Empirical Analysis of Sorting Algorithms: Quick Sort, Merge Sort, and Heap Sort

Andrea A. Venti Fuentes

University of Miami

CSC 401 – Computer Science Practicum I

Dr. Dilip Sarkar

October 21, 2024

Empirical Analysis of Sorting Algorithms: Quick Sort, Merge Sort, and Heap Sort

Introduction

This report compares the empirical performance of Quick Sort (random and deterministic pivots), Merge Sort, and Heap Sort. We evaluated these algorithms on 96 datasets (12 for each input size, ranging from 10,000 to 800,000) using randomly generated integers between 0 and 5000. For each dataset, we measured the number of comparisons and the running time of each algorithm. In deterministic Quick Sort, the last element was chosen as the pivot. Additionally, we assessed performance on sorted (non-decreasing) and reverse-sorted (non-increasing) inputs.

Comparisons and Running Time

On random non-sorted datasets, Merge Sort consistently required the fewest comparisons and exhibited competitive running times, as expected with its $O(n \log n)$ complexity (***Figure 1 and Figure 2***). Heap Sort performed more comparisons and was generally slower, particularly on smaller datasets. Quick Sort, with both random and deterministic pivots, performed well in average cases, but lagged behind Merge Sort.

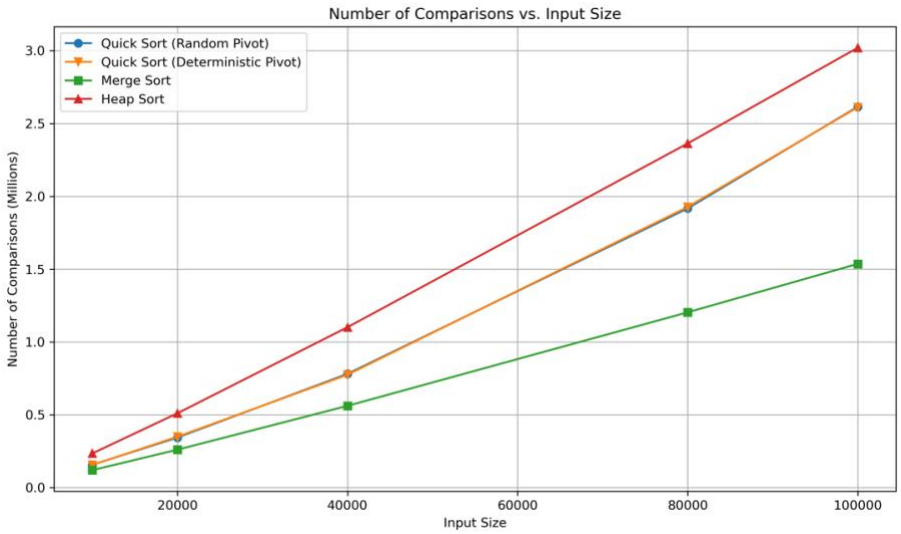

Figure 1: *Comparisons for Input Sizes 10,000 to 100,000*

For larger datasets (***Figure 2***), Merge Sort continued to outperform both versions of Quick Sort and Heap Sort. Notably, both Quick Sort algorithms (random and deterministic pivot) performed nearly identically across all input sizes, with little to no observable performance difference. Merge Sort, however, consumed the most memory due to its $O(n)$ auxiliary space requirement during merging (***Figure 3***).

