1. Who is your programming partner? Which of you submitted the source code of your program?
My programming partner is Abdulaziz Aljanahi. He submitted our source code for the program.

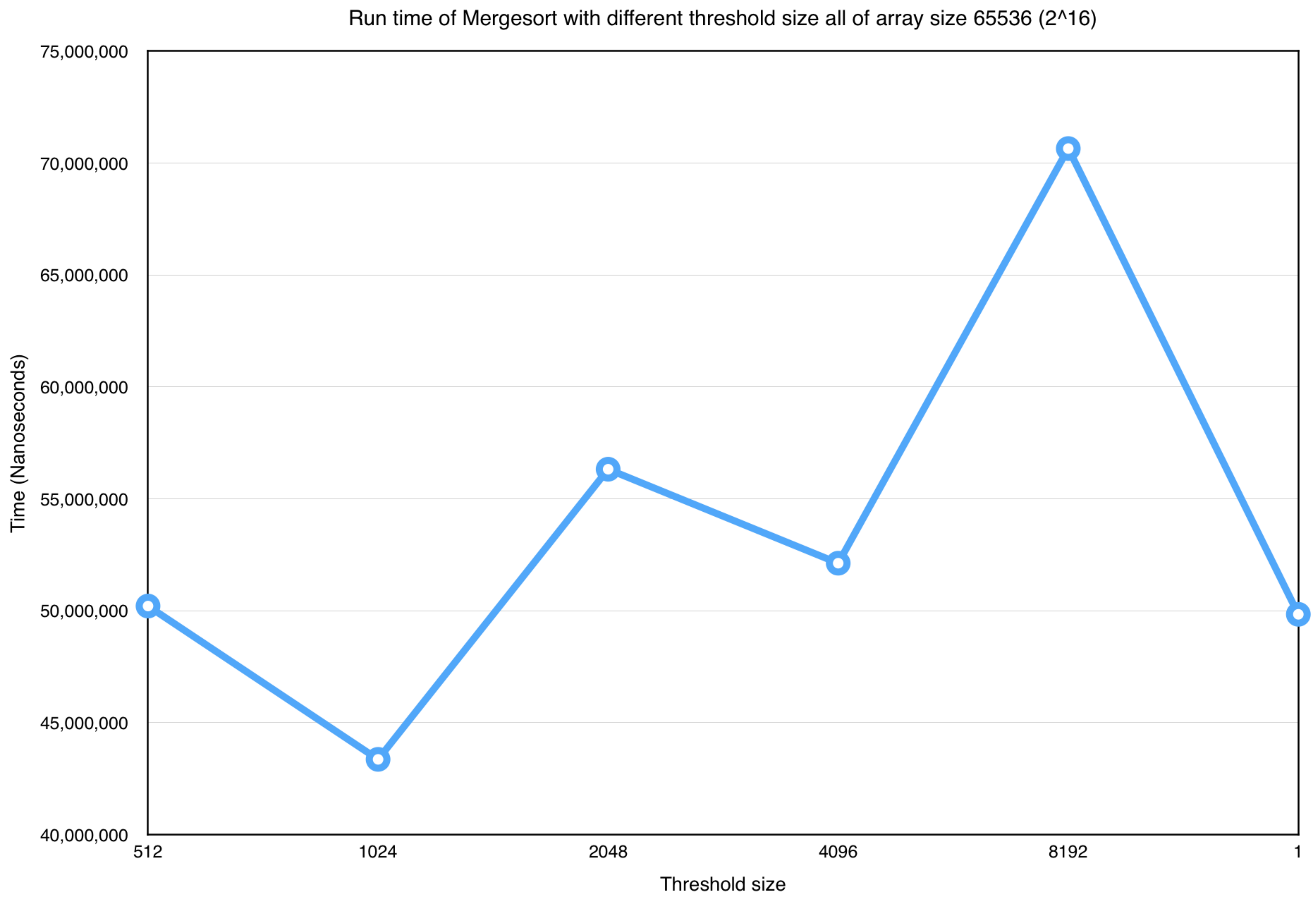
2. Evaluate your programming partner. Do you plan to work with this person again?
I have worked with Aziz for the previous two assignments. He is a very capable and friend programmer. I look forward to working with him again.

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?
The pros of pair programming is obviously that two is better than one. By combining what we know, we are able solve problems faster than we could have if we were alone. However, the cons are that we need to meet up and sometimes our schedule may conflict. There is not much I didn't like about pair programming, aside from that we needed to meet up and code together. If I were working alone, I could've been coding at the comfort of my own home. I will try to be a better programming partner by talking to my partner more to know what problems they are having.

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).
Using array size of 65536 (2^16), I tested out the run times of mergesort by changing the threshold value. I found out that the threshold of 1024, or 2^10 seem to give the fastest runtime. The run time of a full mergesort (where the threshold is 1) is better than a large threshold (of 8192).

See graph below, the full mergesort result is the last entry of the x-axis.

