**Sebastian Barney**
UID **u1277271**
Date 10/06/22
CS 2420

# Analysis Document — Assignment 5

**1.Discuss the pair programming experience:**

We didn’t have to spend a whole lot of time on the assignment, we mostly had all of the methods implemented on the first day. Together, we likely spent 5 hours on the assignment. We both had limited time this week, so we managed to work a bit individually, which I liked. Independently we work just about as effectively as working together.

I’ll definitely keep working with her

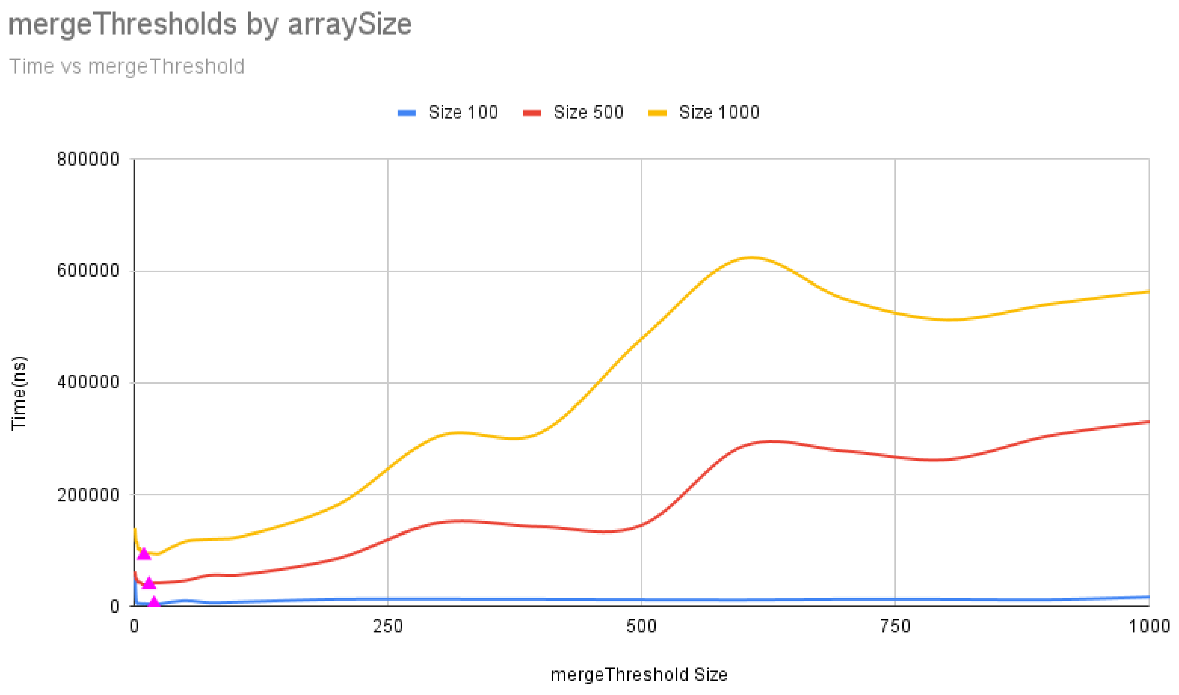
**2.What are the expected growth rates of merge sort in the best, average, and worst cases? How does the ordering of the list to be sorted affect the running time of merge sort?**

mergeSort should always be nLogn, unless implemented with an insertionSort threshold, in which case it can get to n^2 in the worst case. The order of the list never has an effect on the runTime of mergeSort since its best and worst case take the same number of operations to complete.

