Algorithms, Fall 2024

Programming Assignment #1 Report

B12901069 段奕鳴

1. Runtime and Memory Usage

All codes are run on, and all following data are obtained using:
EDA union lab machine 40057 (edaU7)
CPU: Intel Xeon Silver 4210R
Memory: ECC DDR4 2400MHz, 64GB
OS: Ubuntu 20.04.6 LTS

Table 1: Runtime and Memory of each sorter and input size

	IS		MS		BMS		QS		RQS	
	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)
4000. case2	0.000079	5904	0.000918	5904	0.000888	5904	0.014116	6032	0.000217	5904
4000. case3	0.016866	5904	0.000970	5904	0.000785	5904	0.012779	5904	0.000647	5904
4000. case1	0.007910	5904	0.001652	5904	0.001571	5904	0.000822	5904	0.000890	5904
16000. case2	0.000101	6056	0.001609	6056	0.001654	6056	0.126710	6932	0.001753	6056
16000. case3	0.207794	6056	0.003220	6056	0.002431	6056	0.105264	6056	0.000994	6056
16000. case1	0.103396	6056	0.004287	6056	0.003231	6056	0.001720	6056	0.001750	6056
32000. case2	0.000084	6188	0.003409	6188	0.002958	6188	0.487702	8004	0.002104	6188
32000. case3	0.810606	6188	0.003457	6188	0.003287	6188	0.401164	6188	0.002528	6188
32000. case1	0.405923	6188	0.005680	6188	0.005736	6188	0.002724	6188	0.003374	6188
1000000. case2	0.001458	12144	0.081008	14004	0.079823	14000	473.209	72468	0.046743	12144
1000000. case3	788.648	12144	0.095135	14004	0.091131	14000	279.936	12144	0.049433	12144
1000000. case1	394.075	12144	0.176208	14004	0.170132	14000	0.088976	12144	0.102213	12144

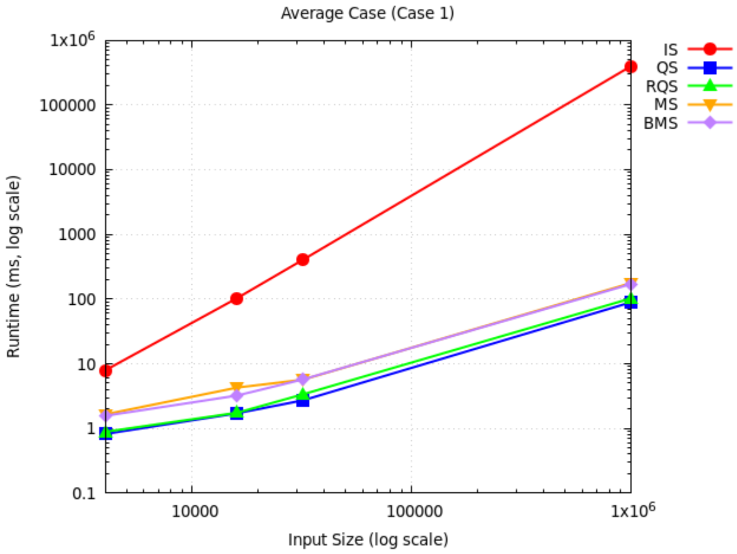

Figure 1: Average Case (random order)

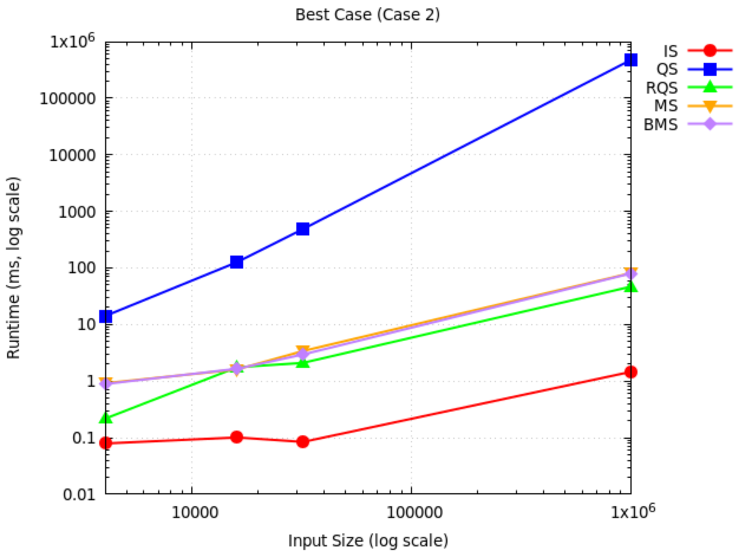

Figure 2: Best Case (sorted order)

