1. Fibonacci Series (Recursive & Non-Recursive)

Aim:

To write recursive and non-recursive Python programs to generate Fibonacci numbers and analyze time and space complexity.

Theory:

The Fibonacci sequence is a series where each term is the sum of the previous two terms: F(n) = F(n−1) + F(n−2). Base cases are F(0)=0 and F(1)=1.

Approaches:

1. Recursive: Uses function calls repeatedly.
2. Iterative: Uses loop and variables to calculate terms efficiently.

Complexity:

Recursive → Time: O(2■), Space: O(n)
Iterative → Time: O(n), Space: O(1)

Sample Output:

Fibonacci Series: 0 1 1 2 3 5 8 ...

Viva Questions:

1. What is the Fibonacci series?
→ It is a series where each term is the sum of the previous two terms.

2. What is the difference between recursion and iteration?
→ Recursion calls itself; iteration uses loops.

3. Why is recursive Fibonacci inefficient?
→ It repeats computations, giving O(2■) time.

4. What are base cases in recursion?
→ F(0)=0, F(1)=1.

5. Which method is faster?
→ Iterative, because it uses O(n) time and O(1) space.

2. Huffman Encoding using Greedy Strategy

Aim:

To implement Huffman Encoding for data compression using the greedy method.

Theory:

Huffman coding is a lossless compression algorithm assigning shorter codes to frequent symbols and longer codes to rare ones.

Steps:

1. Count frequency of each symbol.
2. Build a min-heap.
3. Merge two smallest nodes repeatedly.
4. Assign binary codes (0-left, 1-right).

Complexity:

Time: O(n log n)
Space: O(n)

Sample Output:

Character | Frequency | Code
A | 5 | 0
B | 2 | 10
C | 1 | 11

Viva Questions:

1. What is Huffman encoding?
→ A greedy-based compression technique.

2. Why is it called greedy?
→ It always combines the smallest frequency nodes first.

3. Is Huffman coding lossless?
→ Yes, it preserves original data.

4. What is the time complexity?
→ O(n log n).

5. What type of tree is used?
→ Binary tree.

3. Fractional Knapsack using Greedy Method

Aim:

To solve the fractional knapsack problem using a greedy approach to maximize total profit.

Theory:

In this problem, items can be broken into fractions. The greedy approach selects items based on highest value-to-weight ratio.

Steps:

1. Input items, weights, and values.
2. Compute value/weight ratio.
3. Sort by ratio.
4. Pick items or fractions until full.

Complexity:

Time: O(n log n)
Space: O(n)

Sample Output:

Maximum value in knapsack = 240.0

Viva Questions:

1. What is the greedy choice here?
→ Selecting the item with the highest value-to-weight ratio.

2. What is the difference between fractional and 0/1 knapsack?
→ Fractional allows partial items.

3. Time complexity?
→ O(n log n).

4. Can greedy solve 0/1 knapsack?
→ No, it fails there.

5. Why is greedy optimal here?
→ Because it satisfies the greedy-choice property.

4. 0/1 Knapsack using Dynamic Programming

Aim:

To solve the 0/1 Knapsack problem using dynamic programming for optimal profit.

Theory:

Items are indivisible. DP builds a table dp[i][w] storing max profit using first i items and weight limit w.

Recurrence Relation:

dp[i][w] = max(value[i-1] + dp[i-1][w-weight[i-1]], dp[i-1][w]) if weight[i-1] ≤ w

Complexity:

Time: O(nW)
Space: O(nW)

Sample Output:

Maximum Profit = 220

Viva Questions:

1. What does 0/1 mean?
→ Either take or leave the item.

2. Why DP and not greedy?
→ Greedy fails for 0/1 case.

3. What is the time complexity?
→ O(nW).

4. What is stored in DP table?
→ Maximum profit for subproblems.

5. What principle is used?
→ Principle of optimality.

5. N-Queens Problem using Backtracking

Aim:

To place N queens on an N×N chessboard so that no two queens attack each other.

Theory:

N-Queens is a constraint satisfaction problem solved using backtracking. Queens must not share row, column, or diagonal.

Steps:

1. Place first queen in a safe position.
2. Try placing next queen.
3. If unsafe, backtrack and try another position.
4. Repeat until all queens are placed.

Complexity:

Time: O(N!)
Space: O(N^2)

Sample Output:

One possible solution (for N=4):
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0

Viva Questions:

1. What is the N-Queens problem?
→ To place N queens so none attack each other.

2. Which method is used?
→ Backtracking.

3. Time complexity?
→ O(N!).

4. How is a safe position checked?
→ Check same row, column, and diagonals.

5. What is backtracking?
→ Trying all possibilities and undoing invalid moves.