Design and Analysis of Algorithms (CS206)

Assignment - 4

**U19CS012**

1. Assist an architect in *drawing the skyline* of a city given the locations of the buildings in the city. All buildings are **rectangular** in shape and they share a **common bottom** (a flat surface).

A building is specified by an **ordered triplet** (*Li*, *Ri*, *Hi*) where *Li* and *Ri* are the left and right (x) coordinates, respectively, of the building *i* (0 *< Li < Ri*) and *Hi* is the height of the building.

• For example, the input can be as follows.

(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)

The pseudocode/program should give the minimum number of points on graph (coordinates) as output to assist the architect in drawing the skyline.

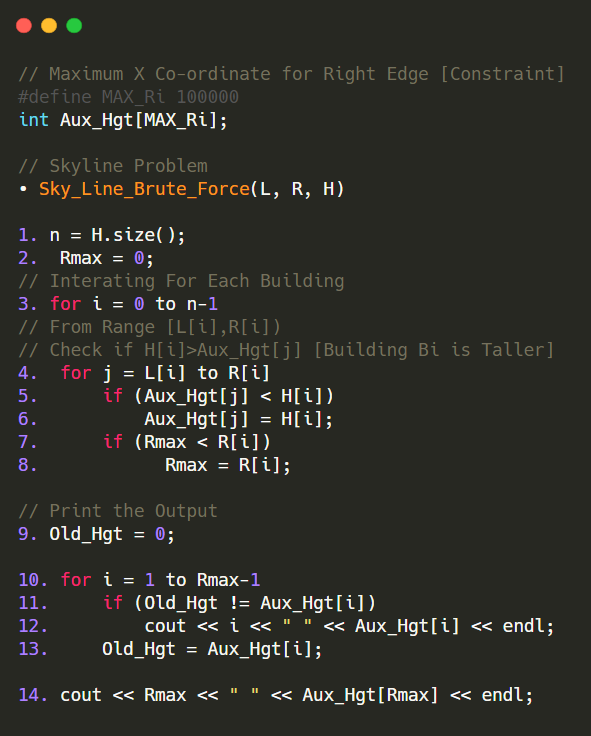
**APPROACH**

We can use an array of 10,000 elements [**Aux\_Hgt**] to represent the *height of each individual discrete x-coordinate*.

For each x-coordinate, we take the highest of all the heights of all the buildings within the range.

For each adjacent x-coordinates, report if there is a change in the height.

1.1. (T) Write a pseudocode (using an incremental/conventional approach) to find the skyline. Analyze the time complexity.



**Analysis**:

Assume **n** = Number of Buildings in Input Sequence, **m** = rightmost x-coordinate [maximum Ri]

From Above Pseudo Code, We are Traversing from Left to Right to Update the Heights. For Worst Case, n Equal Sized Building with l=0 to r = m – 1 coordinates.

Therefore, Running Time = O(n\*m) = O(n2) , if (m>n)

1.2. (L) Write a program using an incremental (conventional) approach to find the

skyline.

*#include* <bits/stdc++.h>

using namespace std;

typedef long long int ll;

*// Maximum X Co-ordinate for Right Edge [Constraint]*

*#define* MAX\_Ri 10010

int Aux\_Hgt[MAX\_Ri];

*// Skyline Problem*

void Sky\_Line\_Brute\_Force(vector<int> &L, vector<int> &R, vector<int> &H)

{

    int n = H.size();

    int Rmax = 0;

*// Interating For Each Building*

*for* (int i = 0; i < n; i++)

    {

*// From Range [L[i],R[i])*

*// Check if H[i]>Aux\_Hgt[j] [Building Bi is Taller]*

*for* (int j = L[i]; j < R[i]; j++)

        {

*if* (Aux\_Hgt[j] < H[i])

                Aux\_Hgt[j] = H[i];

*if* (Rmax < R[i])

                Rmax = R[i];

        }

    }

    int Old\_Hgt = 0;

*for* (int i = 1; i < Rmax; i++)

    {

*if* (Old\_Hgt != Aux\_Hgt[i])

        {

            cout << i << " " << Aux\_Hgt[i] << endl;

            Old\_Hgt = Aux\_Hgt[i];

        }

    }

    cout << Rmax << " " << Aux\_Hgt[Rmax] << endl;

*return*;

}

int main()

{

*// Number of Points*

    ll n;

    cin >> n;

    vector<int> L(n, 0);

    vector<int> R(n, 0);

    vector<int> H(n, 0);

*// li = x-Position Of Left Edge*

*// ri = x-Position Of Right Edge*

*// hi = Building's Height*

*for* (int i = 0; i < n; i++)

