Created by Dmytro Krainyk

[All code is in my repo on github]

Assignment I: Getting things in order

Task 3 Report: “Dangerous Quickminds”

Introduction

This task empirically investigates the relation between the number of pivots in QuickSort and the sorting time. Specifically, I compared the classic **Single-Pivot** strategy against the modern **Dual-Pivot** strategy.

Implementation Details

1. Single-Pivot QuickSort

I implemented the standard partition scheme which splits the array into two parts: elements smaller than the pivot and elements larger.

- Complexity: $O(n \log n)$.

2. Dual-Pivot QuickSort

I implemented the algorithm used in many modern standard libraries (e.g., Java’s `Arrays.sort`). It uses two pivots (P_1, P_2) to divide the array into three parts:

1. Elements $< P_1$
2. Elements between P_1 and P_2
3. Elements $> P_2$

Performance Analysis

I benchmarked both algorithms on random arrays up to 1,000,000 elements.

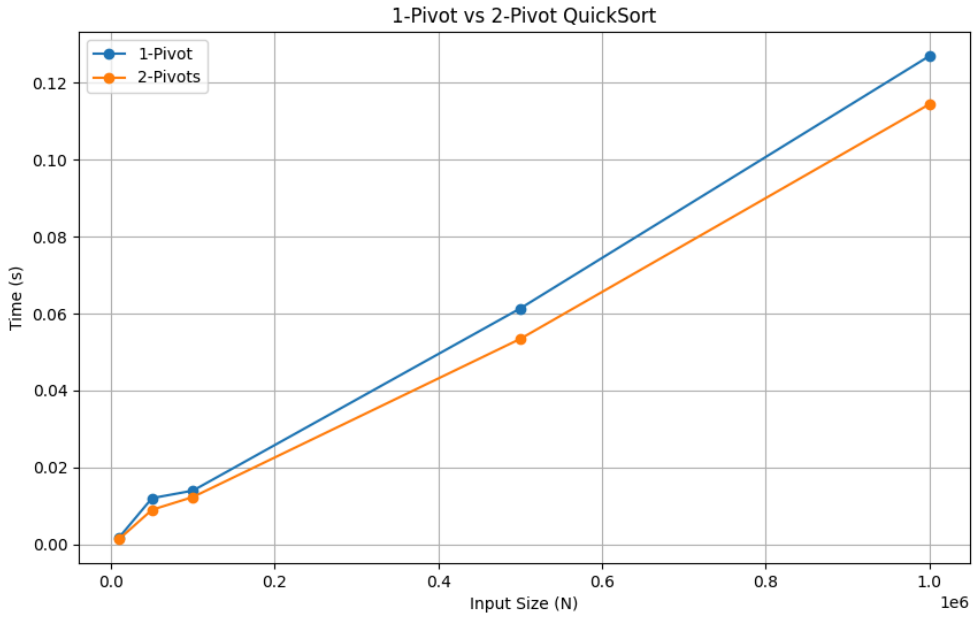

Figure 1: Fig 1. Performance: 1-Pivot (Blue) vs 2-Pivot (Orange).

Observations:

1. **Linear-Logarithmic Growth:** Both algorithms strictly follow the $O(n \log n)$ complexity curve.
2. **Dual-Pivot Efficiency:** On larger datasets (visible on the right of the graph), the Dual-Pivot implementation often performs slightly faster. This is because it reduces the height of the recursion tree (log base 3) and can be more cache-efficient on modern CPUs, despite performing more comparisons per step.

Conclusion

The experiment confirms that the Dual-Pivot strategy is a valid optimization for QuickSort on primitive types, offering a tangible speedup on large datasets.