Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

Lab II: Analysis of Sorting Algorithm

COMP 314

Submitted by:

Manish Shivabhakti
Roll no: 63
CE, III year/ II Semester

Date: 29th May 2024

1. Introduction

We analyze the performance of two sorting algorithms: Merge Sort and Quick Sort. The performance analysis is conducted based on the execution time of each algorithm for inputs of different sizes. We generate random inputs and consider best and worst-case scenarios to understand how the algorithms behave under various conditions.

2. Experimental Setup

We implemented both the Merge Sort and Quick Sort algorithms in Python. We generated random input sequences of increasing sizes ranging from 1000 to 2000 elements with interval of 100. For each input size, we recorded the execution time of both sorting algorithms. Additionally, we evaluated the performance of each algorithm for best-case scenarios (sorted input) and worst-case scenarios (reverse sorted input).

3. Results and Analysis

Merge Sort

Average Case Performance: In the average case, Merge Sort exhibits a linearithmic(linear and logarithmic) time complexity, O(n log n). The execution time increases in a linearithmic fashion with the increase in input elements.

Best Case Performance: Merge Sort consistently performs with a time complexity of O(n log n), even when the input is already sorted. As observed, the execution time increases in a linearithmic manner with the input size. While it may appear almost linear on a large-scale graph, it retains its linearithmic nature (as shown in Figure 1).

Worst Case Performance: Merge Sort maintains its time complexity of O(n log n) regardless of the order of input. Whether the input is randomly ordered or in reverse sorted order, the execution time grows in a linearithmic manner, ensuring reliable performance.

Figure 1. Merge Sort Analysis

Quick Sort

Average Case Performance: Quick Sort demonstrates a linearithmic time complexity, O(n log n), in the average case. The execution time increases in a linearithmic fashion with the input size, making it efficient for large datasets.

Best Case Performance: In the best-case scenario, Quick Sort also exhibits a time complexity of O(n log n). This occurs when the pivot divides the array into two nearly equal halves at every step, ensuring optimal performance.

	n	time
0	1000	0.000930
1	1100	0.001063
2	1200	0.001211
3	1300	0.001367
4	1400	0.001502
5	1500	0.001653
6	1600	0.001748
7	1700	0.001810
8	1800	0.001847
9	1900	0.001958

Figure 2. Best Case Dataset (linearithmic)

Worst Case Performance: In the worst-case scenario, Quick Sort's performance degrades to quadratic time complexity, O(n^2). This typically happens when the pivot selection consistently results in the most unbalanced partitions, such as when the input is already sorted or reverse sorted and the smallest or largest element is always chosen as the pivot.

	n	time
0	1000	0.001907
1	1100	0.001822
2	1200	0.002016
3	1300	0.002035
4	1400	0.002122
5	1500	0.002177
6	1600	0.002511
7	1700	0.002610
8	1800	0.002743
9	1900	0.003032

Figure 3. Average Case Dataset

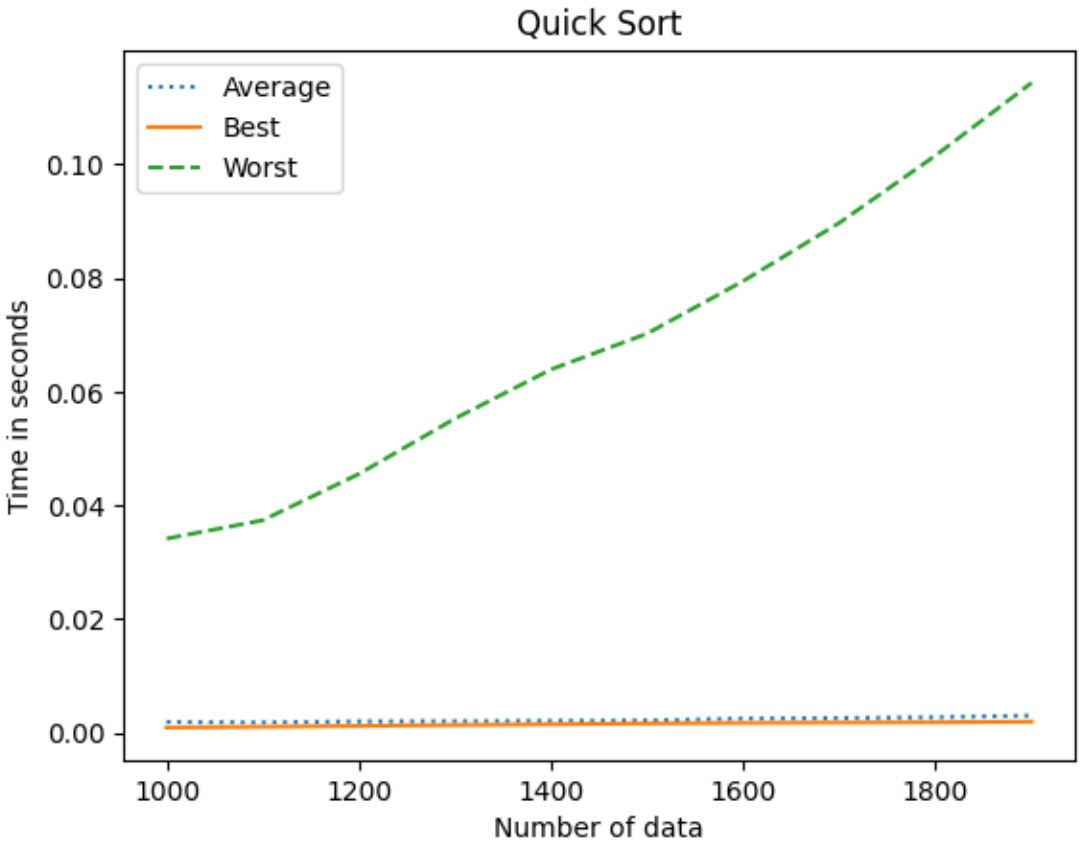

Figure 4. Quick Sort Analysis

4. Conclusion

a. Both Merge Sort and Quick Sort exhibit linearithmic time complexity, O(n log n), in the average case. However, in the worst-case scenario, Merge Sort maintains its O(n log n) complexity, whereas Quick Sort can degrade to quadratic time complexity, O(n^2), depending on the pivot selection.

b. Merge Sort consistently performs well with O(n log n) time complexity in both best and worst cases, making it more predictable. On the other hand, Quick Sort, while also having O(n log n) in the best case, can outperform Merge Sort in practice due to lower constant factors and better cache performance when the pivot divides the array into two nearly equal halves.

5. Code

GitHub link: https://github.com/TheManysh/algorithm_and_complexity_lab/tree/main/lab-2