Chasen Chamberlain
u0583257
Assignment 05 Analysis
CS 2420

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)

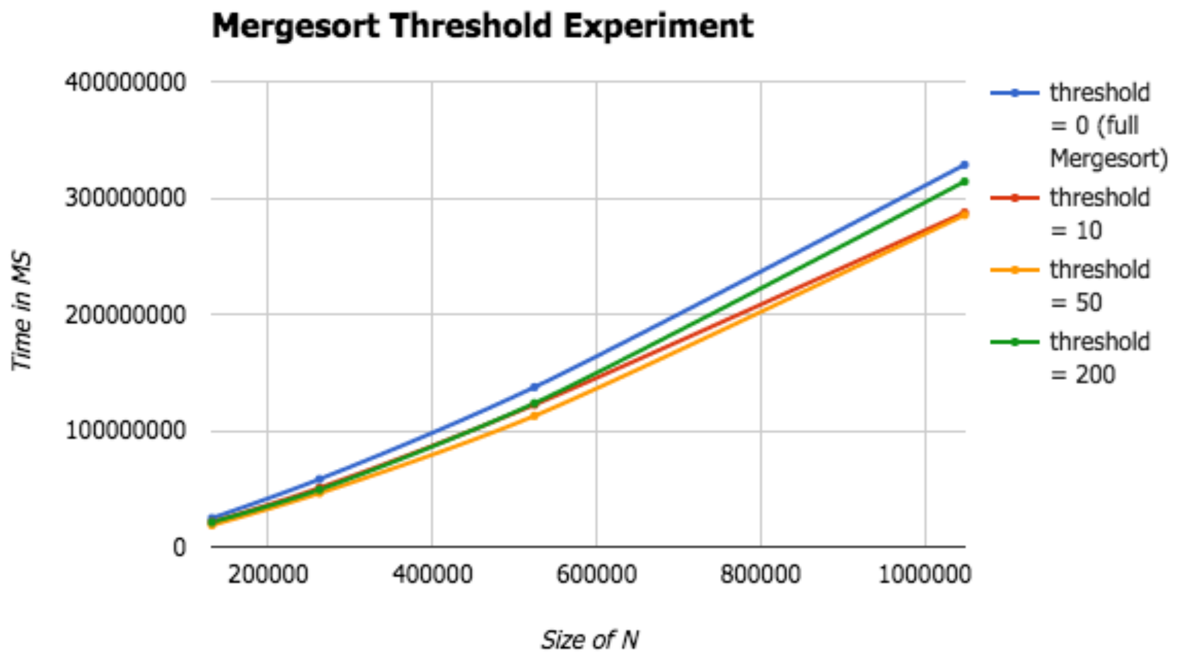
1. Who is your programming partner? Which of you submitted the source code of your program?
Rebekah Peterson. She is submitting the code.

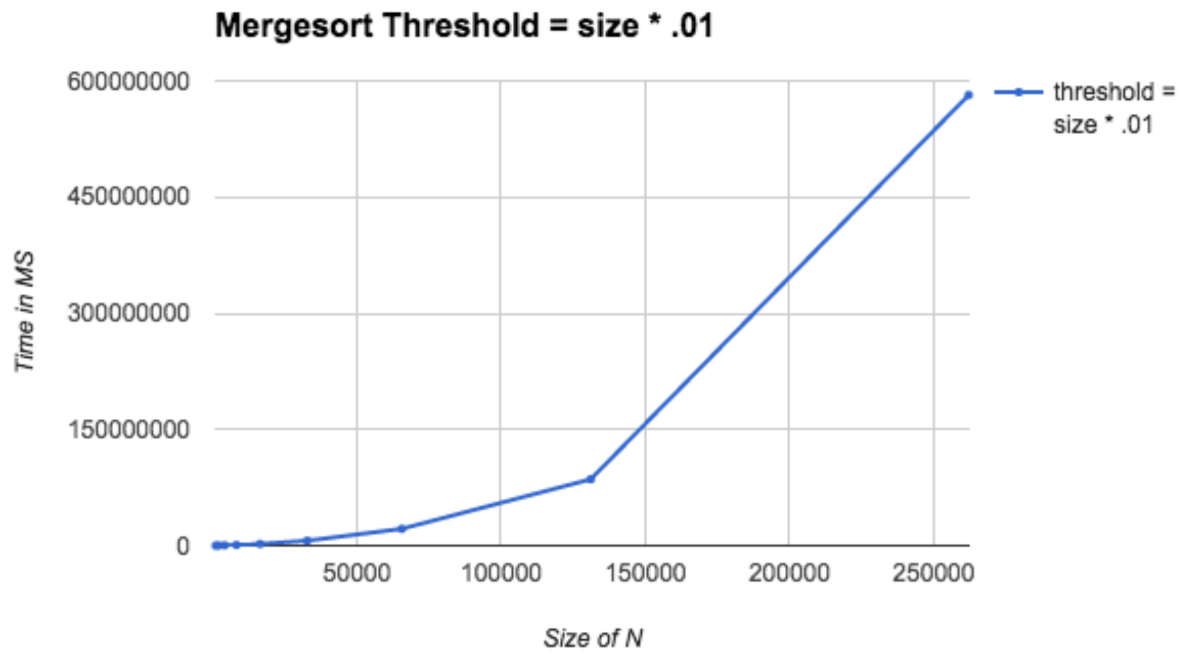
2. Evaluate your programming partner. Do you plan to work with this person again?
She did a great job of making sure the program was nice and tidy. She also was great help with debugging the details of the mergesort and quicksort.
I would work with her again on the final project if possible.

