Name: Preyash   Registration Number: 20BPS1022

DAA L45-46

KNAPSACK USING BRANCH AND BOUND

Algorithm:

1) First, determine the value and weight of the construction.

2) Sort the items in ascending order in the array. 3) Now, using the greedy technique, determine the maximum profit that is equal to the upper bound.

4) Take an item from the queue.

5) Calculate the profit of the item at the next level; if the profit is larger than the maximum profit, the maximum profit is updated.

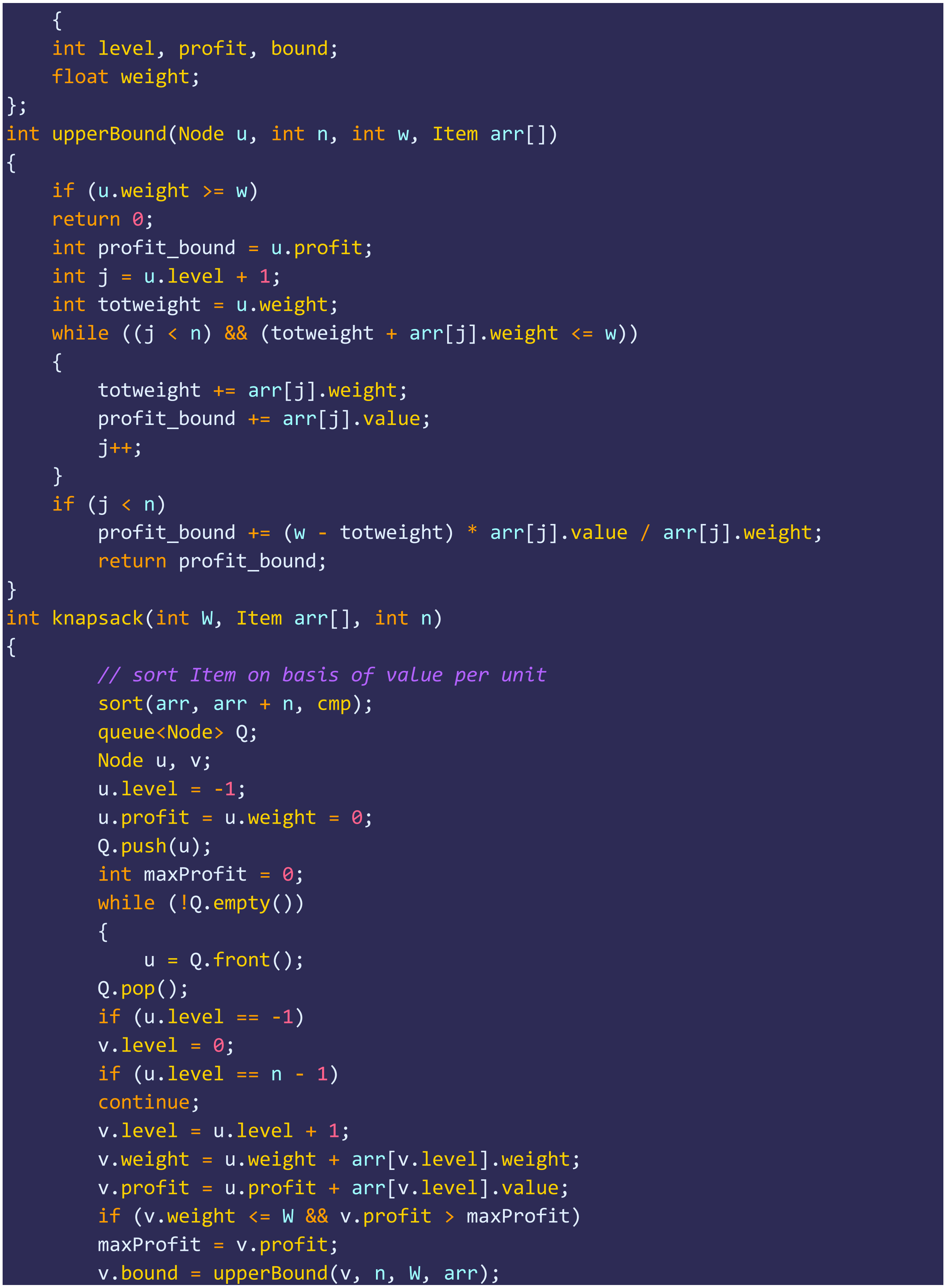
6) If the following node's bound is greater than the maximum profit, add the next level node to Q.

7) Otherwise, toss it out.