    {

        cin >> L[i] >> R[i] >> H[i];

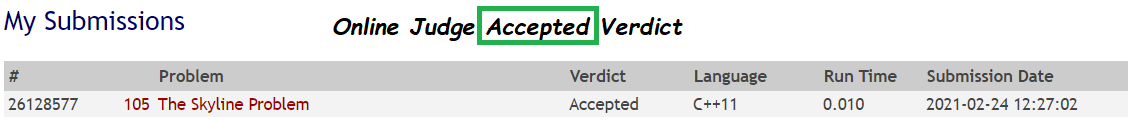
    }

    Sky\_Line\_Brute\_Force(L, R, H);

*return* 0;

}

Only Change in UVa Problem 105 is {Li,Hi,Ri} Instead if {Li,Ri,Hi} [As Mentioned in this Assignment] [Run-Time = 0.010 seconds]



**Sample Test Case**: [Left Edge | Right Edge | Height]

8

1 5 11

2 7 6

3 9 13

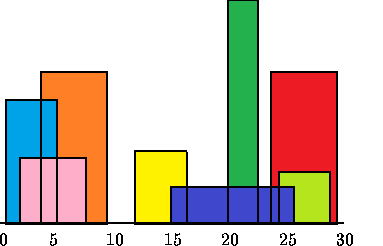
12 16 7

14 25 3

19 22 18

23 29 13

24 28 4



**Expected Output**: [X Co-ordinate | Height]

1 11

3 13

9 0

12 7

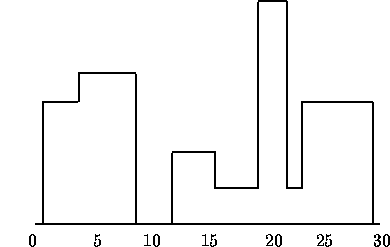
16 3

19 18

