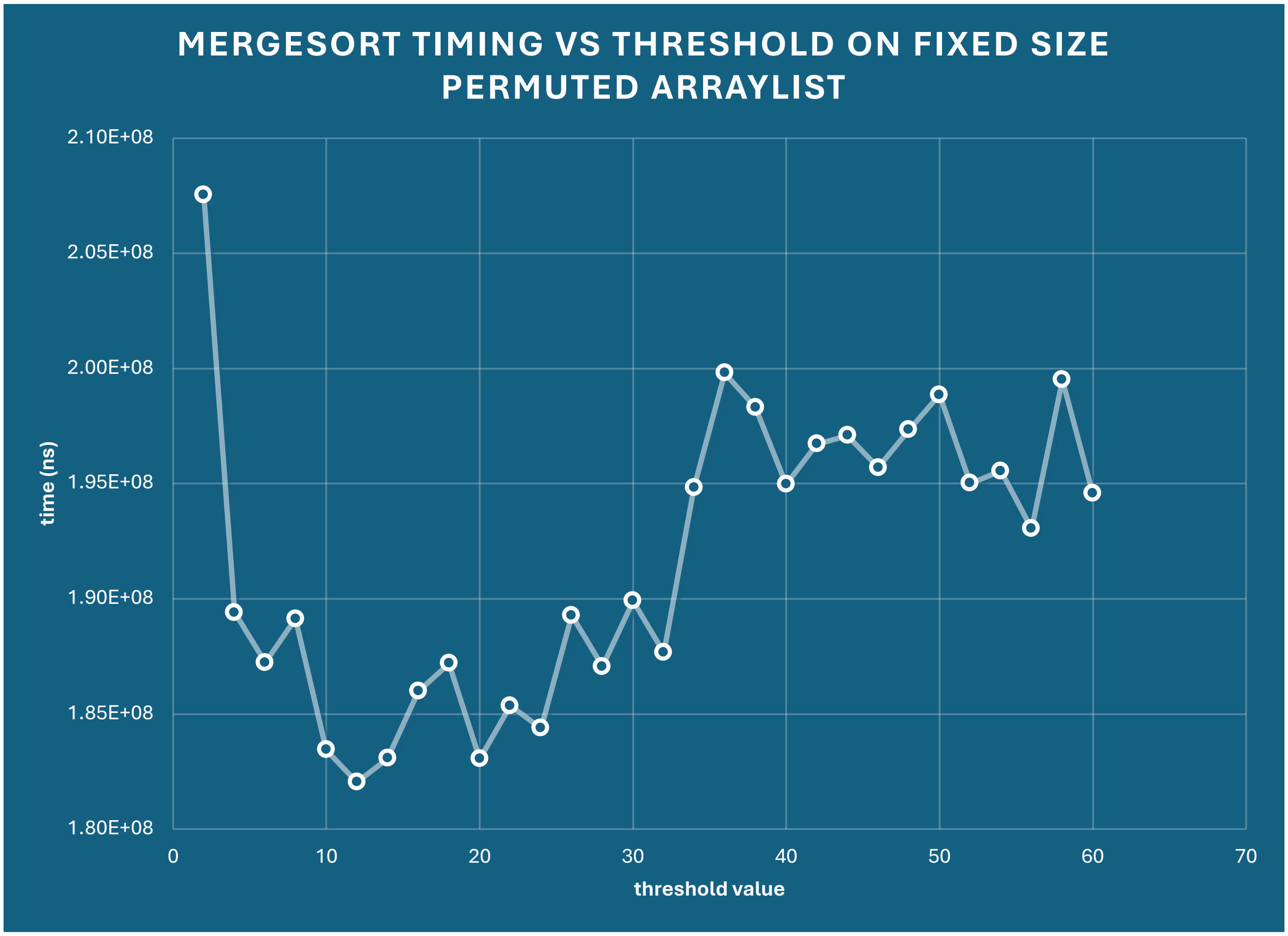
Analysis doc questions
Who are your team members?
Brandon Mountan, Owen Brown

