Programming Assignment 4

Jayant Agrawal 14282
Shubham Pandey 14679

1 Quick Sort versus Merge Sort

No of iterations : 2000

	$n = 10^2$	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$
Average running time of Quick Sort(s)	0.000006	0.000090	0.001187	0.014851	0.178958
Average running time of Merge Sort(s)	0.000007	0.000102	0.001330	0.016136	0.195267
Average number of comparisons during Quick Sort	645.35	10980.48	155472.53	2016323.09	24766309.45
The value of $1.39nlog_2n$	923	13852.44	184699.20	2308740.02	27704880.31
Average number of comparisons during Merge Sort	541.97	8707.60	120454.70	1536395.64	18674389.63
The value of $nlog_2n$	664.38	9965.78	132877.12	1660964.04	19931568.57
Number of times Merge Sort outperformed Quick Sort	22	35	5	3	2

1.1 Inference

- Number of Comparisons in Quick Sort are more than Merge Sort, but still Quick Sort takes less time, for all n.
  - Merge Sort takes time for copying data from the temporary array to the original array.
  - For Quick sort almost half of the comparisons do not initiate any further operation (swapping, copying etc).
- Merge Sort may perform better when it is used on linked lists, where memory manipulations are easier and faster.
- With increasing n, the probability of tha sample being close to the worst case sample for quick sort reduces, so does the probability of merge sort outperforming quick sort.

2 Distribution of the running time of Quick Sort

No of iterations : 2000

2.1 Table

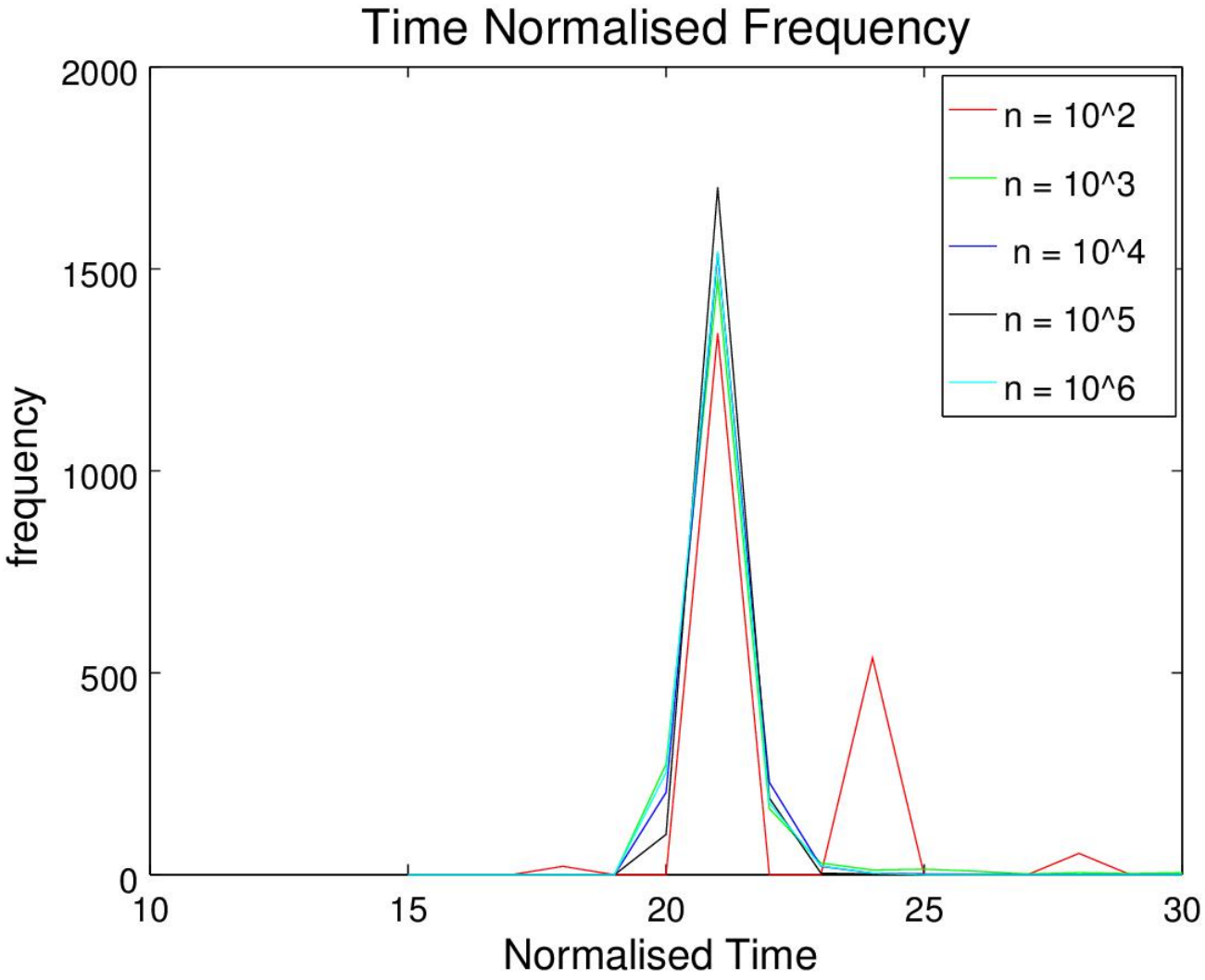
	$n = 10^2$	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$
Average running time of Quick Sort(s)(μ)	0.000006	0.000090	0.001187	0.014851	0.178958
$1.39nlog_2n$	923	13852.44	184699.20	2308740.02	27704880.31
Average number of comparisons during Quick Sort	645.35	10980.48	155472.53	2016323.09	24766309.45
# cases where run time exceeds average by 5%	638	253	259	198	204
# cases where run time exceeds average by 10%	638	89	30	8	26
# cases where run time exceeds average by 20%	101	48	5	2	0
# cases where run time exceeds average by 30%	101	25	1	1	0
# cases where run time exceeds average by 50%	48	10	0	0	0
# cases where run time exceeds average by 100%	15	0	0	0	0