3. Evaluate the pros and cons of the 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 think pair programming is essential for larger assignments that require more work. Also it is essential to writing good code and test cases to have multiple pair of eyes on the code.
Scheduling continues to be the worst part of pair programming, it can just be hard to find someone who’s schedule matches my own.
To be a better partner for the next pair programming assignments I will try to step through debugger our loud and explain how things are working. Also just keeping communication flowing and schedules aligning.

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).

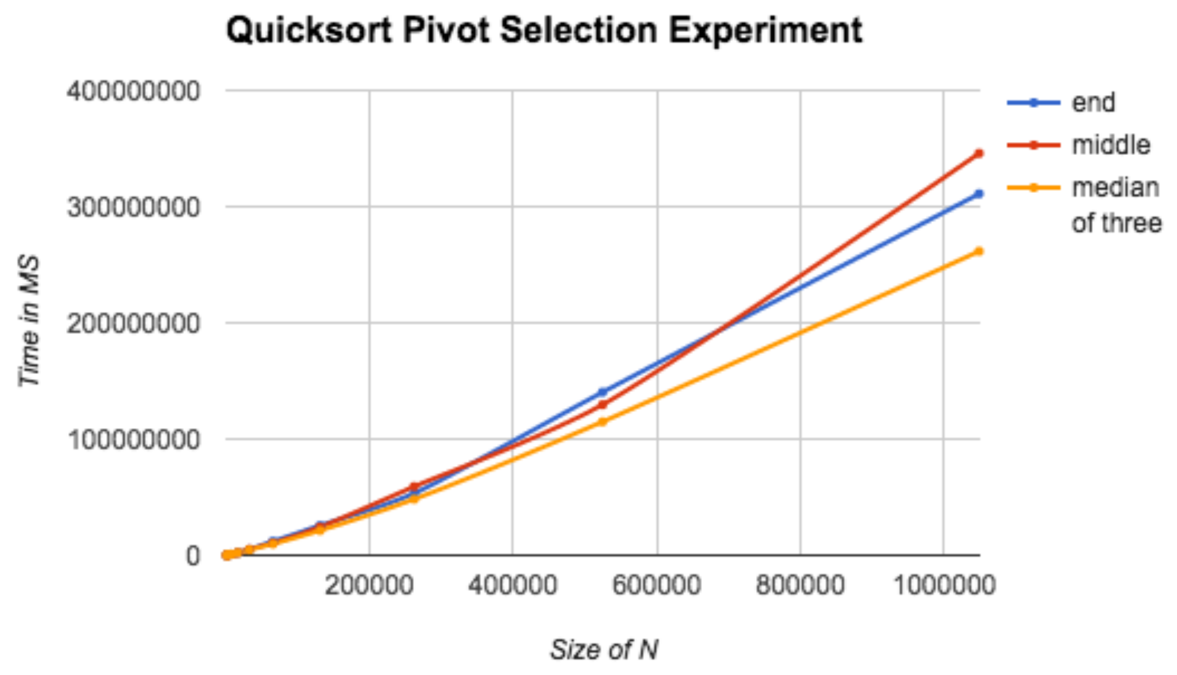




I had to split these graphs into two, because when we had the the threshold of size *.01 with the date for the other thresholds you could not see the trends of the lines because of the how bad the threshold of size *.01 performs.