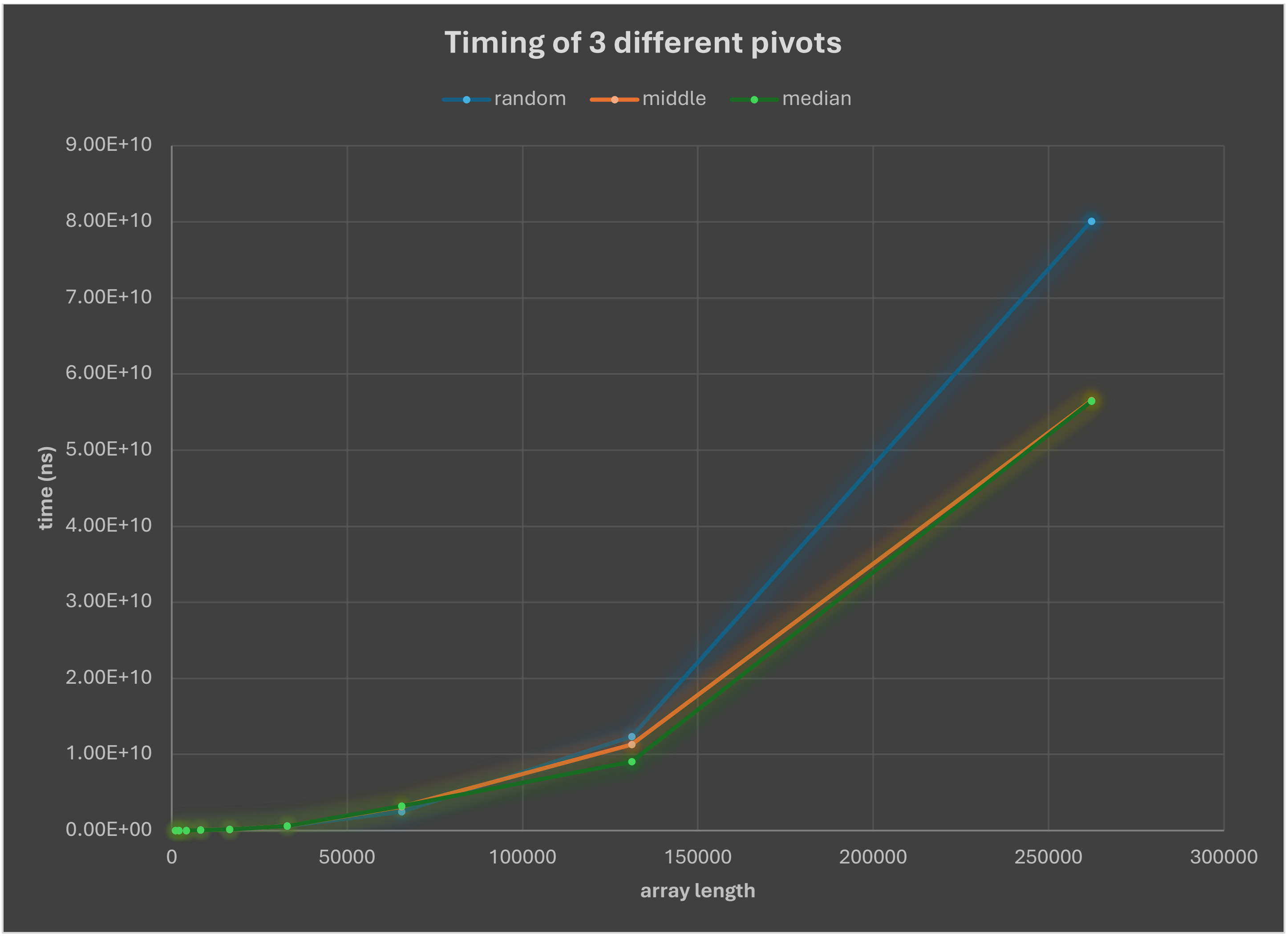
Mergesort Threshold Experiment
Determine the best threshold value for which mergesort switches over to insertion sort. Your list sizes should cover a range of input sizes to make meaningful plots, and should be large enough to capture accurate running times. To ensure a fair comparison, use the same 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 we already demonstrated, 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).