2.1.1 Inference

	$n = 10^2$	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$
Variance(σ^2)	2.7×10^{-9}	6.18×10^{-8}	1.98×10^{-6}	2.05×10^{-4}	0.036
Coefficient of Restitution (C_X)(σ/μ)	8.66	2.76	1.16	0.96	1.0

- For a given 'n', no of cases decreases as we move away from average showing that the probability of the worst case is very low.
- For increasing 'n', no of cases exceeding a given % above μ decreases showing that the run time moves further closer to the average value.
- As n increases, C_X decreases, which further confirms the fact that for a large data size the distribution of the running time gets sharper with mode being closer to the average.

2.2 Plots



2.2.1 Plot Description

For each 'n', taking 5% of the average as the bin size for the histogram, frequency vector is calculated. Then, all such vectors are plotted on the graph. The x-range is 10-30 for better pictorial representation.

2.2.2 Inference

- For smaller 'n' value, the plot shows relatively lesser sharpness, showing that for small n , the probability for the sample to be closer to the worst case is higher.(But, this does not matter, since for small 'n' run time is already very less).
- As evident from the graph, that the peak of the plot is higher for greater 'n'. This implies that for increasing 'n', frequency near average increases.

3 Further Observations

1. Theoretically we have calculated the time complexity based on no of comparisons, but we have ignored the time taken in memory input/output. In quick sort,we are comparing with the same pivot element, whereas in merge sort we are comparing different elements. The cache is utilised well in the case of quick sort.
2. The worst case time for quick sort is $O(n^2)$. Some of the cases where this happens, can be improved with better selection of pivot(randomized or median). Moreover, if the data can be randomized initially to avoid the worst case(sorted array) for quick sort.
3. For any 'n', the probability of the sample being sorted is $1/(n!)$. So, for larger 'n' , it is less likely to get the worst case for quick sort.