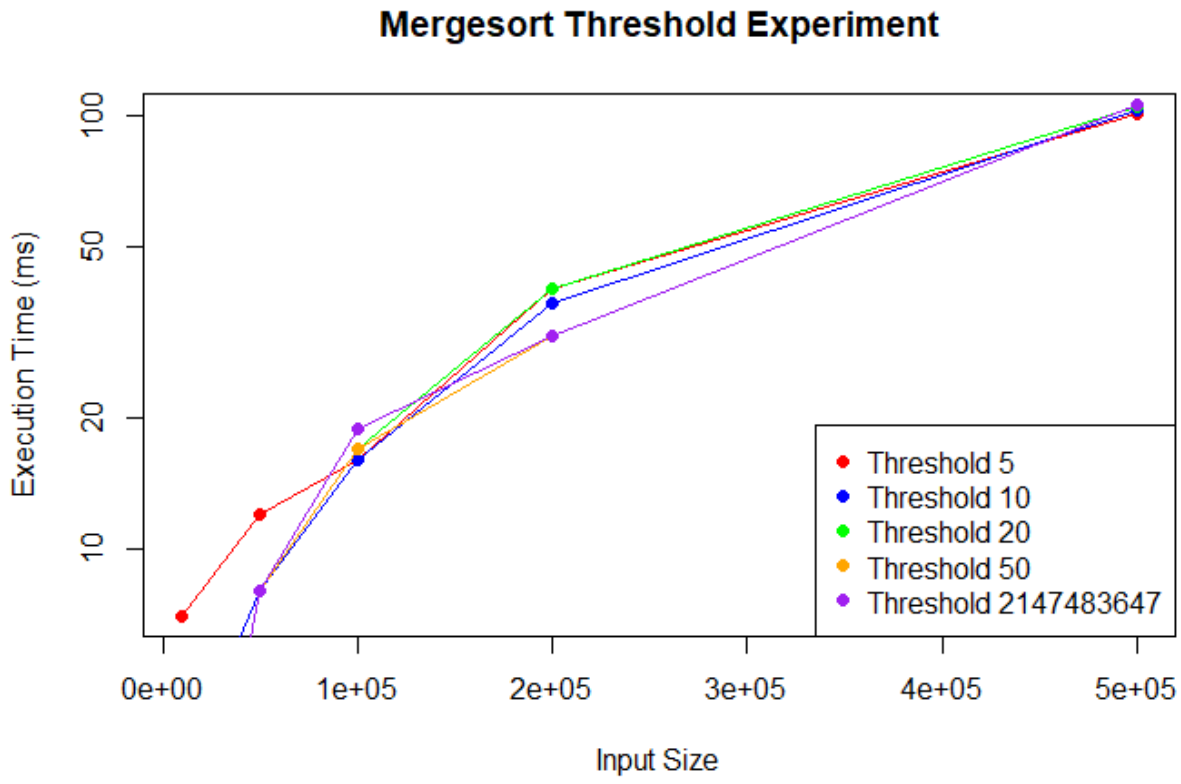
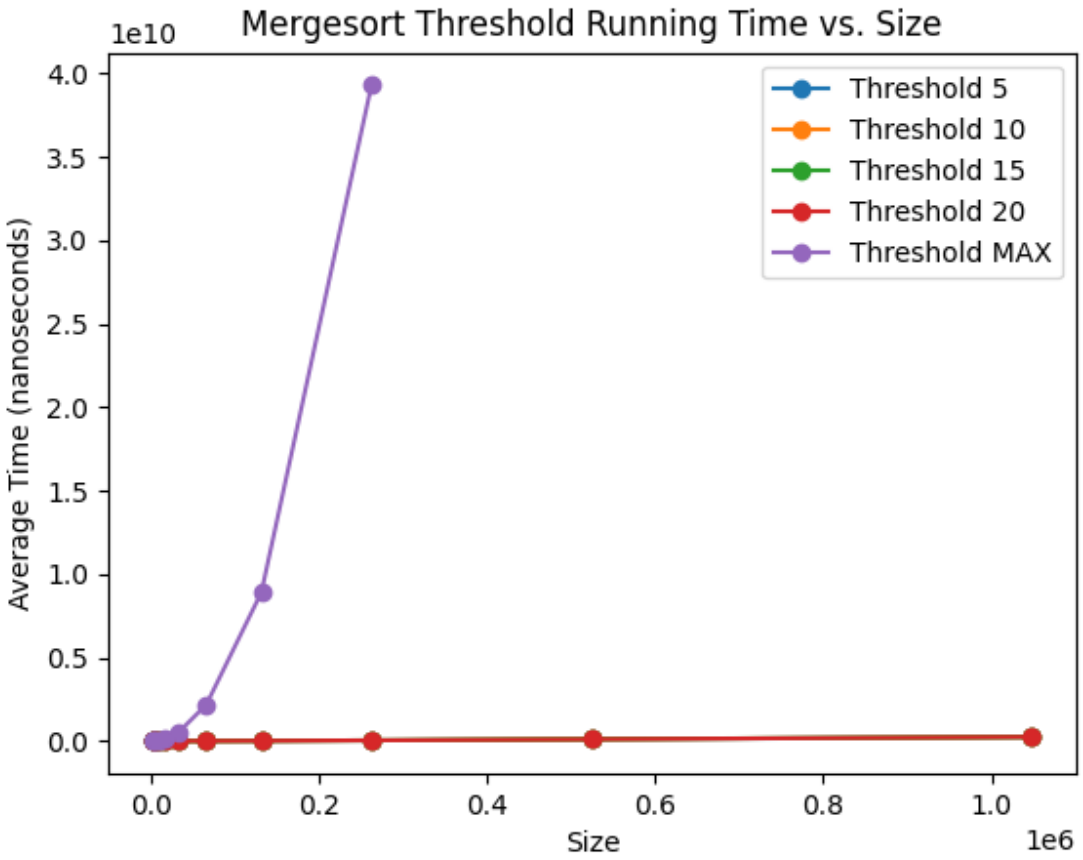
