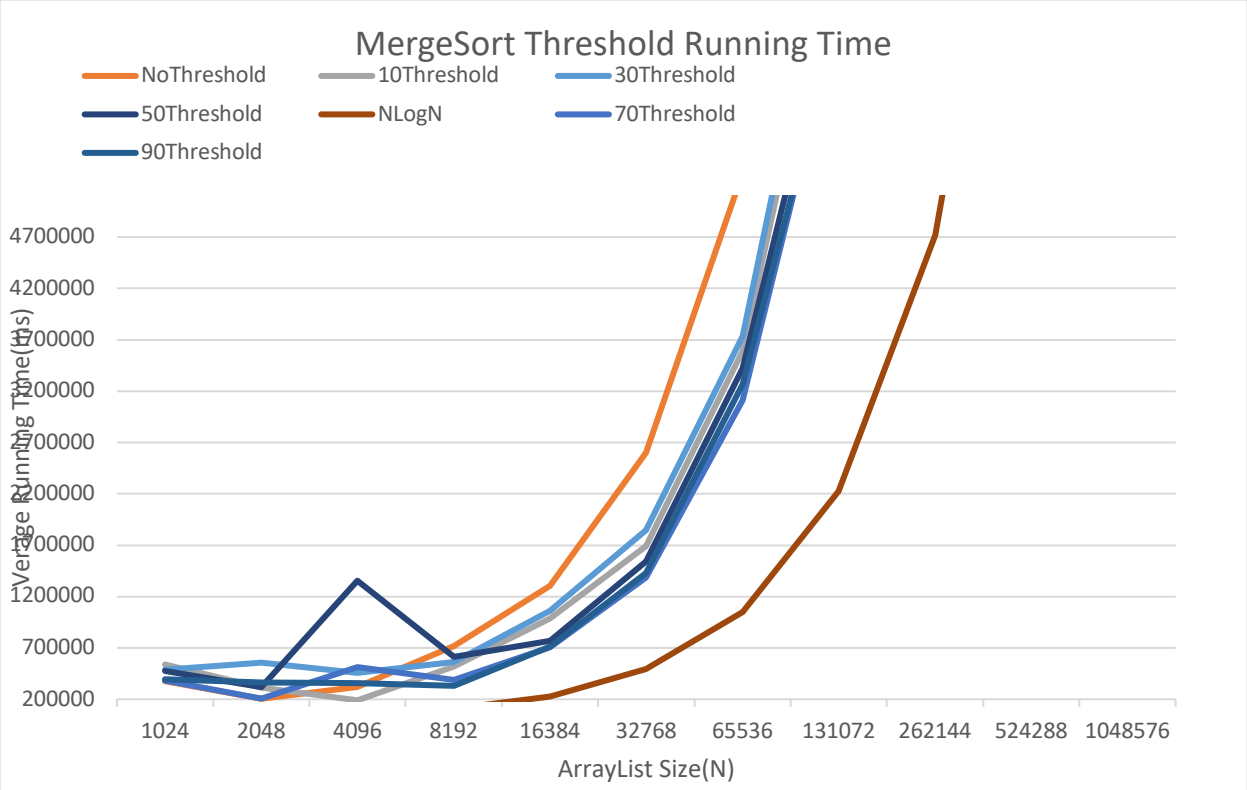
When you are satisfied that your program is correct, write a detailed analysis document. The analysis document is 40% of your assignment grade. Ensure that your analysis document addresses the following.

Note that if you use the same seed to a Java Random object, you will get the same sequence of random numbers (we will cover this more in a lab soon). Use this fact to generate the same permuted list every time, after switching threshold values or pivot selection techniques in the experiments below. (i.e., re-seed the Random with the same seed)

