Assignment 3: Understanding Algorithm Efficiency and Scalability

# Part 1: Randomized Quicksort Analysis

## 1. Implementation

Randomized Quicksort was implemented by choosing a pivot randomly from the subarray being sorted. The implementation handles edge cases such as empty arrays, arrays with duplicate values, already sorted arrays, and reverse-sorted arrays.

## 2. Analysis

Randomized Quicksort has an average-case time complexity of O(n log n). This is because on average, the random pivot tends to divide the array into reasonably balanced parts. The recurrence relation for expected time is:  
T(n) = T(k) + T(n-k-1) + Θ(n),  
where k is the number of elements less than the pivot. Using indicator random variables and probabilistic analysis, we derive the expected number of comparisons to be approximately 1.39n log n.

## 3. Comparison

We compared Randomized and Deterministic Quicksort (first-element pivot) using different input types:  
- Randomly generated arrays  
- Sorted arrays  
- Reverse-sorted arrays  
- Arrays with duplicates  
  
Empirical results showed that Randomized Quicksort performs more consistently across input types, whereas Deterministic Quicksort degrades to O(n^2) in sorted or reverse-sorted inputs. Minor discrepancies between theory and practice may occur due to system factors and Python overheads.

# Part 2: Hashing with Chaining

## 1. Implementation

The hash table was implemented using chaining with linked lists (Python lists) for collision resolution. A simple modulo-based hash function was used. The implementation supports:  
- insert(key, value)  
- search(key)  
- delete(key)

## 2. Analysis

Under the assumption of Simple Uniform Hashing, the expected time for insert, search, and delete is O(1 + α), where α is the load factor (n/m). As α increases, the average length of chains grows, slowing operations. To maintain efficiency:  
- Choose good hash functions (e.g., universal hashing)  
- Keep the load factor low (α < 0.7)  
- Dynamically resize the table when it exceeds a threshold