演算法 PA1

B09901080 電機三 吳宣逸

Data Records

- The following data are run on EDA union lab machine.
- QS implemented here is Randomized Quicksort.

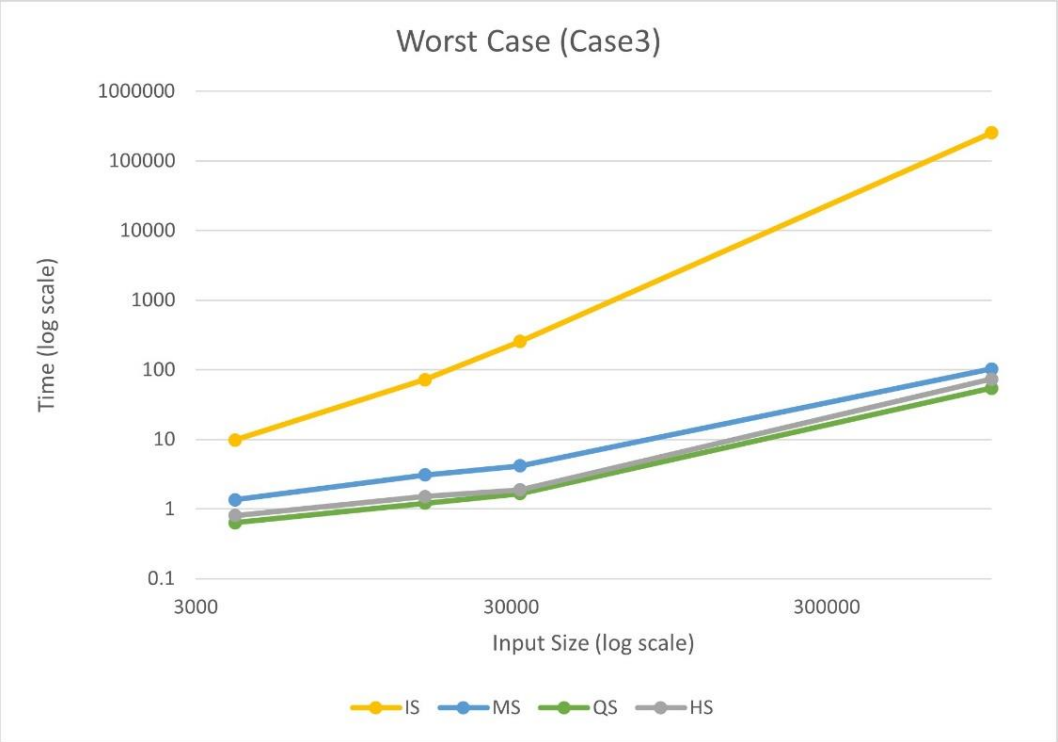
Input size	IS		MS		QS		HS	
	CPU time (ms)	Memory (KB)	CPU time (ms)	Memory (KB)	CPU time (ms)	Memory (KB)	CPU time (ms)	Memory (KB)
4000.case2	0.142	5904	1.013	5904	0.756	5904	0.604	5904
4000.case3	9.822	5904	1.36	5904	0.637	5904	0.808	5904
4000.case1	5.307	5904	2.11	5904	1.161	5904	0.94	5904
16000.case2	0.178	6056	2.567	6056	1.797	6056	2.246	6056
16000.case3	72.309	6056	3.085	6056	1.211	6056	1.516	6056
16000.case1	35.834	6056	4.065	6056	2.106	6056	1.515	6056
32000.case2	0.184	6188	4.39	6188	2.256	6188	2.913	6188
32000.case3	255.188	6188	4.156	6188	1.665	6188	1.892	6188
32000.case1	135.075	6188	6.024	6188	3.435	6188	2.433	6188
1000000.case2	1.115	12144	97.37	14004	51.173	12144	75.13	12144
1000000.case3	253297	12144	103.145	14004	54.35	12144	73.234	12144
1000000.case1	126524	12144	170.991	14004	94.562	12144	139.105	12144