Ben told us to do it this way

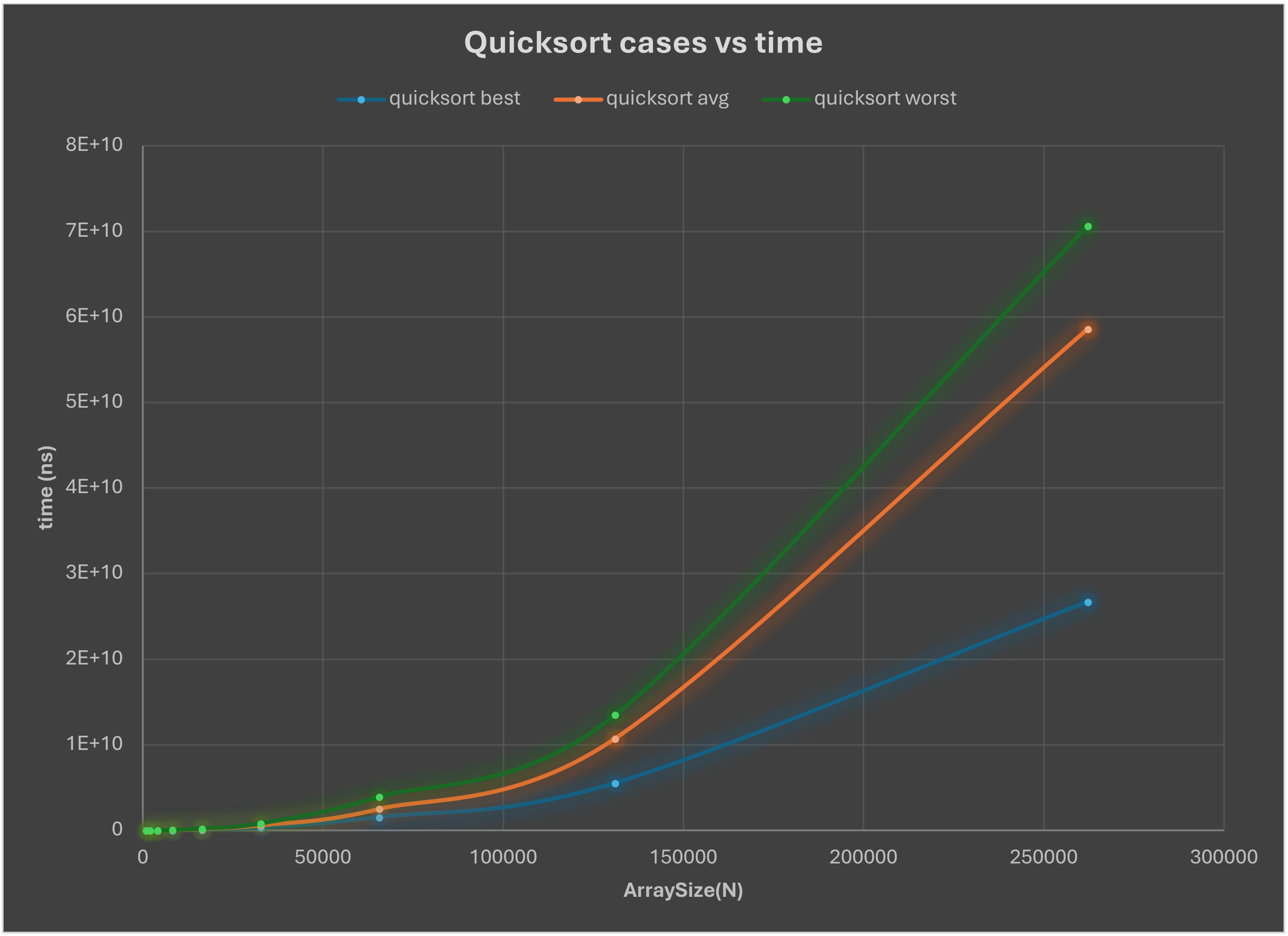
We used a fixed array size of 2^20. We then timed our mergesort method for threshold values between 2 and 60, incrementing by 2 for each data point. This gave us the expected outcome which is when mergesort has a threshold of 2, it uses merge sort the entire time except for the last two elements when it uses insertion sort. It becomes optimized when threshold is equal to 12. It goes up a little from there then back down almost to the lowest point (when threshold equals 20) then back up. The time is averaged over 10 mergesort calls.

