**Practical:- 11**

**Aim:-**

Write a program to implements knapsack using greedy.

**Code:-**

#include <bits/stdc++.h>

using namespace std;

struct Item {

    int value, weight;

    Item(int value, int weight)

    {

    this->value=value;

    this->weight=weight;

    }

};

bool cmp(struct Item a, struct Item b)

{

    double r1 = (double)a.value / (double)a.weight;

    double r2 = (double)b.value / (double)b.weight;

    return r1 > r2;

}

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

{

    sort(arr, arr + n, cmp);

    int curWeight = 0;

    double finalvalue = 0.0;

    for (int i = 0; i < n; i++) {

        if (curWeight + arr[i].weight <= W) {

            curWeight += arr[i].weight;

            finalvalue += arr[i].value;

        }

        else {

            int remain = W - curWeight;

            finalvalue += arr[i].value\* ((double)remain/ (double)arr[i].weight);

            break;

        }

    }

    return finalvalue;

}

int main()

{

    int W = 50;

    Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };

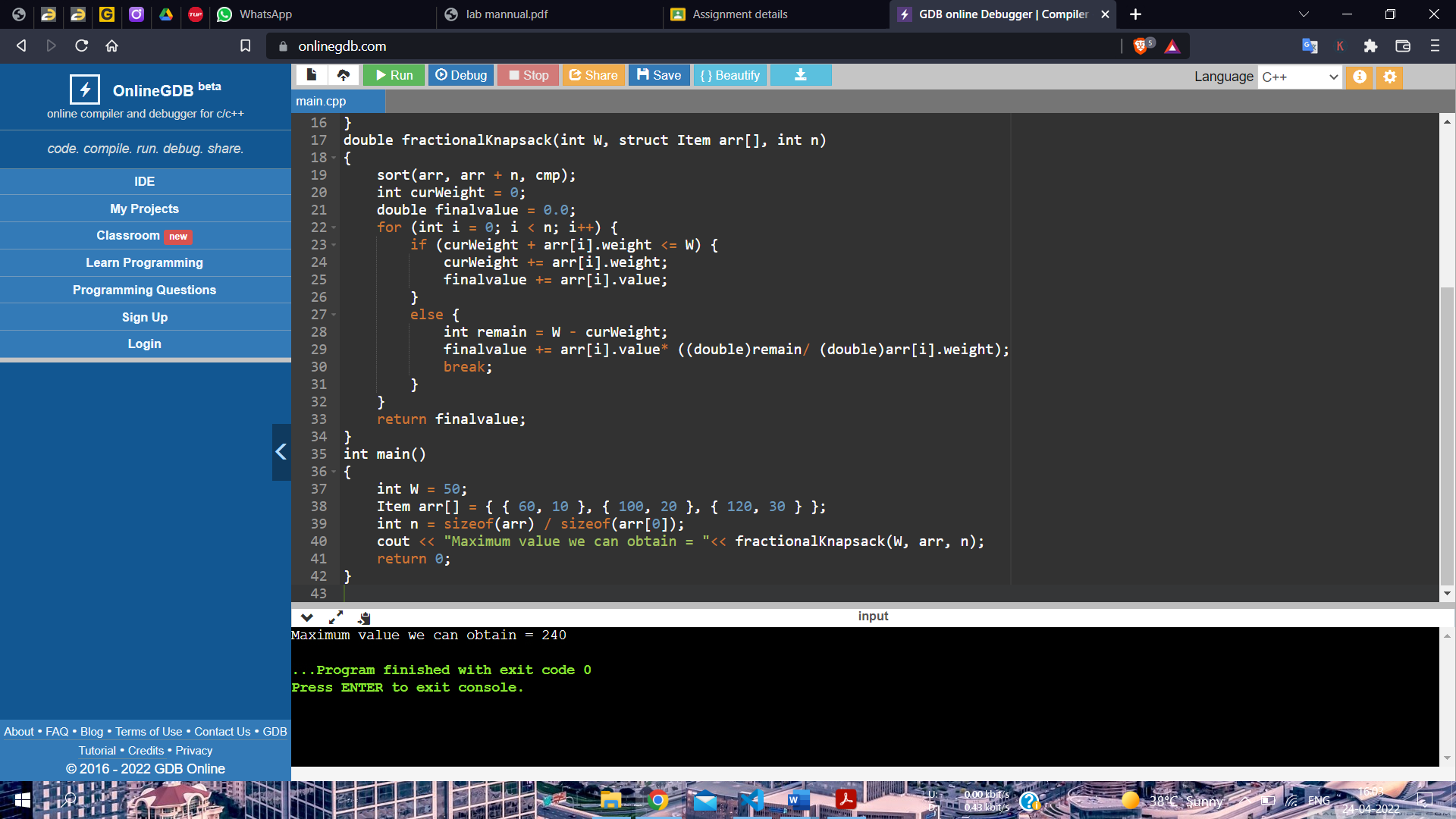
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Maximum value we can obtain = "<< fractionalKnapsack(W, arr, n);

