CS648 Assignment - 1

Team Name:

$\mathscr{Z}\mathscr{J}\mathscr{y}\mathscr{e}\mathscr{e}\mathscr{o}\ \mathscr{H}\mathscr{n}\mathscr{e}\mathscr{e}\mathscr{u}\ \mathscr{Y}\mathscr{x}\mathscr{b}\mathscr{I}\mathscr{z}\ \mathscr{V}\mathscr{R}$

Soham Samaddar 200990

Aditya Tanwar 200057

January 2023

Q1. Randomized Quick Sort versus Quick Sort

Compare the two sorting algorithms based on their run-time, number of comparisons, and time taken to "double-sort" for different n.

Solution: The sorting algorithms (acronyms used to save space) were run on arrays of sizes $\{10^2, 10^3, 10^4, 10^5, 10^6\}$, with 2000 iterations for each value. The double sorting time for $n = 10^6$ for QS however has been extrapolated from theoretical expectations (elaborated further in the optional section). The following results were obtained:

$n \rightarrow$	10^2	10^3	10^4	10^5	10^6
$2n \log_e n$	9.21×10^2	1.38×10^4	1.84×10^5	2.30×10^6	2.76×10^7
Average running time of QS	10.83	1.90×10^2	1.90×10^3	2.22×10^4	2.63×10^5
Average running time of RQS	13.71	1.60×10^2	2.00×10^3	2.35×10^4	2.79×10^5
Average comparisons during QS	6.46×10^2	1.09×10^4	1.56×10^5	2.02×10^6	2.48×10^7
Average comparisons during RQS	6.46×10^2	1.10×10^4	1.56×10^5	2.01×10^6	2.48×10^7
Average double sort time by QS	42.33	2.42×10^3	1.65×10^5	1.58×10^7	9.84×10^8
Average double sort time by RQS	23.35	2.80×10^2	3.23×10^3	3.45×10^4	4.09×10^5

Table 1: Statistics of RQS and QS

Note: The times (running and double sort) have been written in μs for $n \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$.
Note: This value was interpolated by finding line of best fit between $\log_e n$ and available double sort times (of QS). The rationalization behind this approach was the assumption that the quadratic term of n (due to the second time that QS sorts the array) will be the most dominating one.

Our inferences have been stated below:

- The **running time** of the algorithms is *almost the same* except that RQS takes marginally more time than QS, which can be explained by the extra computation RQS does while choosing a pivot.
- Another supporting claim in favour of the inference above is the quantity,

  $$\Delta(\text{Run time})/(\text{Run time of RQS})$$

  which seems to be approaching a constant value (≈ 0.057) as n grows. Thus, the proportion of time spent in computing a random pivot is 5.7% for RQS.

- Further, the running time grows linearly with $n \log_e n$ with the constant (of proportionality) reaching towards 0.02 (when runtime is in μs.) as n grows.
- The **number of comparisons** is almost the *same* for both QS and RQS. This is also "expected" theoretically since both algorithms' behavior is the same for most inputs. The difference is most likely statistically insignificant.
- The **number of comparisons** grows *linearly* with $n \log_e n$, with the constant (of proportionality) apparently reaching ≈ 2 as n grows.
- Lastly, the double sort time for RQS grows linearly with single sort time (and thus with $n \log_e n$), while the double sort time for QS seems to be growing quadratically with n.
  This is also expected theoeretically since RQS is indifferent to the permutation of input array, while for QS, a sorted array is the worst case (for runtime) where after each partitioning, the size of the subproblem decreases only by a constant amount, and hence the quadratic runtime.

Q2. Randomized Quick Sort versus Merge Sort
Compare the two sorting algorithms based on their run-time and number of comparisons for different n.

Solution: The sorting algorithms (acronyms used to save space) were run on arrays of sizes $n \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$, with 2000 iterations for each value of n, and the following results were obtained:

$n \rightarrow$	10^2	10^3	10^4	10^5	10^6
Average running time of QS	10.72	1.65×10^2	1.99×10^3	2.20×10^4	2.61×10^5
Average running time of MS	56.09	5.80×10^2	5.14×10^3	5.08×10^4	5.30×10^5
Average comparisons during QS	6.49×10^2	1.10×10^4	1.56×10^5	2.02×10^6	2.48×10^7
$2n \log_e n$	9.21×10^2	1.38×10^4	1.84×10^5	2.30×10^6	2.76×10^7
Average comparisons during MS	3.16×10^2	4.93×10^3	6.46×10^4	8.15×10^5	9.88×10^6
$n \log_2 n$	6.64×10^2	9.96×10^3	1.33×10^5	1.66×10^6	1.99×10^7
No. of times MS outperformed QS	0	0	0	0	0

Table 2: Statistics of RQS and MS

Note: All the running times have been written in μs for $n \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$.