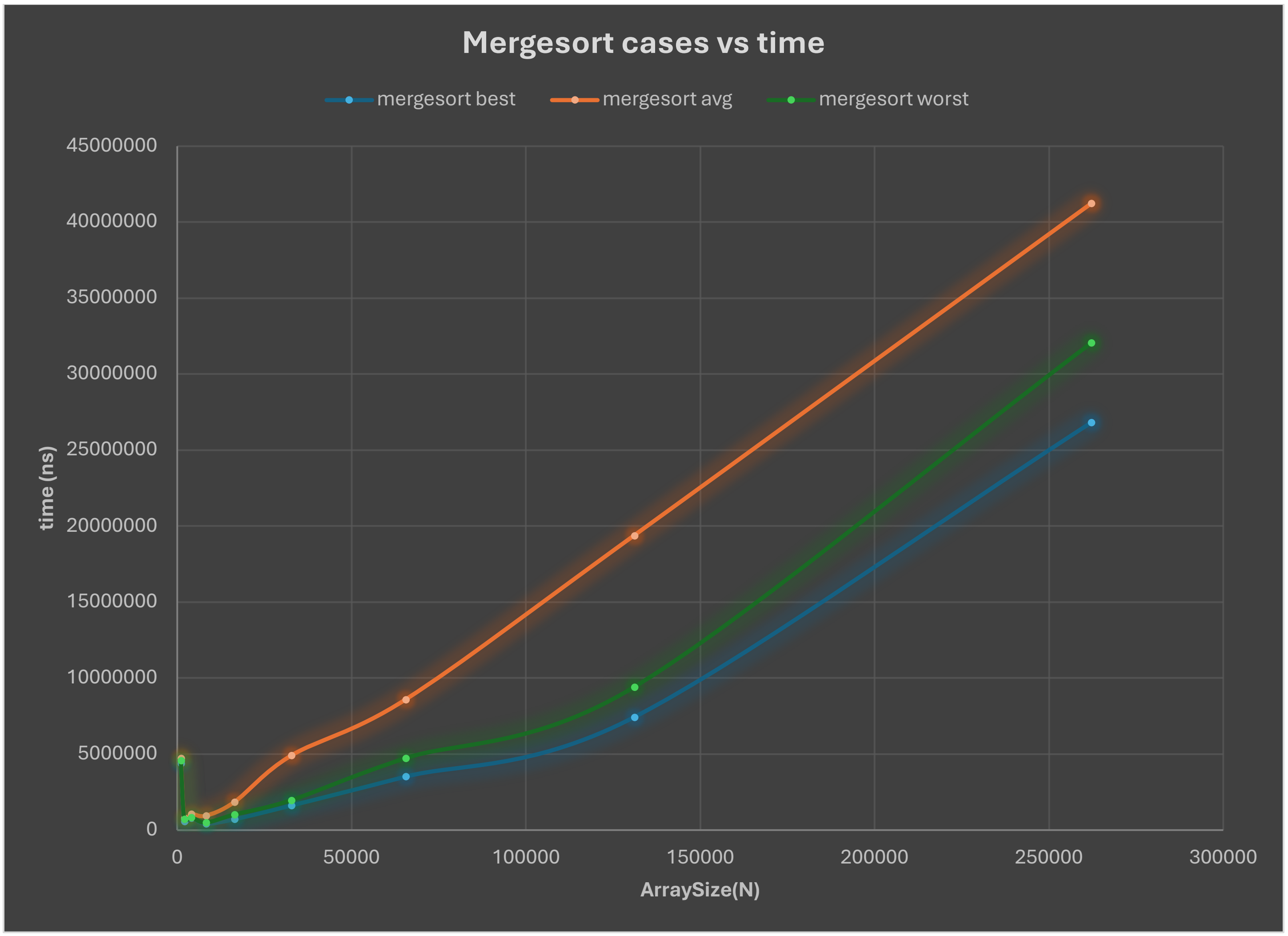
Quicksort Pivot Experiment
Determine the best pivot-choosing strategy for quicksort. (As in #2, use large list sizes, the same set of permuted-order lists for each strategy, and the timing techniques demonstrated before.)
Plot the running times of your quicksort for three different pivot-choosing strategies on permuted-order lists (one line for each strategy).



We were unable to use FIRST or LAST due to stack overflow error so the three pivot strategies we used are middle, median, and random. Random is worse than middle and median and middle and median seem to be exactly the same so we chose to use middle as the optimal pivot strategy.

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 #2, use large list sizes, the same list sizes for each category and sort, and the timing techniques demonstrated before.
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).
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.





The result were what we expected. I had to make a plot for each method because our merge sort method was significantly faster than our quick sort method. If all six lines are charted on the same plot, the merge sort lines are all lined up with the x axis. You can see the merge sort times are around 3E7 and quick sort times are around 5E10. I'm curious why our merge sort average was worse than our merge sort worst case.