Analysis Time Complexity of Merge Sort and Quick Sort

Objective:

This report is based on the analysis the time complexity of two different algorithm named ‘Merge sort and Quick sort’. Though these two algorithms are used for making a list sorted but according to their runtime and case analysis these two algorithms take different time to sort a list which depends on the size of the list. So, the main objective is to identify the time taken by two different algorithm and measure their complexity according to the runtime.

Machine Configuration:

Windows edition	: Windows 10 Pro
Processor	: Inter® Core™ i5-7200U CPU @ 2.50GHz
Installed memory (RAM):	8.00 GB (7.88 GB usable)
System Type	: 64-bit operation System, x64-based processor

Merge Sort:

Theoretically, merge sort makes a list ordered by ‘divide and conquer’ technique and though there is three possible scenario that the list may be sorted (ascending order) or nearly sorted , random ordered and sorted in reverse order (descending order) but this sorting algorithm has a tight upper bound to sort the list and this is $\Theta(n \log n)$ where n is the size of the list with some auxiliary space O(n)

Time Complexity: Θ (n log n)

Space Complexity: O (n)

Quick Sort:

As, merge sort this sorting algorithm works using divide and conquer technique but this sorting algorithm takes different time according to the list order (sorted, random or reverse sorted order) and so this sorting algorithm has different time complexity in big-Oh sense. Though merge sort is not in-place sorting algorithm but quick sort is in-place sorting algorithm.

As, mentioned earlier, according to data-set this algorithm takes different time to sort any sort. So, for average and best case scenario this algorithm runs O(n log n) order in big-Oh sense and takes O(n^2) time when appears in worst case scenario.

Time Complexity: O (n log n) for best and average case

O(n^2) for worst case

Space Complexity: O (1) auxiliary space

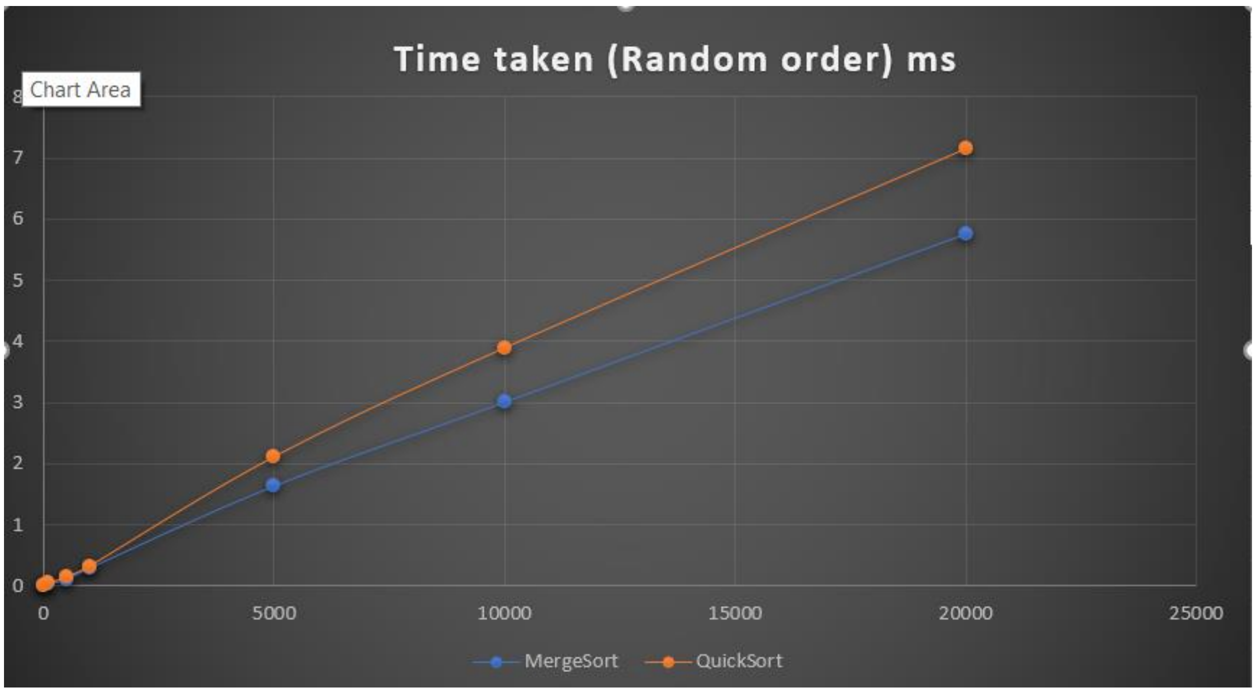
Data Table for Merge and Quick Sort:

