Coding Assignment for Practice
(time and Space Complexity)
(Dr. GC Jana)

Coding Assignment 1: Analyzing Time Complexity
Objective: Implement simple algorithms and analyze their time complexity.

---

1. **Linear Search Implementation**:
    - Write a Python function to implement linear search on an unsorted list.
    - Analyze the time complexity of your implementation.

2. **Binary Search Implementation**:
- Implement binary search on a sorted list.
- Analyze the time complexity of binary search in the best, worst, and average cases

3. **Comparative Analysis**:
- Compare the time complexities of linear search and binary search. Write a brief report on which search method is more efficient under different circumstances.

Coding Assignment 2: Sorting Algorithms
Objective: Implement sorting algorithms and analyze their performance.

---

1. **Bubble Sort Implementation**:
- Write a Python function to implement bubble sort.
- Analyze the time complexity in the best and worst cases.
2. **Merge Sort Implementation**:
- Implement merge sort in Python.
- Analyze the time complexity and compare it with bubble sort.
3. **Performance Comparison**:
- Write a Python script to generate random lists of different sizes.
- Test both sorting algorithms on these lists and measure their execution times.
- Present your findings in a report, explaining which algorithm performs better and why.

Coding Assignment 3: Exploring O(log n) Complexity
Objective: Implement and analyze algorithms with logarithmic time complexity.

---

1. **Exponentiation by Squaring**:
- Implement an algorithm to calculate a^b in logarithmic time using exponentiation by squaring.
- Analyze the time complexity of your algorithm.
2. **Binary Search Tree (BST) Operations**:
- Implement insertion and search operations in a binary search tree.
- Analyze the time complexity of these operations.

Coding Assignment 4: Advanced Problem Solving
Objective: Apply algorithmic complexity concepts to solve real-world problems.

---

1. **Dijkstra's Algorithm**:
- Implement Dijkstra's algorithm to find the shortest path in a weighted graph.
- Analyze the time complexity of your implementation.
2. **Knapsack Problem**:
- Implement the 0/1 Knapsack problem using dynamic programming.
- Analyze the time and space complexities of your algorithm.
3. **Optimizing Algorithms**:
- Choose a standard algorithm with a known time complexity (e.g., $O(n^2)$) and propose an optimization to improve its efficiency.
    - Implement the optimized version and compare it with the original in terms of time complexity and execution time.