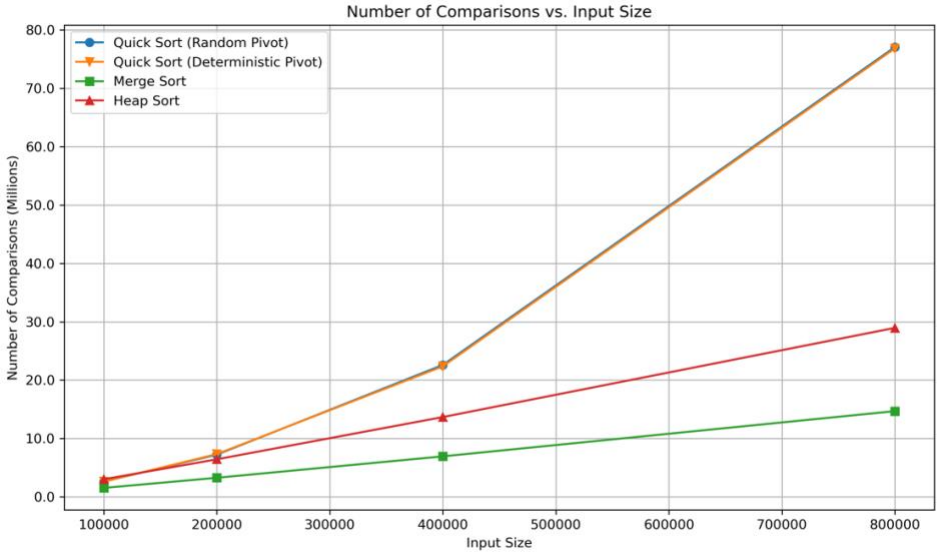

Figure 2: *Comparisons for Input Sizes 100,000 to 800,000*

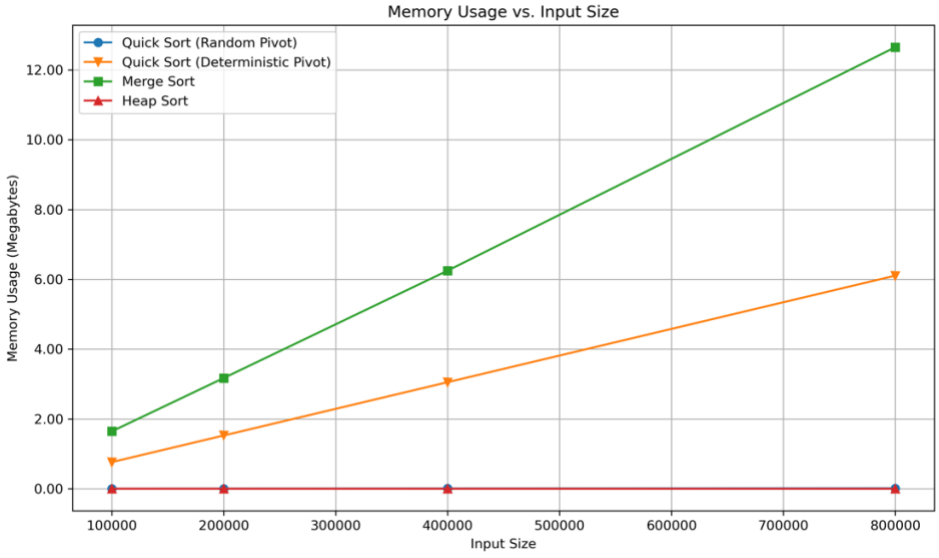

Figure 3: *Memory Usage for Input Sizes 100,000 to 800,000*

Sorted and Reverse-Sorted Data

On sorted and reverse-sorted inputs, Quick Sort with a deterministic pivot performed over 4 billion comparisons on 800,000 elements (***Table* 1**), showcasing significantly higher comparisons and running times, demonstrating its $O(n^2)$ behavior. The random pivot version maintained $O(n \log n)$ performance on reverse-sorted inputs, showing the benefit of randomization. Merge Sort and Heap Sort maintained stable $O(n \log n)$ performance regardless of input order.

Table 1: *Comparisons of Sorting Algorithms on Non-decreasing and Non-increasing Sorted Datasets*

Input Size	Quick Sort (Random) (Non-decreasing)	Merge Sort (Non-decreasing)	Heap Sort (Non-decreasing)	Quick Sort (Deterministic) (Non-decreasing)	Quick Sort (Random) (Non-increasing)	Merge Sort (Non-increasing)	Heap Sort (Non-increasing)	Quick Sort (Deterministic) (Non-increasing)
10000	156.48K	69.29K	244.39K	21.72M	157.62K	69.01K	226.65K	30.29M
20000	360.98K	153.85K	529.05K	49.22M	350.11K	148.02K	493.32K	79.02M
40000	780.64K	337.58K	1.14M	100.12M	790.90K	316.03K	1.07M	177.85M
80000	1.86M	729.75K	2.39M	200.55M	1.93M	672.06K	2.29M	376.94M
100000	2.70M	932.12K	3.02M	251.23M	2.51M	853.90K	2.92M	477.69M
200000	7.23M	1.98M	6.18M	504.23M	7.19M	1.81M	6.24M	980.16M
400000	22.37M	4.19M	12.81M	1.02B	22.83M	3.82M	13.28M	1.99B
800000	76.75M	8.81M	26.73M	2.07B	77.39M	8.03M	28.17M	4.04B

* K: thousands
* M: millions
* B: billions

Conclusion

The empirical results match the theoretical time complexities of these algorithms. Merge Sort consistently performed well, unaffected by input order. Quick Sort's efficiency depended on the pivot selection and input order, with the deterministic version suffering on sorted data. Heap Sort, while stable, generally lagged behind Merge Sort in efficiency. Choosing the right algorithm depends on input characteristics and efficiency requirements.

References

Venti Fuentes, A. (2024). Sorting Algorithm Performance Analysis. GitHub repository. Available at: https://gitfront.io/r/av1155/AbQeNWzfig3F/sorting-algorithm-performance-analysis/