# Assignment 3: Understanding Algorithm Efficiency and Scalability

## Overview:

This assignment is designed to deepen your understanding of how algorithms perform under different conditions. You will analyze and compare the efficiency and scalability of two key algorithms: Randomized Quicksort and Hashing with Chaining. Through this assignment, you will develop the skills necessary to evaluate algorithm performance, implement efficient solutions, and make informed decisions about algorithm selection based on theoretical and empirical analysis.

## Part 1: Randomized Quicksort Analysis

### **1. Implementation**

- Implement the Randomized Quicksort algorithm where the pivot element is chosen uniformly at random from the subarray being partitioned.

- Ensure that your implementation is efficient and handles various edge cases, such as arrays with repeated elements, empty arrays, and already sorted arrays.

### **2. Analysis**

- Provide a rigorous analysis of the average-case time complexity of Randomized Quicksort.

- Clearly explain why the average-case time complexity is \(O(n \log n)\).

- Use concepts such as indicator random variables or recurrence relations in your analysis to demonstrate your understanding.

### **3. Comparison**

- Empirically compare the running time of Randomized Quicksort with Deterministic Quicksort (using the first element as the pivot) on different input sizes and distributions:

- Randomly generated arrays

- Already sorted arrays

- Reverse-sorted arrays

- Arrays with repeated elements

- Discuss the observed differences in running time and relate them to your theoretical analysis. Explain any discrepancies between the empirical results and the expected theoretical performance.

## Part 2: Hashing with Chaining

### **1. Implementation**

- Implement a hash table using chaining for collision resolution.

- Choose a suitable hash function, such as one from a universal hash function family, to minimize collisions.

- Your implementation should efficiently support the following operations:

- Insert: Add a key-value pair to the hash table.

- Search: Retrieve a value associated with a given key.

- Delete: Remove a key-value pair from the hash table.

### **2. Analysis**

- Analyze the expected search, insert, and delete times under the assumption of simple uniform hashing.

- Explain how the load factor (the ratio of the number of elements to the number of slots) affects the performance of these operations.

- Discuss strategies for maintaining a low load factor and minimizing collisions, including dynamic resizing of the hash table.

## Submission Instructions:

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

- Create a new GitHub repository for this assignment.

- Upload the following materials to your repository:

- Your Python implementation of the Randomized Quicksort and Hash Table with Chaining.

- A report detailing your theoretical analysis, empirical comparisons, and discussion of results.

- 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.