1. Who are your team members?
Rongmin Jin

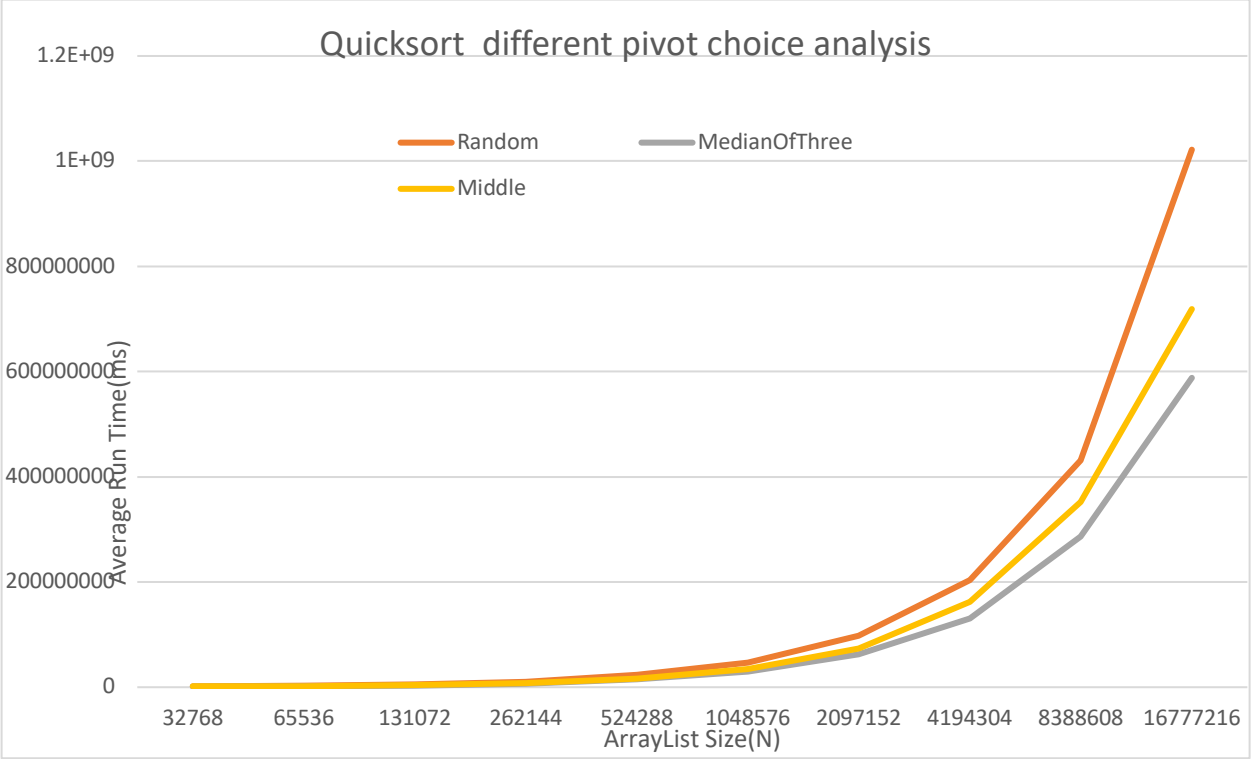
2. Mergesort Threshold Experiment: Determine the best threshold value for which mergesort switches over to insertion sort. Your list sizes should cover a range of input sizes to make meaningful plots, and should be large enough to capture accurate running times. To ensure a fair comparison, use the same set of permuted-order lists for each threshold value. Keep in mind that you can't resort the same ArrayList over and over, as the second time the order will have changed. Create an initial input and copy it to a temporary ArrayList for each test (but make sure you subtract the copy time from your timing results!). Use the timing techniques we already demonstrated, and be sure to choose a large enough value of timesToLoop to get a reasonable average of running times. Note that the best threshold value may be a constant value or a fraction of the list size. Plot the running times of your threshold mergesort for five different threshold values on permuted-order lists (one line for each threshold value). In the five different threshold values, be sure to include the threshold value that simulates a full mergesort, i.e., never switching to insertion sort (and identify that line as such in your plot).



Answer:
In the experiment, I have 100 iterates for one arraylist size. I use 10 different arraylist sizes, ranging from pow(2,10) to pow(2, 20).

From the above graph, we can see that the line with slowest growth-rate is NlogN. The shape matches the rest of lines, so the Big-O behavior is what I expected, O(NlogN) for merge-sort. In terms of deciding the best threshold for the merge sort to switch to insertion sort, we can see that when we merely use merge-sort, it has the fastest growth-rate among other lines. Then incrementing threshold speeds up the run-time until it hits threshold of 90, the run time increases. So based on the graph, 70 is the best threshold value to switch to insertion sort.

3. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksot. (As in #2, use large list sizes, the same set of permuted-order lists for each strategy, and the timing techniques demonstrated before.) Plot the running times of your quicksort for three different pivot-choosing strategies on permuted-order lists (one line for each strategy).



