Assignment4 Analysis Doc

1. Who are your team members?

Me (Corinne Jones) and Lindsay Haslam

2. Do the actual running times of your sorting methods exhibit the growth rates you expected to see?

Yes and no. There are thorough explanations of what I found at each graph.

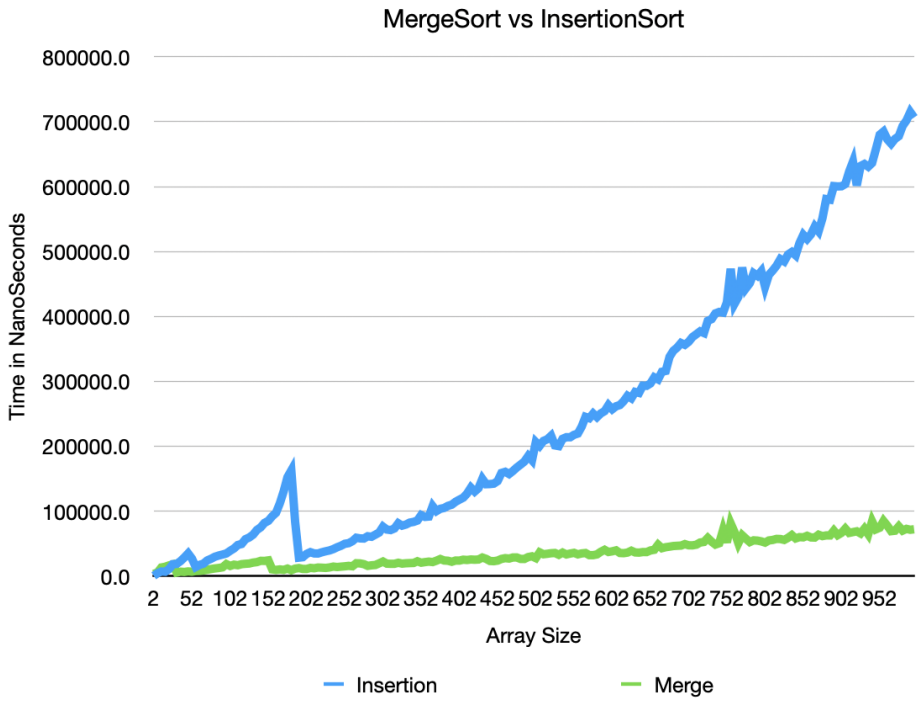
I was both surprised and unsurprised throughout different experiments. I was surprised to notice all of the noise, and couldn't help but wonder what I could have done to eliminate some of the noise. I was unsurprised to notice that InsertionSort time to sort the array did increase exponentially with array growth, whereas MergeSort increased with LogN behavior. This makes sense, considering their big-O analysis. Calculating the percentage between the two was very helpful to see at what point MergeSort began to outperform InsertionSort. It was surprising to me to see that somewhere between 10-30 is where MergeSort began to outperform, making this an ideal threshold value.

It was fascinating to see the difference between the three pivot points. Left and median seemed comparable, whereas random seemed to take more time. I would have expected median and random to be comparable, with left taking more time. This assumption was based on what we had discussed during class.

When comparing MergeSort vs QuickSort, I noticed that there were times when the two were comparable, but QuickSort was much faster during worse case scenarios. I would have expected there to be more dramatic differences from the two during best, worse, and average cases. Based on this experiment, I would choose QuickSort (median pivot) as the faster sorting algorithm, even with a threshold of 20 with MergeSort.

