# 1. Objective

The objective of this assignment is to implement and analyze the performance of the Quicksort algorithm, both in its deterministic and randomized forms. Through this exercise, students will gain a thorough understanding of Quicksort’s behavior in various scenarios and its relevance in practical applications such as big data processing, web services, and system optimization.

# 2. Existing System

Traditional sorting algorithms include Bubble Sort, Insertion Sort, Merge Sort, and Heap Sort. While Bubble and Insertion Sort are easy to implement, they are inefficient for large datasets. Merge Sort provides stable performance with O(n log n) complexity, but requires additional space. Quicksort, on the other hand, is known for its in-place sorting and fast average-case performance, making it a widely used algorithm.

# 3. Drawbacks of the Existing System

Algorithms like Bubble Sort and Insertion Sort suffer from poor time complexity in large datasets. Merge Sort and Heap Sort are more efficient but can be space-intensive. Deterministic Quicksort, although fast on average, may degrade to O(n²) in worst-case scenarios when the pivot selection is poor (e.g., already sorted arrays).

# 4. Proposed System

The proposed system involves implementing both deterministic and randomized Quicksort algorithms. Randomization helps in avoiding worst-case scenarios by randomly selecting a pivot, thus making it more robust and suitable for unpredictable data distributions.

# 5. Architecture (Design, Modules)

The solution is composed of four main modules:  
1. Quicksort implementation (deterministic)  
2. Quicksort implementation (randomized)  
3. Performance analysis (time/space complexity)  
4. Empirical analysis (testing on various data types)