Jared Nielson
Assignment 05 Analysis
CS 2420

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

Christian Hansen was my programming partner. I submitted our source code.

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

Christian was great to work with. This was our third assignment together and once again he was a model partner. There was a point during the assignment where we were completely stuck on mergesort. We couldn't get it to work and we were pulling out hair out. He was the one who ultimately found the source of the bug and was able to fix it.

It is hard to say if we will work together again. The next few pair programming assignments will require different partners. It will be interesting to get work with a new person and gain a new perspective. Christian and I work very well together so I think if we are given the opportunity to work together again we will.

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?

There are definitely benefits to pair programming. The biggest benefit is having someone as a sounding board, someone to bounce ideas off of. Over the course of the three assignments we got stuck several times. We were able to get through these spots by going through our code together and talking through each step. We would launch the debugger and step through the code while talking ourselves through it.

In addition to having a sounding board it is nice to have someone looking over your shoulder checking for errors, helping you come up with variable names, and making sure the code is written with good style. If your partner can't understand your code then there is a good chance no one else will.

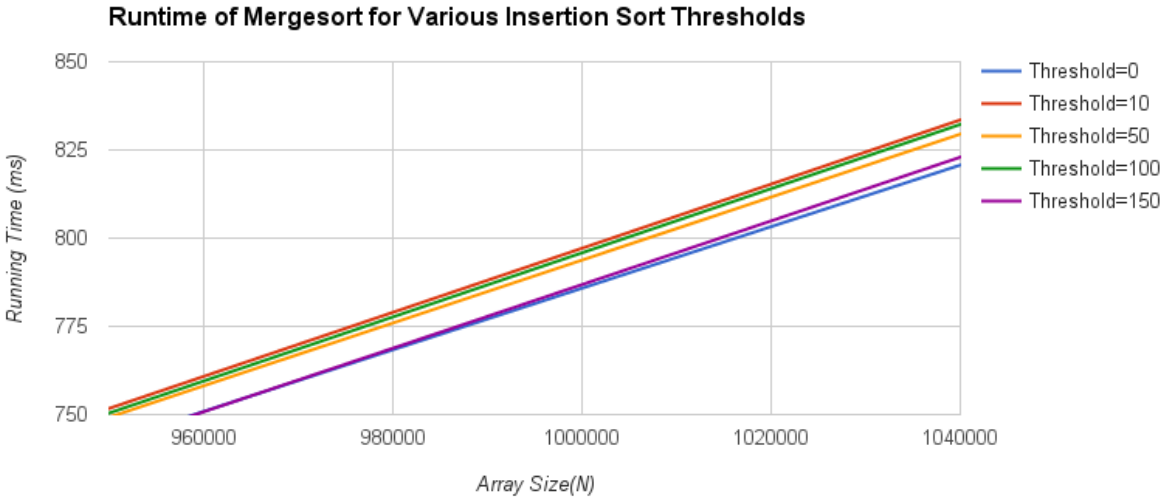
The biggest drawback to pair programming is scheduling. It is hard to schedule enough time to complete the assignment together. I am fortunate enough to have a job with flexible hours, one that allows me to work (or not work) whatever hours I wish. This allowed us to meet and work together for significant amounts of time. However, if I had a different job this wouldn't be possible. Taking time off still comes with consequences such as reduced income over the past few weeks. If we had the ability to pair program remotely I think that would greatly reduce this downside.

Another con is that only one person can be driving at a time. I find I learn best when I'm actually typing and writing something. The algorithm methods are much more ingrained in my brain if physically code them up. Obviously this can be mitigated if both partners are engaged during the pair programming experience, however I did feel that I missed out on some experiences while I was the co-pilot.

For the next pair programming assignments I think the best thing that I could do to be a better partner is to be willing to relinquish the driver seat more often. I like to be the one at the computer, but that robs the other person of valuable experience. Being a better co-pilot and improving those skills will make me a better partner.