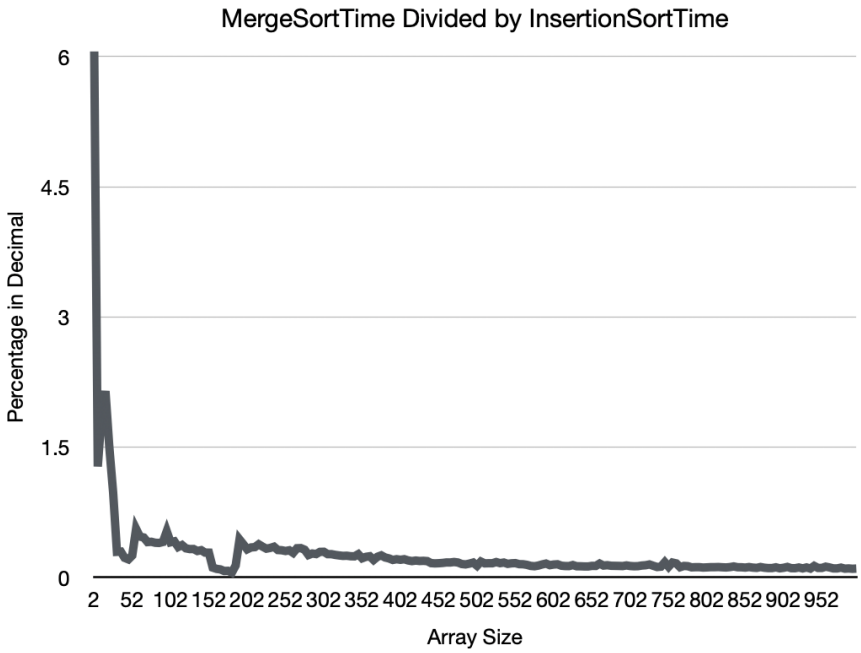
MergeSort Threshold Experiment

Graph 1



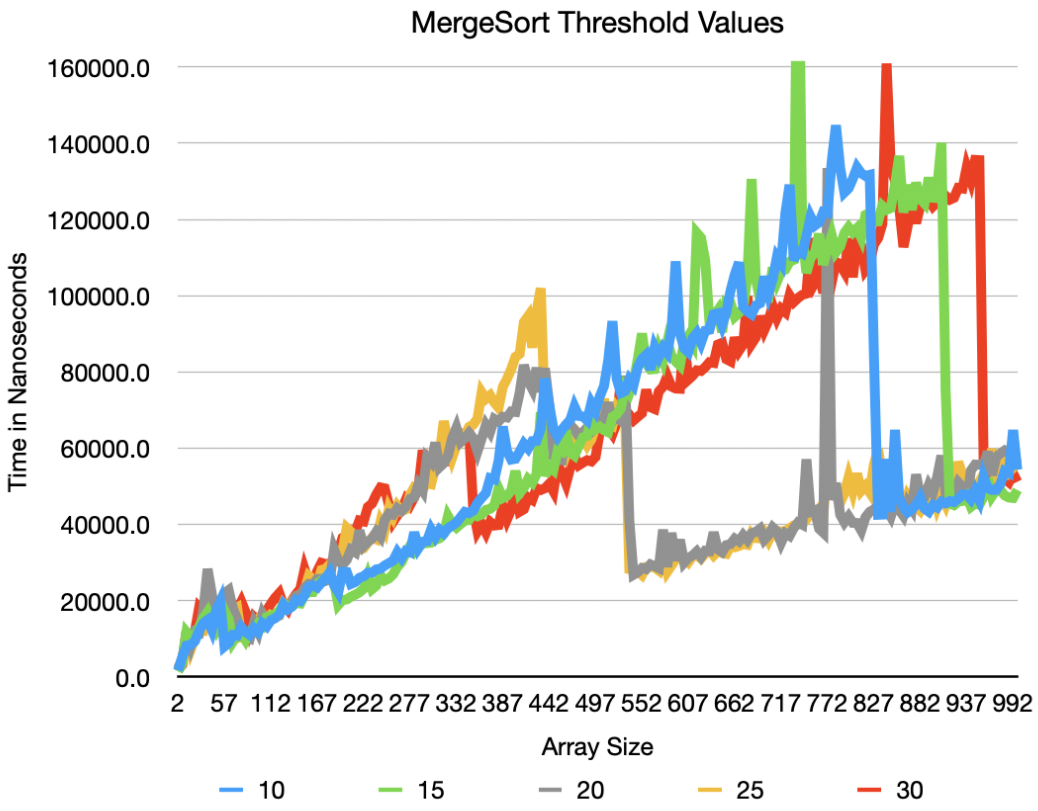
Graph 1 This graph plots the size and run time of MergeSort compared with InsertionSort. The size increments +2 over an unsorted array (i.e., copying portions of it from 2 until 1000 in increments of 2). This was performed to gauge when the size of the array becomes faster with insertion sort compared to merge sort.

Graph 2



Graph 2 This divides the runtime of merge sort and insertion sort in the previous experiment by dividing MergeSort/InsertionSort. This gets a percentage of the compared speed. When zoomed in, you can see that MergeSort begins to be faster between 10-30. I determined what individual thresholds to test in my next experiment based off of Graph 1 and Graph 2.

