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?

   Logan Terry - u0973436
   My partner submitted the source code

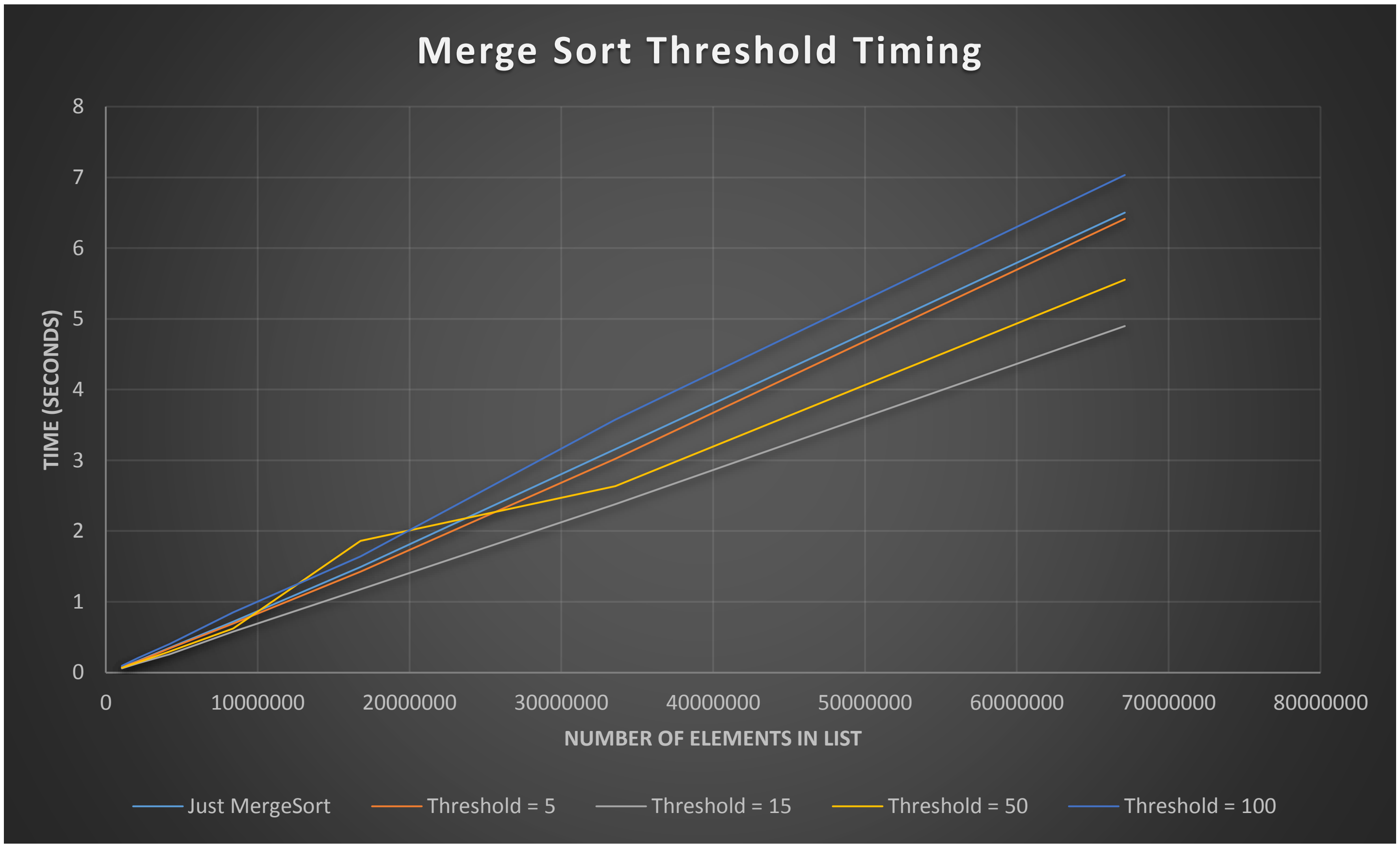
2. Evaluate your programming partner. Do you plan to work with this person again?

   My partner did well on this assignment. It is a little hard to do anything that I think would be better or work because of his knowledge and his view. He usually just wants to do everything his way. Even though it makes it hard to learn while working on this assignment, which is why I prefer just doing assignments by myself. I tend to learn more in the process.