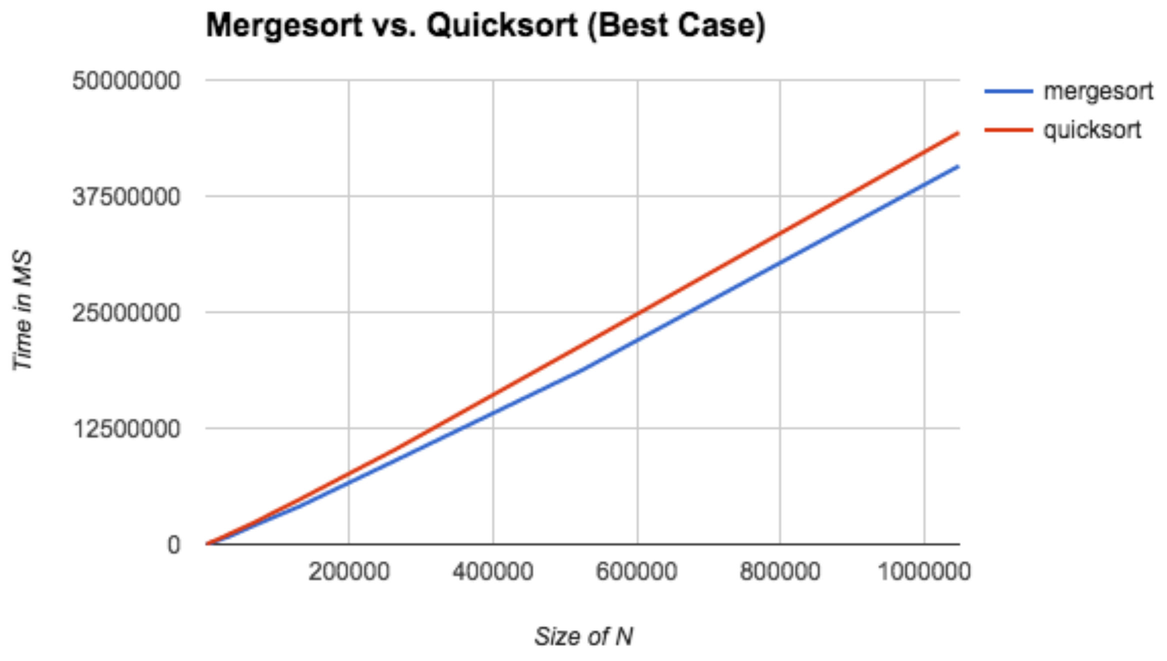
After the tests were done we determined that the best threshold was 50. To have one lower would increase the complexity and to have one higher would also increase the complexity. With Insertion sort being the determinate factor of when this threshold was triggered it makes sense that its a low number, but not insanely low to where mergesort did all the work, but not to high where insertion sort had to do much work and resulting in bad complexity.

5. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksot. (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).