Trendlines

- Average Case (Case1)

- Best Case (Case2)

- Worst Case (Case3)



- Analyze the Slopes of Trendlines

According to theory, there are 3 possible asymptotic notations for time complexity of sorting algorithms implemented in PA1: O(N), O(NlgN), O(N^2). Since all trendlines are plotted in log-scale, we take some transformations:

$$\begin{cases} t = \log(x) \\ n = \log(y) \end{cases}$$

and the slope becomes

$$\frac{dt}{dn} = \frac{\frac{\partial \log(y)}{\partial x}}{\frac{\partial \log(x)}{\partial x}}$$

.

Therefore, after some computation, we get the slope of each time complexities as following:

Time Complexity	Slope ($\frac{dt}{dn}$)
O(N)	1
O(NlgN)	$1+\frac{1}{\ln N}$ (about 1.07~1.12 in PA1)
$O(N^2)$	2

. We observed that the slopes have the relation that O(N) < O(NlgN) < $O(N^2)$, note that asymptotic notations hold when N is large enough. With simple measurement to the slopes of trendlines in charts above, we find that their slopes meet the theoretical time complexities:

	IS	MS	QS	HS
Average Case	N^2	NlgN	NlgN	NlgN
Best Case	N	NlgN	NlgN	NlgN
Worst Case	N^2	NlgN	NlgN	NlgN

▪ Data Structures Used

- Vector (<vector>)

▪ Discussions and My Findings

Among above trendlines, there are some differences with those given in pa1.pdf. So here are the discussions.

• (Randomized) Quicksort

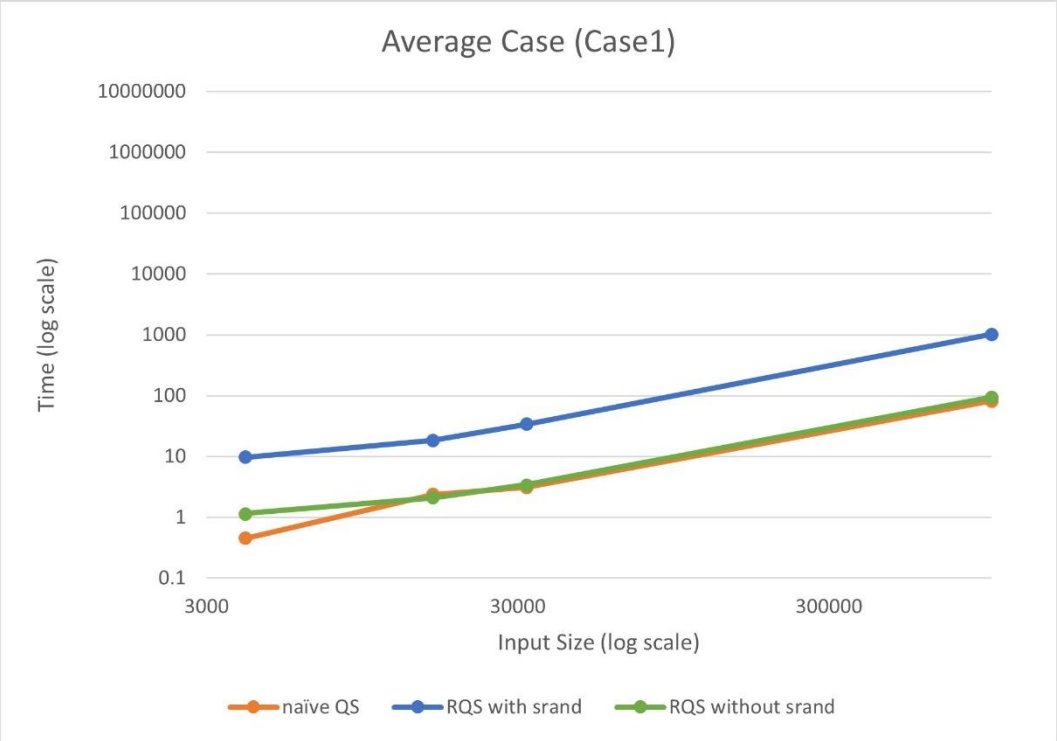
I've modified my Quicksort algorithm 2 times.