Our inferences have been stated below:

- The **running time** of MS is *almost twice* that of QS (as n grows), most likely due to the overhead that MS suffers from the creation of *duplicate subarrays*. QS is inplace which facilitates less memory usage compared to MS and also allows for more cache hits during memory accesses.
- The **number of comparisons** of MS is the same for each input instance (corresponding to a fixed n); this is expected theoretically as well since MS is completely

deterministic, regardless of input array's permutation. On the other hand, the number of comparisons in QS is *larger* than that of MS (almost a *constant ratio* of 2). Both of them, however, grow *linearly* with $n \log_e n$.

- The number of times MS outperforms QS is 0 (in 2000 samples for each n); Though, the number of comparisons made by MS are almost *half*, it is still not able to outperform QS, due to implementation differences, as elaborated above.

Q3. Randomized Quick Sort

Empirically find the deviation of the randomized quick sort from its average running time for different values of n.

Solution: The algorithm (acronym used to save space) was run on arrays of sizes $n \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$, with 2000 iterations for each value of n, and the following results were obtained:

$n \rightarrow$	10^2	10^3	10^4	10^5	10^6
Average running time of RQS	11.43	1.55×10^2	1.93×10^3	2.18×10^4	2.60×10^5
$2n \log_e n$	9.21×10^2	1.38×10^4	1.84×10^5	2.30×10^6	2.76×10^7
Average comparisons during RQS	6.46×10^2	1.10×10^4	1.56×10^5	2.18×10^6	2.60×10^7
Run time exceeds average by 5%	720	625	543	117	79
Run time exceeds average by 10%	454	482	345	36	23
Run time exceeds average by 20%	300	289	131	6	5
Run time exceeds average by 30%	192	178	54	1	1
Run time exceeds average by 50%	65	74	9	1	0
Run time exceeds average by 100%	24	10	1	0	0

Table 3: Statistics of RQS

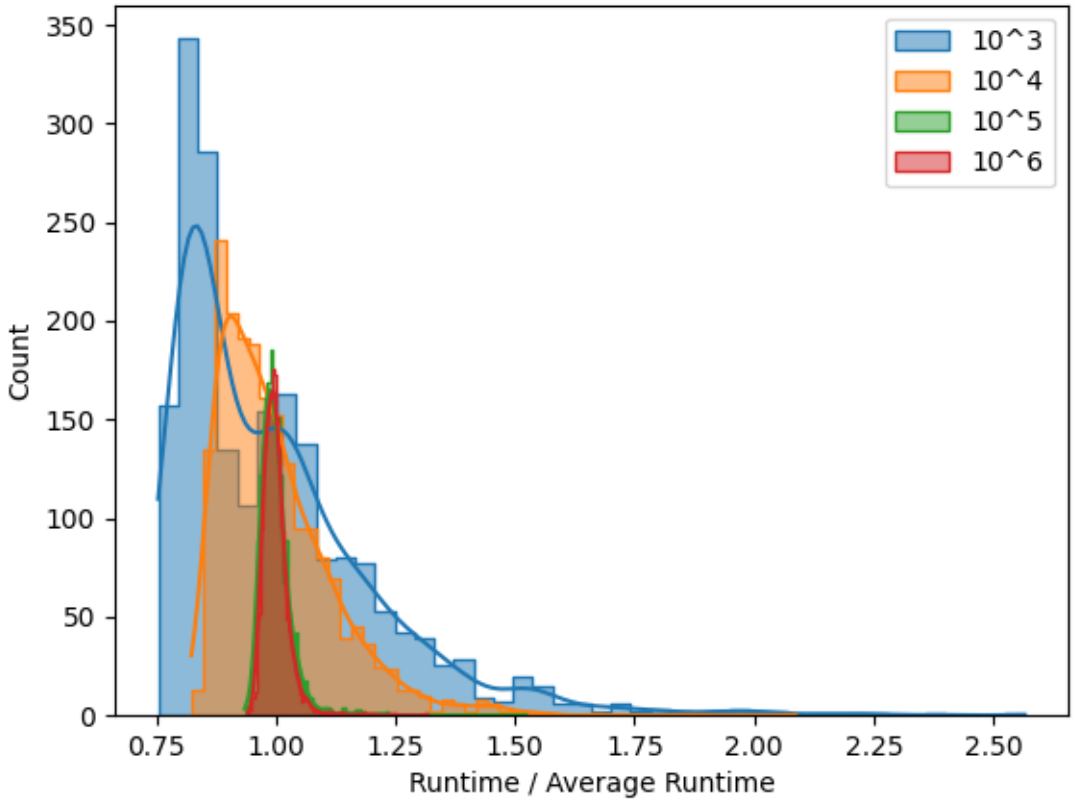
Note: All the running times have been written in μs for $n \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$.