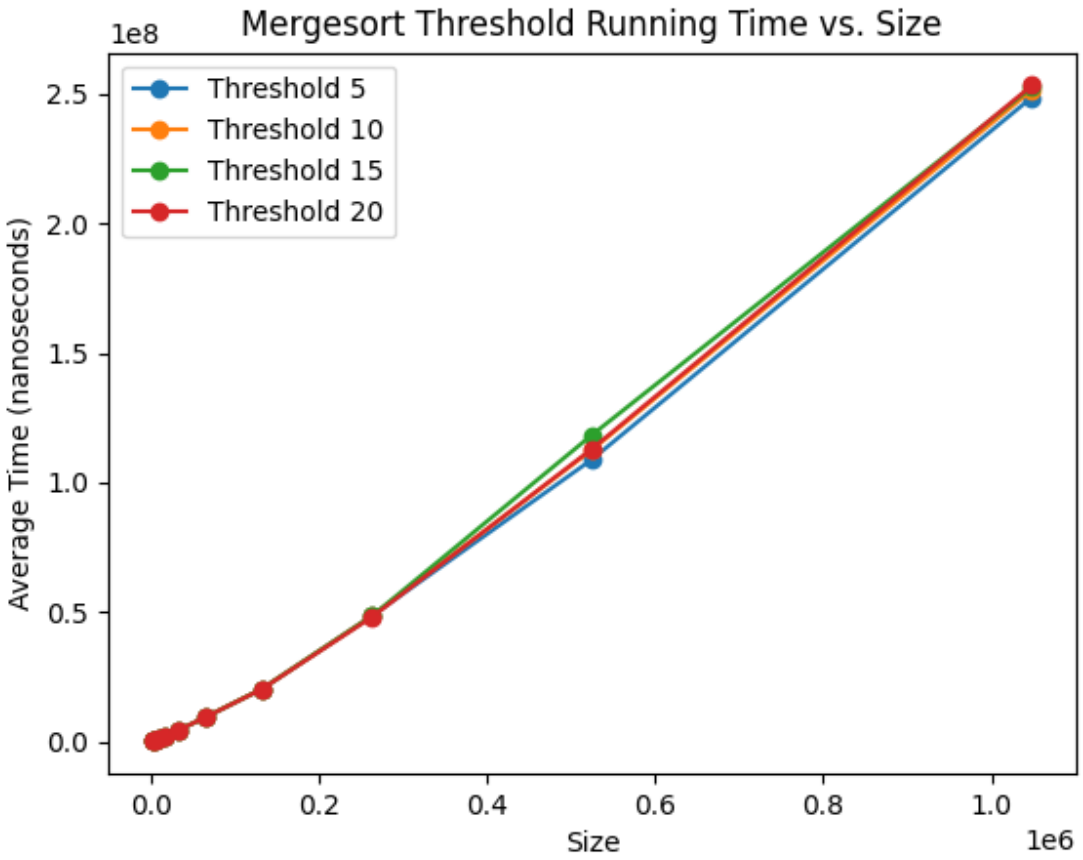
Who are your team members? Brian Erichsen Fagundes & Mina Akbari

