Algorithms Programming Assignment#1

B12901148 Kuan Yu, Chou

A. Runtime and Memory Usage

1. Table of runtime and memory usage

Run on EDA Union

Input size	IS		MS		BMS		QS		RQS	
	CPU Time(ms)	Memory(KB)	CPU Time(ms)	Memory(KB)	CPU Time(ms)	Memory(KB)	CPU Time(ms)	Memory(KB)	CPU Time(ms)	Memory(KB)
4000.case2	19.089	21474904412	5.026	21474904412	5.232	21474904412	18.071	21474904412	20.451	21474904412
4000.case3	32.364	21474904412	4.184	21474904412	4.213	21474904412	44.28	21474904412	22.644	21474904412
4000.case1	21.105	21474904412	4.415	21474904412	4.653	21474904412	20.047	21474904412	11.411	21474904412
16000.case2	154.554	21474904412	13.353	21474904412	15.075	21474904412	46.216	21474904412	50.195	21474904412
16000.case3	300.82	21474904412	12.92	21474904412	12.499	21474904412	313.457	21474904412	47.664	21474904412
16000.case1	152.393	21474904412	16.634	21474904412	14.964	21474904412	43.853	21474904412	49.956	21474904412
32000.case2	606.041	21474904412	22.681	21474904412	19.667	21474904412	82.223	21474904412	82.371	21474904412
32000.case3	1165.6	21474904412	18.689	21474904412	17.921	21474904412	1178.19	21474904412	77.391	21474904412
32000.case1	592.534	21474904412	21.79	21474904412	21.959	21474904412	82.396	21474904516	82.869	21474904412
1000000.case2	809288	21474911620	393.935	21474931512	387.568	21474935412	525127	21474935704	2164.45	21474911620
1000000.case3	811042	21474911620	394.927	21474931512	385.535	21474935412	529585	21474935700	2220.23	21474911620
1000000.case1	405532	21474911620	536.367	21474931512	518.98	21474935412	2296.19	21474911620	2234.53	21474911620

