**Name:** Rohan Arun Nalawade

**Roll No:** 231012

**PRN:** 22310407

**SY IT A**

**Assignment 8B**

**Code Implementation:**

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

struct Item {

    int weight, value;

    double ratio;

    Item(int w, int v) : weight(w), value(v) {

        ratio = (double)v / w;

    }

};

bool cmp(Item a, Item b) {

    return a.ratio > b.ratio;

}

struct Node {

    int level, profit, weight;

    double bound;

    Node(int l = -1, int p = 0, int w = 0) : level(l), profit(p), weight(w), bound(0) {}

};

struct CompareBound {

    bool operator()(Node a, Node b) {

        return a.bound < b.bound;

    }

};

double bound(Node u, int n, int W, vector<Item> &items) {

    if (u.weight >= W) return 0;

  double profitBound = u.profit;

    int j = u.level + 1;

    int totWeight = u.weight;

 while (j < n && totWeight + items[j].weight <= W) {

        totWeight += items[j].weight;

        profitBound += items[j].value;

        j++;

    }

    if (j < n)

        profitBound += (W - totWeight) \* items[j].ratio;

    return profitBound;

}

int knapsack(int W, vector<Item> &items) {

    sort(items.begin(), items.end(), cmp);

    priority\_queue<Node, vector<Node>, CompareBound> Q;

    Node u, v;

    int n = items.size();

    v.bound = bound(v, n, W, items);

    Q.push(v);

    int maxProfit = 0;

    while (!Q.empty()) {

        v = Q.top(); Q.pop();

        if (v.bound > maxProfit) {

            u.level = v.level + 1;

            if (u.level >= n) continue;

            // Include current item

            u.weight = v.weight + items[u.level].weight;

            u.profit = v.profit + items[u.level].value;

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

                maxProfit = u.profit;

            u.bound = bound(u, n, W, items);

            if (u.bound > maxProfit)

                Q.push(u);

            // Exclude current item

            u.weight = v.weight;

            u.profit = v.profit;

            u.bound = bound(u, n, W, items);

            if (u.bound > maxProfit)

                Q.push(u);

        }

    }

    return maxProfit;

}

int main() {

    int n, W;

    cout << "Enter number of items: ";

    cin >> n;

    vector<Item> items;

    for (int i = 0; i < n; ++i) {

        int value, weight;

        cout << "Enter value and weight for item " << i + 1 << ": ";

        cin >> value >> weight;

        items.push\_back(Item(weight, value));

    }

    cout << "Enter capacity of knapsack: ";

    cin >> W;

    int maxProfit = knapsack(W, items);

    cout << "Maximum profit is: " << maxProfit << endl;

    return 0;

}

**Output:**