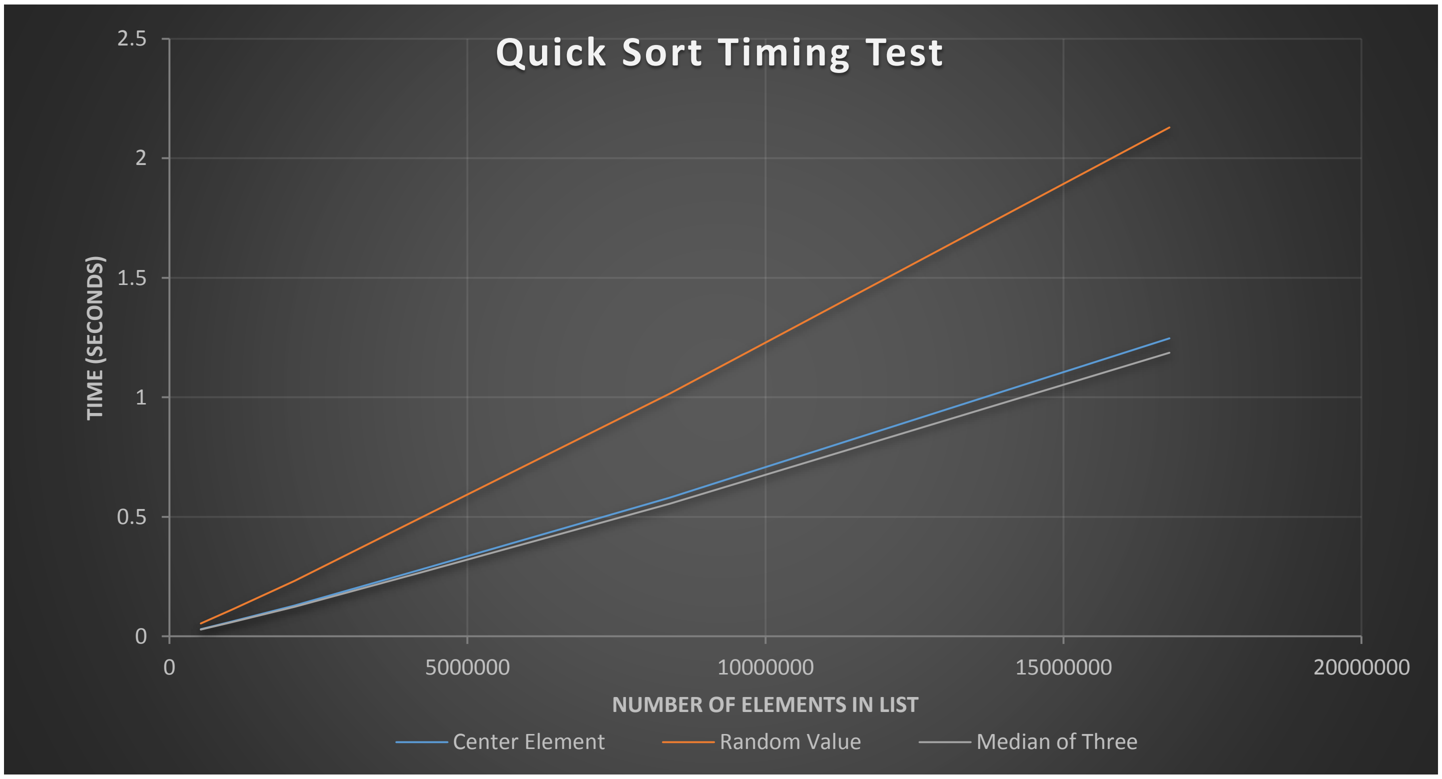
3. Evaluate the pros and cons of the pair programming you've done so far. What did you like, what didn't work out so well? You'll be asked to pair on three more of the remaining seven assignments. How can you be a better partner for those assignments?

   I do like that it seems the assignment gets done quicker when working with a partner, especially when your partner is knowledgeable about programming. However, I also find I learn less when working with a partner as noted above. It just seems like it goes by quick and when I'm unsure on something my partner is able to just figure it out but without much explanation or understanding in the process. I enjoy the struggle of trying to figure things out on my own as it stretches my thinking and capacity for learning. I guess in order to be a better partner myself, I can not only contribute maybe more to the at keyboard experience, but also help ensure my partner understands everything that is going on in the program itself.

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.
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).

