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

## Overview:

This assignment focuses on the Quicksort algorithm, its implementation, performance analysis, and a randomized version. You will explore the theoretical underpinnings of Quicksort, implement both deterministic and randomized versions, and analyze their performance under different conditions.

The skills and knowledge gained from this assignment are directly applicable to a range of modern software technologies. Whether you're optimizing algorithms for large-scale data processing frameworks like Apache Hadoop or Spark, enhancing the performance of search algorithms in web services, or developing resource-efficient applications for mobile and embedded systems, the principles of Quicksort and its analysis will be invaluable.

By the end of this assignment, you will have a deep understanding of the Quicksort algorithm, its implementation nuances, and its practical applications in today's technology-driven world. This knowledge will equip you to make informed decisions about algorithm selection and optimization in your future projects, contributing to the development of efficient and scalable software solutions.

## Quicksort Implementation and Analysis

### **1. Implementation**

- Implement the Quicksort algorithm using Python. Your implementation should be clear, efficient, and correctly follow the steps for selecting a pivot, partitioning the array, and recursively sorting the subarrays.

### **2. Performance Analysis**

- 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.

### **3. Randomized Quicksort**

- Implement a randomized version of Quicksort where the pivot is chosen randomly from the subarray being sorted.

- Analyze how randomization affects the performance of Quicksort and reduces the likelihood of encountering the worst-case scenario.

### **4. Empirical Analysis**

- Empirically compare the running time of the deterministic and randomized versions of Quicksort on different input sizes and distributions (e.g., random, sorted, reverse-sorted).

- Discuss the observed results and relate them to your theoretical analysis.

## Deliverables:

- Well-documented source code for both the deterministic and randomized versions of Quicksort.

- A detailed report discussing your implementation, performance analysis, and the impact of randomization on Quicksort.

- Clear explanations of the time complexity analysis for both versions of Quicksort.

## Submission Instructions:

### **1. GitHub Repository:**

- Create a new GitHub repository for this assignment.

- Upload the following materials to your repository:

- Your Python implementation of both the deterministic and randomized Quicksort algorithms.

- A report detailing your design choices, implementation details, and analysis.

- A README file with instructions on how to run your code and a summary of your findings.

### **2. Submit the GitHub Repository Link:**

- Submit the link to your GitHub repository as your final submission.