Mergesort Threshold Experiment

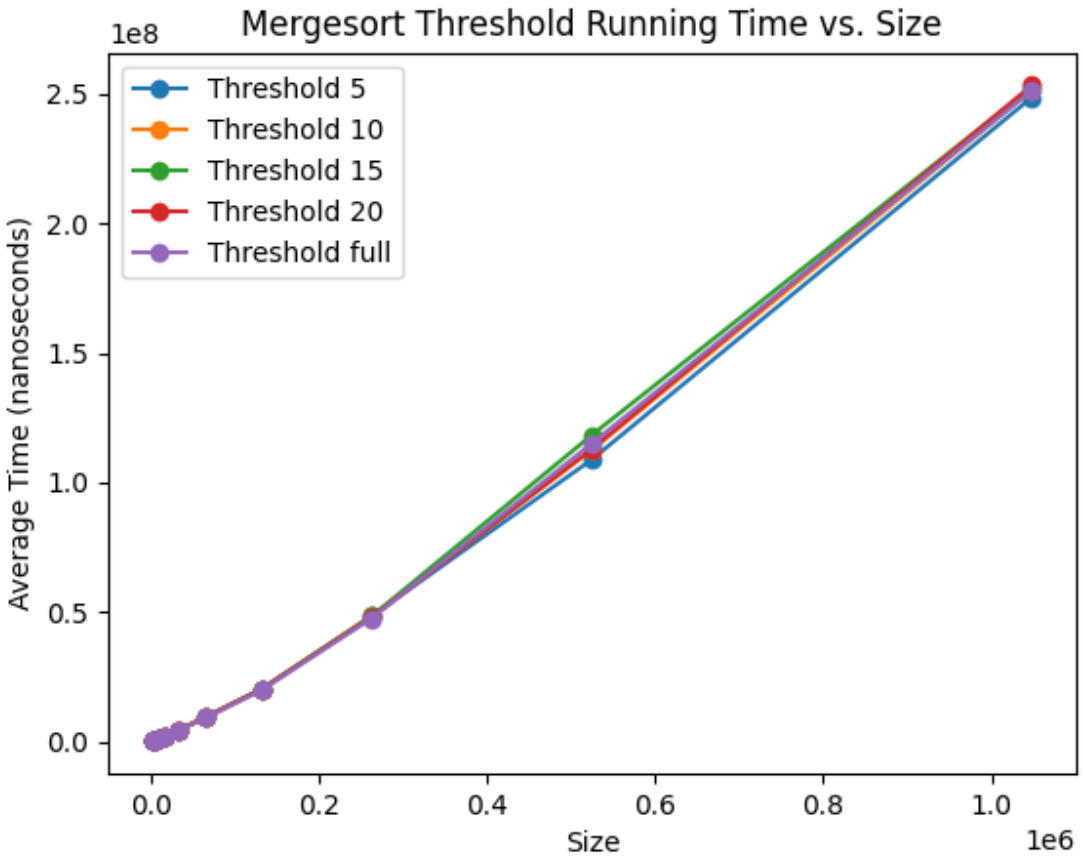




Here the threshold is set to Integer.MAX_VALUE; so it pretty much does insertion sort all the way through; which shows how much slower it is compared to mergesort method.