4. Mergesort Threshold Experiment: Determine the best threshold value for which mergesort switches over to insertion sort.

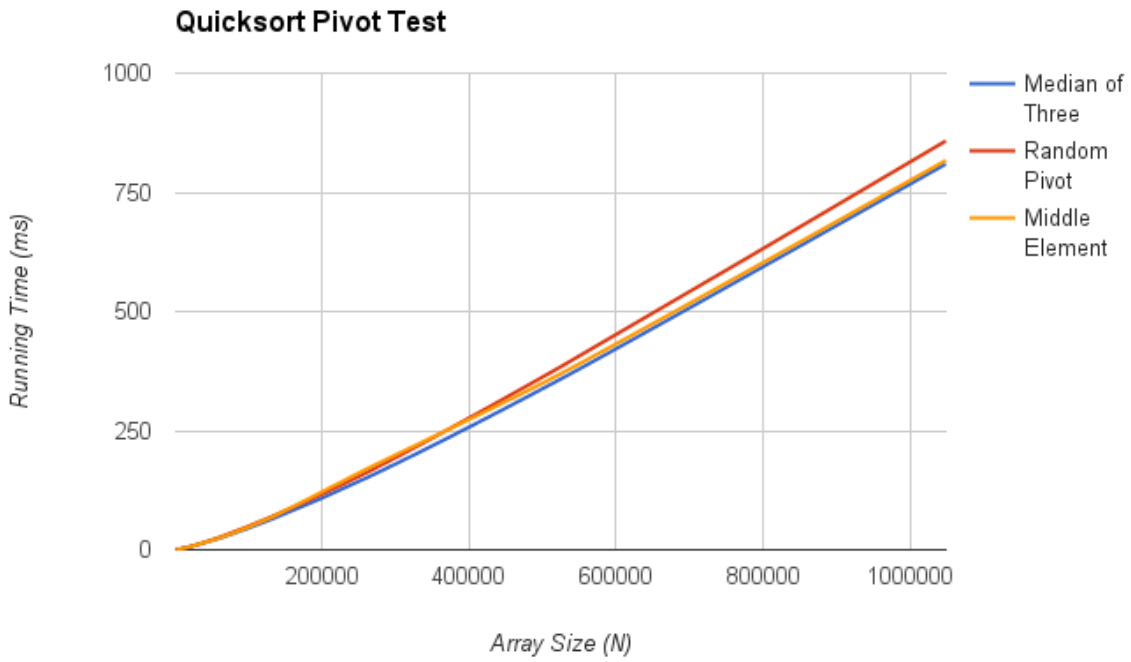
Based on our experiments it seemed that the best threshold for insertion sort was 150 elements. However, this may have been due to differences in calls to System.nanoTime(). When we used 150 as the threshold for insertion sort when comparing mergesort to quicksort the threshold that performed best was 10. In addition for the majority of data points (not shown in the graph) the threshold of 10 was either the best or a top performer. Below is a graph of the running time of mergesort based on various thresholds at large N. The graph is slightly zoomed in to see the differences in the thresholds.



5. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksort.

We compared three methods for choosing the pivot, Median of Three, Random Pivot and Middle Element. The best performers were Median of Three and Middle Element with Median of Three ultimately edging out Middle Element. This makes sense because choosing the Median of Three guarantees that the pivot selected is not the lowest or highest value in the subarray. Choosing such a pivot (the lowest or highest) is what results in the worst-case performance, $O(N^2)$, for quicksort.