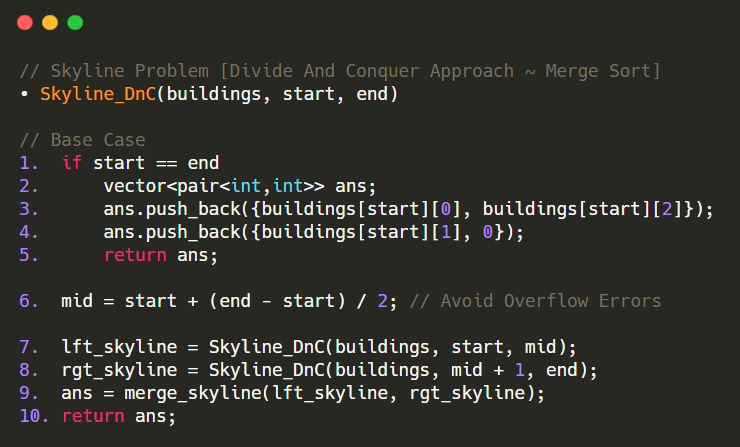
22 3

23 13

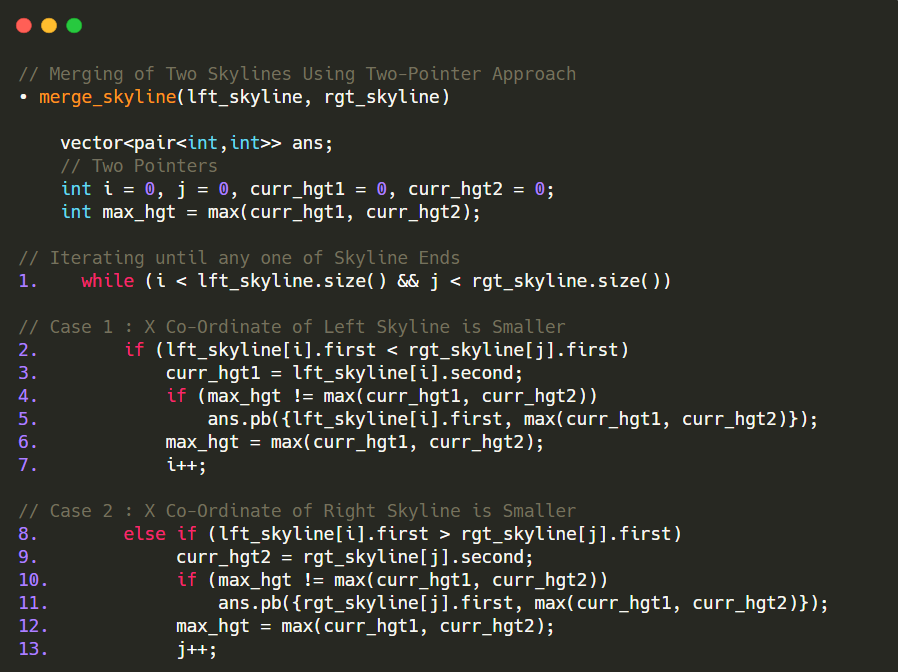
29 0

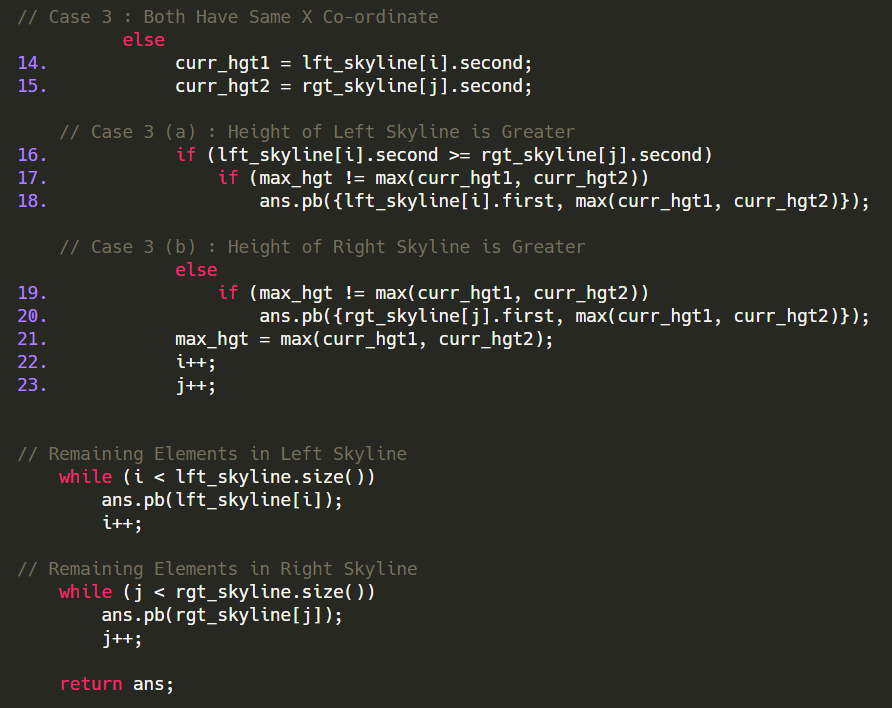


1.3. (T) Write a pseudocode to find the skyline using the divide and conquer approach. Analyze the time complexity.

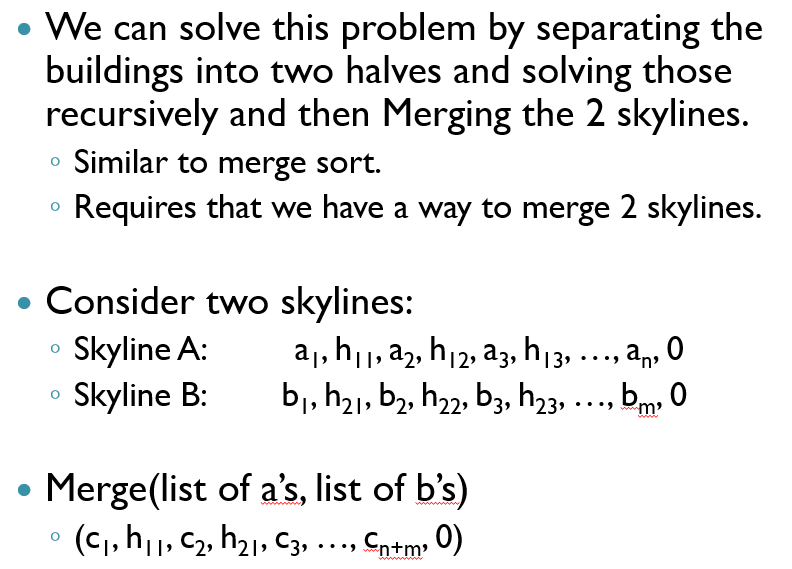


MERGE PART





**APPROACH**



**Complexity Analysis**:

The Above Algorithm has Similar Structure as Merge Sort, Hence the Time Complexity of this Approach is O(n\*log(n)). [Seen in Run-Time Difference]

1.4. (L) Write a program using the divide-and-conquer approach to find the skyline.