    return 0;

}

**Output:-**



**Practical:- 12**

**Aim:-**

Program for finding minimum cost spanning tree using greedy method using Prim’s algorithm.

**Code:-**

#include <bits/stdc++.h>

using namespace std;

#define V 5

int minKey(int key[], bool mstSet[])

{

    int min = INT\_MAX, min\_index;

    for (int v = 0; v < V; v++)

        if (mstSet[v] == false && key[v] < min)

            min = key[v], min\_index = v;

          return min\_index;

}

void printMST(int parent[], int graph[V][V])

{

cout<<"Edge \tWeight\n";

for (int i = 1; i < V; i++)

cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";

}

void primMST(int graph[V][V])

{

    int parent[V];

    int key[V];

    bool mstSet[V];

    for (int i = 0; i < V; i++)

    key[i] = INT\_MAX, mstSet[i] = false;

    key[0] = 0;

    parent[0] = -1;

    for (int count = 0; count < V - 1; count++)

    {

        int u = minKey(key, mstSet);

        mstSet[u] = true;

        for (int v = 0; v < V; v++)

            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];

    }

    printMST(parent, graph);

}

int main()

{

    cout<<"Edges Along with weight"<<endl;

    int graph[V][V] = { { 0, 2, 0, 6, 0 },

            { 2, 0, 3, 8, 5 },

            { 0, 3, 0, 0, 7 },

                                           { 6, 8, 0, 0, 9 },