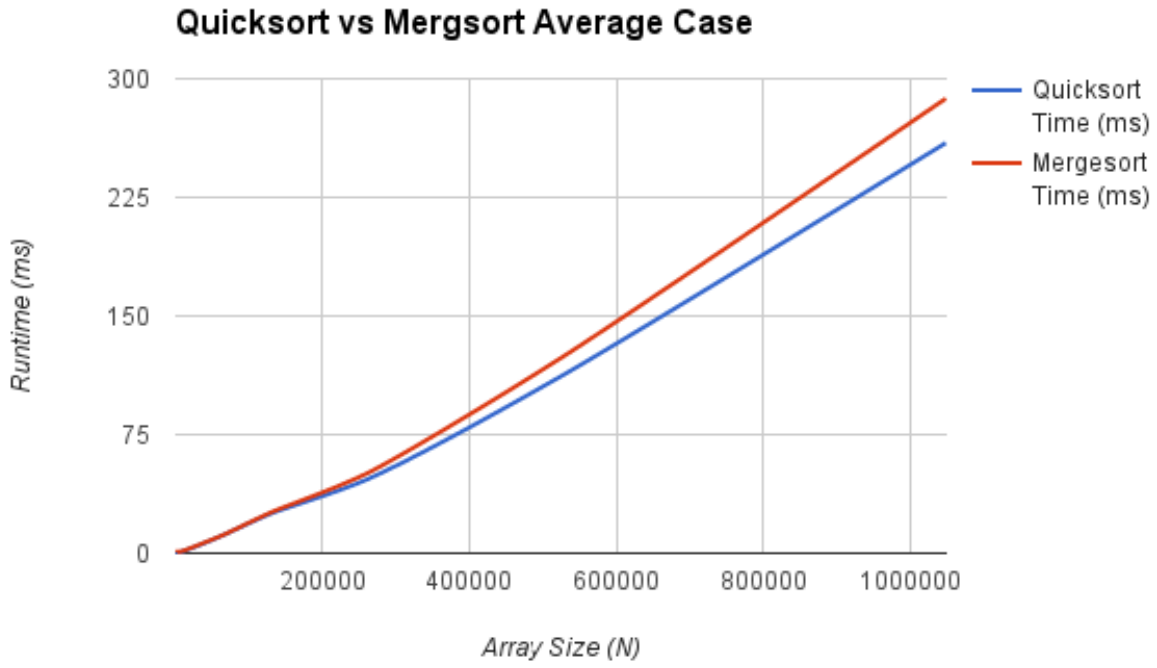
The chart below shows the runtime in milliseconds for each of the three pivot selection methods at various array sizes. The experiment indicates that choosing the median of three for the pivot results in the best runtime.



6. Mergesort vs. Quicksort Experiment: Determine the best sorting algorithm for each of the three categories of lists (best-, average-, and worst-case).

Based on our experiments quicksort outperformed mergesort in the average, best and worst case tests. However, the worst case test did depend on the pivot. Mergesort outperformed quicksort when the median of three and random pivots were chosen. However, when the middle element was used as the pivot quicksort out performed mergesort. This makes sense considering the middle element in a reverse sorted list will be the median.

The charts below show the average, best and worst case runtimes of both quicksort and mergesort for various ArrayList sizes. In each case the N log N behavior is clearly visible. It is also easy to tell that quicksort outperforms mergesort in all three cases.



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.

Both quicksort and mergesort are sub quadratic sorting algorithms. Both take advantage of the divide and conquer problem solving paradigm. Each swap for either algorithm removes more than one inversion. This is because adjacent elements are not swapped (usually). Because of this both algorithms have an average case complexity of N log N. Mergesort has a worst case complexity of N log N (in fact it's best and worst case complexity is the same) while quicksort has a worst case complexity of N^2. However, this worst case complexity comes not from an inverted array but from a poor pivot choice. Using strategic methods poor pivot choices can be eliminated leading to a general complexity of N log N.

In the above charts it is clear that both quicksort and mergesort follow N log N behavior. They grow in what looks like al near fashion, though under close inspection the graph is more curved and increases more than a linear curve would. In addition when examining the runtime data it is clear that as the input size doubles the runtime slightly more than doubles (indicative of N log N behavior). If the sorting algorithms were quadratic the runtime would quadruple every time the input size doubled.

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

We spent approximately 12 hours on this assignment.