Rebekah Peterson
Analysis 5: Quicksort and Mergesort
CS 2420

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

Chasen Chamberlain; I submitted the code.

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

Chasen stays ahead on the assignment and puts in a lot of work. He also uses the debugger a lot, which helped me learn to use it more. I definitely could work with him again.

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?

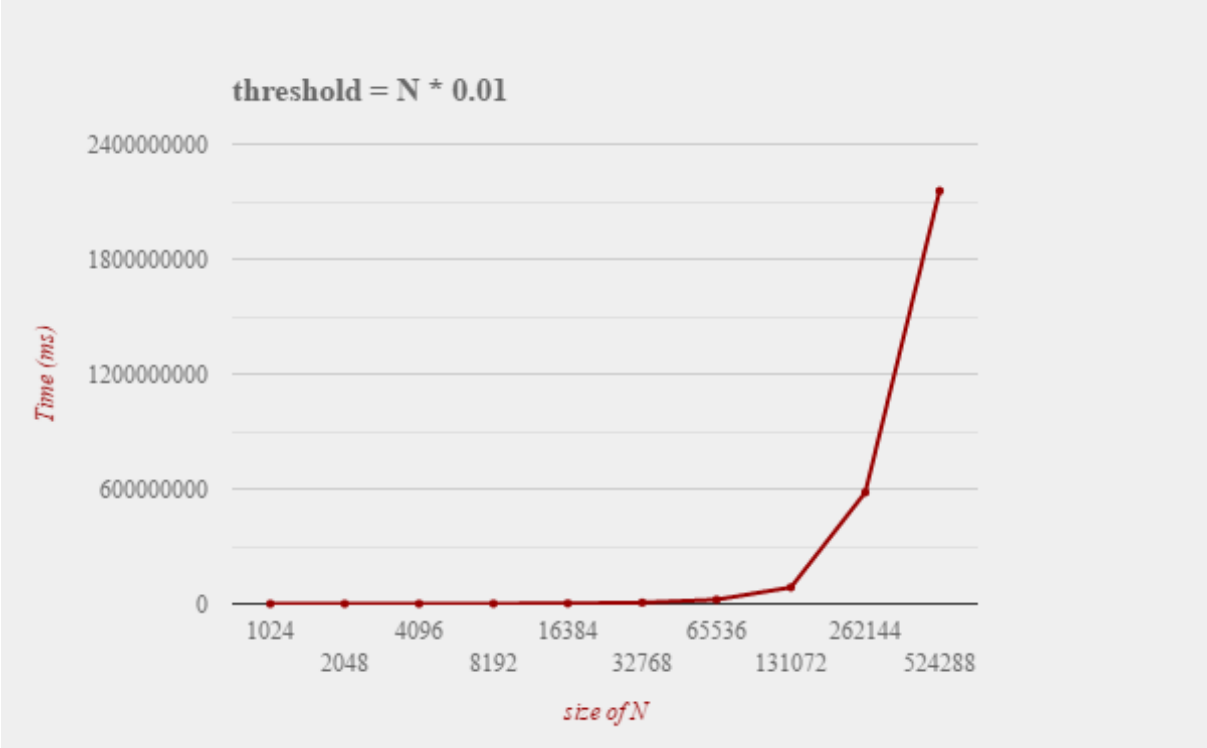
It was helpful to have to make specific time for working on the assignment. It prevented last minute cramming and bad code. It was also extremely helpful to work through bugs with a partner. Explaining everything out loud helped us both visualize the problem, and fix it! The hardest part was probably trying to mesh programming style. Generally, we would just format it using eclipse to fix the inconsistencies. Some of the times we met, I felt a little unprepared. It would have been better to read through the assignment before and come up with testing ideas, other potential edge case issues, and overall design.

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 re-sort 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 times To Loop 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 merge sort 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).

