Data Structures and Algorithms

Assignment 5

~Ankur Kumar (210152)

The Objective of the Assignment

The followings are the objectives of this assignment.

1. To investigate whether Quick sort has better/poor performance than Merge sort in reality ?
2. To investigate how often Quick sort deviates from its average-case behavior in reality ?

Tasks to be done

The aim of this part is to compare Quick sort with Merge sort. For this purpose, you have to empirically compare the efficiency parameters of the quick sort and merge sort on a sequence of n uniformly randomly generated numbers for various values of n. Fill up the following tables on the basis of your experimental results. The C code has to be submitted along with the assignment.

1 Quick Sort versus Merge Sort

Notes for this section:

- Number of iterations per value of n: 500
- Units of time is microseconds (μs)

1.1 Comparisons

$n \longrightarrow$	10^2	10^3	10^4	10^5	10^6

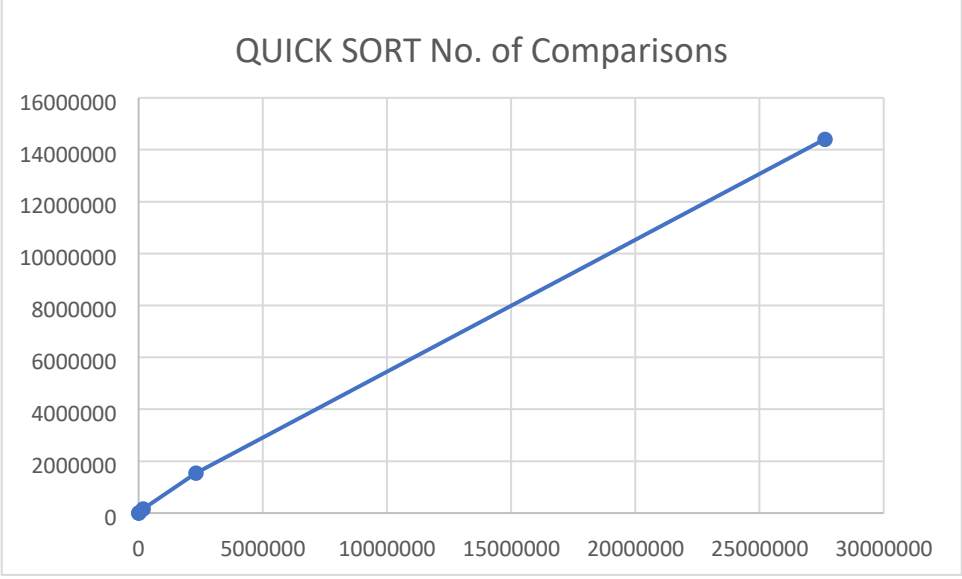
Average number of comparisons during Quick Sort	698	11490	161700	2159000	14405000
$2n\log_e n$	920	13,800	1,84,200	23,02,500	2,76,30,000
Average number of comparisons during Merge Sort	541	8707	120450	1536360	8837000
$n\log_2 n$	664	9965	132877	1660964	19931568

From the graph constructed from the data it is evident that both the algorithms make similar no. of comparisons (at average) which is of Order nlogn.

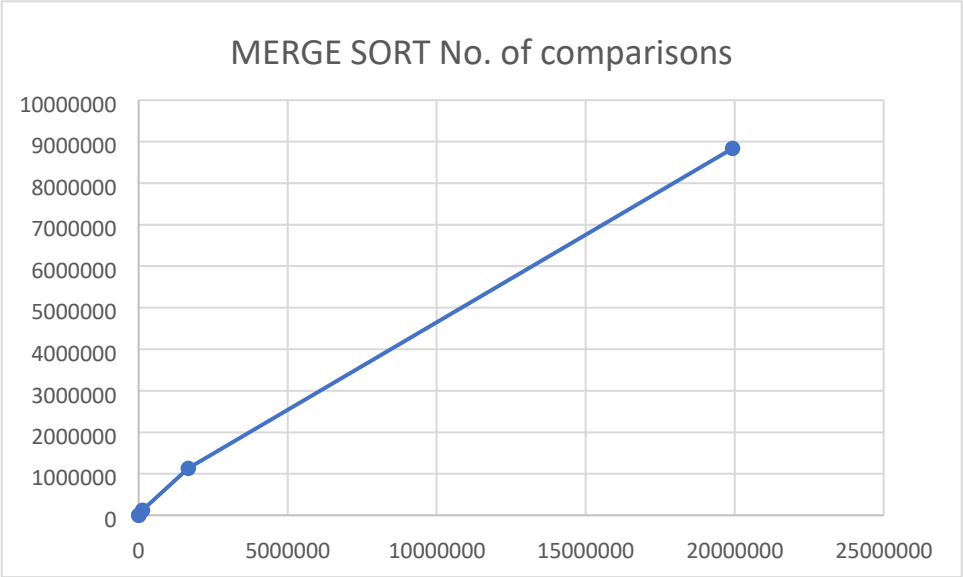
Also the number of comparisons made by MERGE SORT is consistently **lower** (initially the difference is small but becomes significant for larger n) than that of QUICK SORT indicating the faster execution of merge sort over quick sort.

