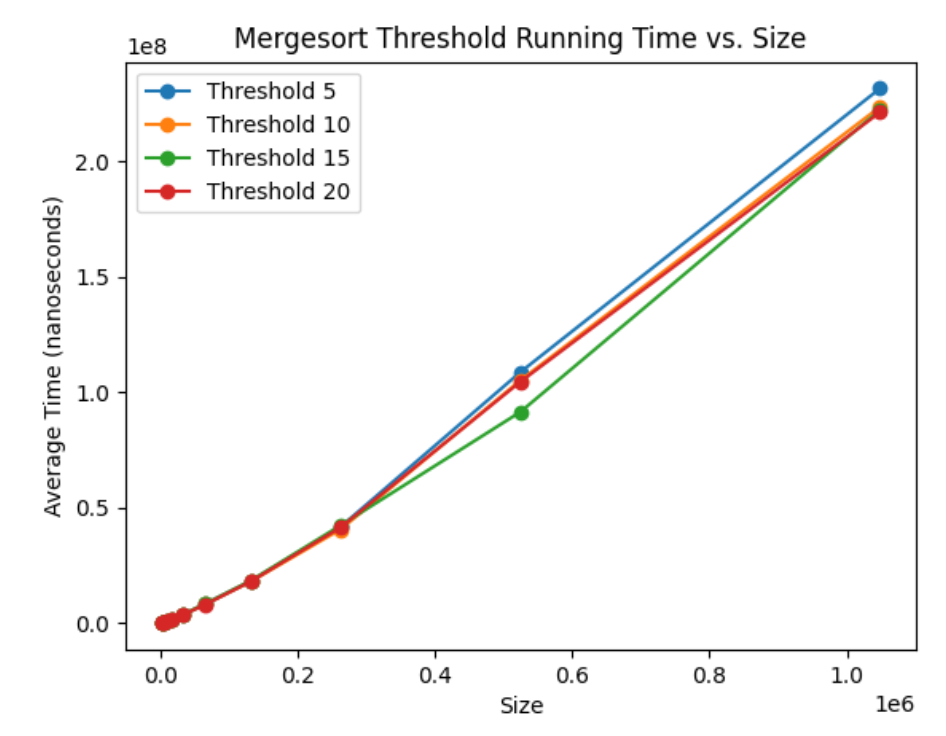
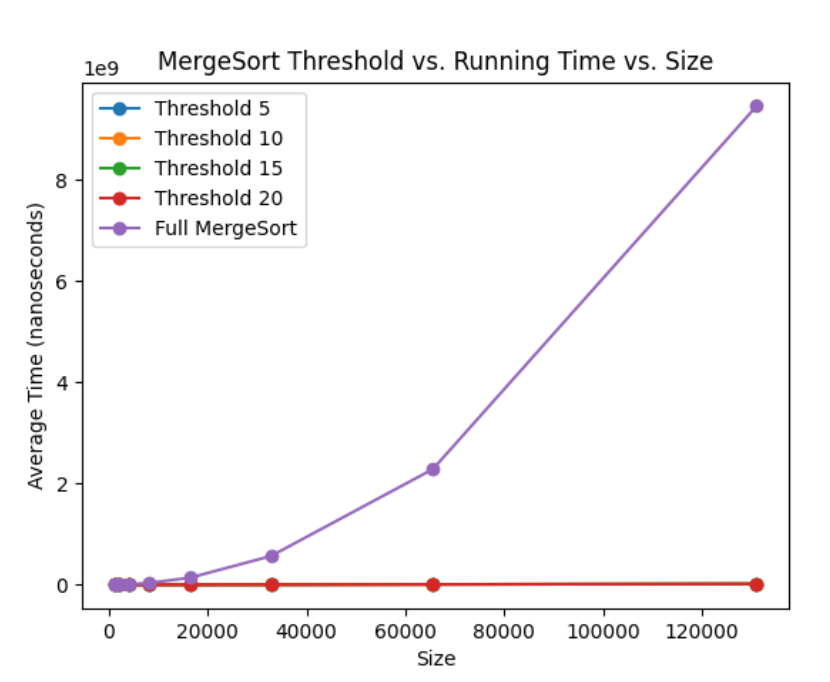
Partner:XiaoHan Chen