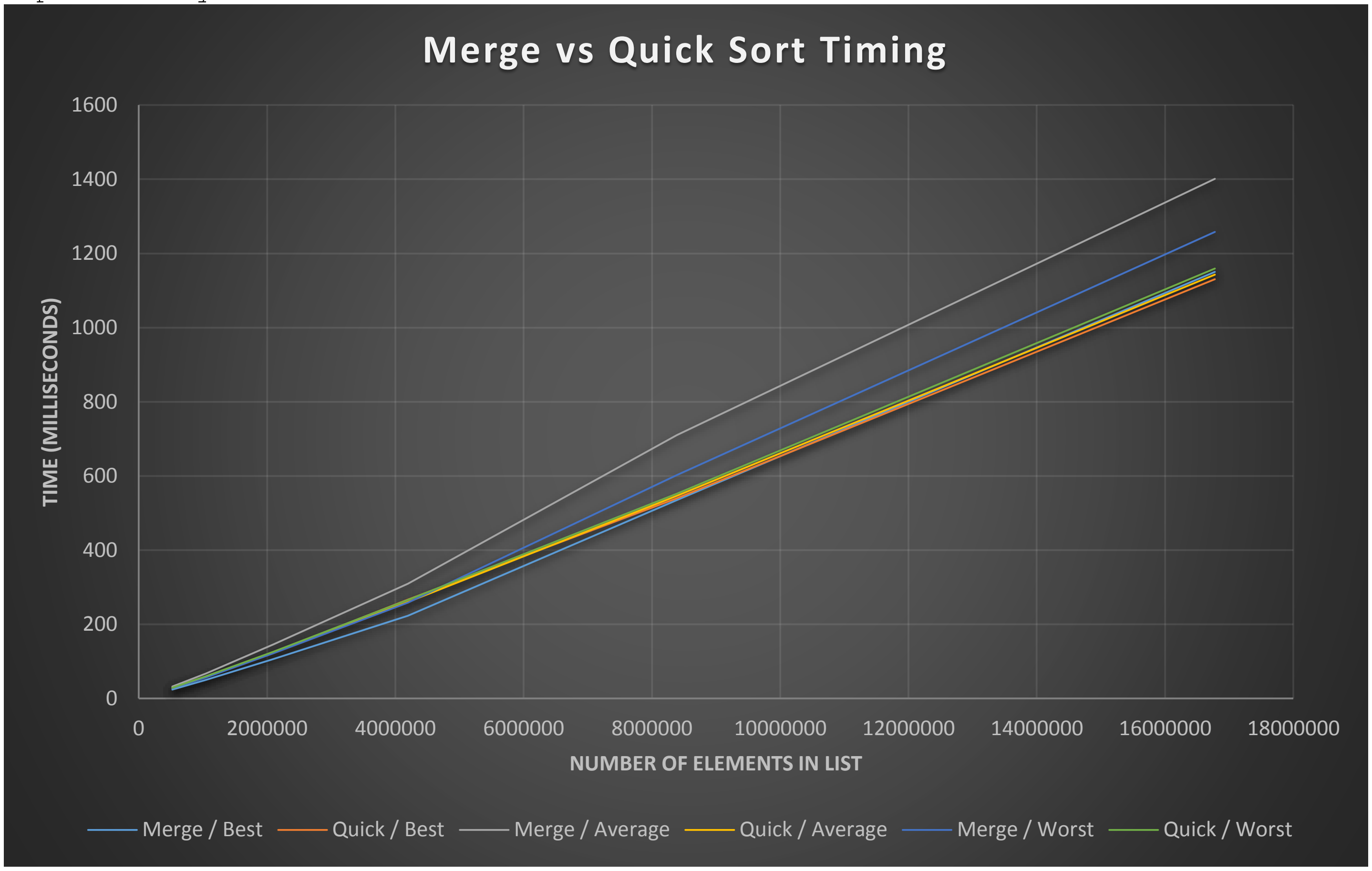


5. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksort. (As in #3, use large list sizes, the same set of permuted-order lists for each strategy, and the timing techniques demonstrated in Lab 1.) Plot the running times of your quicksort for three different pivot-choosing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



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).

I used my threshold value of 15 for mergesort testing in this step and I used the median of three strategy in my quicksort for all three tests. The following graph represents my results.



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.

They more or less showed the expected growth rates as the expected growth rate should be N Log N for both mergesort and quicksort.

The quicksort method definitely seems consistent in all three tests. This is due to my median of three strategy that I chose to implement on all three tests. If I had chosen the random number strategy it would not have been quick as clean.

The Mergesort tests are not as consistent but I assume this is the general case as it doesn’t get to have a special strategy of choosing middle elements and having to use a temp array. Also, I didn’t get the crazy borderline cases of quicksort going to N^2.

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

Approximately 10 hours. I spent about 4-5 hours with my partner and then we did all the timing and analysis individually as well as programming the three different sort methods for quicksort.

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.