***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

***Algorithms and Problem Solving [15B17CI471]***

***Lab - Week 10***

**Solution 1)**

#include <bits/stdc++.h>

using namespace std;

int minProductSubset(int a[], int n)

{

    if (n == 1)

        return a[0];

    int max\_neg = INT\_MIN, min\_pos = INT\_MAX, count\_neg = 0,

        count\_zero = 0, prod = 1;

    for (int i = 0; i < n; i++) {

        if (a[i] == 0) {

            count\_zero++;

            continue;

        }

        if (a[i] < 0) {

            count\_neg++;

            max\_neg = max(max\_neg, a[i]);

        }

        if (a[i] > 0)

            min\_pos = min(min\_pos, a[i]);

        prod = prod \* a[i];

    }

    if (count\_zero == n || (count\_neg == 0 && count\_zero > 0))

        return 0;

    if (count\_neg == 0)

        return min\_pos;

    if (!(count\_neg & 1) && count\_neg != 0)

        prod = prod / max\_neg;

    return prod;

}

int main()

{

    int a[] = { -1, -1, -2, 4, 3 };

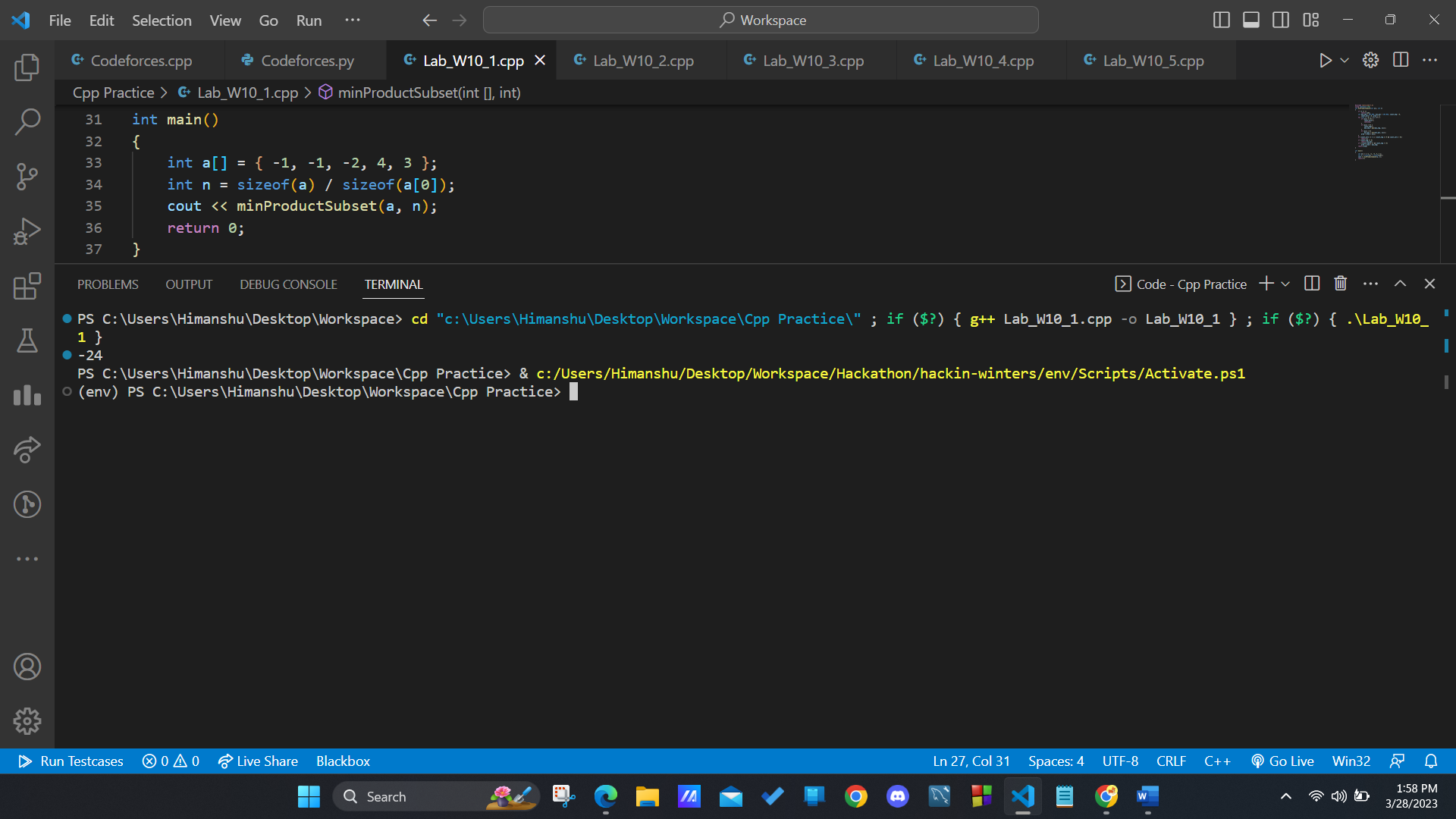
    int n = sizeof(a) / sizeof(a[0]);