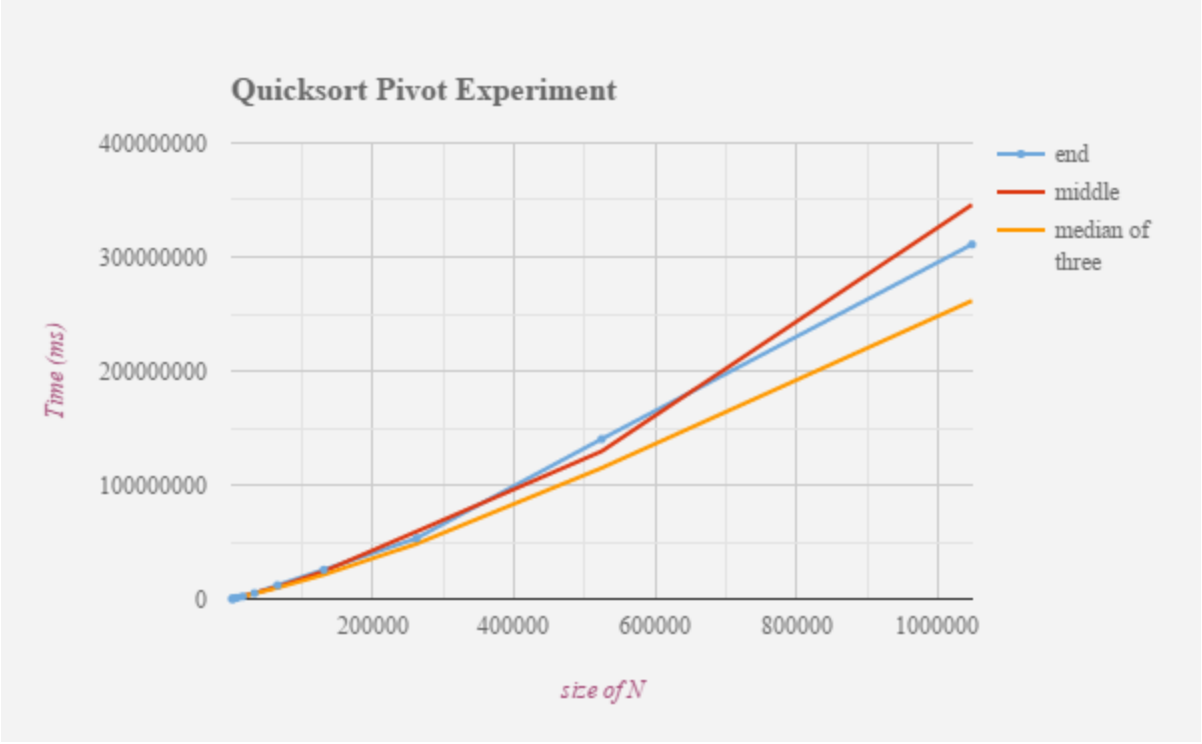
According to our timing experiment, a threshold size of 50 provided the best running times. This is seen from the graph. Our experiment threshold variables included 0, 10, 50, 200, and 0.01 * size of N. The last variable was included as a separate graph because it made the other data unreadable. It was obviously the worst option for large N. Threshold = 0 is the case where mergesort is fully carried out (insertion sort is never used).



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).

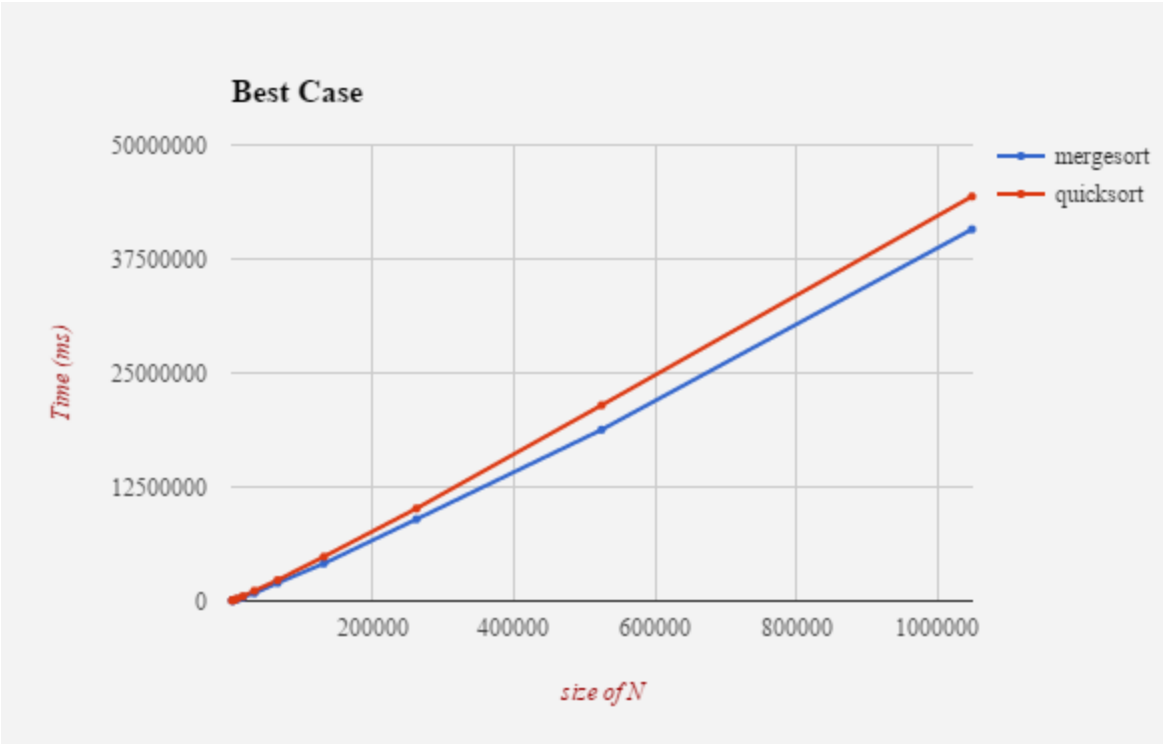
According to our timing experiment, a pivot strategy of “median of three” provided the best running times. This is seen from the graph. Our experiment pivot selection strategies included choosing the element at the end index, the element at the middle index, or the median of three elements.