Case	N	10	100	500	1000	5000	10000	20000
	Sort							
Random Time(ms)	Merge	0.0024	0.0245	0.1032	0.2859	1.6316	3.0088	5.7708
	Quick	0.0022	0.041	0.1404	0.32	2.1133	3.8993	7.1671
Ascending Time(ms)	Merge	0.0024	0.0141	0.1219	0.2198	10.56	2.0865	4.0866
	Quick	0.0025	0.2748	2.7186	14.0956	269.488	1198.86	4289.97
Descending Time(ms)	Merge	0.0025	0.0187	0.0807	0.1811	1.4644	2.8558	3.7871
	Quick	0.0025	0.0983	1.5067	5.9413	150.125	629.364	2384.4

Graphical Representation:

1: Time taken is Random case scenario:

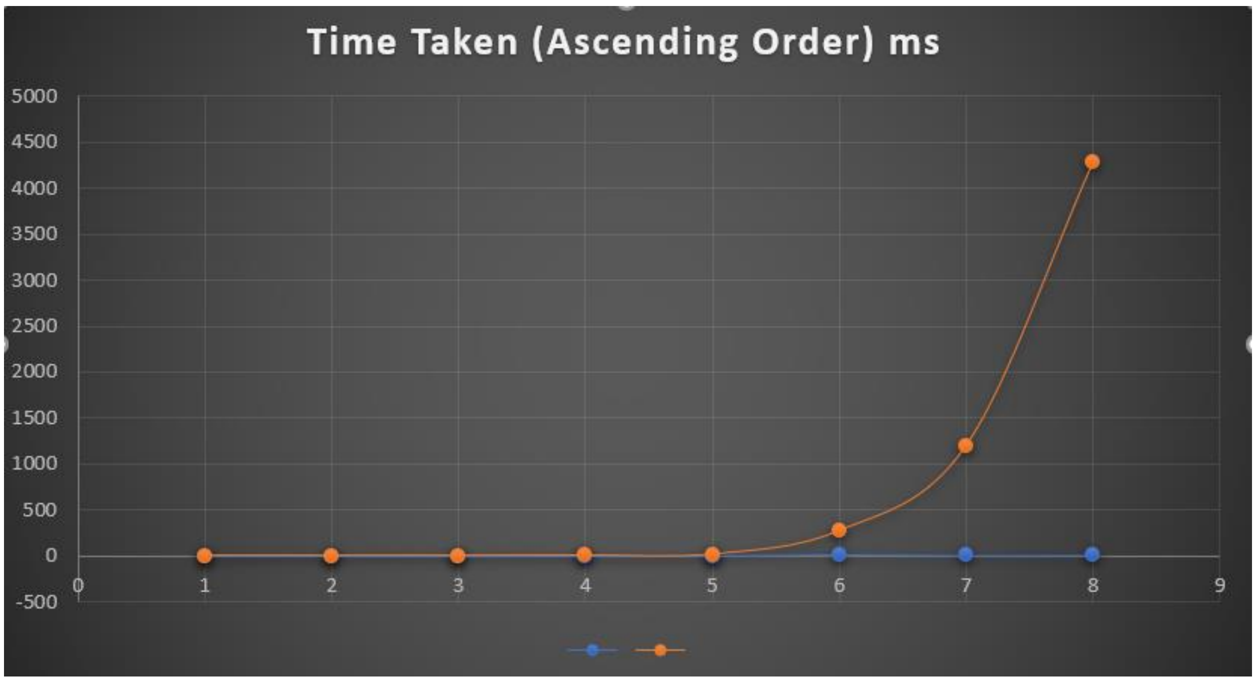
Time taken for average (random ordered case) indicates that both algorithm may run in same order in big-Oh sense and theoretically this is obviously O(n log n)