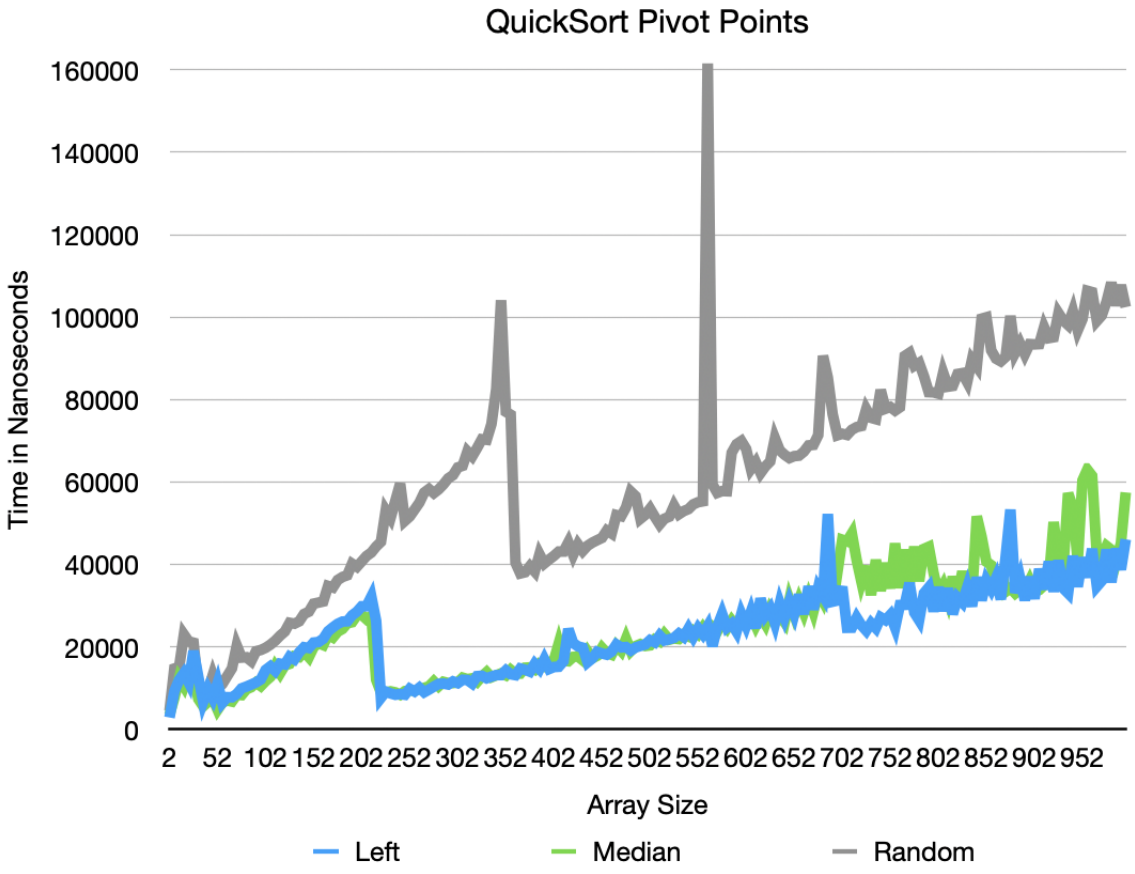
Graph 3



Graph 3 Plots MergeSort with a threshold of 10, 15, 20, 25, and 30. These numbers were chosen based on the previous experiments, which suggested this is around the array size where insertion sort becomes faster. It appears that 20 or 25 would be the best threshold value.