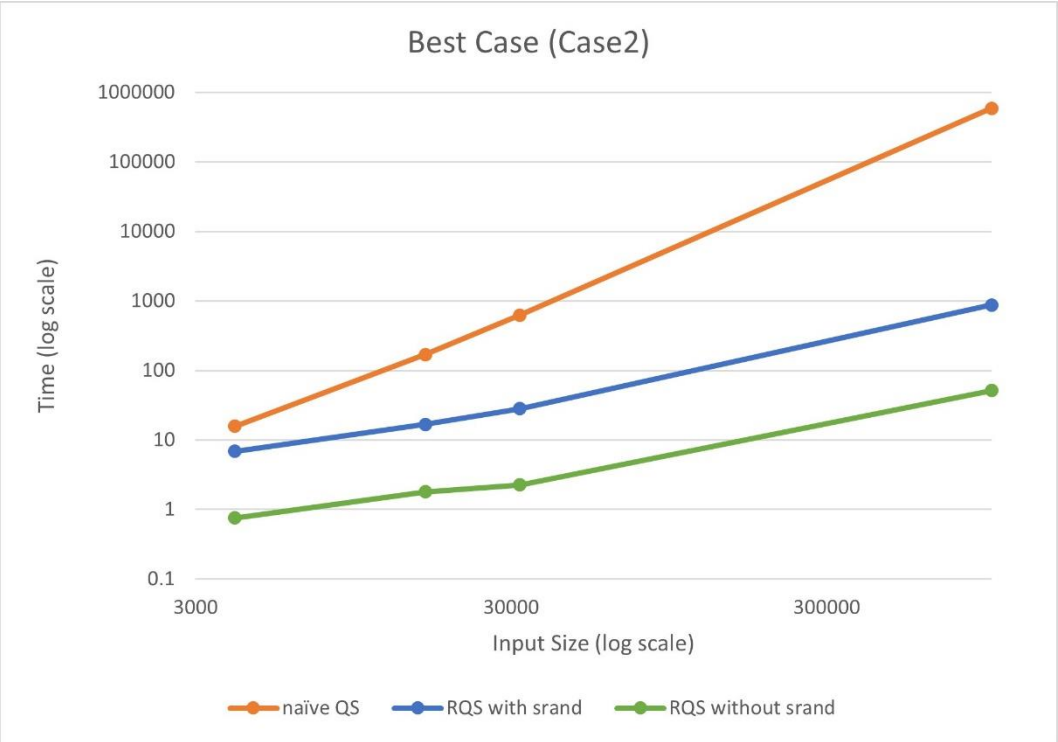
My first modification is using random pivot in SortTool::Partition, realized with std::srand() and std::rand(). This version of Quicksort has got a significant advance in CPU runtime of case2 and case3, but is still not "quick" enough.