    cout << minProductSubset(a, n);

    return 0;

}

**Output :**



**Solution 2)**

#include <bits/stdc++.h>

using namespace std;

struct Activitiy

{

    int start, finish;

};

bool activityCompare(Activitiy s1, Activitiy s2)

{

    return (s1.finish < s2.finish);

}

void printMaxActivities(Activitiy arr[], int n)

{

    sort(arr, arr + n, activityCompare);

    cout << "Following activities are selected :\n";

    int i = 0;

    cout << "(" << arr[i].start << ", " << arr[i].finish

         << ")";

    for (int j = 1; j < n; j++)

    {

        if (arr[j].start >= arr[i].finish)

        {

            cout << ", (" << arr[j].start << ", "

                 << arr[j].finish << ")";

            i = j;

        }

    }

}

int main()

{

    Activitiy arr[] = {{5, 9}, {1, 2}, {3, 4}, {0, 6}, {5, 7}, {8, 9}};

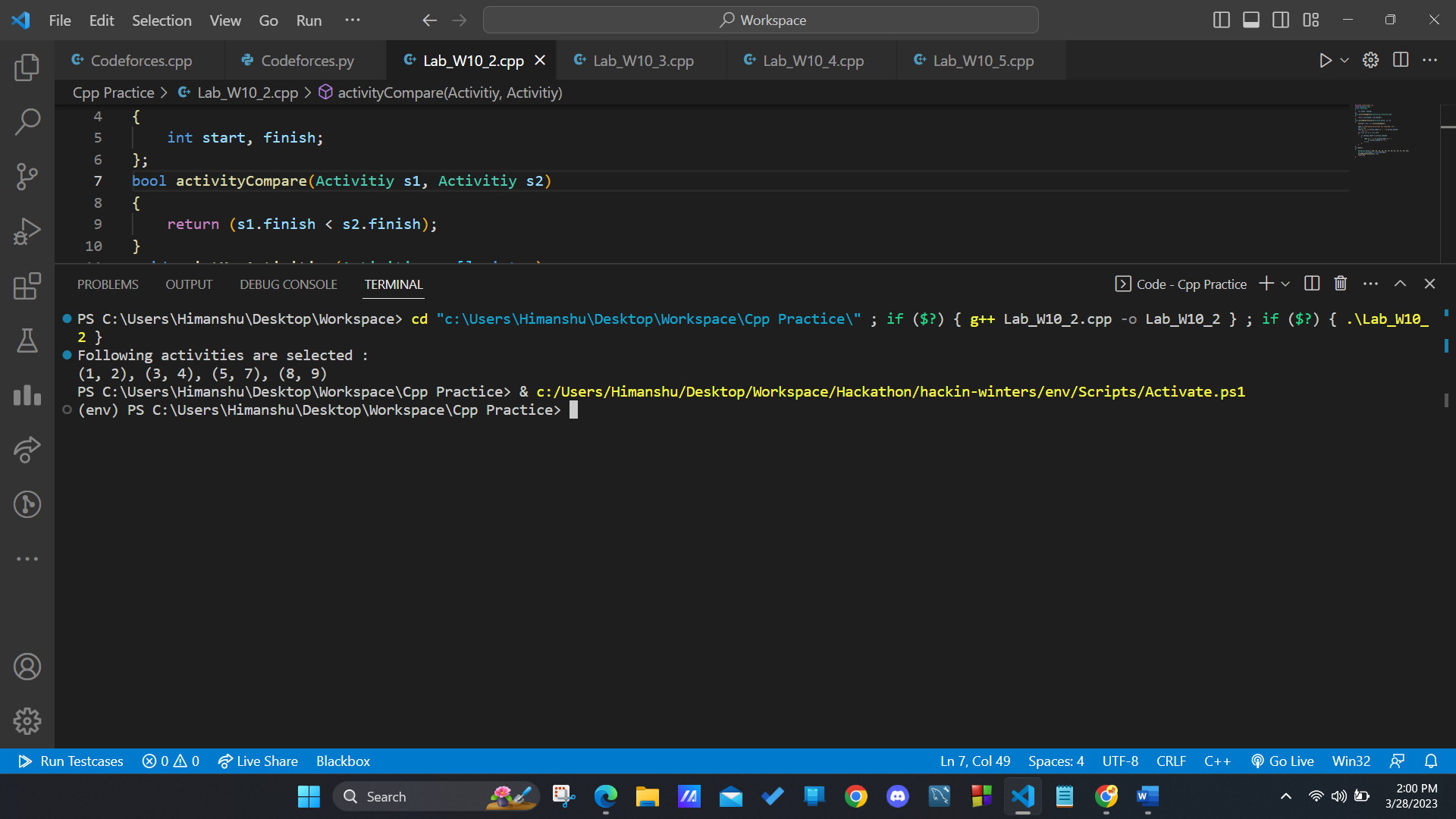
    int n = sizeof(arr) / sizeof(arr[0]);

