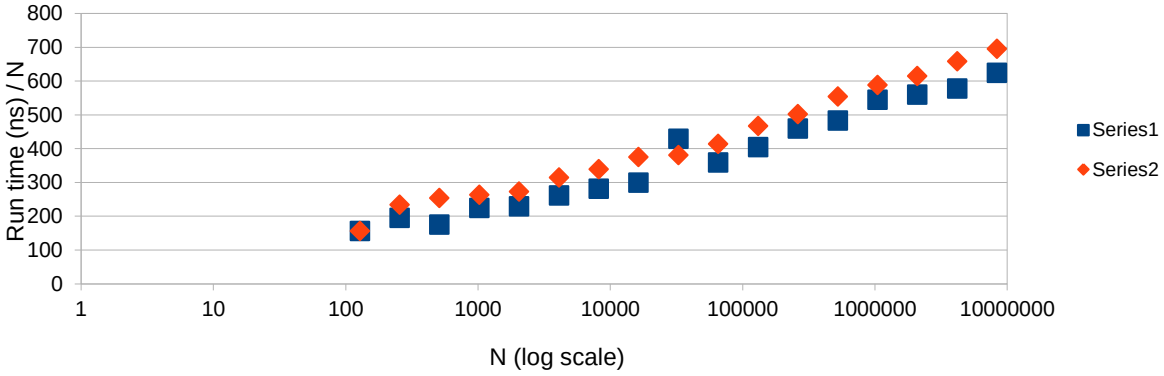
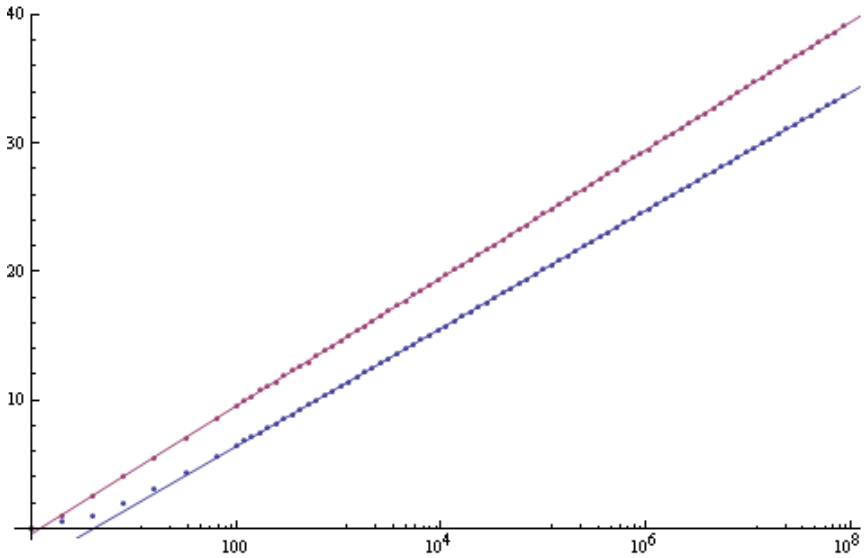
Q2) First we look at the runtime (in nanoseconds) versus size of the input instance (n), for random instances [without averaging over all random instances of size n]. However, in case of quick sort, pivots are always chosen randomly, which is equivalent to shuffling the input instance.



The plot of runtime/N versus log(N) looks quite linear , as is expected because the theoretical time complexity of the merge sort is NLog(N) and the average case time complexity of quick sort is also O(N*Log(N)). However, for the further studies, this approach of measuring run times is dispensed with in favour of counting the number of comparisons in case of both algorithms. The disadvantage of plotting run times is that they are sensitive to operating system, they are also slightly sensitive to the cache size (L1, L2, RAM), as will be evident from slightly different slopes in three different ranges for N. The other disadvantage with measuring run time is that it is very difficult to accurately measure time intervals on small scales like nanoseconds, and often require looping which makes it more time consuming. Finally, if we follow this approach, and try to average over all random instances, it will take exponential time!

It is known from cost time analysis study that overheads < swaps < comparisons. Moreover, in my implementation of quick sort, the # of comparisons > # of swaps – 2. Hence a program is written to model the sort algorithm, which will give the number of comparisons.

Plotted below is the **average** number of comparisons executed by both the algorithms (averaged over all random input instances) . Note that the dots are the experimental data points while the straight line connecting them is the experimental fitting.