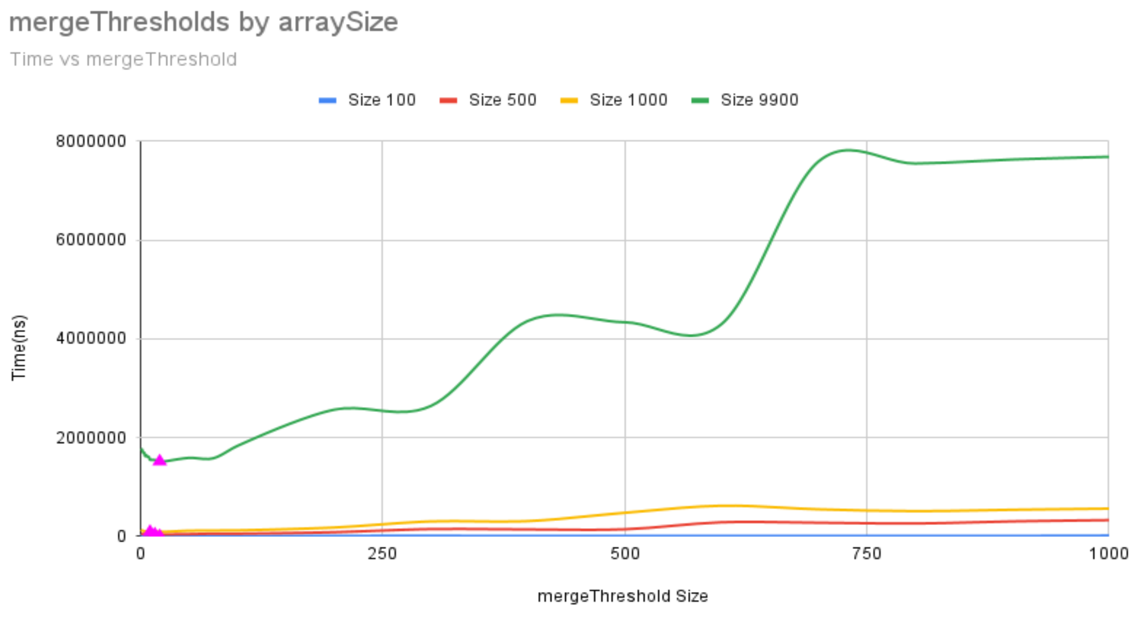
**3.Explain how you invoked insertion sort when the threshold for small sublists in merge sort was reached.  In particular, what did you do to ensure that you are not invoking insertion sort for the entire list being sorted?**

Our insertionSort method sorted between a starting index and ending index of an array. When called it will sort only between those two indices. We basically just added an if statement in our mergeSort method that invoked an insertionSort between the two indices only if that threshold was reached.

4.**<u>Merge sort threshold experiment:</u> Determine the best threshold value for which merge sort switches over to insertion sort. (Use large list sizes, but not so large that you cannot collect the timing data in a reasonable period of time.) To ensure a fair comparison, use the same set of permuted lists for each threshold value. Plot the running times of your merge sort for five different threshold values on permuted lists (one line for each threshold value). In the five different threshold values, be sure to include the threshold value that simulates a full merge sort (and identify that line as such in your plot).**



We actually tested from thresholdSize 1 to 1000. The minimum time value will correspond to the best thresholdSize for whichever size Array. In the graphs, the pink triangle shows the best thresholdSize for the Size of array. The best thresholdSize changes according to the size array you're trying to sort, which would make sense considering it would be better to mergeSort 10000 entries rather than insertionSort them. This, however didn't dramatically change, it mostly stayed constant around 10-20 for array sizes between 100 and 100,000 as shown in the graph below.



