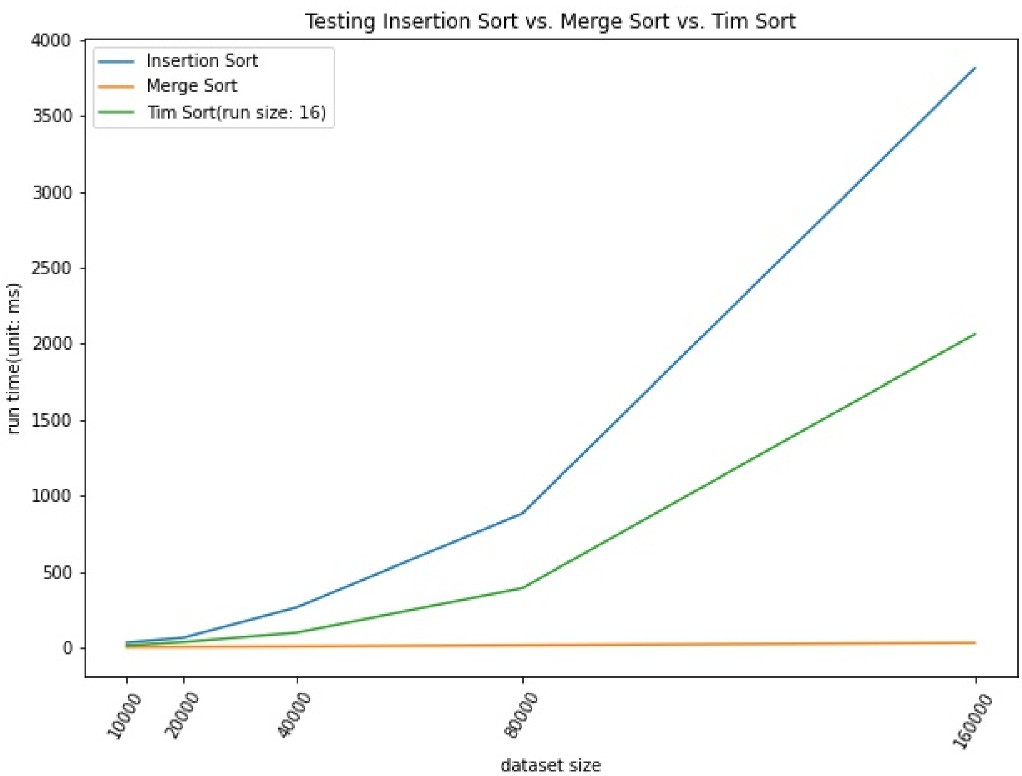
Runtime Analysis

Task 1. Testing Insertion Sort vs. Merge Sort vs. Tim Sort

- Dataset sizes = {10000, 20000, 40000, 80000, 160000}
- time unit: millisecond

Dataset Size	Insertion Sort	Merge Sort	Tim Sort (run size: 16)
10000	33	3	13
20000	65	4	36
40000	264	7	98
80000	883	15	392
160000	3813	31	2062



As the picture depicts above, when dataset sizes doubles, these three sort algorithms' performance varies from each other.

Insertion sort takes $O(n)$ time complexity in best cases which all elements are sorted while takes $O(n^2)$ time complexity in worst cases which all elements are sorted in reversed order. Therefore, in average cases, Insertion sort takes $O(n^2)$ time complexity and as dataset size doubles, run time takes as four times as ever.

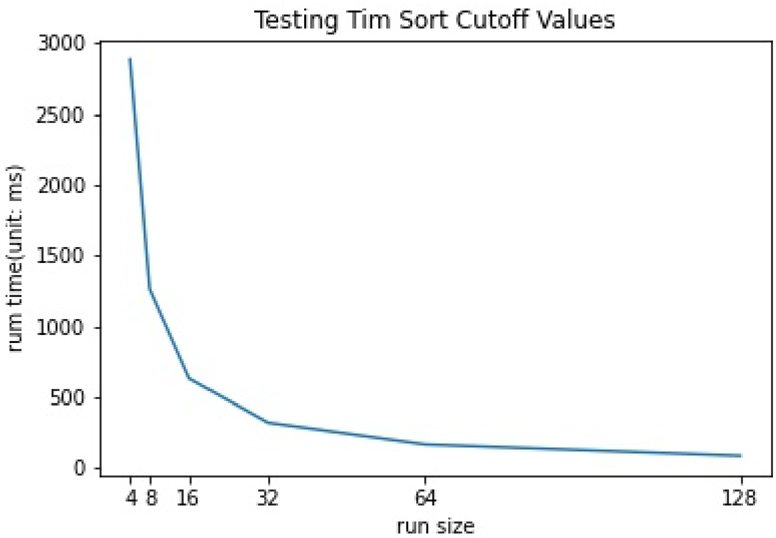
Merge sort takes $O(nlogn)$ time complexity no matter how the array orders, so when dataset size doubles, run time takes nearly no more time than ever.

Tim sort takes less time than Insertion sort when the array to sort is equal. It depends on the natural in-order within the array. Tim sort's time complexity is recorded at $O(nlogn)$, making it's average time complexity equal to that of Quick sort and Merge sort. As we can imagine, when run size doubles up to a considerable magnitude, it will speed over Merge sort.

Testing Tim Sort Cutoff Values

- Dataset size = 200,000
- Cutoff Values for Run Size = {4, 8, 16, 32, 64, 128}

Run Size	Time (unit: ms)
4	2886
8	1263
16	630
32	318
64	165
128	85



As the picture depicts, when run sizes increase, time to sort a specified array will be cut down significantly. That's because the greater the run size, there will be more sub-arrays. Cutting array to sub-arrays takes $O(logn)$ time but merge sub-arrays takes $O(n)$ time. Therefore, when run size increases, there will be less sub-arrays and less merge operation.