Practical No. 4

Aim : Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

Code :

```
def knapSack(W, wt, val, n):

    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i - 1] <= w:
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]],
                              K[i - 1][w])
            else:
                K[i][w] = K[i - 1][w]


    res = K[n][W]
    w = W
    selected_items = []

    for i in range(n, 0, -1):
        if res <= 0:
            break
        if res == K[i - 1][w]:
            continue
        else:
            selected_items.append(i - 1)  # 0-indexed for easy access
            res -= val[i - 1]
            w -= wt[i - 1]

    return K[n][W], selected_items

if __name__ == "__main__":
    # Take number of items
    n = int(input("Enter number of items: "))
```

```
    val = list(map(int, input("Enter values of items (space-separated):
").split()))
    wt = list(map(int, input("Enter weights of items (space-separated):
").split()))
    W = int(input("Enter capacity of knapsack: "))
    if len(val) != n or len(wt) != n:
        print("Error: Number of values/weights does not match number of
items!")
    else:
        max_value, selected_items = knapSack(W, wt, val, n)

        print("\nMaximum value that can be put in knapsack =", max_value)
        print("Selected items:")

        for i in selected_items[::-1]:  # reverse to maintain original order
            print(f"Item {i+1}: Value = {val[i]}, Weight = {wt[i]}")
```

Output :

```
Enter number of items: 5
Enter values of items (space-separated): 60 100 120 80 50
Enter weights of items (space-separated): 10 20 30 40 10
Enter capacity of knapsack: 50

Maximum value that can be put in knapsack = 230
Selected items:
Item 1: Value = 60, Weight = 10
Item 3: Value = 120, Weight = 30
Item 5: Value = 50, Weight = 10
```