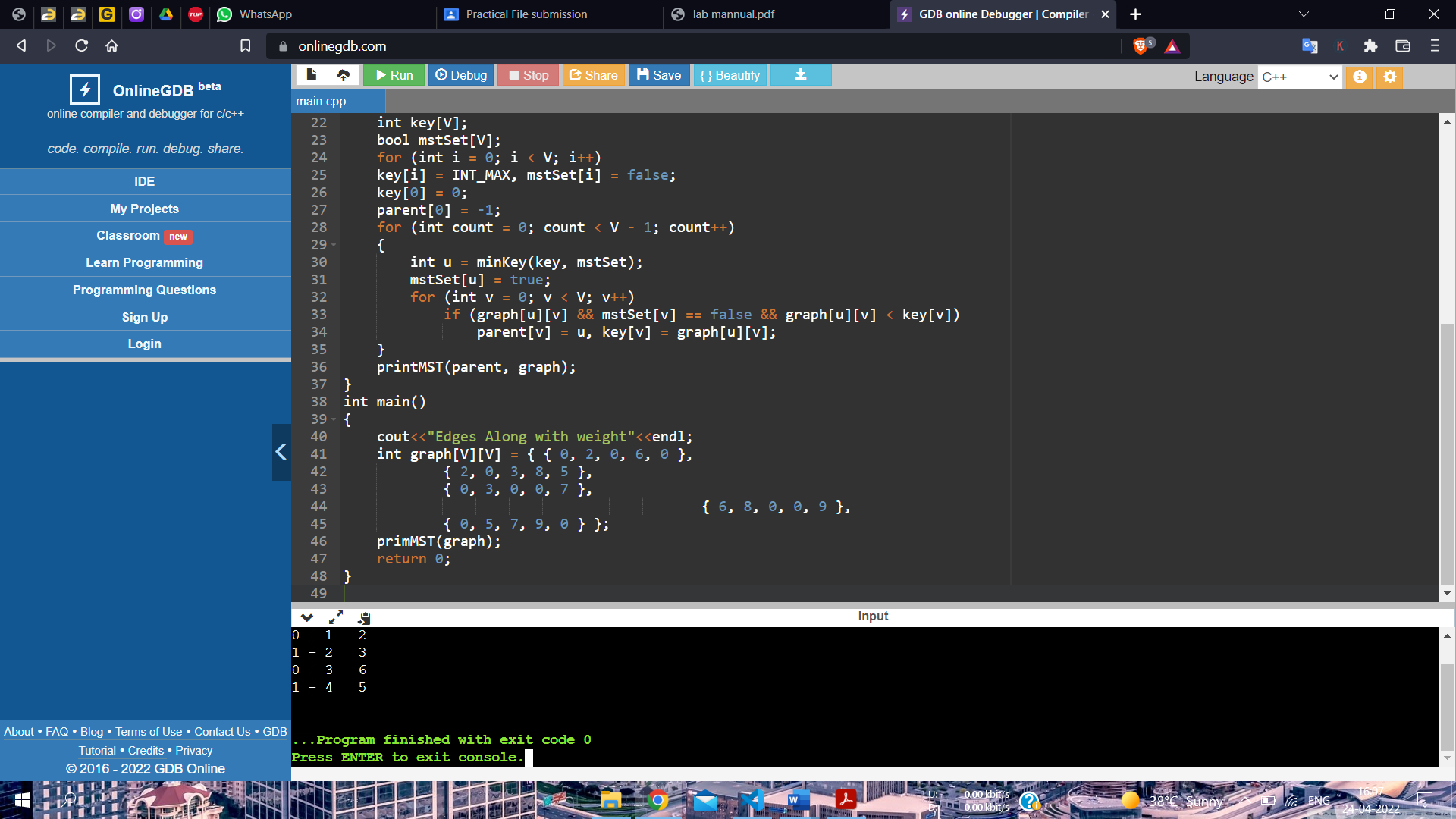
            { 0, 5, 7, 9, 0 } };

    primMST(graph);

    return 0;

}

**Output:-**



**Practical:- 13**

**Aim:-**

Program for finding minimum cost spanning tree using greedy method using Kruskal’s algorithm.

**Code:-**

#include <bits/stdc++.h>

using namespace std;

class DSU {

    int\* parent;

    int\* rank;

public:

    DSU(int n)

    {

                           parent = new int[n];

        rank = new int[n];

        for (int i = 0; i < n; i++) {

            parent[i] = -1;

            rank[i] = 1;

        }

    }

    int find(int i)

    {

        if (parent[i] == -1)

            return i;

                 return parent[i] = find(parent[i]);

    }

    void unite(int x, int y)

    {

        int s1 = find(x);

        int s2 = find(y);

        if (s1 != s2) {

            if (rank[s1] < rank[s2]) {

                parent[s1] = s2;

                rank[s2] += rank[s1];

            }

            else {

                parent[s2] = s1;

                rank[s1] += rank[s2];

            }

        }

    }

};

class Graph {

    vector<vector<int> > edgelist;

    int V;

public:

    Graph(int V) { this->V = V; }

    void addEdge(int x, int y, int w)

    {

               edgelist.push\_back({ w, x, y });

    }

    void kruskals\_mst()

