# 1.1

**CODE**

## **DAA LAB ASSIGNMENT 4**

Admn No: U20CS110

Name: KRISHNA PANDEY

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

*bool* comp(*const* vector<*int*> *&*v1, *const* vector<*int*> *&*v2)

{

*return* v1[0] < v2[0];

}

vector<vector<*int*>> generateSkyline(vector<vector<*int*>> *&*buildings)

{

*int* N = buildings.size(); vector<vector<*int*>> wall; *int* left, right, height; *for* (*int* i = 0; i < N; i++)

{

left = buildings[i][0]; right = buildings[i][1]; height = buildings[i][2];

vector<*int*> v1, v2; v1.push\_back(left); v1.push\_back(height); v1.push\_back(1); *// 1 for left wall*

v2.push\_back(right); v2.push\_back(height); v2.push\_back(0); *// 0 for right wall*

wall.push\_back(v1); *// Storing the left and height*

wall.push\_back(v2); *// Storing the right and height*

}

sort(wall.begin(), wall.end(), comp); *// Comparator to avoid duplicate / redundant points with same x coordinate*

vector<vector<*int*>> skyline;

multiset<*int*> leftWallHeight = {0}; *// Initializing multiset*

*int* top = 0; *// Current max height among walls for* (*int* i = 0; i < wall.size(); i++) *// Traverse through the sorted walls*

{

*// If left wall is found if* (wall[i][2] == 1)

{

leftWallHeight.insert(wall[i][1]); *// Insert the height*

}

*// If right wall is found else*

{

leftWallHeight.erase(leftWallHeight.find(wall[i][1])); *// Remove the*

*height*

}

*if* (\*leftWallHeight.rbegin() != top) *// If top changes then we have to mark the point*

{

top = \*leftWallHeight.rbegin(); vector<*int*> v3; v3.push\_back(wall[i][0]); v3.push\_back(top); skyline.push\_back(v3);

}

}

*return* skyline;

}

*void* printSkyline(vector<vector<*int*>> *&*buildings)

{

vector<vector<*int*>> skyline = generateSkyline(buildings); cout << "The coordinates for generating the skyline are \n"; *int* n = skyline.size();

*for* (*auto* it : skyline)

{

cout << "(" << it[0] << "," << it[1] << ")";

cout << endl;

}

}

*int* main()

{

vector<vector<*int*>> buildings; buildings = {

{33, 41, 5},

{4, 9, 21},

{30, 36, 9},

{14, 18, 11},

{2, 12, 14},

{34, 43, 19},

{23, 25, 8},

{14, 21, 16},

{32, 37, 12},

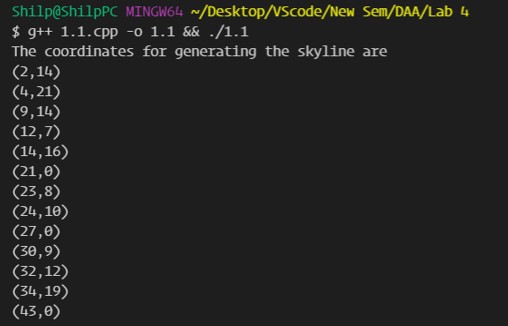
{7, 16, 7},

{24, 27, 10}};

printSkyline(buildings);

}

# OUTPUT



**1.2**

# CODE

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

*struct* point

{

*int* x;

*int* y;

};

vector<vector<*int*>> mergesky(vector<vector<*int*>> *&*skylinel, vector<vector<*int*>>

*&*skylineh)

{

*int* hl = 0, hh = 0; *// to strore the previous heights of skylines int* i = 0, j = 0;

vector<vector<*int*>> merged; *// to store the final output*

*while* (i < skylinel.size() && j < skylineh.size())

{

*if* (skylinel.empty() || skylineh.empty())

{

*break*;

}

vector<*int*> temp;

*if* (skylinel[i][0] < skylineh[j][0]) *// storing the x coordinate and updating the height accordingly*

{

temp.push\_back(skylinel[i][0]);

*if* (skylinel[i][1] < hh) *// if height is less than last height of other skyline*

temp.push\_back(hh);

*else*

temp.push\_back(skylinel[i][1]);

hl = skylinel[i][1]; i++;

}

*else if* (skylinel[i][0] > skylineh[j][0]) *// storing the x coordinate and updating the height accordingly*

{

temp.push\_back(skylineh[j][0]);

*if* (skylineh[j][1] < hl) *// if height is less than last height of other skyline*

temp.push\_back(hl);

*else*

temp.push\_back(skylineh[j][1]);

hh = skylineh[j][1]; j++;

}

*else*

{

temp.push\_back(skylineh[j][0]); temp.push\_back(max(skylinel[i][1], skylineh[j][1])); *// if x*

*coordinate is same the one with higher height gets stored*

hl = skylinel[i][1]; hh = skylineh[j][1]; i++;

j++;

}

merged.push\_back(temp);

}

*if* (i >= skylinel.size()) *// to store the left out points*

{

*while* (j < skylineh.size())