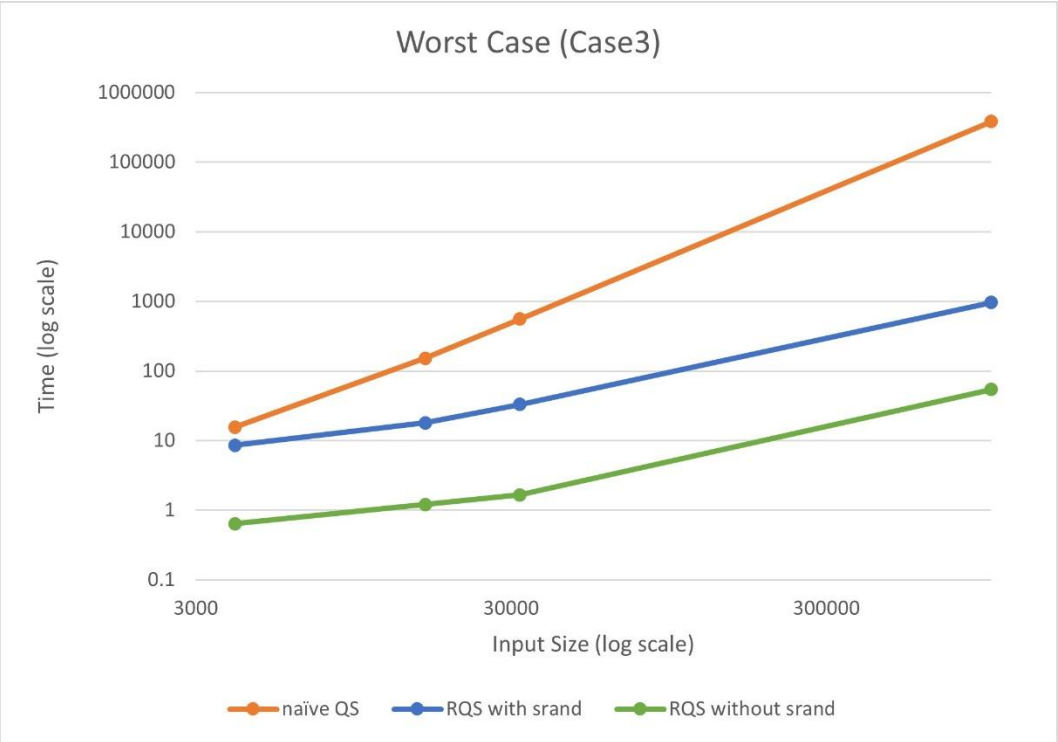
My second modification is removing std::srand() from my code. std::srand() is the function to set seed for std::rand() in C/C++ <stdlib>. Although we used to set seed before generating a random number, but it's not the must. Usually, the purpose for setting a seed beforehand is either to get more random result with another random parameter such as current time or to get a fixed result with a constant seed. However, an extremely high-quality randomized pivot doesn't imply a good CPU runtime. Instead, the function std::srand() takes too much time, and slow down the whole algorithm.

Comparison of 3 Quicksort algorithms is shown in the following table and charts.

Input size	naïve QS		RQS with srand		RQS without srand	
	CPU time (ms)	Memory (KB)	CPU time (ms)	Memory (KB)	CPU time (ms)	Memory (KB)
4000.case2	15.697	5968	6.843	5904	0.756	5904
4000.case3	15.651	5904	8.536	5904	0.637	5904
4000.case1	0.452	5904	9.734	5904	1.161	5904
16000.case2	168.71	6684	16.735	6056	1.797	6056
16000.case3	152.32	6304	17.977	6056	1.211	6056
16000.case1	2.361	6056	18.417	6056	2.106	6056
32000.case2	624.855	7500	28.265	6188	2.256	6188
32000.case3	558.842	6740	33.167	6188	1.665	6188
32000.case1	3.117	6188	34.064	6188	3.435	6188
1000000.case2	588223	56840	871.523	12144	51.173	12144
1000000.case3	381237	27252	961.383	12144	54.35	12144
1000000.case1	81.509	12144	1012.25	12144	94.562	12144







- Insertion Sort

In the given 3 charts, the trendline of Insertion sort crosses with other three trendlines, but it doesn't happen in my charts.

I also observe that all my trendlines are not straight lines as shown in pa1.pdf, especially when input size is small. There are some reasons:

1. In theory, or in the given charts, insertion is "the fastest" among these 4 sorting algorithms when input size N is small. I think the problem is that we start our test from 4000, which itself seems to be a "big" number; therefore, we doesn't see this result in our charts .

2. The “complexity” we say are asymptotic, i.e., when input size is small, small terms still affect the runtime, so most trendlines are not straight when input size is small.
3. Because I run data on EDA union lab machine, an environment with multiple unknown tasks running simultaneously, measurement errors are inevitable.

In short, simply for the reason that it’s not possible to get linear results, it’s reasonable that some properties of mine don’t perfectly fit the charts given in pa1.pdf.