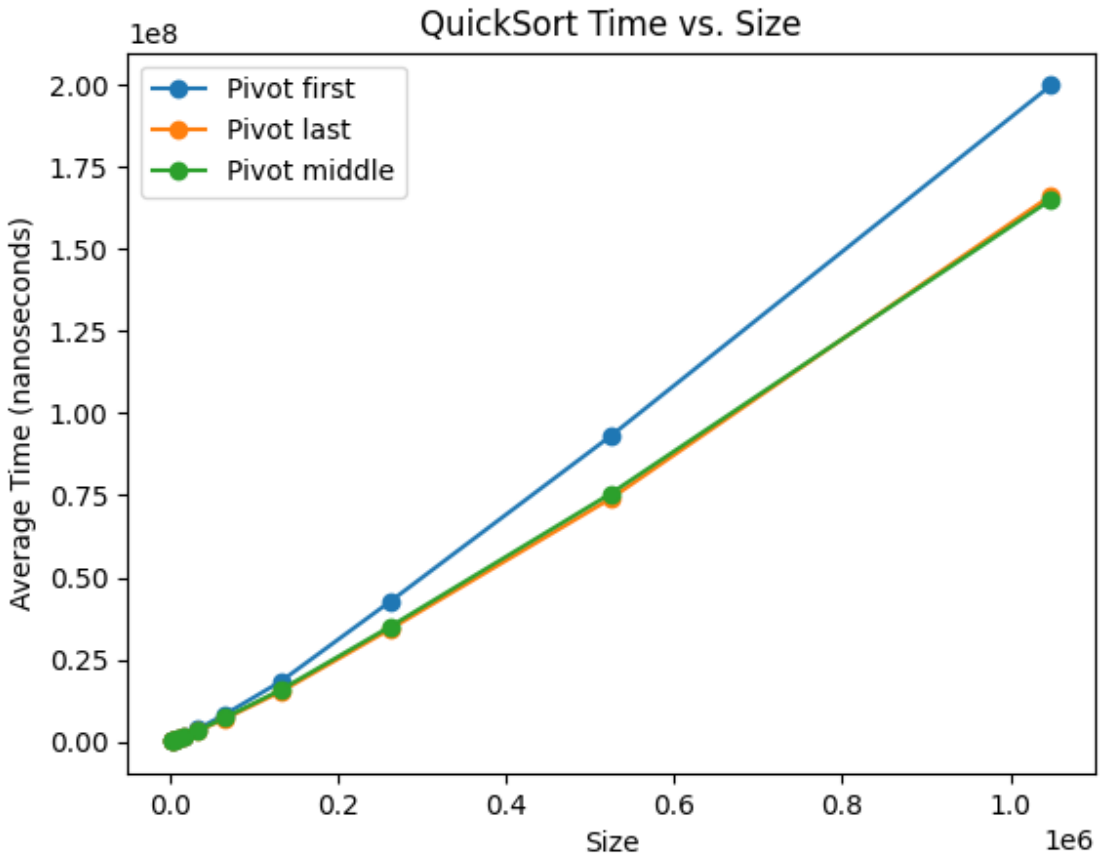


Here the plot shows the lines for 4 different thresholds where the result is not much different than each other but the blue line representing a threshold of 5 is slightly more efficient than the other thresholds.



