When you are satisfied that your program is correct, write a detailed analysis document. The analysis document is 40% of your assignment 5 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 the next lab). 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 is your programming partner? Which of you submitted the source code of your program?

My programming partner was Ben Allred. He submitted the source code for the program.

2. Evaluate your programming partner. Do you plan to work with this person again?

Ben was an amazing partner. He's smart, works hard, and plans ahead. I definitely plan on working with this person again.

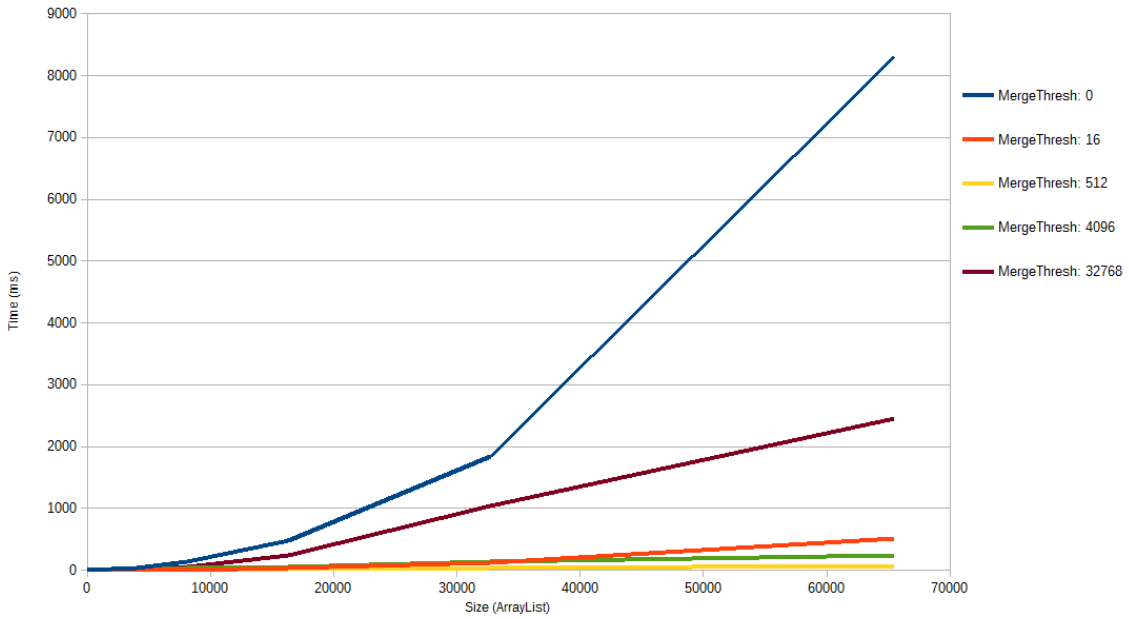
3. Evaluate the pros and cons of the pair programming you've done so far. What did you like, what didn't work out so well? You'll be asked to pair on three more of the remaining seven assignments. How can you be a better partner for those assignments?

The pros of pair programming are that I have someone I can work with. It's more efficient to collaborate and work with someone else at times. The cons are scheduling. We both have other classes and I have a full time job so finding time to meet is very difficult. I can be a better programming partner for future assignments by working ahead of time and communicating with my partner more often.

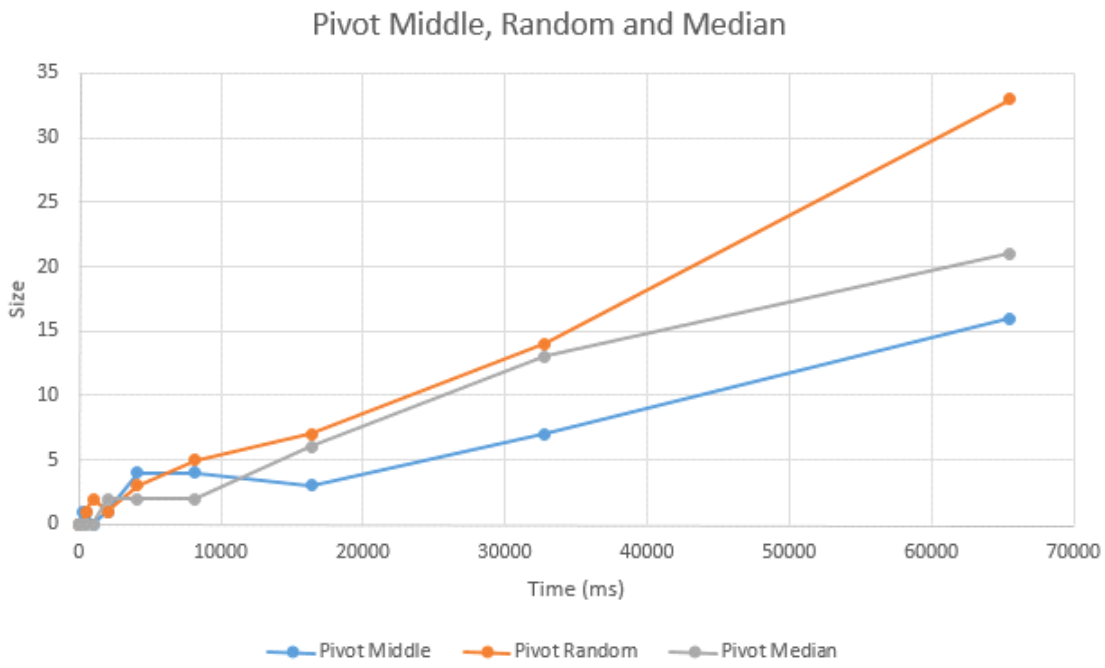
4. Mergesort Threshold Experiment: Determine the best threshold value for which mergesort switches over to insertion sort. Your list sizes should cover a range of input sizes to make meaningful plots, and should be large enough to capture accurate running times. To ensure a fair comparison, use the same 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 demonstrated in Lab 1 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.
â€¨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).