    {

                            sort(edgelist.begin(), edgelist.end());

        DSU s(V);

        int ans = 0;

        cout << "Following are the edges in the constructed Minimum Spanning Tree<< endl;

        for (auto edge : edgelist) {

            int w = edge[0];

            int x = edge[1];

            int y = edge[2];

            if (s.find(x) != s.find(y)) {

                s.unite(x, y);

                ans += w;

                cout << x << " -- " << y << " == " << w

                    << endl;

            }

        }

        cout << "Minimum Cost Spanning Tree: " << ans;

    }

};

int main()

{

    Graph g(4);

              g.addEdge(0, 1, 10);

    g.addEdge(1, 3, 15);

    g.addEdge(2, 3, 4);

    g.addEdge(2, 0, 6);

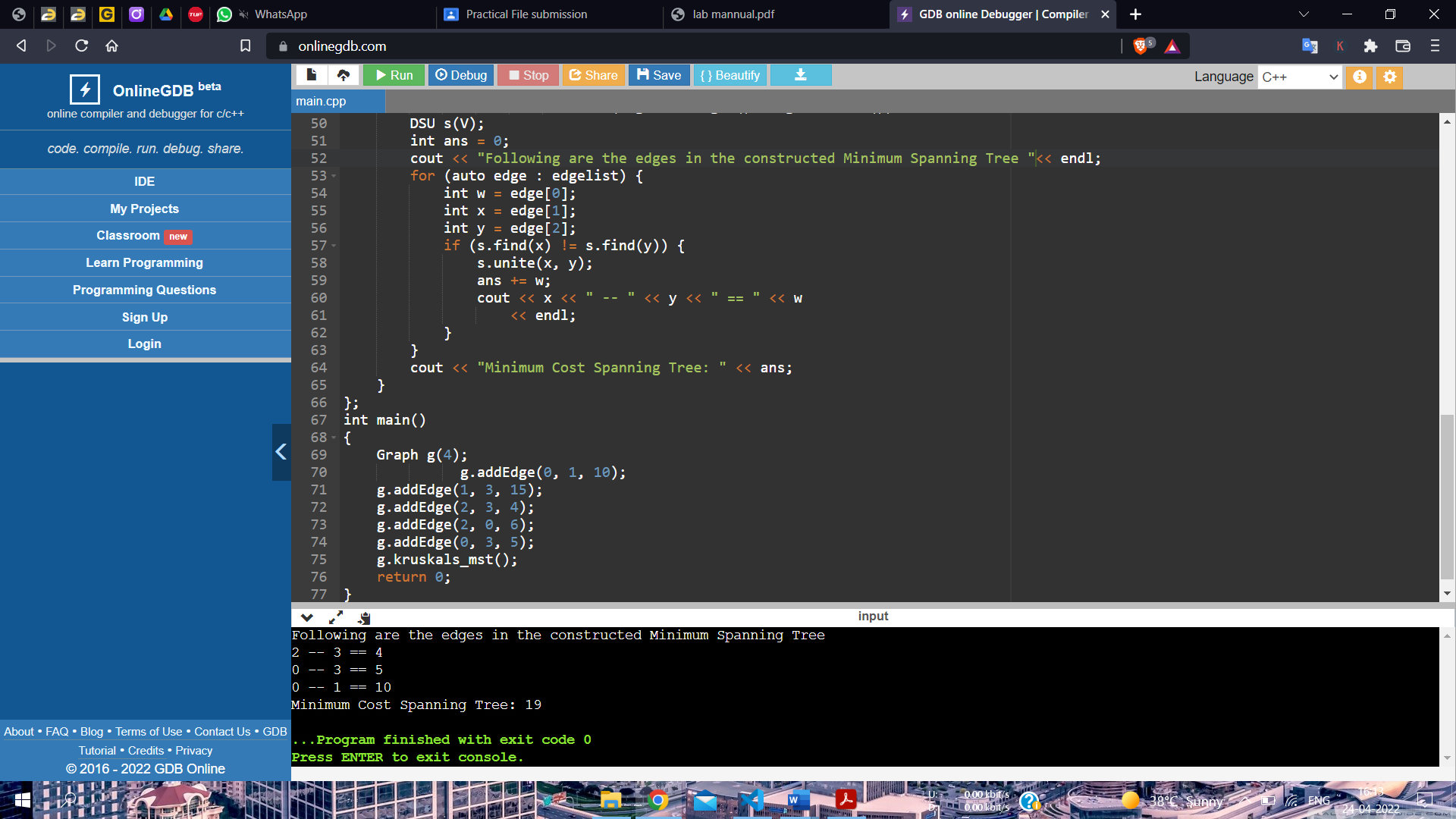
    g.addEdge(0, 3, 5);

