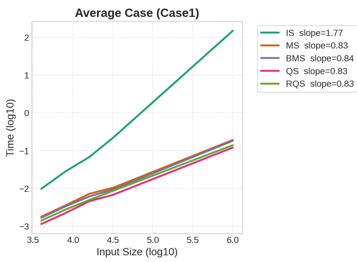
B11901136 電機四 梁程捷 PA1 Report

The machine used in EDAUnion: (port 40062)

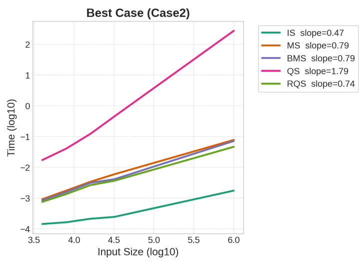
(1) Runtime and memory usage of sorting algorithms

The experimental results generally align with theoretical expectations, though several noteworthy deviations merit discussion. Insertion Sort (IS) demonstrates a best-case slope of approximately $O(n^{0.5})$, which falls notably below the theoretical $O(n)$ complexity. Similarly, both Bottom-up Merge Sort (BMS) and Randomized Quick Sort (RQS) exhibit empirical growth rates between $O(n^{0.8})$and $O(n)$, rather than the anticipated $O(nlogn)$ behavior.

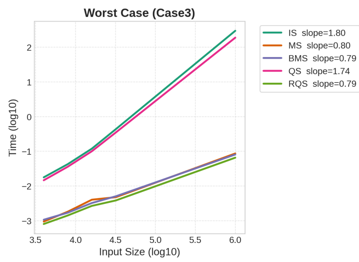
These discrepancies can be attributed to the limited scale of the input sizes tested. In asymptotic analysis, the dominant terms only become clearly observable at sufficiently large values of n. For the input ranges employed in this study, constant factors and lower-order terms continue to exert significant influence on the measured runtimes, thereby obscuring the true asymptotic behavior.



(a) Average Case



(b) Best Case



(c) Worst Case

Input size	IS CPU time (s)	IS Mem (KB)	MS CPU time (s)	MS Mem (KB)	BMS CPU time (s)	BMS Mem (KB)	QS CPU time (s)	QS Mem (KB)	RQS CPU time (s)	RQS Mem (KB)
4000.case1	0.009706	6076	0.00177	6076	0.00167	6076	0.001136	6076	0.001396	6076
4000.case2	0.000144	6076	0.000919	6076	0.000847	6076	0.017214	6076	0.000753	6076
4000.case3	0.017814	6076	0.000947	6076	0.00107	6076	0.014664	6080	0.0008060	6076
16000.case1	0.067478	6228	0.00716	6228	0.0061	6228	0.004531	6228	0.004886	6228
16000.case2	0.000213	6228	0.003388	6228	0.00312	6228	0.121938	6228	0.00263	6228
16000.case3	0.11945	6228	0.004037	6228	0.00324	6228	0.100619	6600	0.00268	6228
32000.case1	0.2221	6360	0.010435	6360	0.00942	6360	0.006802	6360	0.00868	6360
32000.case2	0.000245	6360	0.005955	6360	0.00411	6360	0.451272	6360	0.003657	6360
32000.case3	0.437036	6360	0.004778	6360	0.00507	6360	0.350793	7160	0.003849	6360
1000000.case1	149.071	12316	0.190892	14176	0.1815	14172	0.120333	12316	0.13909	12316
1000000.case2	0.001752	12316	0.077648	14176	0.07147	14172	273.099	12316	0.046387	12316
1000000.case3	298.205	12316	0.08691	14176	0.0812	14172	188.78	28848	0.065661	12316

(2) Merge Sort and Bottom-Up Merge Sort

From the experimental results, both Merge Sort (MS) and Bottom-up Merge Sort (BMS) demonstrate similar time complexity trends across all test cases, with slopes approximating O(n log n) behavior as expected theoretically. However, contrary to the theoretical expectation that BMS should exhibit superior performance due to its iterative nature eliminating recursion overhead, the empirical data shows nearly identical runtime performance between the two variants. This performance parity can be attributed to the implementation approach of mine where both algorithms utilize the same fundamental `Merge()` operation

(3) Quick Sort and Randomized Quick Sort
According to textbook analysis, standard Quick Sort exhibits distinct time complexity characteristics across different input scenarios. While it achieves O(n log n) performance in the average case, it degrades to O(n^2) in both best and worst cases. In contrast, Randomized Quick Sort maintains O(n log n) expected time complexity across all input patterns.
The fundamental distinction arises from their partitioning strategies. Randomized Quick Sort employs probabilistic pivot selection that, with high probability, yields balanced partitions regardless of input distribution. This randomization ensures that even for pre-sorted or reverse-sorted arrays—the pathological cases for standard Quick Sort—the algorithm maintains approximately equal-sized subproblems through the recursion tree.
Conversely, the deterministic Hoare partition scheme used in standard Quick Sort proves vulnerable to specific input patterns. When processing already sorted or uniformly structured data, the fixed pivot selection can consistently produce maximally unbalanced partitions (splitting into 1 and n-1 elements). This imbalance cascades through the recursion, resulting in the quadratic time complexity observed in worst-case scenarios.

(4) Data Structure and other findings
Data Structure
Dynamic arrays (vector) are used throughout the implementation of algorithms, which is convenient as we do not need to initialize an array with a specific size like in C language.
Findings
During the implementation of Randomized Quick Sort, a critical optimization was identified in the pivot selection process. The standard textbook approach typically generates random pivots within the range [low, high] (inclusive). However, empirical testing revealed that constraining the pivot selection to [low, high - 1] (excluding the last element) yields significant performance improvements.
This optimization addresses a subtle edge case in the partitioning logic. When the recursion reaches subarrays of size 2, the original approach could select the last element as the pivot, potentially creating an unbalanced partition that degenerates into the same recursive case repeatedly. By excluding the final element from pivot consideration, we ensure that partitions of size 2 are always split into non-empty subarrays, preventing redundant recursive calls and maintaining better balance in the partition tree.