Note: No. of comparison and time complexity when not stated is assumed to be average one when referred here and beyond.

(P.T.O.)



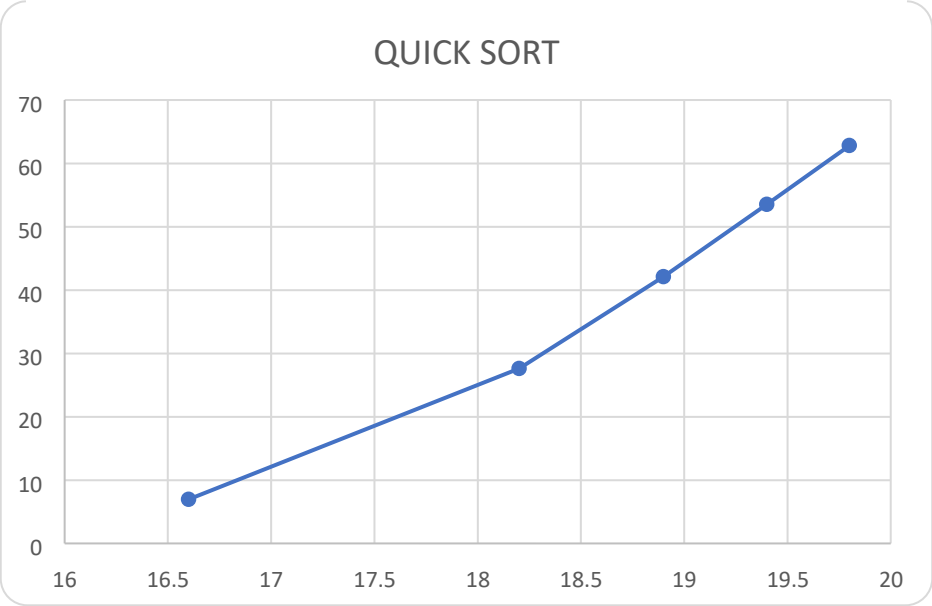
No of comparisons vs nlogn



No of comparisons vs nlogn

1.2 Number of comparisons and time complexity of quick sort

$n \longrightarrow$	10^5	$3 * 10^5$	$5 * 10^5$	$7 * 10^5$	$9 * 10^5$
Average running time of Quick Sort (in seconds)	6.9	26.6	39.1	51.5	62.8
n*logn (in multiples of 10^5) (log with base = 2)	16.6	18.2	18.9	19.4	19.8



T(n) *vs* nlogn

A: As we can see from the linearity of the T(n) vs nlogn graph that the worst case time complexity may be **O(n^2)** for QUICK SORT but on average, over a large number of iterations; it is of the order **O(nlogn)**

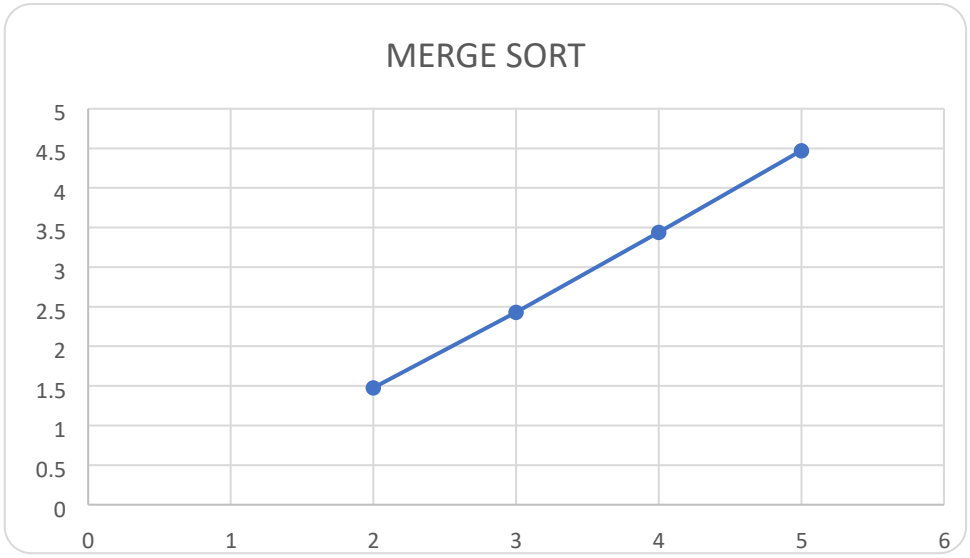
Also from 1.1 No. of comparisons is also of the order O(nlogn); same as time complexity