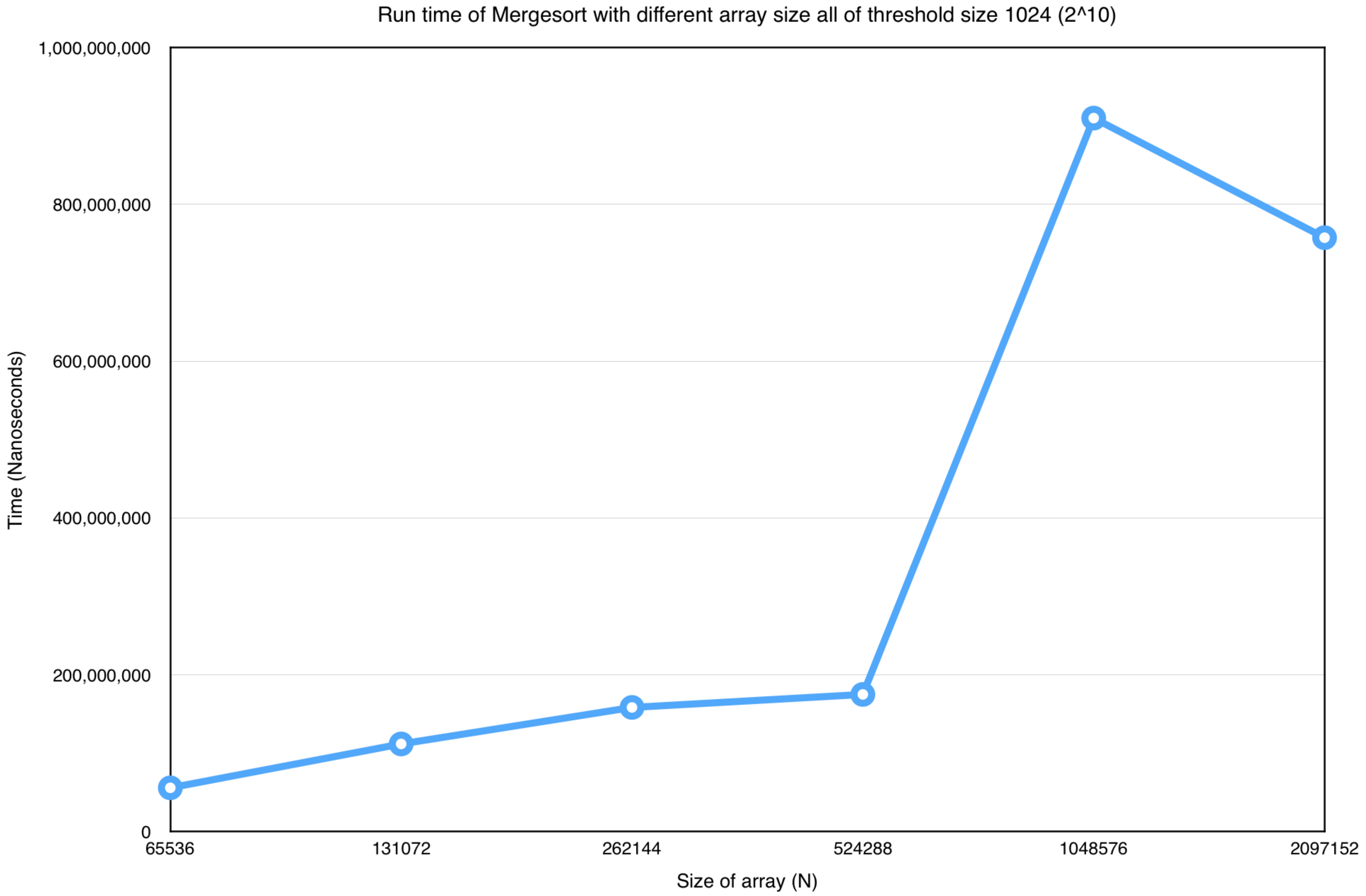

Run time of Mergesort with different threshold size all of array size 65536 (2^16)
Time (Nanoseconds)
75,000,000
70,000,000
65,000,000
60,000,000
55,000,000
50,000,000
45,000,000
40,000,000
512
1024
2048
4096
8192
1
Threshold size

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).
Between choosing a random pivot, median of three, and the first element, we found out that the median of three method gives the best run time. This is no surprising, as we guarantee that the pivot is not worst case, there is at the very least, 1 element greater or 1 element lesser than the pivot.

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).
Both average and best case displayed NlogN behavior from both quicksort and mergesort. However mergesort is much faster in worst case than quicksort, which is what we expected.

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.
We expected to see a O(NlogN) behavior most of the time. Mergesort has a O(NlogN) behavior regardless of best, average or worst case, while quicksort should see O(NlogN) on best and average case, but O(N^2) in worst case. In our graphs mergesort displays a O(NlogN) growth, as we increased N (array size) by powers of 2. (See graph below)

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


Run time of Mergesort with different array size all of threshold size 1024 (2^10)
1,000,000,000
800,000,000
600,000,000
400,000,000
200,000,000
0
Time (Nanoseconds)
65536
131072
262144
524288
1048576
2097152
Size of array (N)