**ASSIGNMENT 5**

Name: Abhishek Santosh Gaikwad

PRN: 22210975

Div:- TY A [1]

Roll No.: 371017

**Title:**  Implement 0/1 Knapsack problem using Following algorithmic strategies.

(a) Dynamic programming

(b) Back tracking

(C) Branch and bound

**CODE:**

def knapSack(W, wt, val, n, memo={}):

    # Base case: if no items are left or the knapsack capacity is 0, return 0

    if n == 0 or W == 0:

        return 0

    # Check if the result for the subproblem (n, W) is already computed and memoized

    if (n, W) in memo:

        return memo[(n, W)]

    # If the weight of the nth item exceeds the capacity W, we cannot include it

    if wt[n - 1] > W:

        result = knapSack(W, wt, val, n - 1, memo)  # Skip the item and move to the next

    else:

        # Consider both options: including the nth item and excluding it

        # 1. Include the nth item, subtract its weight, and add its value

        # 2. Exclude the nth item and proceed with the remaining items

        result = max(

            val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1, memo),  # Include the item

            knapSack(W, wt, val, n - 1, memo)  # Exclude the item

        )

    # Memoize the result of the current subproblem for future reference

    memo[(n, W)] = result

    return result

# Example usage

val = [60, 100, 120]  # Values of the items

wt = [10, 20, 30]     # Weights of the items

W = 50                # Maximum weight capacity of the knapsack

n = len(val)          # Number of items

# Call the knapSack function to compute the maximum value

result = knapSack(W, wt, val, n)

# Print the result

print("Maximum value that can be obtained:", result)

**OUTPUT:**