Our inferences have been stated below:

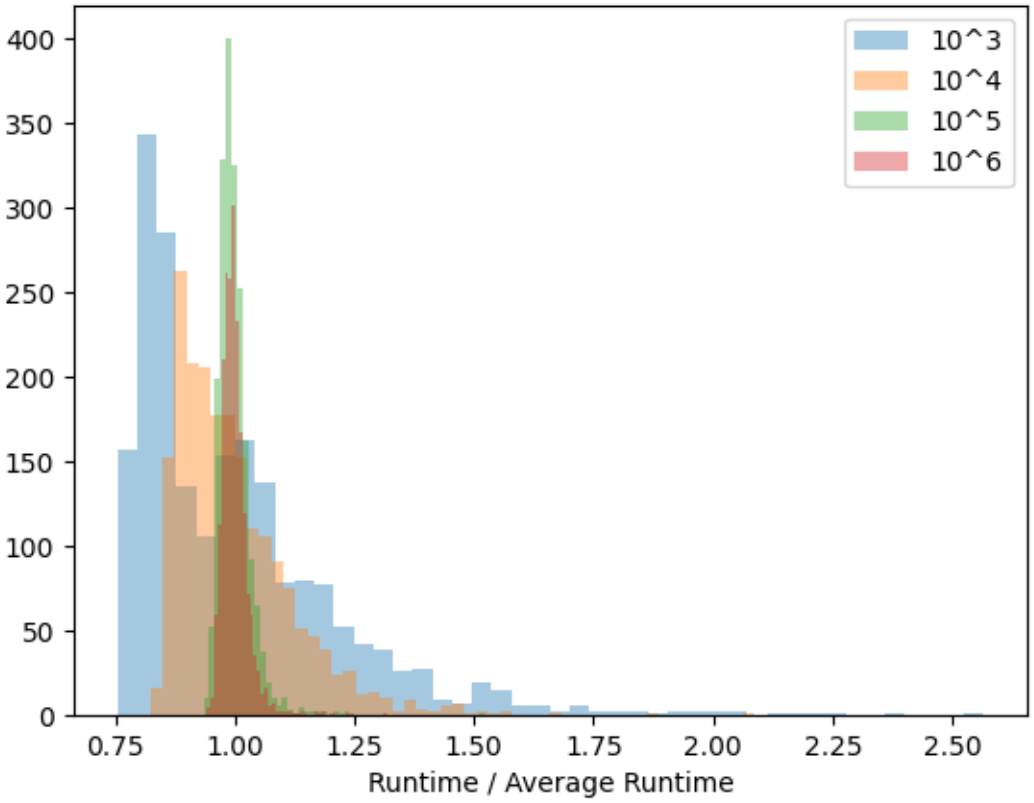
- With increase in the value of n, the deviation from the average runtime decreases. For $n = 10^6$, 94.6% of the runtimes are within 5% of the average runtime. This fact is even more apparent from the graphs in the following section.
- For smaller values of n the variance in the runtime is quite high. This is probably because for smaller n, there is a greater probability for the array to be an outlier (nearly sorted/reversed) which greatly affects the runtime of QS.
- The average number of comparisons is **strictly less than** $2n \log_e n$ which is consistent with theory since the expected number of comparisons is bounded above by $2n \log_e n - O(n)$.