#### **Mergesort Threshold Experiment**



## Quicksort Pivot Experiment

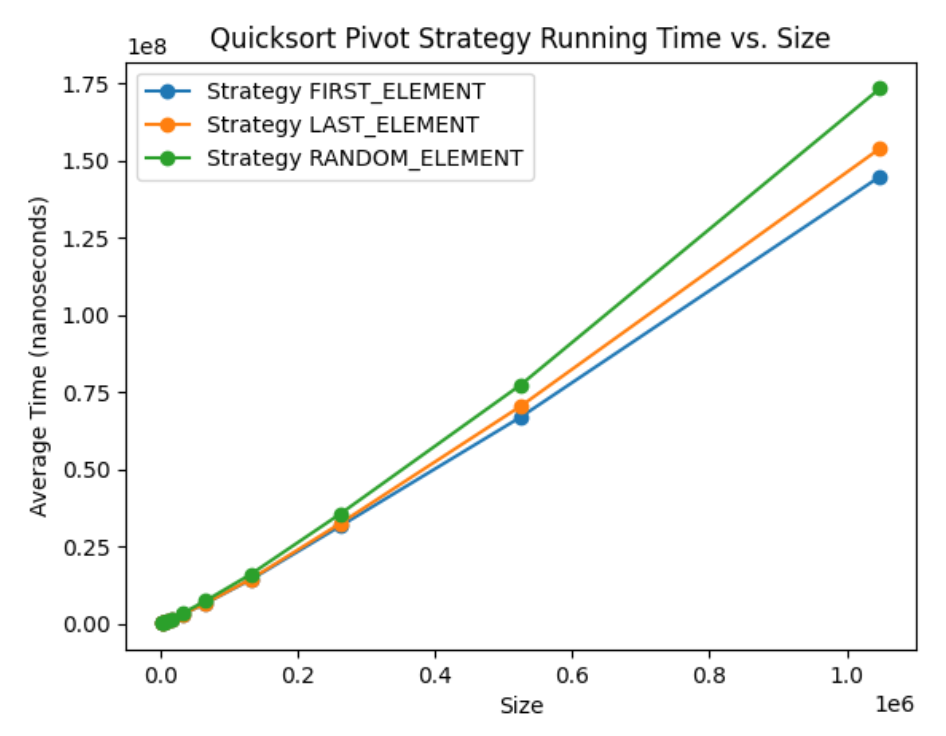


Because merge sort breaks down the array into smaller sub-arrays and then merges them back together. While this is efficient for large arrays, it involves a significant overhead for managing recursive calls and merging small arrays.

