ESO207
Assignment-5

Instructor

Prof. Surender Baswana

Assignment Partners

Aditya Bangar (210069)

Pratham Sahu (210755)

Task 1.1

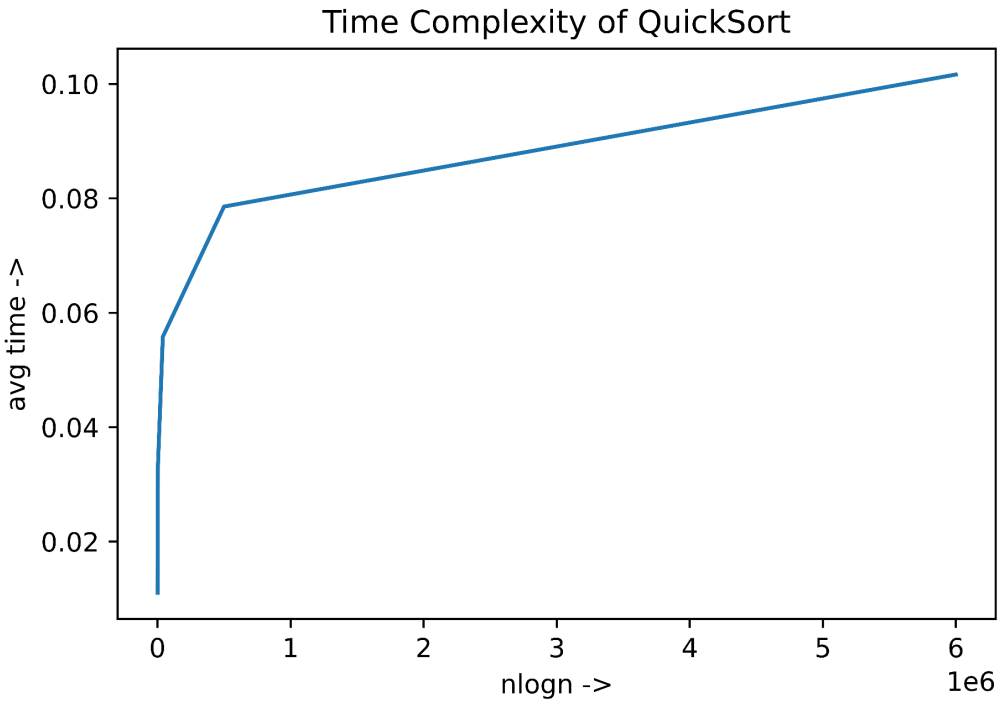
n −>	10^2	10^3	10^4	10^5	10^6
Average number of comparisons during QuickSort	579	9105	118005	1389694	13841359
$2nlog_e n$	921.034037	13815.510557	184206.807439	2302585.092994	27631021.11592
Average number of comparisons during MergeSort	356	5044	69008	853904	10066432
$nlog_2 n$	664.385619	9965.784285	132877.123795	1660964.047444	19931568.569324

Inference

- The total number of comparisons in QuickSort as seen from the above data is visibly more than that of MergeSort. This data also confirms our theoretical time-complexity analysis of QuickSort vs MergeSort that is $2nlog_e n = 1.39nlog_2 n$ is the time complexity of QuickSort which is 1.39 times the time complexity of MergeSort.
- Therefore, from the given data of number of comparisons we can infer that MergeSort should be the faster algorithm as it carries out less number of comparisons.

Task 1.2

n ->	10^5	$3 * 10^5$	$5 * 10^5$	$7 * 10^5$	$9 * 10^5$
Average running time of QuickSort	0.011034	0.033183	0.055794	0.078551	0.101627

