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

Syed Noor UI Hassan

University of the Cumberlands

Algorithms and Data Structures (MSCS-532-B01)

November 17, 2024

**Quicksort Algorithm:- Implementation, Analysis and Randomization**

Below is the screenshot of deterministic and randomized quicksort implementation

A computer screen shot of a program code

Description automatically generated

# Performance Analysis

The choice of pivot is crucial for its performance.   
• Best case  
When the pivot always splits the array in half, we get the ideal O(nlogn) complexity. This happens if the pivot is always the median.   
• Average case  
Even when the pivot isn't perfectly splitting the array, the average time complexity still holds at O(nlogn). This is because the division is usually well-balanced enough over multiple recursive calls.   
• Worst case  
 The dreaded O(n2) occurs when the pivot consistently divides the array poorly. Think sorted or reverse-sorted arrays where the pivot is always the smallest or largest element.   
**Space Complexity**   
The space complexity of Quicksort is typically O(logn) due to the recursion stack, assuming the pivot divides the array reasonably evenly. However, in the worst case, where the array is not divided well, the space complexity can degrade to O(n).   
  
**Empirical Analysis**   
Both algorithm is compared using the timeit function in Python to determine the performances. The input sizes vary from 1000, 5000, 10000, and 20000 for both. Below is the screenshot from the test.   
  
A screenshot of a computer

Description automatically generated  
  
  
**Observations based on the result**   
**Random Arrays**

The performance of deterministic and randomized Quicksort is fairly close across all input sizes.

This is to be anticipated, as the data is distributed equitably by random input, and the selection of pivot (deterministic or randomized) frequently gives similar results.

**Sorted and Reverse-Sorted Arrays.**

The performance of Deterministic Quicksort on sorted and reverse-sorted data is unexpectedly high.

O(n2) performance is typically the result of a weak pivot choice in these cases.

Nevertheless, Randomized Quicksort consistently requires a prolonged time to complete on sorted and reverse-sorted arrays. This is likely the result of the less efficient partitioning on virtually ordered data and the overhead of randomly selecting a pivot. In situations where deterministic Quicksort is already effectively managing the input, the randomness introduces unnecessary complexity.

**Conclusion**   
Deterministic Quicksort exhibits efficient management even when the input is ordered, demonstrating surprising performance on sorted and reverse-sorted data.  
  
Randomized Quicksort is marginally slower in general, especially on sorted arrays where the random pivot selection may not be required. It would be more advantageous when confronted with adversarial inputs or specific scenarios that are intended to elicit the worst-case behavior of deterministic Quicksort.