QuickSort Pivot Experiment

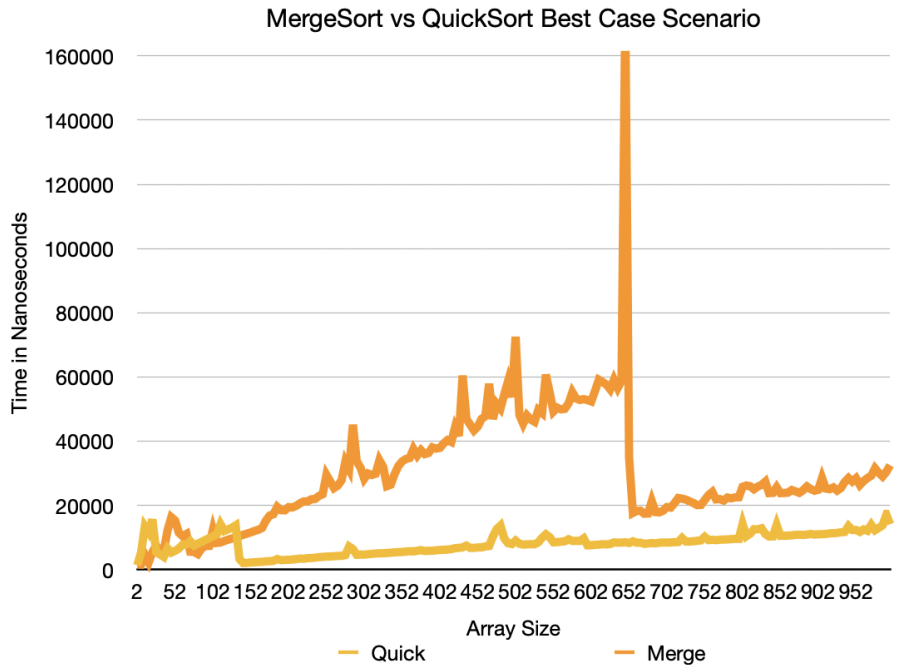
Graph 4



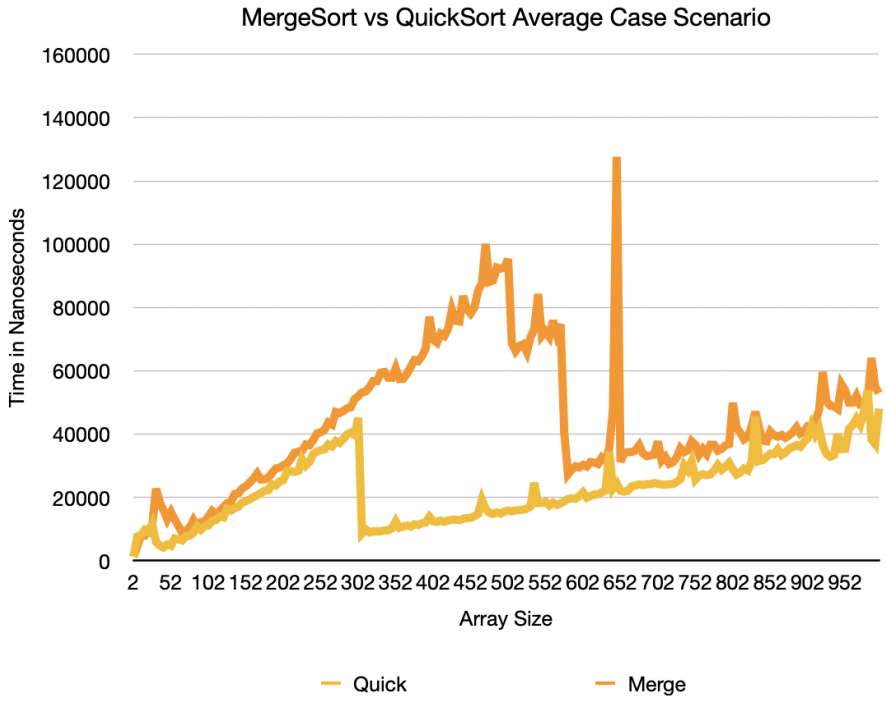
Graph 4 Plots the QuickSort algorithm at three different pivot points: the left most, the median, and a random pivot point. This was done on an “average” case array, where the numbers are in a random order.

MergeSort vs Quicksort Experiment

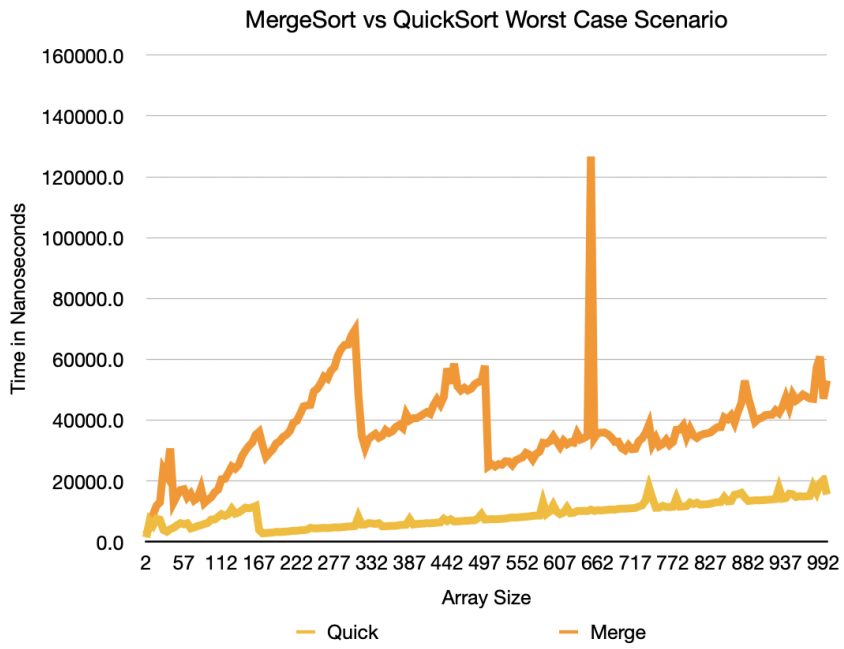
Graph 5



Graph 6



Graph 7



Graph 5, 6, & 7 Demonstrate MergeSort (threshold of 20) and QuickSort (median pivot value) during a best case (i.e., ascending order), average case (i.e., random order), and worst case (i.e, descending order) respectively. Overall, QuickSort appeared to be faster during the three different cases. During the best case scenario, MergeSort appeared to be close with QuickSort, demonstrating the gap between the two might have been closed with a further increase in array size. The average case between the two were extremely close, with QuickSort being only marginally faster. With the worst case scenario, QuickSort was faster from a small array to a large array.