    printMaxActivities(arr, n);

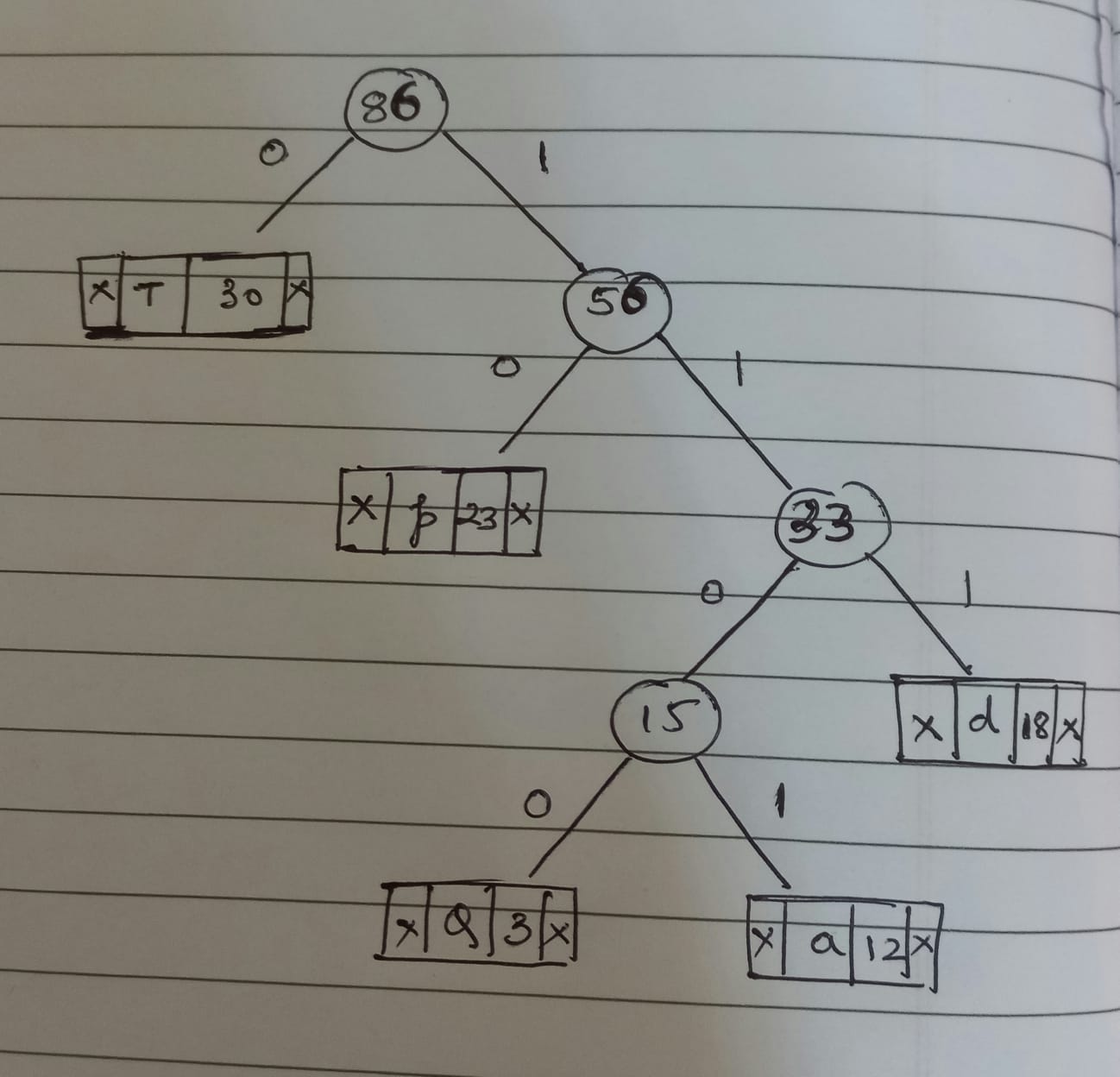
    return 0;

