**def knapsack\_backtracking(weights, values, capacity, n):**

**"""**

**Function to solve the 0/1 Knapsack problem using backtracking.**

**:param weights: List of item weights**

**:param values: List of item values**

**:param capacity: Maximum weight capacity of the knapsack**

**:param n: Number of items**

**:return: Maximum value that can be obtained**

**"""**

**# Recursive helper function to explore all combinations**

**def helper(i, cur\_weight, cur\_value):**

**# Base case: when all items have been considered**

**if i == n:**

**return cur\_value**

**# If adding the current item exceeds the capacity, skip it**

**if cur\_weight + weights[i] > capacity:**

**return helper(i + 1, cur\_weight, cur\_value)**

**# Two possibilities:**

**# 1. Include the current item**

**include = helper(i + 1, cur\_weight + weights[i], cur\_value + values[i])**

**# 2. Exclude the current item**

**exclude = helper(i + 1, cur\_weight, cur\_value)**

**# Return the maximum value between including and excluding the item**

**return max(include, exclude)**

**# Start recursion from the 0th item with 0 current weight and 0 current value**

**return helper(0, 0, 0)**

**# Example usage**

**weights = [1, 2, 3, 5]**

**values = [1, 6, 10, 16]**

**capacity = 7**

**n = len(values)**

**print(f"Maximum value using Backtracking: {knapsack\_backtracking(weights, values, capacity, n)}")**