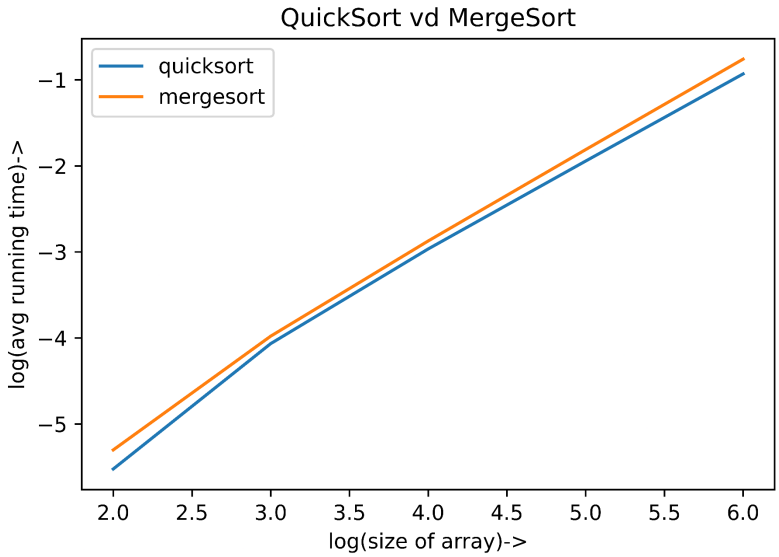


Inference

- This graph shows the time complexity of QuickSort is of the order of nlogn.

Task 1.3

n –>	10^2	10^3	10^4	10^5	10^6
Average running time of QuickSort	0.000003	0.000086	0.001083	0.011379	0.117252
Average running time of MergeSort	0.000005	0.000105	0.001341	0.015458	0.174066
Number of times MergeSort outperforms QuickSort	4	24	1	0	0

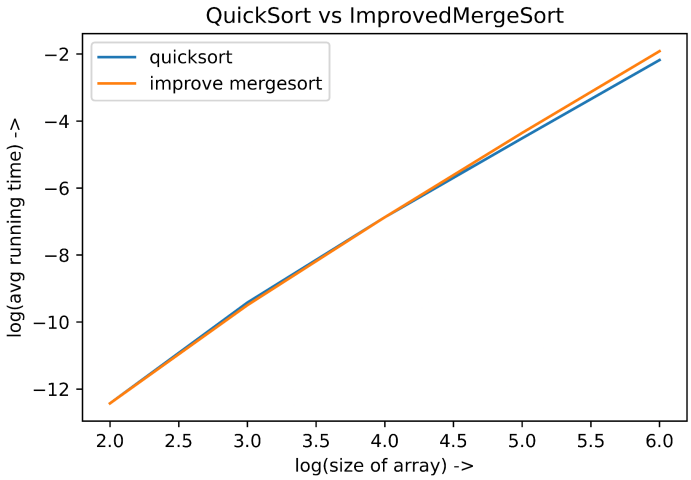


Inference

- From the above data, we see that in practice MergeSort is slower in sorting an array than QuickSort.
- This can be due to the *merge* function in MergeSort program in which it takes time at every step it merges 2 arrays, because it has to allocate memory space for the new array (*which takes time*) and also it writes the values in them.
- But on the other hand QuickSort does not create a new memory space and only carries out the swapping of values, which is a less computationally expensive process.

Task 1.4

n ->	10^2	10^3	10^4	10^5	10^6
Average running time of QuickSort	0.000004	0.000081	0.001035	0.010946	0.113124
Average running time of ImprovedMergeSort	0.000004	0.000075	0.001035	0.012851	0.147368
Number of times ImprovedMergeSort outperforms QuickSort	214	475	316	0	1



Inference

- We have improved mergeSort by carrying out Insertion for small sized arrays(*length < 20*), and thus saves the time of memory allocation to new sorted and merged array.
- The performance as seen from the data has improved for small values of n, but as the values of n increases, ImprovedMergeSort also fails to beat QuickSort.
- This shows that the task of allocation of memory space to new arrays is indeed a time taking task for the computer as compared to just reading and writing.

Task 2

n ->	10^2	10^3	10^4	10^5	10^6
Average running time of QuickSort	0.000004	0.000086	0.001066	0.011197	0.116043
No. of cases where run time exceeds 5%	222	262	31	21	27
No. of cases where run time exceeds 10%	222	117	11	8	2
No. of cases where run time exceeds 20%	222	8	1	1	0
No. of cases where run time exceeds 30%	3	2	1	0	0
No. of cases where run time exceeds 50%	3	1	0	0	0
No. of cases where run time exceeds 100%	1	0	0	0	0

Inference

- Since we know that the worst case time-complexity of QuickSort algorithm is $O(n^2)$. This might make us consider that using MergeSort is a better option as it might be slower but is quite consistent with the time taken.
- But as seen from the above data, we can conclude that in practice using QuickSort is still a reliable option for large values of n, as time taken for sorting the array still lies very close to average case time-complexity.