}

**Output :**



**Solution 3)**

****

**Solution 4)**

#include <bits/stdc++.h>

using namespace std;

struct Item

{

    int profit, count;

    Item(int profit, int count)

    {

        this->profit = profit;

        this->count = count;

    }

};

static bool cmp(struct Item a, struct Item b)

{

    double r1 = (double)a.profit / (double)a.count;

    double r2 = (double)b.profit / (double)b.count;

    return r1 > r2;

}

double fractionalKnapsack(int W, struct Item arr[], int N)

{

    sort(arr, arr + N, cmp);

    double finalprofit = 0.0;

    for (int i = 0; i < N; i++)

    {

        if (arr[i].count <= W)

        {

            W -= arr[i].count;

            finalprofit += arr[i].profit;

        }

        else

        {

            finalprofit += arr[i].profit \* ((double)W / (double)arr[i].count);

            break;

        }

    }

    return finalprofit;

}

int main()

{

    int W = 40;

    Item arr[] = {{20000, 10}, {10500, 5}, {89000, 26}, {4000, 8}, {23000, 12}};

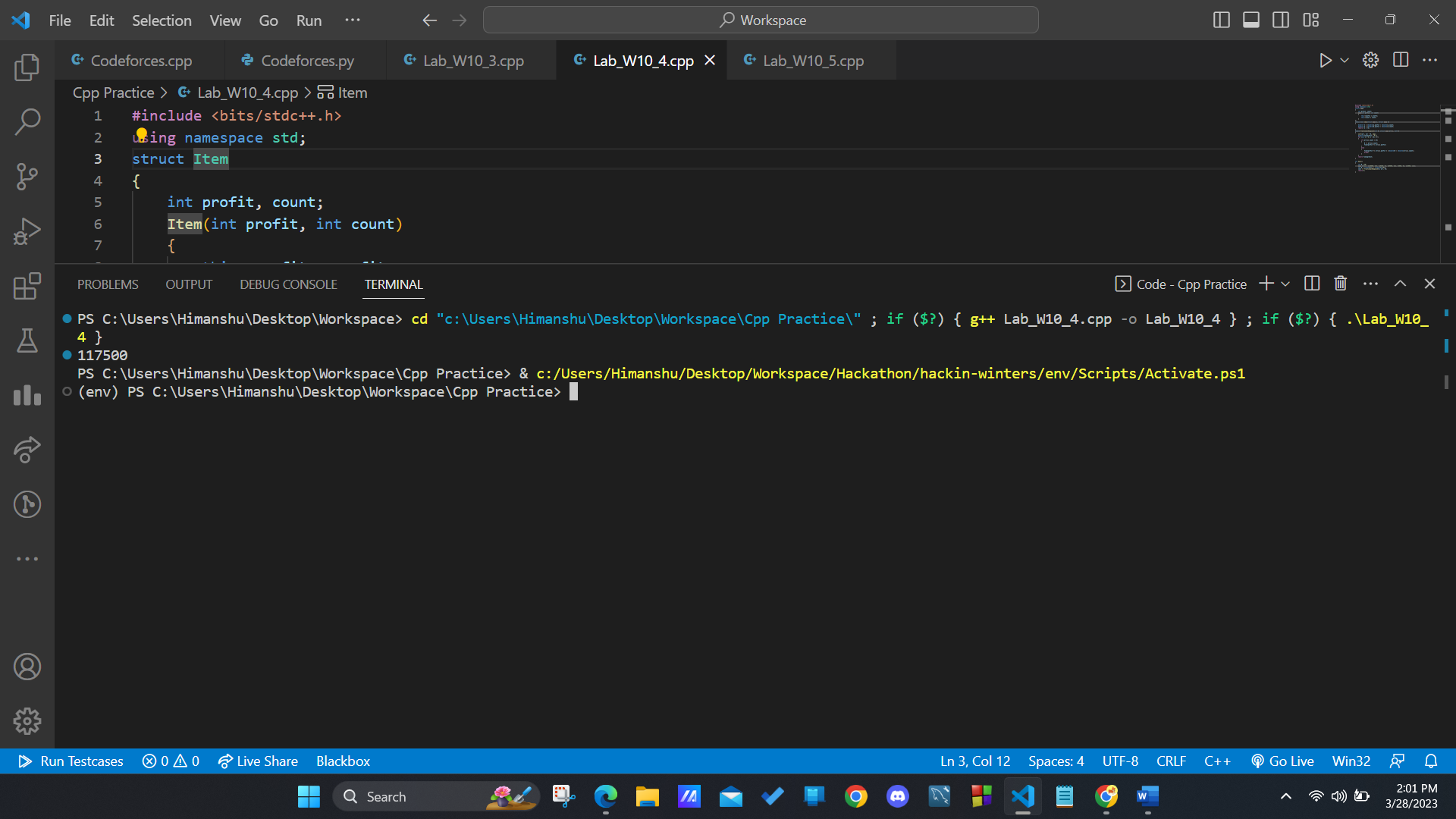
    int N = sizeof(arr) / sizeof(arr[0]);

    cout << fractionalKnapsack(W, arr, N);

    return 0;

}

**Output :**



**Solution 5)**

#include <iostream>

#include <list>

using namespace std;

class Graph

{

    int V;

    list<int> \*adj;

public:

    Graph(int V) { this->V = V; adj = new list<int>[V]; }

    ~Graph()     { delete [] adj; }

    void addEdge(int v, int w);