2.Time taken in Ascending case scenario:

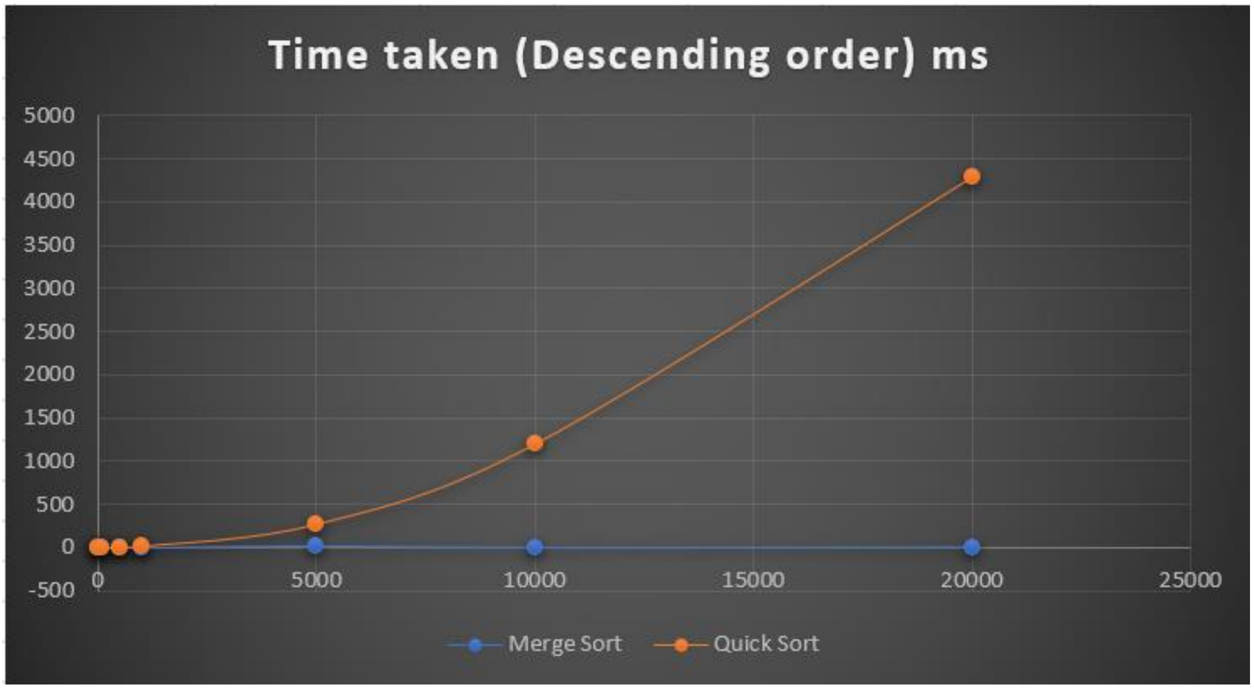
Time taken for this scenario, indicates that obviously merge sort takes less time than quick sort to make a list ordered.

Here, Orange line indicates quick sort time and green line indicates time for merge sort



3. Time taken in Descending case scenario:

Time taken for this scenario, indicates that obviously merge sort takes less time than quick sort to make a list ordered.



Analysis:

According to the time analysis and graphical representation of the time taken by the two graph, it is clear that merge sort takes O(n log n) for all scenario so we represent it by a tight upper bound Θ(n log n) but quick sort takes O(n^2) when the list is sorted or reverse sorted (descending order) and thus merge sort outperform this sorting algorithm.

Submitted By

Al Arafat Tanin

1705070