# **Assignment 5: Quicksort Algorithm, Implementation, Analysis, and Randomization**

Jacob Jeffers

MSCS 532

Dr. Vanessa Cooper

Fall 2024

11/17/2024

Assignment 5: Quicksort Algorithm, Implementation, Analysis, and Randomization

Part 1: Implementation

*Instructions: Implement the Quicksort algorithm using Python. Your implementation should be transparent and efficient, and you should correctly follow the steps for selecting a pivot, partitioning the array, and recursively sorting the subarrays.*

The Quicksort algorithm was implemented into VS Code.

A screenshot of a computer program

Description automatically generated

Part 2: Performance Analysis

*Instructions: Provide a detailed analysis of the time complexity of Quicksort in the best, average, and worst cases. Explain why the average-case time complexity is \(O(n \log n)\) and the worst-case time complexity is \(O(n^2)\). Discuss the space complexity and any additional overheads associated with the algorithm.*

Quicksort's efficiency depends on selecting the pivot and the resulting partitioning (Cormen et al., 2022). The best case occurs when each pivot divides the array into two nearly equal halves and performs at O(n logn). For the average case, this occurs when the pivot splits the array into reasonably balanced partitions (a quarter or three-quarters of the array's size) (Cormen et al., 2022). The worst-case scenario is O(n^2). It occurs when the partition is exceptionally unbalanced (one containing no or few elements, with the other containing all or almost all elements) (Cormen et al., 2022).

Part 3: Randomized Quicksort

*Instructions: Implement a randomized version of Quicksort where the pivot is chosen randomly from the sorted subarray. Analyze how randomization affects the performance of Quicksort and reduces the likelihood of encountering the worst-case scenario.*

The randomized Quicksort algorithm was implemented in VS Code.

A screenshot of a computer program

Description automatically generated

Choosing a random pivot point reduces the likelihood of achieving the worst-case scenario. It does this by reducing the dependence on input patterns. For example, since the pivot is chosen randomly, it does not suffer from poor performance for sorted or reverse-sorted arrays.

Part 4: Empirical Analysis

*Instructions: Empirically compare the running time of the deterministic and randomized versions of Quicksort with different input sizes and distributions (e.g., random, sorted, reverse-sorted). Discuss the observed results and relate them to your theoretical analysis.*

To begin, I started with the algorithms created in parts one and three and merged them into one file on VS Code. I then defined varying input sizes, distributions, and specific experiments to capture the empirical evidence for Quicksort's deterministic and randomized versions. A snippet of the results can be found below, as well as a table representation.

A screen shot of a computer

Description automatically generated

| **Size** | **Distribution** | **Deterministic Time (s)** | **Randomized time (s)** |
| --- | --- | --- | --- |
| 1,000 | Random | 0.00095 | 0.00115 |
| 1,000 | Sorted | 0.00056 | 0.00096 |
| 1,000 | Reverse-Sorted | 0.00054 | 0.00095 |
| 10,000 | Random | 0.01138 | 0.01274 |
| 10,000 | Sorted | 0.00739 | 0.01230 |
| 10,000 | Reverse-Sorted | 0.00714 | 0.01281 |
| 100,000 | Random | 0.12843 | 0.12314 |
| 100,000 | Sorted | 0.09036 | 0.15151 |
| 100,000 | Reverse-Sorted | 0.09404 | 0.14625 |

For random distributions, both deterministic and randomized versions performed similarly across all input sizes, with slightly lower times for the deterministic version in smaller sizes. Surprisingly, the deterministic version consistently ran faster than the randomized version on sorted input, especially at larger sizes (100,000 elements). The randomized version may have taken longer due to the extra cost of choosing a random pivot point. Like the sorted input, the deterministic Quicksort outperformed the randomized Quicksort at even larger sizes. This was surprising as well. I hypothesize that the deterministic Quicksort outperformed the randomized Quicksort because it was more optimized from a coding standpoint.

**GitHub Repository for Assignment 5:** <https://github.com/jakejeffers/MSCS-532-Assignment-5>

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms, fourth edition*. MIT Press.