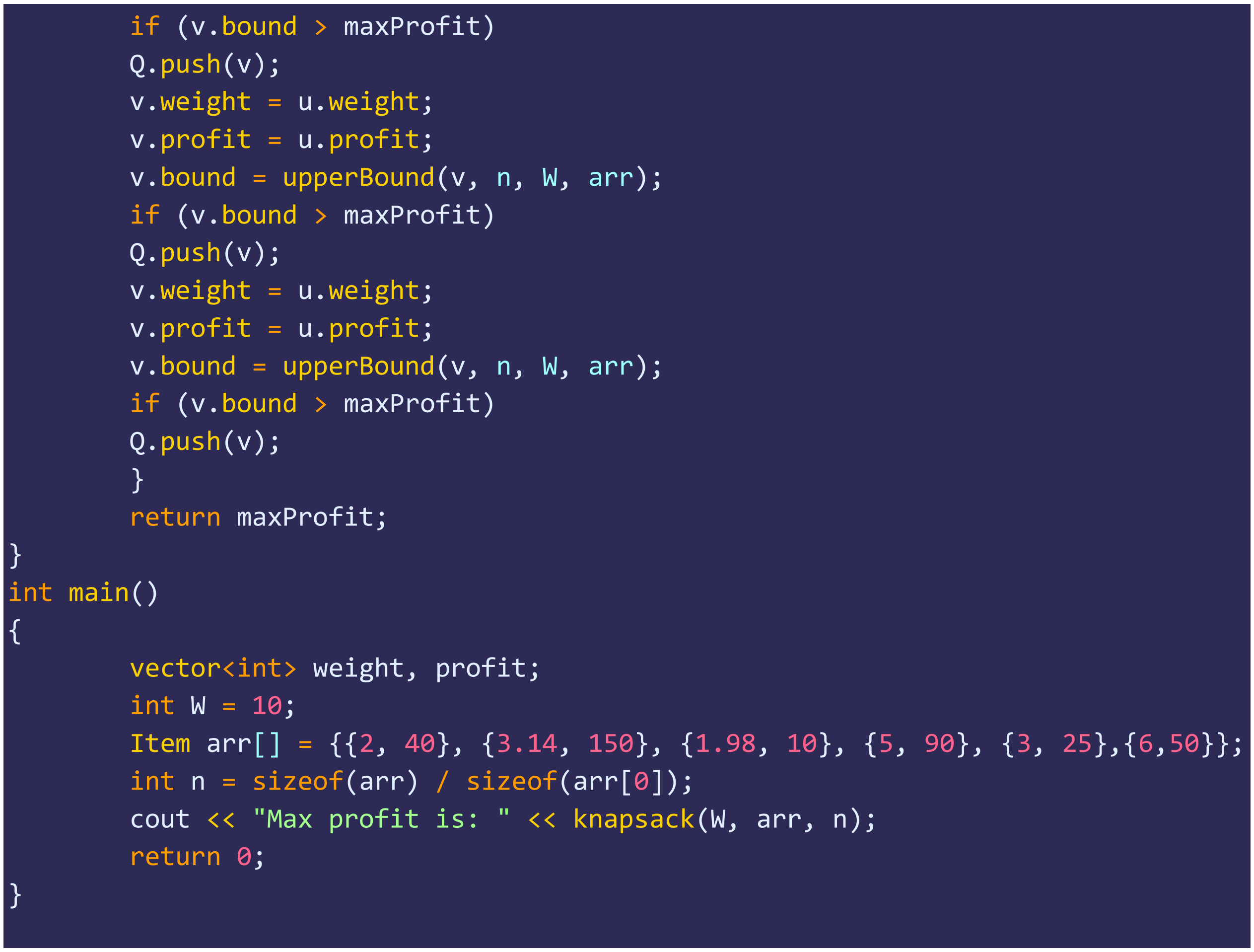
8) If the next level node is not deemed a part of the solution, add it to Q.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
using namespace std;
// knapsack using branch and bound
struct Item
    {
    float weight;
    int value;
};
// calculating upper bound using greedy method
bool cmp(Item a, Item b)
    {
    double a1 = (double)a.value / a.weight;
    double b1 = (double)b.value / b.weight;
    return a1 > b1;
}
struct Node
```

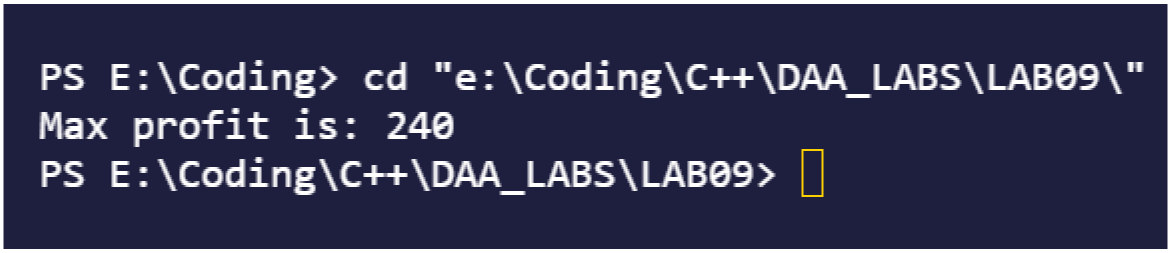


```
    {
    int level, profit, bound;
    float weight;
};
int upperBound(Node u, int n, int w, Item arr[])
{
    if (u.weight >= w)
    return 0;
    int profit_bound = u.profit;
    int j = u.level + 1;
    int totweight = u.weight;
    while ((j < n) && (totweight + arr[j].weight <= w))
    {
        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 Item on basis of value per unit
        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 = upperBound(v, n, W, arr);
```



```
        if (v.bound > maxProfit)
        Q.push(v);
        v.weight = u.weight;
        v.profit = u.profit;
        v.bound = upperBound(v, n, W, arr);
        if (v.bound > maxProfit)
        Q.push(v);
        v.weight = u.weight;
        v.profit = u.profit;
        v.bound = upperBound(v, n, W, arr);
        if (v.bound > maxProfit)
        Q.push(v);
        }
        return maxProfit;
}
int main()
{
        vector<int> weight, profit;
        int W = 10;
        Item arr[] = {{2, 40}, {3.14, 150}, {1.98, 10}, {5, 90}, {3, 25},{6,50}};
        int n = sizeof(arr) / sizeof(arr[0]);
        cout << "Max profit is: " << knapsack(W, arr, n);
        return 0;
}
```

Output:



```
PS E:\Coding> cd "e:\Coding\C++\DAA_LABS\LAB09\"
Max profit is: 240
PS E:\Coding\C++\DAA_LABS\LAB09> 
```