    g.kruskals\_mst();

    return 0;

}

**Output:-**



**Practical:- 15**

**Aim:-**

Write a program for optimal merge patterns.

**Code:-**

#include <bits/stdc++.h>

using namespace std;

int minComputation(int size, int files[])

{

    priority\_queue<int, vector<int>, greater<int> > pq;

    for (int i = 0; i < size; i++) {

        pq.push(files[i]);

    }

    int count = 0;

    while (pq.size() > 1) {

        int first\_smallest = pq.top();

        pq.pop();

        int second\_smallest = pq.top();

        pq.pop();

        int temp = first\_smallest + second\_smallest;

        count += temp;

        pq.push(temp);

    }

    return count;

}

int main()

{

    int n = 6;

    int files[] = { 2, 3, 4, 5, 6, 7 };

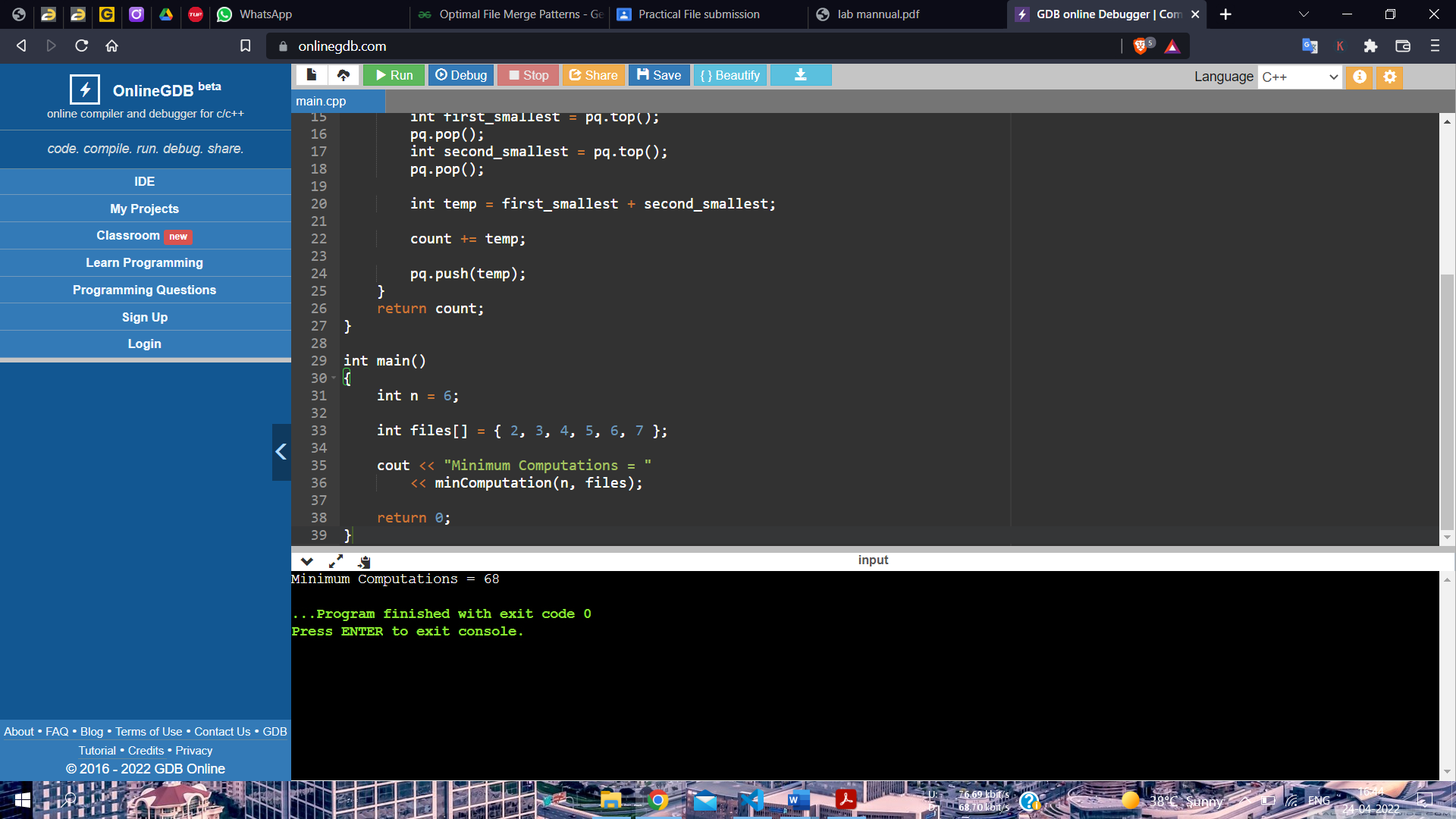
    cout << "Minimum Computations = "