{

vector<*int*> temp; temp.push\_back(skylineh[j][0]); temp.push\_back(skylineh[j][1]); merged.push\_back(temp);

j++;

}

}

*if* (j >= skylineh.size()) *// to store the left out points*

{

*while* (i < skylinel.size())

{

vector<*int*> temp; temp.push\_back(skylinel[i][0]); temp.push\_back(skylinel[i][1]); merged.push\_back(temp);

i++;

}

}

*int* ind = 1; vector<*int*> redun; redun.push\_back(0);

*while* (ind < merged.size())

{

*points*

}

*if* (merged[ind][1] == merged[ind - 1][1]) *// to remove the redundant*

redun.push\_back(1);

*else*

redun.push\_back(0); ind++;

*for* (i = 0; i < redun.size(); i++)

{

*if* (redun[i] == 1) merged[i][0] = -1;

}

*return* merged;

}

vector<vector<*int*>> createSkyline(*int* l, *int* h, vector<vector<*int*>> *&*buildings)

{

vector<vector<*int*>> skyline;

*if* (l > h)

{

*return* skyline; *// empty vector*

}

*else if* (l == h) *// when it reduces to single building and terminating condition*

{

vector<*int*> v1, v2; v1.push\_back(buildings[l][0]); v1.push\_back(buildings[l][2]);

skyline.push\_back(v1); *// storing the left coordinate and height*

v2.push\_back(buildings[l][1]); v2.push\_back(0);

skyline.push\_back(v2); *// storing right coordinate and height as 0 return* skyline;

}

*else*

{

*int* mid = l + ((h - l) / 2);

vector<vector<*int*>> skylinel = createSkyline(l, mid, buildings); vector<vector<*int*>> skylineh = createSkyline(mid + 1, h, buildings); *return* mergesky(skylinel, skylineh);

}

}

*void* printSkyline(vector<vector<*int*>> *&*buildings)

{

vector<vector<*int*>> skyline = createSkyline(0, buildings.size() - 1, buildings);

cout << "The coordinates for generating the skyline are \n";

*int* n = skyline.size();

*for* (*int* i = 0; i < n; i++)

{

*if* (skyline[i][0] != -1) *// checking redundancy*

{

cout << skyline[i][0] << " " << skyline[i][1]; cout << endl;

}

}

}

*int* main()

{

vector<vector<*int*>> buildings; buildings = {

{33, 41, 5},

{4, 9, 21},

{30, 36, 9},

{14, 18, 11},

{2, 12, 14},

{34, 43, 19},

{23, 25, 8},

{14, 21, 16},

{32, 37, 12},

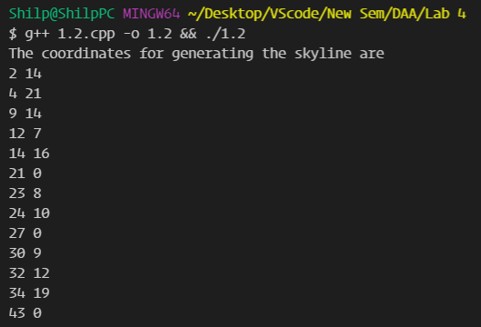
{7, 16, 7},

{24, 27, 10}};

printSkyline(buildings);

}

# OUTPUT



**2.1**

# CODE

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

*int* main()

{

*int* m1, n1, m2, n2;

cout << "Enter rows and columns of matrix A \n"; cin >> m1 >> n1;

cout << "Enter rows and columns of matrix B \n"; cin >> m2 >> n2;

*if* (n1 != m2)

{

cout << "Cannot multiply matrices \n";

*return* 0;

}

*int* a[m1][n1];

*int* b[m2][n2];

*int* c[m1][n2];

cout << "Enter elements of matrix A \n";

*for* (*int* i = 0; i < m1; i++)

{

*for* (*int* j = 0; j < n1; j++)

{

cin >> a[i][j];

}

}

cout << "Enter elements of matrix B \n";

*for* (*int* i = 0; i < m2; i++)

{

*for* (*int* j = 0; j < n2; j++)

{

cin >> b[i][j];

}

}

*int* m = m1; *int* n = n2; *int* i, j, k;

*for* (i = 0; i < m; i++)

{

*for* (j = 0; j < n; j++)

{

c[i][j] = 0;

*for* (k = 0; k < n1; k++)

c[i][j] += a[i][k] \* b[k][j];

}

}

cout << "The product matrix is \n";

*for* (i = 0; i < m; i++)

{

*for* (j = 0; j < n; j++)

{

cout << c[i][j] << " ";

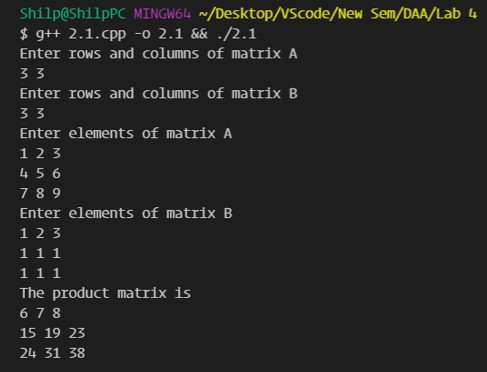
}

cout << endl;