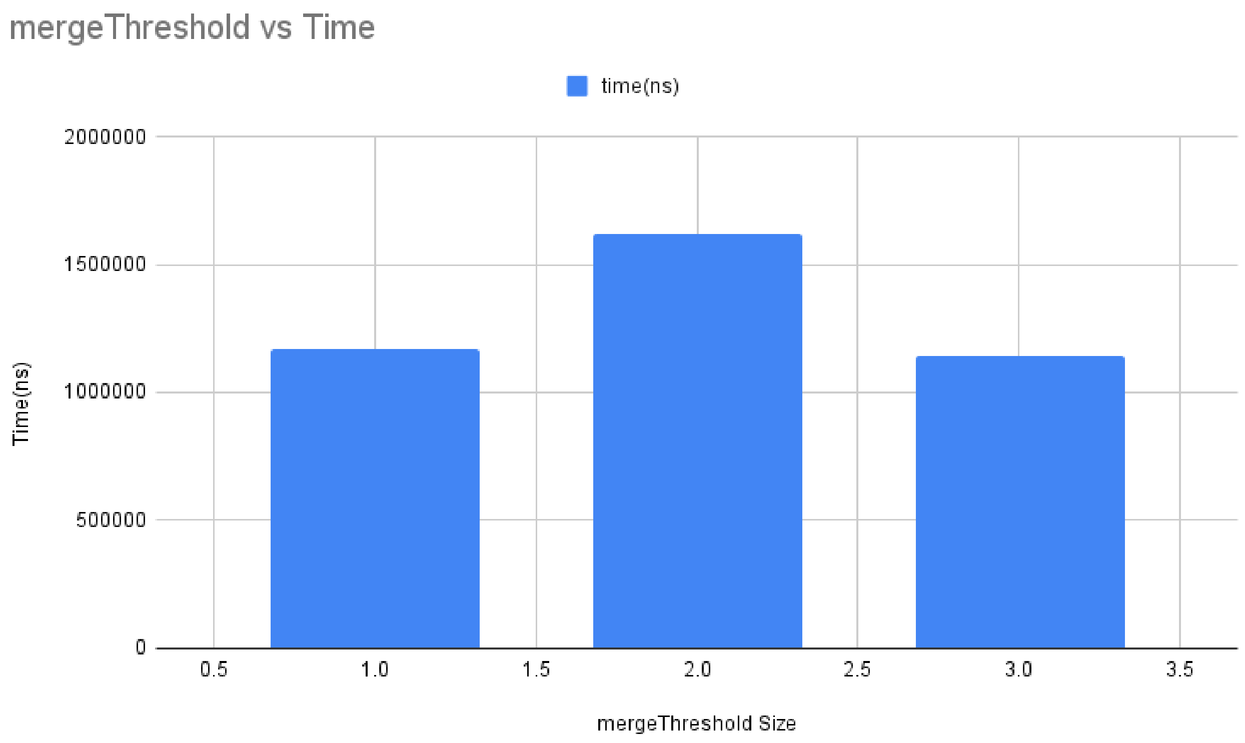
**5.What are the expected growth rates of quicksort in the best, average, and worst cases? How does the ordering of the list to be sorted affect the running time of quicksort? How does the choice of pivot affect the running time of quicksort?**

quickSort can be n^2 at worst and nlogn at average and best. The ordering of the list has an immediate effect on the runTime of quicksort since the steps are proportional to the number of inversions.

**6.Explain the three strategies that you used to choose the pivot in your quicksort implementation. What is the Big-O behavior of each strategy? How does each pivot-selection strategy affect the overall Big-O behavior of quicksort?**

quickSort can be implemented with a pivot chosen from the first index, the average of the first, middle, and last indices, or the pivot chosen at random. The random index method will be the worst BigO behavior on average, as choosing it randomly will more likely result in a choice that creates the most inversions. Choosing the first index every time will be slightly better than random, but it won't be as good as averaging the first, middle, and last indices because the value of the first index might be far from the median value of the array.