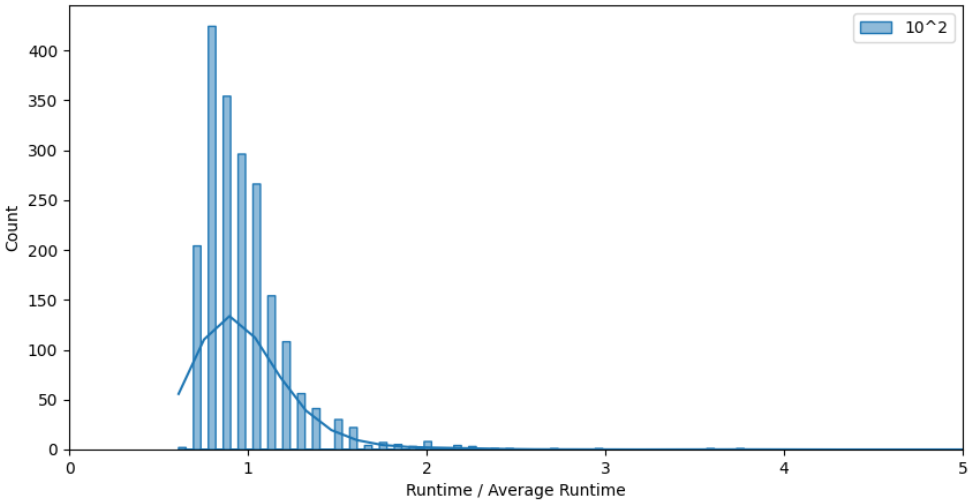
Optional Section

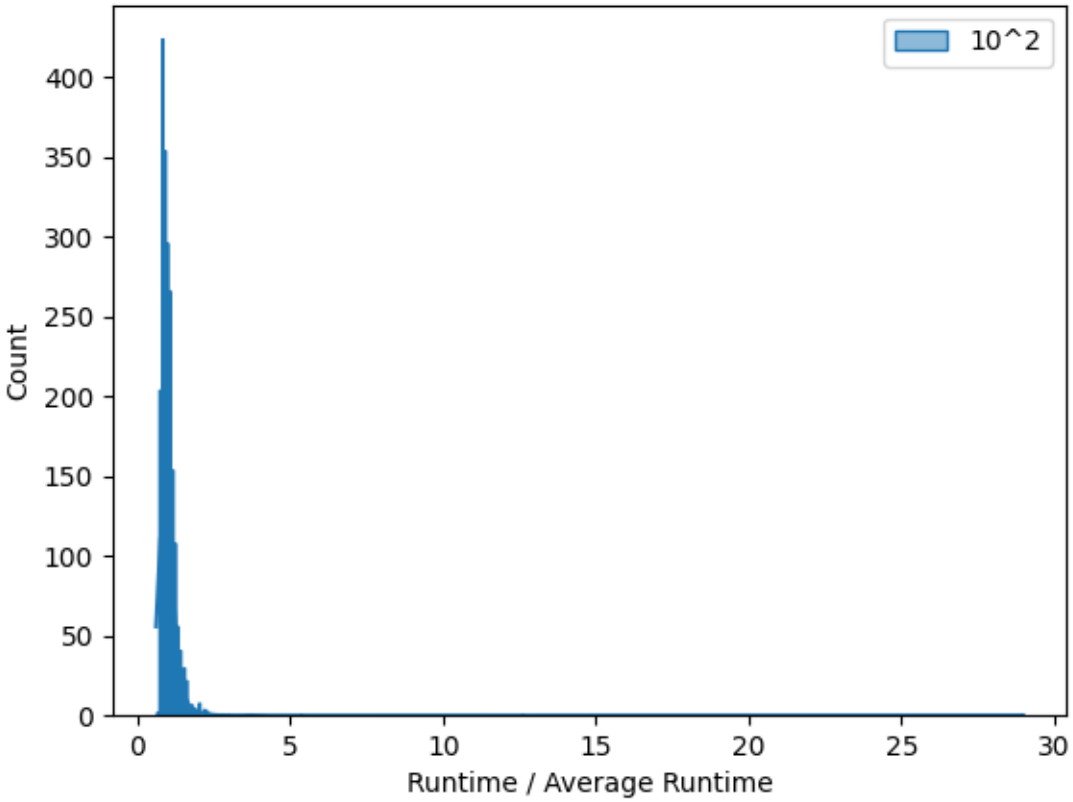
The data from **Q3** was used to make the following histogram plots. On the x-axis, are the "normalized" values of runtimes (with respect to average runtime for that particular n), and the y-axis contains count for each value of the ratio.
Additionally, the first two graphs contain the same data, but just represent the data differently for $n \in \{10^3, 10^4, 10^5, 10^6\}$. The graphs for $n = 10^2$ had to be drawn separately because of the huge difference in statistics (for instance, the ratio for $n = 10^2$ goes up to ≈ 28, which completely disrupted the scale for the axes. Hence, the data has been shown in two graphs, one with a truncated scale for better visibility around the mean, and one with the complete scale for the sake of simplicity.



It is very easy to see that the graph concentrates around 1.00 (i.e., Runtime $\approx$ Average Runtime) for larger values of n. Interestingly enough, instead of being symmetric with higher variance for smaller values of n, the graphs instead exhibit a positive skew.


10^3
10^4
10^5
10^6
400
350
300
250
200
150
100
50
0
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
Runtime / Average Runtime


10^2
Count
400
350
300
250
200
150
100
50
0
0
1
2
3
4
5
Runtime / Average Runtime


10^2
400
350
300
250
200
150
100
50
0
Count
0
5
10
15
20
25
30
Runtime / Average Runtime