ASSIGNMENT NO. 4

```
import java.util.*; // Import Scanner and List utilities

public class Knapsack {

    // Function to solve 0/1 Knapsack Problem using Dynamic Programming
    public static int knapsack(int W, int[] weights, int[] values, int n) {
        // Create a DP table of size (n+1) x (W+1)
        int[][] dp = new int[n + 1][W + 1];

        // Build DP table bottom-up
        for (int i = 1; i <= n; i++) {
            for (int w = 0; w <= W; w++) {
                if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(
                            dp[i - 1][w],
                            values[i - 1] + dp[i - 1][w - weights[i - 1]]
                    );
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }

        // Display DP Table
        System.out.println("\nDP Table:");
        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= W; w++) {
                System.out.printf("%5d", dp[i][w]);
            }
            System.out.println();
        }

        // Backtrack to find selected items
        int res = dp[n][W];
        int w = W;
        List<Integer> selectedItems = new ArrayList<>();

        for (int i = n; i > 0 && res > 0; i--) {
            if (res != dp[i - 1][w]) { // Item was included
                selectedItems.add(i - 1);
                res -= values[i - 1];
                w -= weights[i - 1];
            }
        }

        // Print selected items
        System.out.println("\nItems included in the knapsack:");
        for (int i = selectedItems.size() - 1; i >= 0; i--) {
            int idx = selectedItems.get(i);
            System.out.println("Item " + (idx + 1) +
                    " -> Weight: " + weights[idx] +
                    ", Profit: " + values[idx]);
        }

        return dp[n][W]; // Return maximum profit
    }

    // Main function
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Input number of items
        System.out.print("Enter number of items: ");
        int n = sc.nextInt();

        // Input weights
        int[] weights = new int[n];
        System.out.println("Enter weights of items:");
        for (int i = 0; i < n; i++) {
            weights[i] = sc.nextInt();
        }

        // Input values (profits)
        int[] values = new int[n];
        System.out.println("Enter values (profits) of items:");
        for (int i = 0; i < n; i++) {
            values[i] = sc.nextInt();
        }

        // Input maximum capacity of knapsack
        System.out.print("Enter maximum weight capacity of knapsack: ");
        int W = sc.nextInt();

        // Compute maximum profit using knapsack DP
        int maxValue = knapsack(W, weights, values, n);

        // Display result
        System.out.println("\nMaximum value (profit) that can be obtained = " +
maxValue);
        System.out.println("Time Complexity: O(n * W)");
        System.out.println("Space Complexity: O(n * W)");

        sc.close();
    }
}
```

OUTPUT:

```
Enter number of items: 4
Enter weights of items:
1 3 4 5
Enter values (profits) of items:
1 4 5 7
Enter maximum weight capacity of knapsack: 7

DP Table:
   0   0   0   0   0   0   0   0
   0   1   1   1   1   1   1   1
   0   1   1   4   5   5   5   5
   0   1   1   4   5   6   6   9
   0   1   1   4   5   7   8   9

Items included in the knapsack:
Item 2 -> Weight: 3, Profit: 4
Item 3 -> Weight: 4, Profit: 5

Maximum value (profit) that can be obtained = 9
Time Complexity: O(n * W)
Space Complexity: O(n * W)


=== Code Execution Successful ===
```