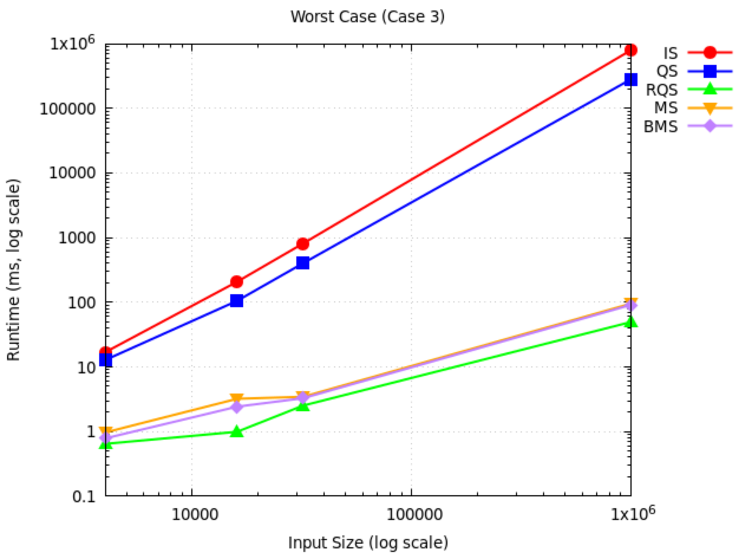

Figure 3: Worst Case (reversely sorted order)

Table 2: Slope of trendline (linear fitting)

	IS	QS	RQS	MS	BMS
Case 1 (Average)	1.97	0.88	0.89	0.86	0.88
Case 2 (Best)	0.56	1.91	0.93	0.84	0.85
Case 3 (Worst)	1.96	1.83	0.82	0.83	0.87

2. Match or Not Match of Slope with Theoretical Complexity

From Table 2, we get the slope of $\log T(n)$ vs $\log n$ for different sorters under different cases of input.

From the textbook, we know that the Insertion Sort has expected time $\Theta(n^2)$ for the average case and the worst case. And we get a slope close to 2, which indicates the slope matches the theory. However, the best case has slope 0.56, while we know the time complexity for the best case should be $\Theta(n)$. This is because when the input size is relatively small, the overall runtime can be affected significantly by the constant, which we usually neglect while analyzing time complexity of an algorithm.

For the Merge Sort and the Bottom-up Merge Sort, we know that no matter the case of input, they guarantee to run in $\Theta(n\, logn)$. That is, $T(n)$ is bounded by $n \log n$. If we take log on both sides:

$$\log T(n) = \log c + \log n + \log \log n$$

However, we know that the iterated logarithm term is comparatively small. Therefore, under these cases, although their slopes are approximately 1, they should not misinterpret as $\Theta(n)$.

For the QS, in the best case (implies that the input data are already sorted, not the "best" for the QS) and the worst case, the slopes are approximately 2, which match the theory. Because certain input will lead to the worst-case behavior of the QS, we switch to use the RQS (more discussion on this in problem 4). The average case of the QS, and the average, the best, the worst case of the RQS, have slope approximately equal to 1. But as the same argument of the MS/BMS part, they should have $\Theta(n\, logn)$ time complexity, instead of $\Theta(n)$.

3. Comparison between MS (Merge Sort) and BMS (Bottom-up Merge Sort)

No matter in the average case, the best case, or the worst case, MS and BMS have same order of time complexity. By class material, we know they both have $O(n\, lgn)$ time complexity. Also, both MS and BMS require extra $O(n)$ space for temporary arrays used for merging smaller arrays.

However, MS recursively breaks the array into smaller subarrays, and thus MS run recursively, which require extra stack size. As a result, when our environment is stack-limited, or when we are dealing with large input datasets, we should consider BMS as a better choice, which eliminates the possible overhead due to deep recursion. Besides, without recursive function call, the implementation of Bottom-up Merge Sort might be considered slightly easier.

4. Comparison between QS and RQS

For the average case and the best case, QS and RQS need almost the same running time. While for the worst case, which the input data are in reverse order, RQS still runs relatively fast. However, QS, runs far slower, almost close to the runtime of Insertion Sort.

This implies that when the worst case occurs, QS might take up to $\Theta(n^2)$. And this is why we introduce RQS. Although randomly choosing the pivot does not make the worst case time complexity any better than the original $O(n^2)$. Such a worst case happens when the data are always split into the extreme case, such as n-1 elements are less than the pivot, and no element is larger than the pivot.

However, the advantage of RQS is that we will not find a particularly input can elicit the worst case behavior. By contrast, when we apply the normal QS, which always choose the last element as the pivot, we do know how to elicit the worst-case behavior. That is, if the input data are already sorted (or already sorted reversely), the worst-case behavior will surely happen.

5. Other Data Structure used and Findings

In the part of Insertion Sort, although we know that it has the $O(n^2)$ average time complexity, however, the implementation of code will affect the real running time, while they are still on the same time order.

For example, a sample of naïve implementation of IS will be

```
For i from 1 upto data.size()-1
    int j = i
    while( j>0 and arr[j]<arr[j-1] )
        exchange arr[j] with arr[j-1]
        j = j – 1
```

However, in each “EXCHANGE” in line 4 require 3 steps, as follows:

```
EXCHANGE(data_1, data_2)
    tmp = data_1
    data_1 = data_2
    data2 = tmp
```

If we implement IS in another way:

```
For i from 1 upto data.size()-1
    int j = i
    int tmp = arr[i]
    while( j>0 and arr[j]<arr[j-1] )
        arr[j] = arr[j-1]
        j = j - 1
    arr[j] = tmp
```

In this implementation, each time we only need 1 step to move the element backward (e.g. arr[9]>arr[10], so we move the data of arr[9] to arr[10]). And after each while loop, index j will be at the place where arr[i] belongs to, so we move arr[i] to arr[j]. This way, we avoid the 3-steps operation for each exchange, and therefore decrease the constant coefficient of total time costs. However, obviously, we do not improve the overall time complexity performance of IS, which is still $O(n^2)$.