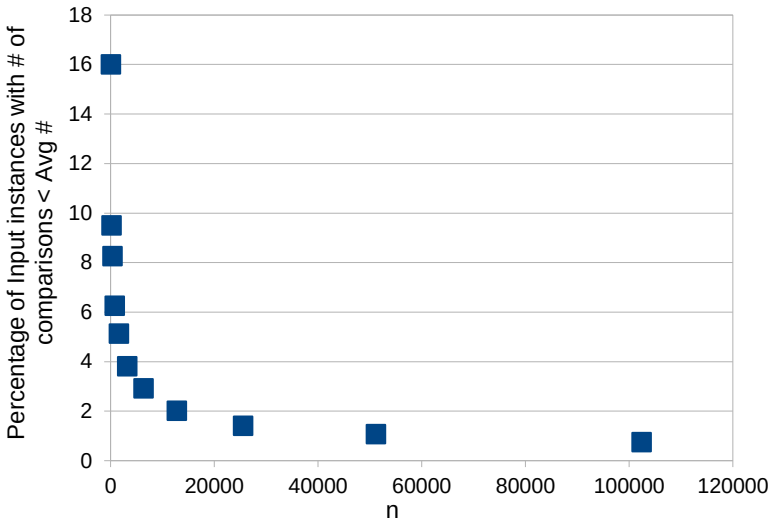


Theoretically it is known that for quick sort, the average number of comparisons, $C_n = (n+1)(2H_{n+1} - 2) - 2n \sim 2n \log_e n + O(n)$. Hence, the fitting of the data has been done to the form: $\frac{C_n}{n} = a \log_e n + b$, to a first approximation.

	Value of a	Std deviation in a
Quick Sort	1.99636(~ 2 as expected)	0.000488171
Merge Sort	2.16387	0.00150803

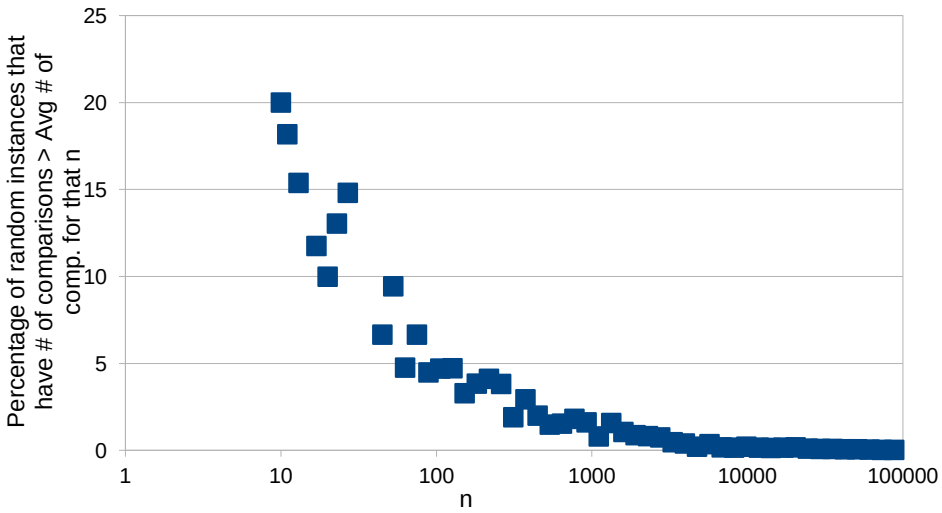
Moreover, from the value of b, we can determine the O(n) term of C_n for quick sort to first approximation as -2.79n. So $C_n \sim 1.996n \log_e n - 2.79n$. Since the nLogn term is dominating we can say that the algorithm is O(nLogn).

Next, for quick sort,we analyse what is the fraction of random input instances of give size n_0, having number of comparisons > c * C_{n_0} , where C_{n_0} is the average number of comparisons for input size n_0 and c > 1. We take c = 1.1 and c=1.01.



We see that the percentage falls rapidly in the beginning as n increases and continues to fall even at large n. For n = 102400, it drops to as low as 0.75 %. This is captured by saying that the quick sort algorithm asymptotically becomes a O(nlogn) deterministic algorithm

Shown below is another sample, now with c=1 and plotted with a *logarithmic scale* on n – axis.



In this case, beginning with 20 % at n=10, the percentage drops to 0.02 % at n=88151 .