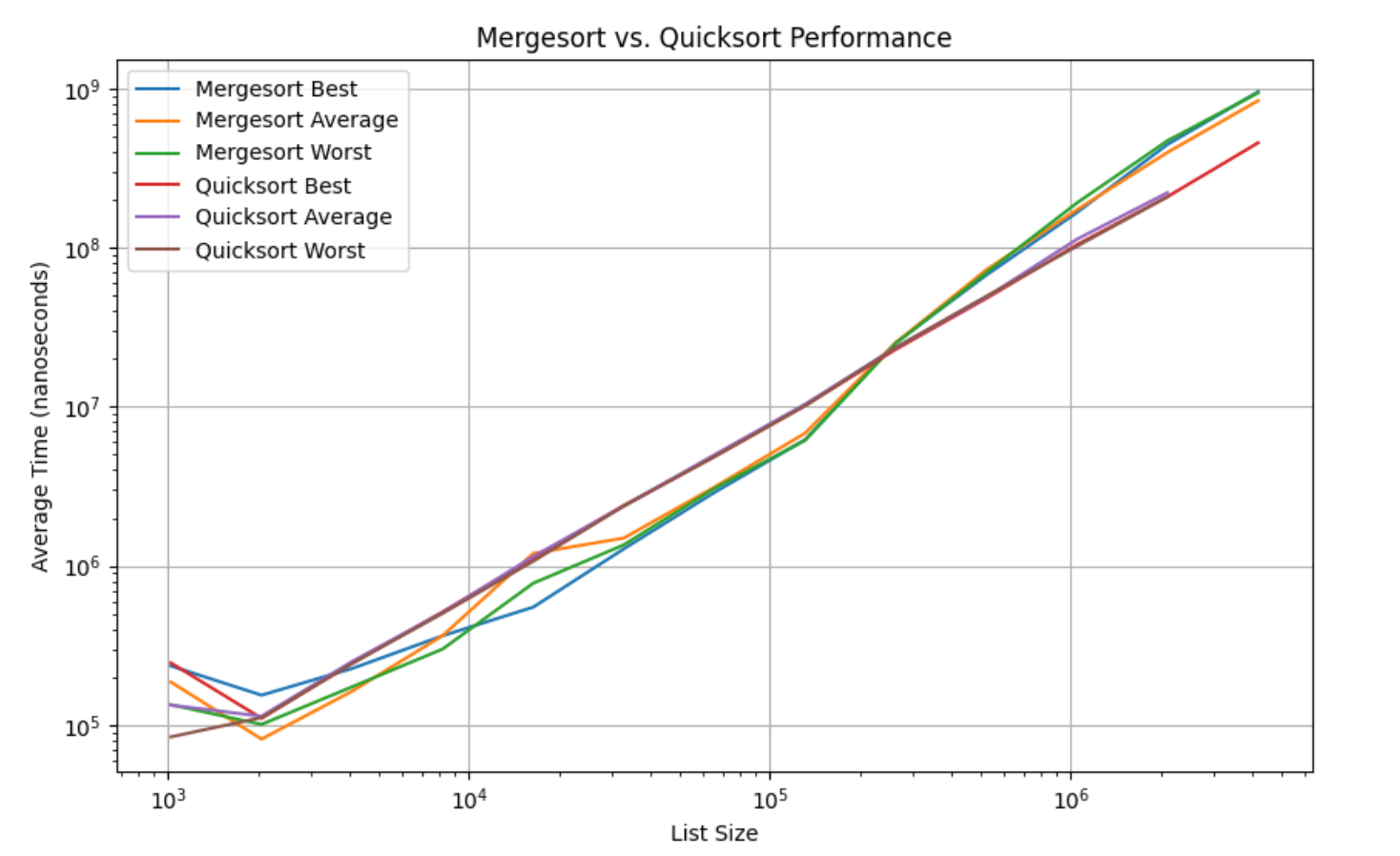
As for insertion sort, despite its worse average and worst-case complexity of O(n^2), is actually more efficient for small arrays. This is due to its lower constant factors and simpler operations (fewer swaps and comparisons, no recursive calls). For small enough arrays, these advantages outweigh its theoretical inefficiency.

Without transitioning to Insertion Sort for smaller sub-arrays, MergeSort is forced to handle all array segments with the same divide-and-conquer approach, which is not as efficient for smaller sizes. Implementing a threshold for transitioning to Insertion Sort allows the algorithm to utilize the most efficient approach for the given array size, leading to overall better. In my case, set threshold as 15 can lead to a optimize case.

Pick pivot randomly can improve performance.



## Mergesort vs. Quicksort Experiment



All curves trend upwards with increasing list sizes, indicating that time complexity increases with the scale of the data.

**Performance with Increasing List Sizes:**

Overall, When the list size is relatively small, merge sort has better performance. With size growing, quick sort perform better than merge sort.

As the list size increases, Quicksort outperforming Mergesort suggests that the efficiencies gained from a well-chosen pivot and in-place sorting start to outweigh the consistent but potentially heavier overhead of Mergesort.

Mergesort’s need for additional space (as it's not an in-place algorithm) and its consistent divide-and-conquer approach might become less efficient with large datasets.

Mergesort and Quicksort show similar performance for best and average cases, suggesting that they are comparably efficient when dealing with sorted data.

**Best and Average Cases:**

The similar performance of both algorithms in best and average cases highlights their efficiency in handling both sorted and randomly ordered data.

This similarity suggests that, in many practical scenarios, either algorithm could be a suitable choice.

**Worst-Case Scenarios:**

The more significant performance degradation of Quicksort in the worst-case scenario (likely due to unfavorable pivot choices leading to O(n^2) complexity) is a known limitation of the algorithm.

This highlights the importance of pivot selection strategy in Quicksort, especially in applications where worst-case performance is critical.