Table 1-Runtime and Memory Usage

2. Trending plot

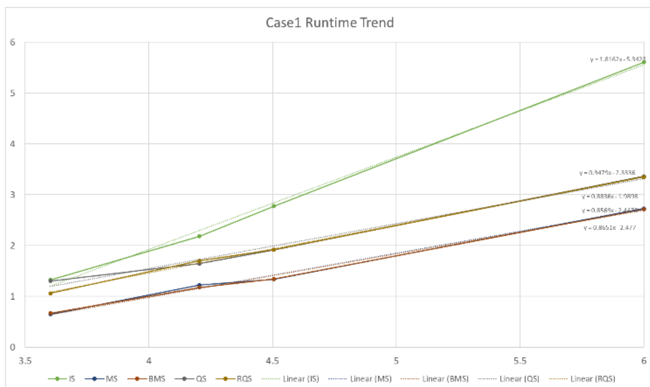

Figure 1-Trend of case1

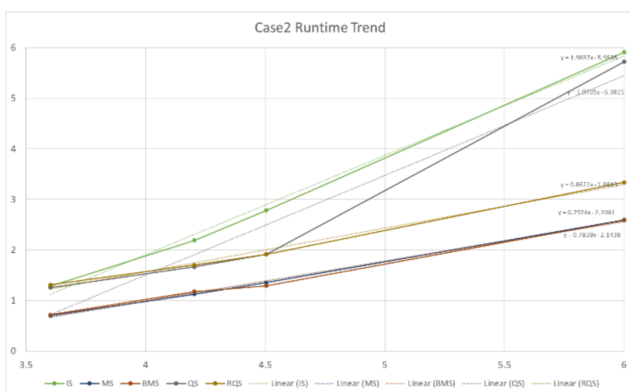

Figure 2-Trend of case2

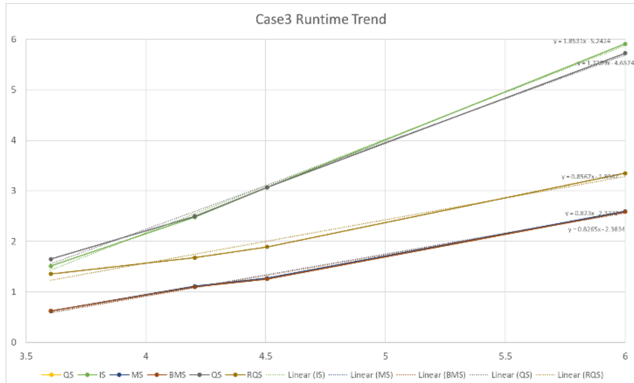

Figure 3-Trend of case3

slope	IS	MS	BMS	QS	RQS
case1	1.82	0.865	0.86	0.88	0.95
case2	1.97	0.8	0.78	1.97	0.86
case3	1.85	0.83	0.82	1.73	0.86

Table 2-Slope of test result

In theory, the time complexity of insertion sort (IS) is $\Theta(n^2)$ and that of merge sort (MS), bottom-up merge sort (BMS), and randomized quick sort (RQS), are $\Theta(n\log(n))$ independent of the cases they meet. As for the quick sort, its time complexity is $\Theta(n\log(n))$ for average case (case1), and $\Theta(n^2)$ for its worst case, which is originally sorted (case2) or reverse-sorted (case 3).

In addition, we may expect the slope for the log-scaled trend line of runtime are 2 for $\Theta(n^2)$ time complexity by the following equation 1. This is approximately in consistent with our result, while it seems like we probably make overestimation. The reason is probably that we take an approximation that b/n and c/n^2 are zero, so the real slope cannot reach to 2.

$$\log(an^2+bn+c) = 2\log\left(n\left(a+\frac{b}{n}+\frac{c}{n^2}\right)^{1/2}\right) \approx 2\log(n)+\log(a)$$

Equation 1-Slope of $\Theta(n^2)$

As for the slope of log-scaled runtime with $\Theta(n\log(n))$ complexity, we may expect the slopes larger than 1 by equation 2. However, we probably make a much loose approximation when ignoring the other terms compared with the condition in $\Theta(n^2)$. The nlog(n) term doesn't dominate for not large enough n, so we make a serious overestimation.

$$\begin{aligned}\log(an\log(n)+bn+c\log(n)+d) &\approx \log(an\log(n))\\ &= \log(a)+\log(n)+\log^{(2)}(n)\end{aligned}$$

Equation 2-Slope of $\Theta(n\log n)$

3. Comparison between merge sort and the bottom-up one

   The runtime of merge sort and the bottom-up merge sort is almost the same because of their commonalities on division as well as conquest. But it is noteworthy that the time usage of merge sort is always slightly more than the bottom-up one. I guess the reason is the merge sort recursively call the function which requires the time to maintain the call stacks and the bottom-up merge sort possesses more efficient cache utilization. And they are not the fault of algorithm.

4. Comparison between quick sort and the randomized one

   There's a huge difference between quick sort and randomized quick sort on originally sorted (case2), and reversed (case3) cases. The key reason why quick sort behaves n^2 on these two cases is how it choose the pivot to divide. In case 2/case 3, it keeps opting for the largest/smallest number as the pivot, resulting the consequence that its divide is always unbalanced. In comparison, the randomized quick sort randomly chooses the pivot. It possibly divides unbalanced, but it probably divides balanced.

5. Data structure and others

   First, we use the vector<int> of C++ STL as the container of our data. It is possibly not the fastest but is the safer container compared to the manually allocated array because the vector automatically helps us maintain the memory. Secondly, we use the index but not the iterator to access the elements in vector, which probably consumes more time due to some process of conversion between index and address may be involved.

   Furthermore, when we recursively call the function, we call by referencing the original vector of data, so the passing time is constant and we can easily edit the contents of the vector. Last, I use memory copying when practicing the merge function of merge sort in the last stage of the merge process, which should be faster than using for loop and index to copy the content of vector in theory.