}

}

# OUTPUT



**2.2**

# CODE

*#include* <bits/stdc++.h> using *namespace* std; typedef *long long* ll;

*int \*\**initmat(*int* n)

{

*int* \*\*temp = new *int* \*[n];

*int* i, j;

*for* (i = 0; i < n; i++)

{

temp[i] = new *int*[n];

}

*for* (i = 0; i < n; i++)

{

*for* (j = 0; j < n; j++)

{

temp[i][j] = 0;

}

}

*return* temp;

}

*int \*\**add(*int \*\**A, *int \*\**B, *int* n)

{

*int* \*\*temp = initmat(n);

*int* i, j;

*for* (i = 0; i < n; i++)

{

*for* (j = 0; j < n; j++)

{

temp[i][j] = A[i][j] + B[i][j];

}

}

*return* temp;

}

*int \*\**sub(*int \*\**A, *int \*\**B, *int* n)

{

*int* \*\*temp = initmat(n);

*int* i, j;

*for* (i = 0; i < n; i++)

{

*for* (j = 0; j < n; j++)

{

temp[i][j] = A[i][j] - B[i][j];

}

}

*return* temp;

}

*int \*\**strassen(*int \*\**A, *int \*\**B, *int* n)

{

*if* (n == 1)

{

*int* \*\*C = initmat(n); C[0][0] = A[0][0] \* B[0][0];

*return* C;

}

*int* \*\*C = initmat(n);

*int* k = n / 2;

*int* \*\*A11 = initmat(k);

*int* \*\*A12 = initmat(k); *int* \*\*A21 = initmat(k); *int* \*\*A22 = initmat(k); *int* \*\*B11 = initmat(k); *int* \*\*B12 = initmat(k); *int* \*\*B21 = initmat(k); *int* \*\*B22 = initmat(k);

*for* (*int* i = 0; i < k; i++)

{

*for* (*int* j = 0; j < k; j++)

{

A11[i][j] = A[i][j];

A12[i][j] = A[i][k + j];

A21[i][j] = A[k + i][j];

A22[i][j] = A[k + i][k + j];

B11[i][j] = B[i][j];

B12[i][j] = B[i][k + j];

B21[i][j] = B[k + i][j];

B22[i][j] = B[k + i][k + j];

}

}

*int* \*\*P1 = strassen(A11, sub(B12, B22, k), k); *int* \*\*P2 = strassen(add(A11, A12, k), B22, k); *int* \*\*P3 = strassen(add(A21, A22, k), B11, k); *int* \*\*P4 = strassen(A22, sub(B21, B11, k), k);

*int* \*\*P5 = strassen(add(A11, A22, k), add(B11, B22, k), k); *int* \*\*P6 = strassen(sub(A12, A22, k), add(B21, B22, k), k); *int* \*\*P7 = strassen(sub(A11, A21, k), add(B11, B12, k), k);

*int* \*\*C11 = sub(add(add(P5, P4, k), P6, k), P2, k);

*int* \*\*C12 = add(P1, P2, k);

*int* \*\*C21 = add(P3, P4, k);

*int* \*\*C22 = sub(sub(add(P5, P1, k), P3, k), P7, k);

*for* (*int* i = 0; i < k; i++)

{

*for* (*int* j = 0; j < k; j++)

{

C[i][j] = C11[i][j];

C[i][j + k] = C12[i][j];

C[k + i][j] = C21[i][j];

C[k + i][k + j] = C22[i][j];

}

}

*return* C;

}

*int* main()

{

*int* i, j, \*\*C, \*\*A, \*\*B;

*int* r1, c1, r2, c2;

cout << "Enter rows and columns of matrix A \n"; cin >> r1 >> c1;

cout << "Enter rows and columns of matrix B \n "; cin >> r2 >> c2;

*if* (c1 != r2)

{

cout << "Cannot multiply the matrices \n";

*return* 0;

}

*int* temp, n = max(r1, max(c1, max(r2, c2)));

*if* (ceil(log2(n)) != floor(log2(n)))

{

n = pow(2, ceil(log2(n)));

}

A = initmat(n); B = initmat(n);

cout << "Enter elements of matrix A \n";

*for* (i = 0; i < r1; i++)

{

*for* (j = 0; j < c1; j++)

{

cin >> A[i][j];

}

}

cout << "Enter elements of matrix B \n";

*for* (i = 0; i < r2; i++)

{

*for* (j = 0; j < c2; j++)

{

cin >> B[i][j];

}

}

cout << "\nThe product Matrix C \n"; C = strassen(A, B, n);

*for* (i = 0; i < r1; i++)

{

*for* (j = 0; j < c2; j++)

{

cout << C[i][j] << " ";

}

cout << endl;

}

}

# OUTPUT