    void greedyColoring();

};

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w);

    adj[w].push\_back(v);

}

void Graph::greedyColoring()

{

    int result[V];

    result[0] = 0;

    for (int u = 1; u < V; u++)

        result[u] = -1;

    bool available[V];

    for (int cr = 0; cr < V; cr++)

        available[cr] = false;

    for (int u = 1; u < V; u++)

    {

        list<int>::iterator i;

        for (i = adj[u].begin(); i != adj[u].end(); ++i)

            if (result[\*i] != -1)

                available[result[\*i]] = true;

        int cr;

        for (cr = 0; cr < V; cr++)

            if (available[cr] == false)

                break;

        result[u] = cr;

        for (i = adj[u].begin(); i != adj[u].end(); ++i)

            if (result[\*i] != -1)

                available[result[\*i]] = false;

    }

    for (int u = 0; u < V; u++)

        cout << "Vertex " << u << " ---> Color "

            << result[u] << endl;

}

int main()

{

    Graph g1(5);

    g1.addEdge(0, 1);

    g1.addEdge(0, 2);

    g1.addEdge(1, 2);

    g1.addEdge(1, 3);

    g1.addEdge(2, 3);

    g1.addEdge(3, 4);

    cout << "Coloring of graph 1 \n";

    g1.greedyColoring();

    Graph g2(5);

    g2.addEdge(0, 1);

    g2.addEdge(0, 2);

    g2.addEdge(1, 2);

    g2.addEdge(1, 4);

    g2.addEdge(2, 4);

    g2.addEdge(4, 3);

    cout << "\nColoring of graph 2 \n";

    g2.greedyColoring();

    return 0;

}

**Output :**