MergeThresh 512 is the best threshold value for which mergesort switches over to insertion sort. As seen by the chart, it is log(N) behavior.

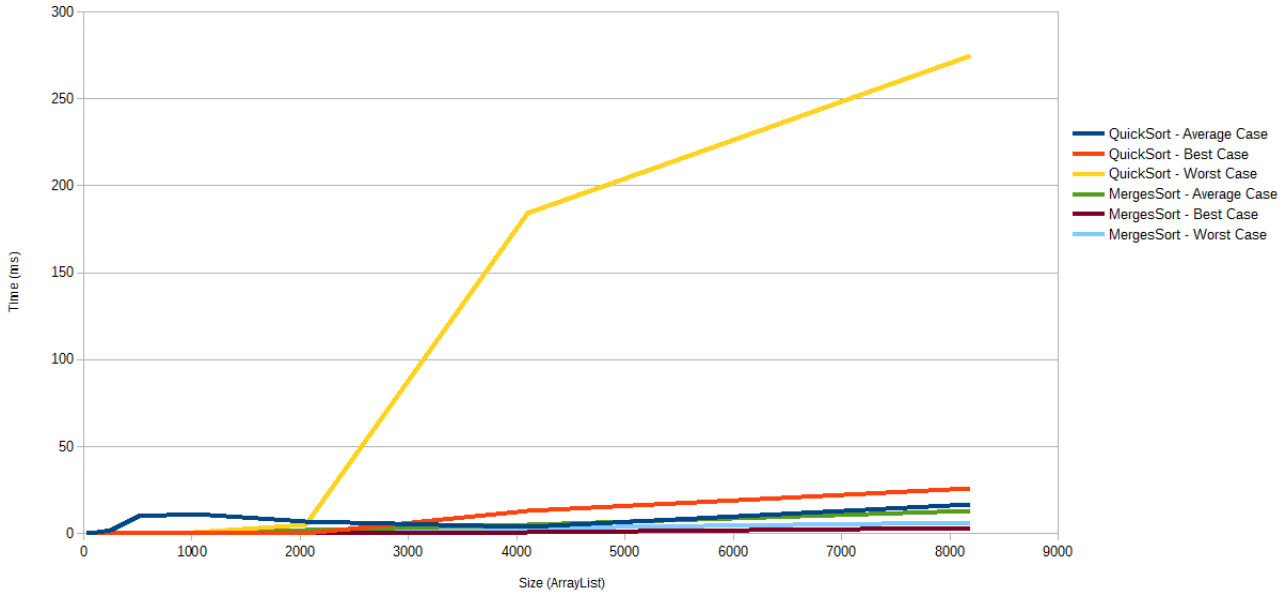


5. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksort. (As in #3, use large list sizes, the same set of permuted-order lists for each strategy, and the timing techniques demonstrated in Lab 1.) Plot the running times of your quicksort for three different pivot-choosing strategies on permuted-order lists (one line for each strategy).Â



For choosing a pivot in Quicksort. It's ideal to use the median because it results in the best case typically O (N log N). The complexity of Quicksort relies on the pivot point. Random selection tends to cause O(N^2) behavior which is the worst case. So if the pivot partions the array into two eually sized subarrays at each stage O (N log N) behavior. Average behavior is having a medium behavior which is O (N log N) behavior.

6. 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 #3, use large list sizes, the same list sizes for each category and sort, and the timing techniques demonstrated in Lab 1. 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).



Quicksort's Best Case: O(N log N)
Quicksort's Average Case: O(N log N)
Quicksort's Worst Case: O(N^2)

Mergesort's Best Case: O(N log N)
Mergesort's Average Case: O(N log N)
Mergesort's Worst Case: O(N log N)

In most cases it seems that Mergesort is more efficient overall for smaller cases since they both are O(N log N) in average case and Mergesort is also O(N log N) in the worst case. In contrast Quicksort' Best and Average Case is O(N log N) with a median and middle pivot; and O(N^2) with a random pivot (much less efficient in this case). However, Mergesort requires 2N space since it has to copy everything from the merged array back to the original takes time. Quicksort does not require any extra space; making it better for larger values.

7. 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.

The actual running times of my sorting methods did exhibit the growth rates we expected to see after several tests. These tests consisted of determining the best, average, and worst case of each of the sorting methods. However, since this assignment did consist of calling the insertion sort method, this did alter the results we expected by making the methods take longer than originally planned (typical quicksort and mergesort algorithms).

8. How many hours did you spend on this assignment?

We spent about 10-12 hours on this assignment.

Programming partners are encouraged to collaborate on the answers to these questions. However, each partner must write and submit his/her own solutions.

Upload your solution (.pdf only) on the assignment 5 page by 11:59pm on September 28th.