Assignment No 4:- Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

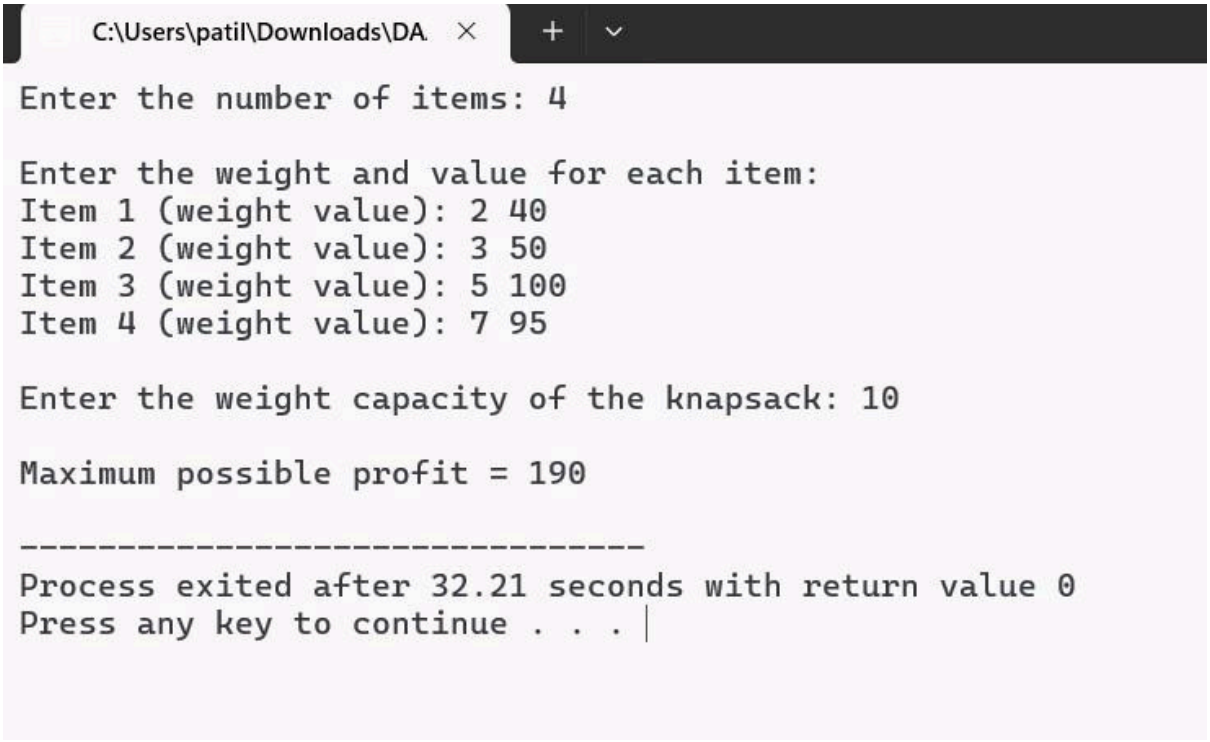
Code:-

```
#include <bits/stdc++.h>
using namespace std;

struct Item
{
   float weight;
   int value;
};

struct Node
{
   int level, profit, bound;
   float weight;
};

bool cmp(Item a, Item b)
{
   double r1 = (double)a.value / a.weight;
   double r2 = (double)b.value / b.weight;
   return r1 > r2;
}

int bound(Node u, int n, int W, Item arr[])
{
   if (u.weight >= W)
      return 0; int

   profit_bound = u.profit;

   int j = u.level + 1; int
   totweight = u.weight;

   while ((j < n) && (totweight + arr[j].weight <= W))
   {
      totweight += arr[j].weight;
      profit_bound += arr[j].value;
      j++;
```

```
    }

    if (j < n)
        profit_bound += (W - totweight) * arr[j].value / arr[j].weight;

    return profit_bound;
}

int knapsack(int W, Item arr[], int n)
{
    sort(arr, arr + n, cmp);

    queue<Node> Q;
    Node u, v;

    u.level = -1;
    u.profit = u.weight = 0;
    Q.push(u);

    int maxProfit = 0;
    while (!Q.empty())
    {
        u = Q.front();
        Q.pop();

        if (u.level == -1)
            v.level = 0;

        if (u.level == n - 1)
            continue;

        v.level = u.level + 1;

        v.weight = u.weight + arr[v.level].weight;
        v.profit = u.profit + arr[v.level].value;

        if (v.weight <= W && v.profit > maxProfit)
            maxProfit = v.profit;

        v.bound = bound(v, n, W, arr);

        if (v.bound > maxProfit)
            Q.push(v);

        v.weight = u.weight;
```

```
        v.profit = u.profit;
        v.bound = bound(v, n, W, arr);
        if (v.bound > maxProfit)
            Q.push(v);
    }
    return maxProfit;
}

int main()
{
    int n, W; cout << "Enter the number
    of items: "; cin >> n;

    Item arr[n]; cout << "\nEnter the weight and value for each
    item:" << endl; for (int i = 0; i < n; i++)
    {
        cout << "Item " << i + 1 << " (weight value): ";
        cin >> arr[i].weight >> arr[i].value;
    }

    cout << "\nEnter the weight capacity of the knapsack: "; cin >> W; cout
    << "\nMaximum possible profit = " << knapsack(W, arr, n) << endl;

    return 0;
}
```

Output:-


C:\Users\patil\Downloads\DA
Enter the number of items: 4
Enter the weight and value for each item:
Item 1 (weight value): 2 40
Item 2 (weight value): 3 50
Item 3 (weight value): 5 100
Item 4 (weight value): 7 95
Enter the weight capacity of the knapsack: 10
Maximum possible profit = 190

Process exited after 32.21 seconds with return value 0
Press any key to continue . . .