        << minComputation(n, files);

    return 0;

}

**Output:-**



**Practical:- 17**

**Aim:-**

Write a program to implement Hamiltonian cycle problem.

**Code:-**

#include <bits/stdc++.h>

using namespace std;

#define V 5

void printSolution(int path[]);

bool isSafe(int v, bool graph[V][V],int path[], int pos)

{

    if (graph [path[pos - 1]][ v ] == 0)

        return false;

    for (int i = 0; i < pos; i++)

        if (path[i] == v)

            return false;

    return true;

}

bool hamCycleUtil(bool graph[V][V],

                int path[], int pos)

{

    if (pos == V)

    {

        if (graph[path[pos - 1]][path[0]] == 1)

            return true;

        else

            return false;

    }

    for (int v = 1; v < V; v++)

    {

        if (isSafe(v, graph, path, pos))

        {

            path[pos] = v;

            if (hamCycleUtil (graph, path, pos + 1) == true)

                return true;

            path[pos] = -1;

        }

    }

    return false;

}

bool hamCycle(bool graph[V][V])

{

    int \*path = new int[V];

    for (int i = 0; i < V; i++)

        path[i] = -1;

    path[0] = 0;

    if (hamCycleUtil(graph, path, 1) == false )

    {

        cout << "\nSolution does not exist";

        return false;

    }

    printSolution(path);

    return true;

}

void printSolution(int path[])

{

    cout << "Solution Exists: Following is one Hamiltonian Cycle \n";

    for (int i = 0; i < V; i++)

        cout << path[i] << " ";

    cout << path[0] << " ";

    cout << endl;

}

int main()

{

                                                 bool graph1[V][V] = {{0, 1, 0, 1, 0},

                        {1, 0, 1, 1, 1},

                        {0, 1, 0, 0, 1},

                        {1, 1, 0, 0, 1},

                        {0, 1, 1, 1, 0}};

    hamCycle(graph1);

    bool graph2[V][V] = {{0, 1, 0, 1, 0},

                        {1, 0, 1, 1, 1},

                        {0, 1, 0, 0, 1},

                        {1, 1, 0, 0, 0},

                        {0, 1, 1, 0, 0}};

               hamCycle(graph2);

    return 0;

}

**Output:-**