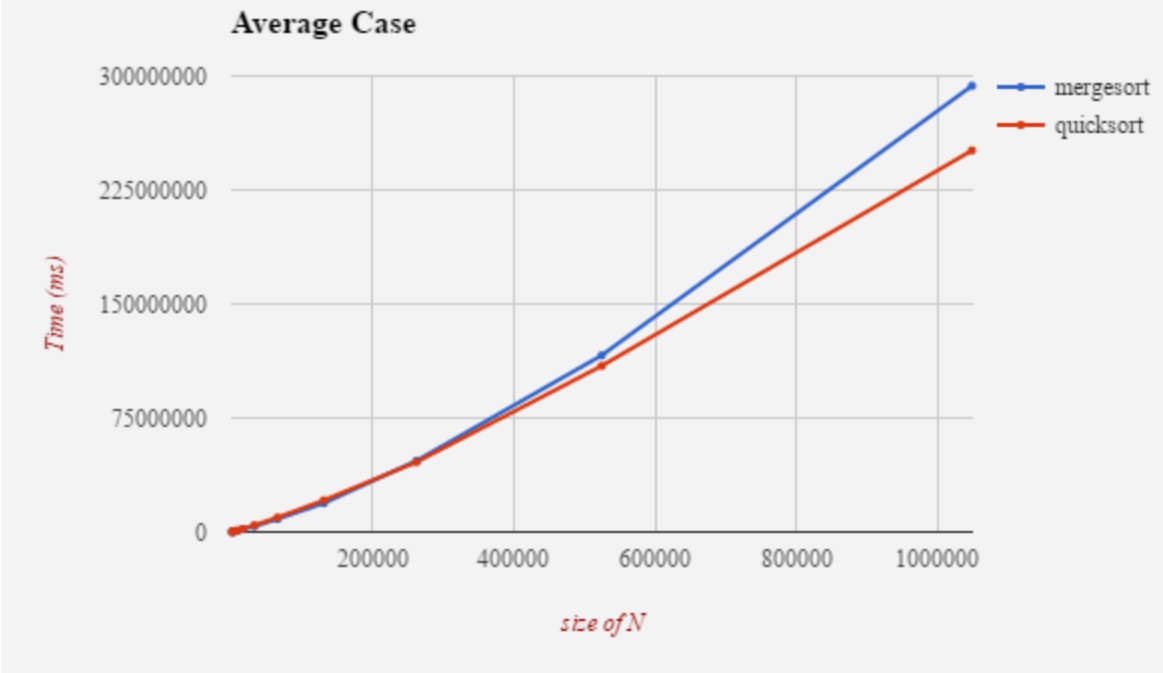


6. Mergesort vs. Quicksort Experiment:

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 quick sort, 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).

Mergesort performed better in the best and worst cases. Quicksort did better in the average case. Quicksort performed horribly in the worst case using “median of three” pivot selection, so we chose the middle index pivot selection method instead for this timing experiment. Although mergesort did perform better overall, it is important to note that mergesort takes up much more space.


Best Case
50000000
37500000
25000000
12500000
0
Time (ms)
200000
400000
600000
800000
1000000
size of N
mergesort
quicksort


Average Case
300000000
225000000
150000000
75000000
0
Time (ms)
200000
400000
600000
800000
1000000
size of N
mergesort
quicksort

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 consistently shows O(N log N) performance, which is expected. The only time it didn't was when the threshold was set to a fraction of N. This caused insertion sort to be called much too early (resulting in N^2 complexity). When a proper threshold is used N log N perseveres. However, mergesort does take up a lot of space as it must be "merged" in a temporary array.

Quicksort usually showed O(N log N) performance, but with bad pivot selection in the worst case the complexity was more apparently O(N^2).

I did expect mergesort to outperform quicksort in the average case as well. However, the difference between running times is not large enough to cause concern.

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

10 - 12 hours