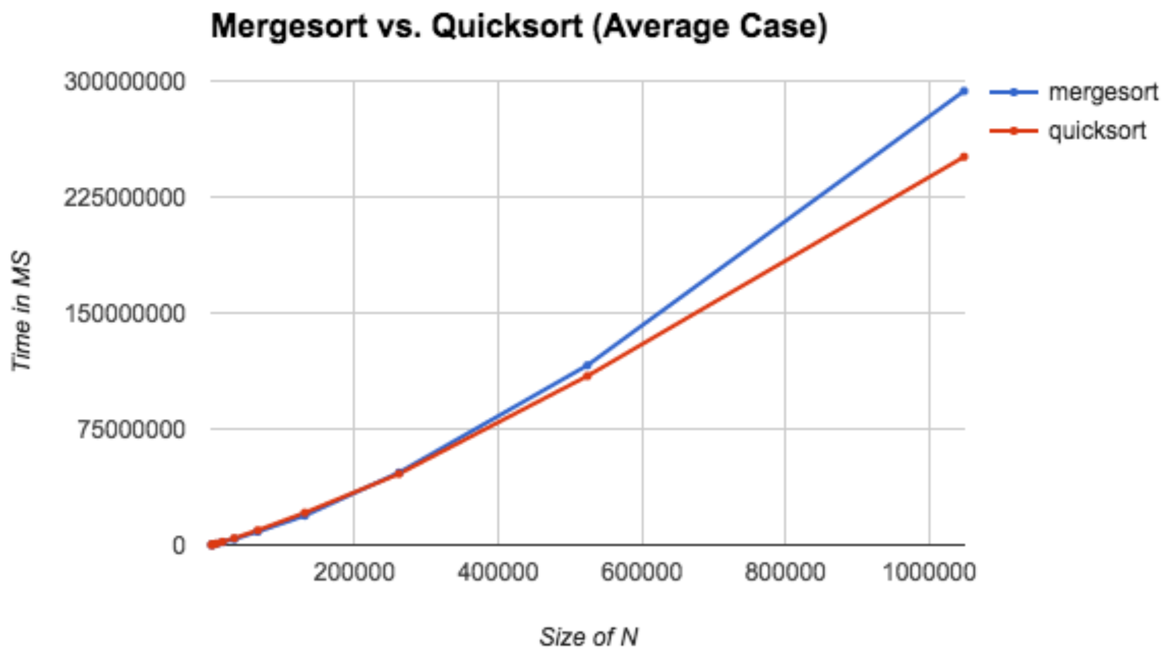


The best pivot selecting strategy was indeed the one discussed in class, median of three. Middle is a safe choice and was close with the median of three multiple times, but if you pick a bad element in the middle it hurts your complexity. Also end is a lot like choosing randomly in the middle and hoping for an element that actually belongs in the middle. Median of three helps turn the random pivot selection more in your favor because it grabs the first, middle, and end element and picks the middle of those.

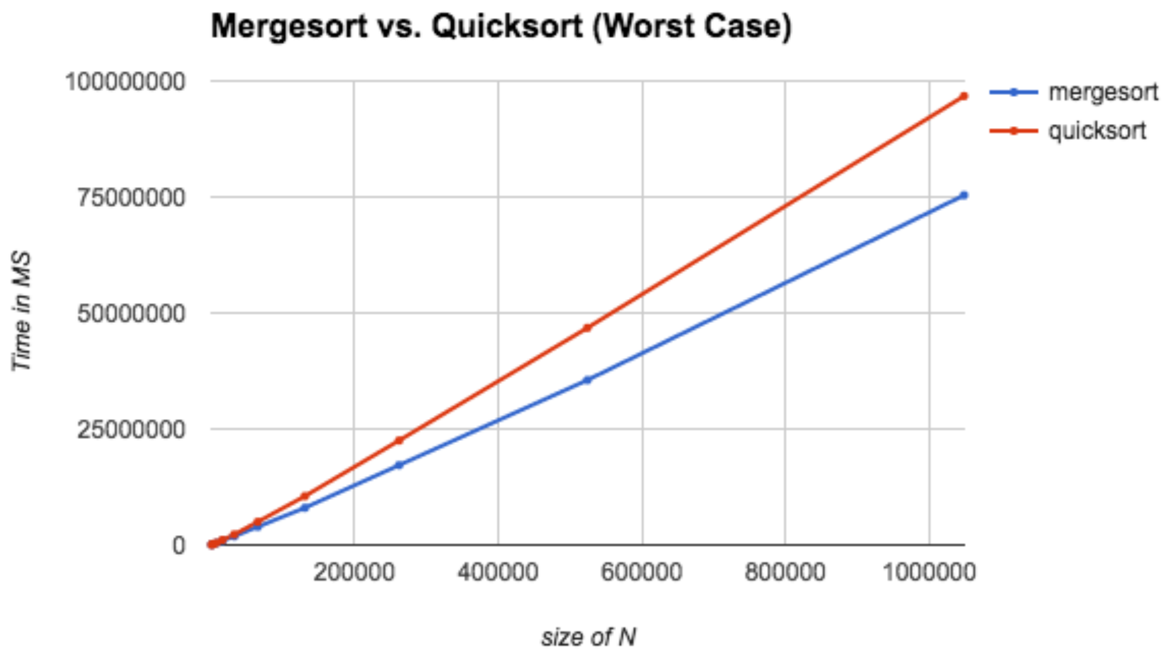
6. Mergesort vs. Quicksort Experiment: Determine the best sorting algorithm for each of the three categories of lists (best-, average-, and worst-case). For the mergesort, use the threshold value that you determined to be the best. For the quicksort, use the pivot-choosing strategy that you determined 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).



For best case I would choose mergesort, but they really do both have the same complexity of O(N log N). Choosing mergesort for best case is fine as long as you can afford the 2N space complexity.



Quicksort is the better option for the average case. For the average case we used the median of three on pivot selection to obtain this behavior.



For the worst case i'd choose mergesort. For the quicksort on the worst case we had to change from median of three to picking the middle element because it was acting very O(N^2). quicksort has high potential to be O(N^2) because the picking of the pivot is never promised to be optimal.

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.
Mergesort does follow the growth rates that were expected as long as the threshold was around the optimal choice. For all cases it gave us O(N Log N). If there isn't a good threshold value then mergesort does all the work through recursion, or insertion sort takes over and both of those instances impact growth rate significantly.

Quicksort also did follow the expected growth rates depending on the pivot that is selected. When the pivot was chosen at the end and the list size order was an average case, it started to get near O(N^2), middle choice and median of three hung around O(N Log N) which is what was expected.

The average case is about the only case that seemed to be a little different with quicksort looking like it is the more efficient choice, but they are both close enough to be both be called O(N Log N).

8. How many hours did you spend on this assignment?
10 - 12 hours.

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.