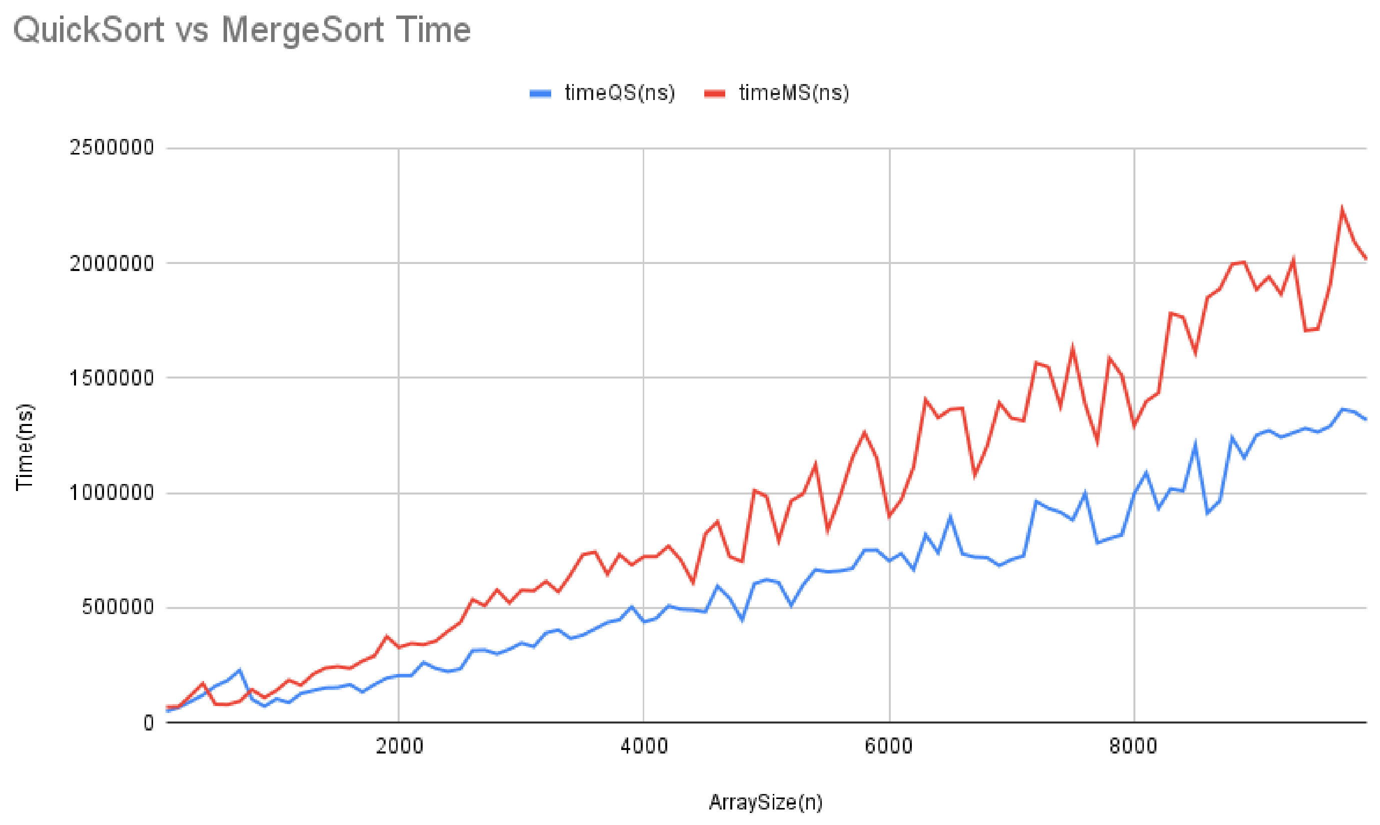
**7.<u>Quicksort pivot experiment:</u> Determine the best pivot-selection strategy for quicksort. (As in #5, use large list sizes and the same set of permuted lists for each strategy.)**
**Plot the running times of your quicksort for three different pivot-selection strategies on permuted lists (one line for each strategy).**



**(1,2,3) ------> (Average of 3 indices, random pivot, first index)**

When tested over a range of arraySizes, it was clear that the 1st method was just barely the best on average. This is to be expected, as explained above the average of the 3 indices will result in a faster quickSort on average.

**8. <u>Merge sort vs. qucksort experiment:</u> Determine the best sorting algorithm for each of the three categories of lists (ascending, permuted, and descending). For the merge sort, use the threshold value that you determined to be the best. For the quicksort, use the pivot-selection strategy that you determined to be the best. (As in #5, use large list sizes and the same list sizes for each category and sort.)**



The test was done with the same Size array, 10000 integers, a pivot method of 1, and a mergeThreshold of 20.

**9.Do the actual running times of your sorting methods exhibit the expected growth rates? Why or why not?**
**How did you determine the growth rate of the actual running times (e.g., trend of the plotted line, convergence of T(N)/F(N), or something else)?**

Yes. When the runTime was divided by the expected complexity we got a near constant value for both mergeSort and quickSort. In both cases we got a time complexity of O(nLogn)