# Quicksort Algorithm Analysis

Susan Subedi

Algorithms and Data Structure

Professor. Satish Penmatsa

University of the Cumberlands

February 2, 2025

# Quicksort Algorithm Analysis

In this analysis, we are going to assess the performance of two implementations of quicksort algorithms, which are deterministic and randomized, on different input distributions (Iliopoulos, 2015). The goal of the analysis is to determine which sort algorithm is better under each of the conditions.

**Implementation**

**Deterministic Quicksort**

The deterministic implementation of Quicksort involves selecting the middle element of the array to be sorted as the pivot point and then partitions the elements into those smaller and greater than the pivot point. This helps the algorithm to improve its efficiency in most cases.

Randomized Quicksort

Under this implementation, a randomly selected pivot point is used to partition the arrays. The advantage of using this approach is that it avoids the worst-case scenario because it ensures there is an even distribution of partitioning (Maucher, 2016).

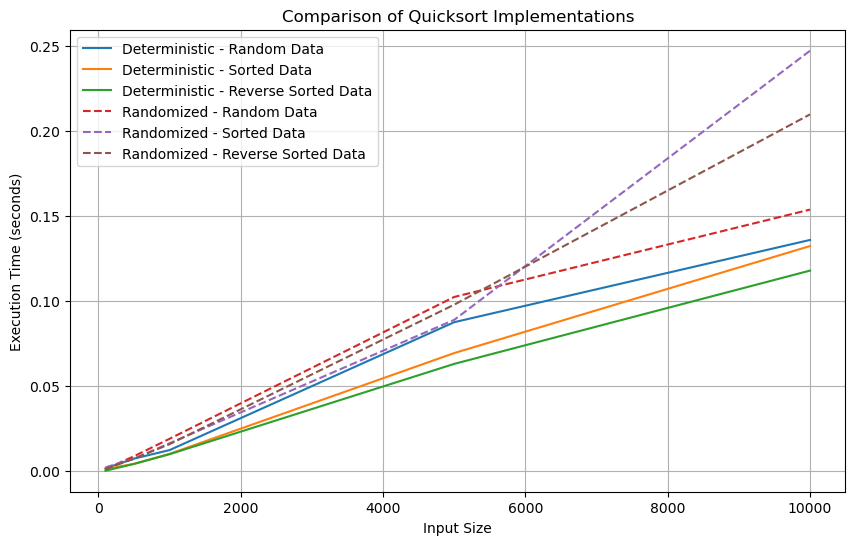
Time complexity analysis

| Case | Deterministic Quicksort | Randomized Quicksort |
| --- | --- | --- |
| Best Case | O(n log n) | O(n log n) |
| Average Case | O(n log n) | O(n log n) |
| Worst Case | O(n2)O(n^2)O(n2) | O(n log n) (highly unlikely) |

The deterministic version shows complexity represented by O(n^2) for sorted or reverse-sorted data because of the issue of poor strategy for selecting pivot. The randomized version avoids this issue by distributing the pivot choices evenly.

Empirical Analysis

Performance Comparison



The results from the graph above indicate that the randomized Quicksort performance is poor compared to that of deterministic Quicksort when dealing with small input. However, for larger input sizes, deterministic Quicksort performance improves, especially on sorted and reverse-sorted data. The randomized Quicksort takes longer to complete its operation. The results indicate that the reverse-sorted data has a negative impact on the execution time when using deterministic Quicksort, increasing execution time.

Observations

The observation from the analysis is that deterministic Quicksort performs better when the data is randomly distributed, while randomized Quicksort performs better in terms of reducing the risks of facing the worst-case complexity. When dealing with a large data set, it is better to use randomized Quicksort, considering that it has faster execution time.

Conclusion

In conclusion, we can mitigate the worst-case scenario by using the randomized approach. However, when the data is sorted in practical applications, it is better to use randomized Quicksort because it has a favorable execution time. Deterministic Quicksort is preferred in terms of efficiency and when working with a completely random dataset.

# References

Iliopoulos, V. (2015). The Quicksort algorithm and related topics. *arXiv preprint arXiv:1503.02504*.

Maucher, M. (2016). On the influence of non-perfect randomness on probabilistic algorithms.