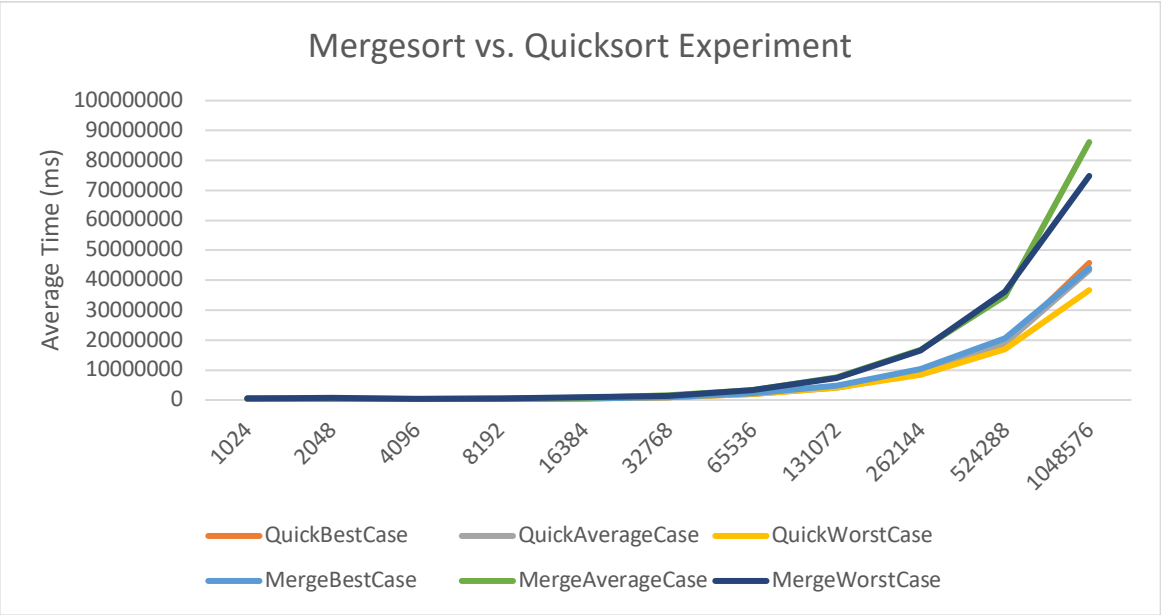
Answer:
I have chosen three different strategies: 1) generating a random int within the range of arraylist input, and let it be the pivot index, 2) choosing the middle index, position wise, as the pivot index, 3) choosing median-of-three, that is, the median of the first, middle, and last elements of the arraylist, and use the median number's index as pivot index.

I have also tested choosing the leftmost index as pivot index, however, since using it more likely results in O(N^2) complexity, and with large input size, it gives me stackoverflow exception.

Based on the above graph, median-of-three has the best run-time performance. This is due to that we want partition puts the pivot in its correct index. Choosing a median increases this possibility.

4. Mergesort vs. Quicksort Experiment: Determine the best sorting algorithm for each of the three categories of lists (best-, average-, and worst-case). For the mergesort, use the threshold value that you determined to be the best. For the quicksort, use the pivot-choosing strategy that you determined to be the best. Note that the best pivot strategy on permuted lists may lead to O(N^2) performance on best/worst case lists. If this is the case, use a different pivot for this part. As in #2, use large list sizes, the same list sizes for each category and sort, and the timing techniques demonstrated before. Plot the running times of your sorts for the three categories of lists. You may plot

all six lines at once or create three plots (one for each category of lists).



Answer:
Here I used 70 input size as threshold for merge sort, and median – of -three pivot strategy for quicksort.
In general, quicksort is faster than mergesort in most cases. For best-case when the input is sorted, mergesort and quicksort perform similarly. For average-case & worst-case, quicksort is significantly faster than mergesort.

5. Do the actual running times of your sorting methods exhibit the growth rates you expected to see? Why or why not? Please be thorough in this explanation.
Answer:
Yes. I expect mergesort for all cases will have O(NlogN) growth rate. Based on the top graph where I ploted NlogN, mergesort() method matches complexity of O(NlogN).
Based on mergesort vs quicksort experiment graph, we can see that with the pivot choice of median-of-three, quicksort() has O(NlogN) complexity for all three cases. It is important to note that mergesort and quicksort complexity do not depend on the orderness of input array.

I expected quicksort() to have nearly O(N^2) complexity when choosing the leftmost pivot for worst case arraylist input. However, with large arraylist input, I got stackoverflow exception.

Team members are encouraged to collaborate on the answers to these questions and generate graphs together. However, each member must write and submit his/her own solutions.