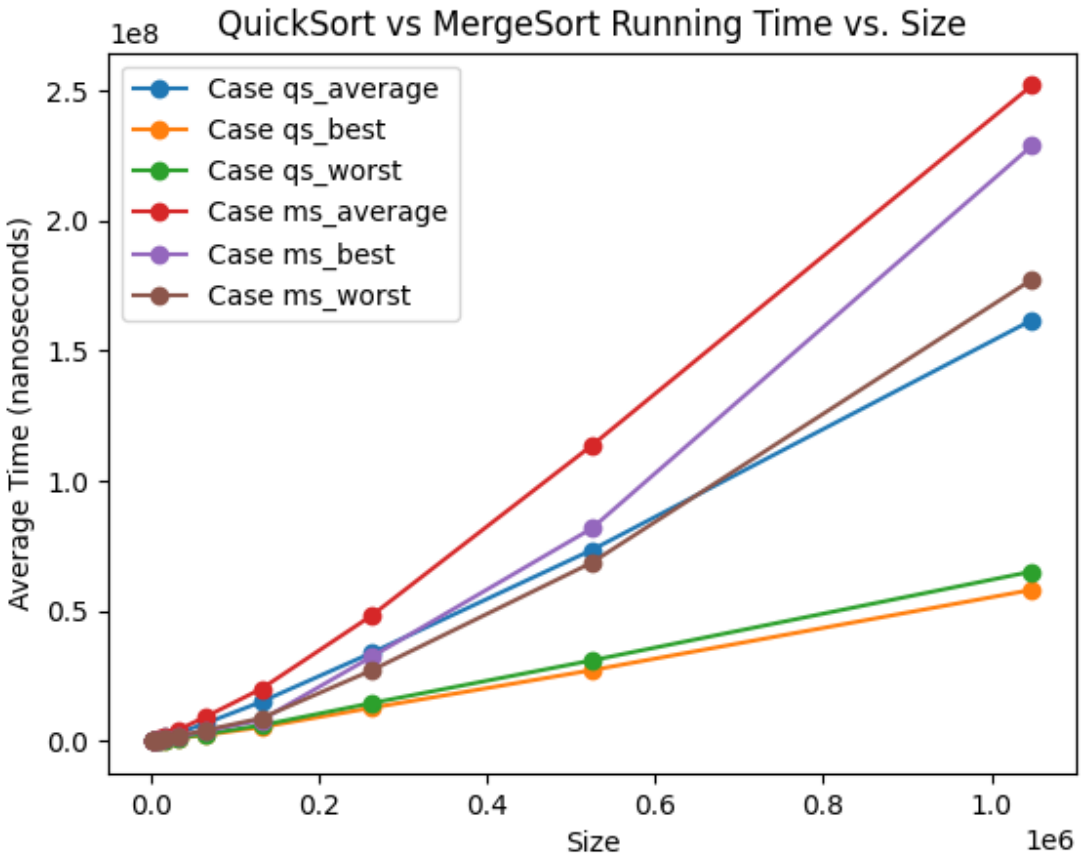
Here the purple line shows using the Mergesort method without switching to insertion sort which points that using the threshold is slight better than not using it.

Quicksort Pivot Experiment



In this experiment, the best strategy for the choice of index pivot is using the middle element as far as the collected data exhibits. Using the middle of the array to pivot showed the best performance of all three and the using the last element also showed a relatively good performance when compared to using the first element. Using the first element yielded consistently the worst run time of all 3 options.

Mergesort vs. Quicksort Experiment



The best algorithms for best, worst or average cases depend on various factors but as far as **average cases** the quicksort it did outperformed all cases of merge sort in the collected data. This usually happens whenever the data is randomly or uniformly distributed; however mergesort has a consistent O(n log n) time complexity for all cases in spite of best, worst, or average cases.

As far as **worst cases**, quicksort can possibly have the worst time complexity of O(n^2) when poor pivot choices lead to unbalanced partitions when mergesort has the guaranteed O (log n).

And finally as far as **best cases** quicksort did outperform mergesort which both exhibit a O (n log n) pattern however the mergesort has it's recursive nature and additional memory requirements that can make it a bit less efficient in terms of space complexity and cache complexity compared to quicksort.

Do the actual running times of your sorting methods exhibit the growth rates you expected to see?

Mostly the actual running times of the implemented sorted methods exhibited the grow rates that we expected. However in the collected data; the quicksort method for worst case exhibited better performance than expected whereas it should exhibit a run complexity of O(n^2) instead of what the collected data indicates. However quicksort has the downside of being not as stable sorting algorithm when compared to other sorting algorithms due to its pivot choices that can possibly lead to unbalanced and possibly unpredictable partitions which can lead degrade of time complexity from O(N Log N) to O(N^2) if poor pivot choices are made; meanwhile the mergesort is a very stable algorithm that preserves the relative order of equal elements. Finally, quicksort often outperforms mergesort as exhibited on the final graph especially for average and best case scenarios due to it's smaller constants factor in it's runtime making it an average of O(N Log N) complexity and in the other hand, Mergesort can be a bit slower than quicksort but more consistently it has the time complexity of O (N Log N).