising when looking at the implementation I used for them. For Bubble (my implementation and just in general), n\*n number of comparisons will have to be made with this data set because it goes pair by pair and compares and switches if the left value is greater than the right; this means on the first iteration, every pair will swap unless they are duplicates and each following iteration the same thing will happen, leading to its worst case of O(n2). For Insertion, it will act similarly to Bubble, as for every comparison, the right value will need to be moved all the way to the left since it will always be a duplicate of the lowest value or the lowest value itself, which once again brings the worst case of O(n2). When it comes to Merge, its average, best, and worse case are all O(n log n) which explains the low variance between the data sets. The slightly increases time with the purely random set may be solely due to the complete randomness that lacks in the other three sets; the reverse does have a mixture of comparisons, the 20-80 set is mostly comparing duplicates, and the 30-random sorted allows for some of the sub-vectors to already be sorted, removing some of the randomness in 30% of the set. The 20-80 and 30-random sorted sets were about average in timing for all the sorting algorithms.