So we see that **no. of comparisons directly and proportionally affects the time complexity** of algorithms as it should because sorting is all about **comparing** and rearranging. So time complexity of both sorting algorithms is **O(nlogn)** on average.

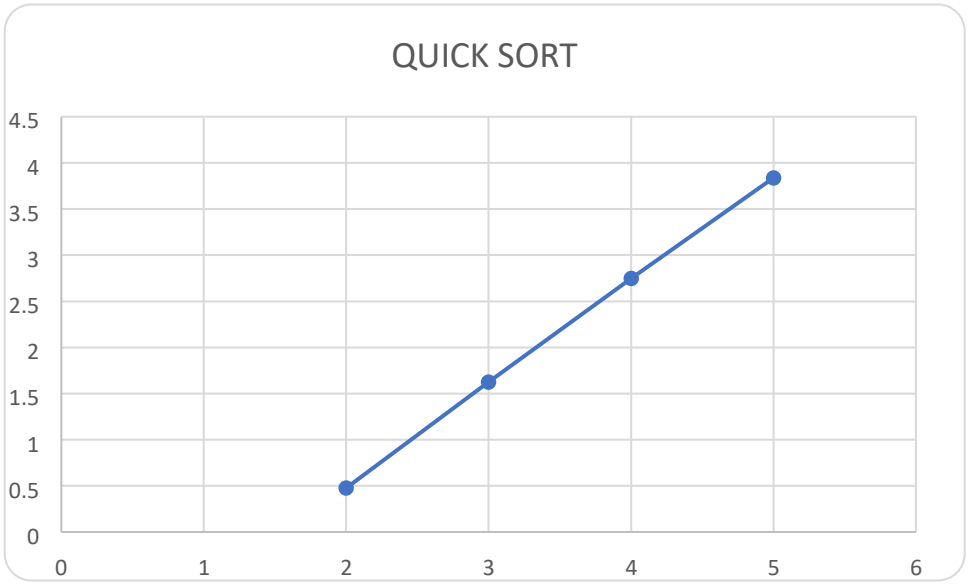
1.3 Time Complexity

$n \longrightarrow$	10^2	10^3	10^4	10^5	$5*10^5$
Average running time of Quick Sort	3	42	560	6900	42900
Average running time of Merge Sort	30	270	2754	29667	124000

Number of times Merge Sort outperformed Quick Sort(out of 500	2	1	0	3	0



Log(Tavg(n)) *vs* logn



Log(Tavg(n)) *vs* logn

Knowing that Quick sort can take time upto O(n^2) we would naturally expect it to take more average time compared to Merge sort (which takes O(nlogn) time always). Bute the reverse is evident from table. This is quite strange.

We know that average T(n) = O(nlogn) for both algorithms

So log(T) = log(O(nlogn))

=> log(T) = O(log(nlogn))

=> log(T) = O(logn + loglogn)

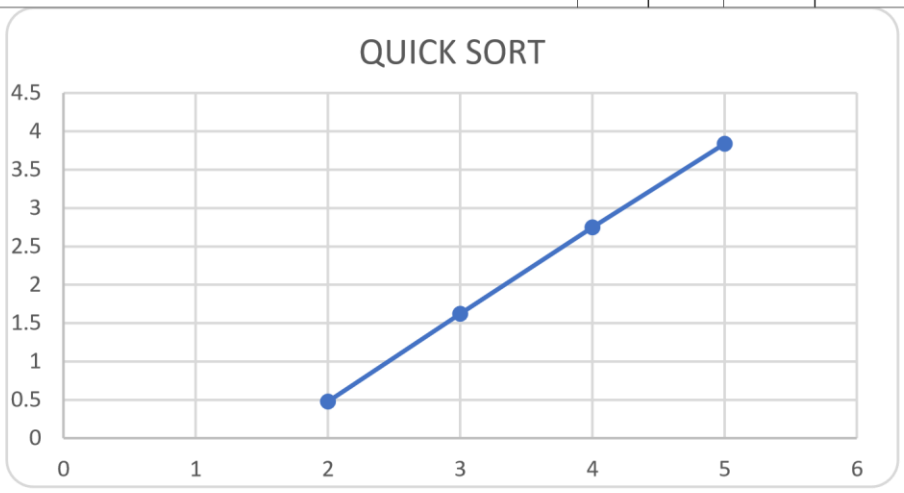
=> log(T) = O(logn) //since logn >> loglogn

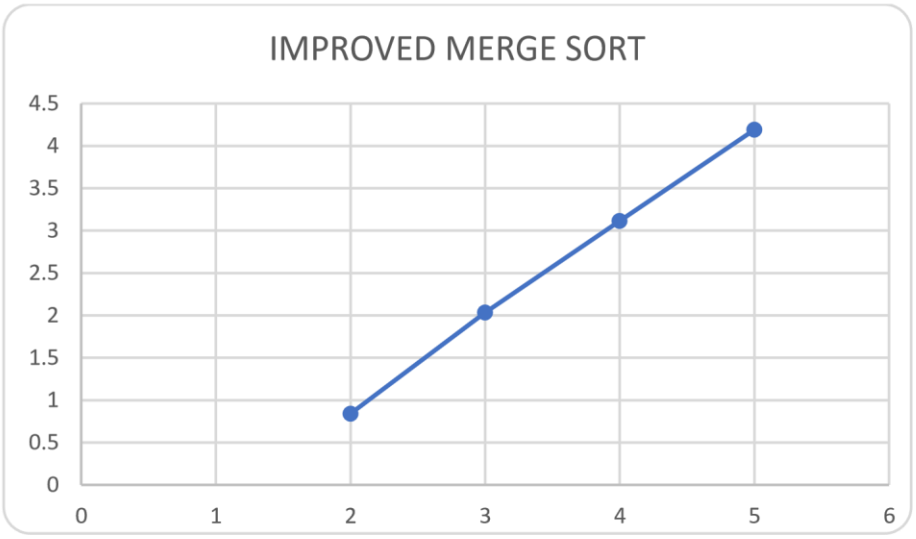
Therefore log(T(n)) is linear wrt log(n) which is obvious from the graphs

As seen from the table the average time of quick sort is always smaller than merge sort and it's only rarely that merge sort is faster and outperforms (in at max 3 out of 500 iterations). Also this has no correlation with array size and reinforces the face that Quick Sort has a lot of lead compared to Merge sort.

1.4 Can you improve merge-sort ?

$n \longrightarrow$	10^2	10^3	10^4	10^5	$5*10^5$
Average running time of Quick Sort	3	42	560	6900	42900
Average running time of Improved-Merge-Sort	7	108	1300	15500	85000
Number of times Improved-Merge Sort outperformed Quick Sort	4	31	73	492	500





In initial merge sort code we declared the array dynamically every time we made a function call to merge( ) and freed the memory after execution.
In the improved version we declared the array in the int main( ) itself and passed the pointer to that array saving us extra Space and some Time .

But, as expected, this is very minor change and still Quick Sort is faster than even the Improved merge sort!
So much for an algorithm whose time complexity is unpredictable(and has worst case time complexity O(n^2)

2. Reliability of Quick Sort

$n \longrightarrow$	10^2	10^3	10^4	10^5	$5*10^5$
Average running time of Quick Sort	3	42	560	6900	42900
No. of cases where run time exceeds average by 5%	266	129	20	51	7
No. of cases where run time exceeds average by 10%	242	100	17	35	2
No. of cases where run time exceeds average by 20%	65	92	13	1	0
No. of cases where run time exceeds average by 30%	37	82	11	1	0
No. of cases where run time exceeds average by 50%	10	61	9	0	0
No. of cases where run time exceeds average by 100%	2	19	5	0	0

Inference:

As seen from the table almost all the executions revolve around the average time only and mere 2-20 executions (out of 500) manage to take even twice as average time. (which is really minute when compared to O(n^2) worst case time since 2nlogn<<n^2)

So the worst case time complexity for Quick sort is O(n^2) but inspite of that somehow more than 90% of the cases take less than twice the average time IF THE DATASET IS RANSOMIZED. So quick sort is the ideal sorting algorithm for large collection of randomized dataset.

So to conclude Quick Sort; which is an algorithm with unpredictable time complexity outperforms Merge Sort; an algorithm which is disciplined and has same time complexity every time.

And this is due to the face that the dataset is randomized and due to that for more than 90% of cases the time taken is still O(nlogn) for Quick sort making it consume remarkably lesser time than Merge Sort on average (Still both have time complexity of same ORDER)