*#include* <bits/stdc++.h>

using namespace std;

*//SHORT HAND*

*#define* pb push\_back

*#define* mp make\_pair

typedef vector<int> vi;

typedef vector<vi> vvi;

typedef pair<int, int> pi;

typedef vector<pi> vpi;

*// Merging of Two Skylines Using Two-Pointer Approach*

vpi merge\_skyline(vpi &lft\_skyline, vpi &rgt\_skyline);

*// Skyline Problem [Divide And Conquer Approach ~ Merge Sort]*

vpi Skyline\_DnC(vvi &buildings, int start, int end);

int main()

{

*// Number of Points*

    int n, lft, rgt, hgt;

    cin >> n;

*// lft = x-Position Of Left Edge*

*// rgt = x-Position Of Right Edge*

*// hgt = Building's Height*

    vvi buildings;

*for* (int i = 0; i < n; i++)

    {

        cin >> lft >> rgt >> hgt;

        vi tmp;

        tmp.push\_back(lft);

        tmp.push\_back(rgt);

        tmp.push\_back(hgt);

        buildings.push\_back(tmp);

    }

    vpi final\_ans = Skyline\_DnC(buildings, 0, buildings.size() - 1);

*for* (auto pr : final\_ans)

    {

        cout << pr.first << " " << pr.second << endl;

    }

*return* 0;

}

*// Skyline Problem [Divide And Conquer Approach ~ Merge Sort]*

vpi Skyline\_DnC(vvi &buildings, int start, int end)

{

*// Base Case*

*if* (start == end)

    {

        vpi ans;

        ans.pb({buildings[start][0], buildings[start][2]});

        ans.pb({buildings[start][1], 0});

*return* ans;

    }

    int mid = start + (end - start) / 2;*// Avoid Overflow Errors*

    vpi lft\_skyline = Skyline\_DnC(buildings, start, mid);

    vpi rgt\_skyline = Skyline\_DnC(buildings, mid + 1, end);

    vpi ans = merge\_skyline(lft\_skyline, rgt\_skyline);

*return* ans;

}

*// Merging of Two Skylines Using Two-Pointer Approach*

vpi merge\_skyline(vpi &lft\_skyline, vpi &rgt\_skyline)

{

    vpi ans;

*// Two Pointers*

    int i = 0, j = 0, curr\_hgt1 = 0, curr\_hgt2 = 0;

    int max\_hgt = max(curr\_hgt1, curr\_hgt2);

*// Iterating until any one of Skyline Ends*

*while* (i < lft\_skyline.size() && j < rgt\_skyline.size())

    {

*// Case 1 : X Co-Ordinate of Left Skyline is Smaller*

*if* (lft\_skyline[i].first < rgt\_skyline[j].first)

        {

            curr\_hgt1 = lft\_skyline[i].second;

*if* (max\_hgt != max(curr\_hgt1, curr\_hgt2))

            {

                ans.pb({lft\_skyline[i].first, max(curr\_hgt1, curr\_hgt2)});

            }

            max\_hgt = max(curr\_hgt1, curr\_hgt2);

            i++;

        }

*// Case 2 : X Co-Ordinate of Right Skyline is Smaller*

*else* *if* (lft\_skyline[i].first > rgt\_skyline[j].first)

        {

            curr\_hgt2 = rgt\_skyline[j].second;

*if* (max\_hgt != max(curr\_hgt1, curr\_hgt2))

            {

                ans.pb({rgt\_skyline[j].first, max(curr\_hgt1, curr\_hgt2)});

            }

            max\_hgt = max(curr\_hgt1, curr\_hgt2);

            j++;

        }

*// Case 3 : Both Have Same X Co-ordinate*

*else*

        {

            curr\_hgt1 = lft\_skyline[i].second;

            curr\_hgt2 = rgt\_skyline[j].second;

*// Case 3 (a) : Height of Left Skyline is Greater*

*if* (lft\_skyline[i].second >= rgt\_skyline[j].second)

            {

*if* (max\_hgt != max(curr\_hgt1, curr\_hgt2))

                {

                    ans.pb({lft\_skyline[i].first, max(curr\_hgt1, curr\_hgt2)});

                }

            }

*// Case 3 (b) : Height of Right Skyline is Greater*

*else*

            {

*if* (max\_hgt != max(curr\_hgt1, curr\_hgt2))

                {

                    ans.pb({rgt\_skyline[j].first, max(curr\_hgt1, curr\_hgt2)});

                }

            }

            max\_hgt = max(curr\_hgt1, curr\_hgt2);

            i++;

            j++;

        }

    }

*// Remaining Elements in Left Skyline*

*while* (i < lft\_skyline.size())

    {

        ans.pb(lft\_skyline[i]);

        i++;

    }

*// Remaining Elements in Right Skyline*

*while* (j < rgt\_skyline.size())

    {

        ans.pb(rgt\_skyline[j]);

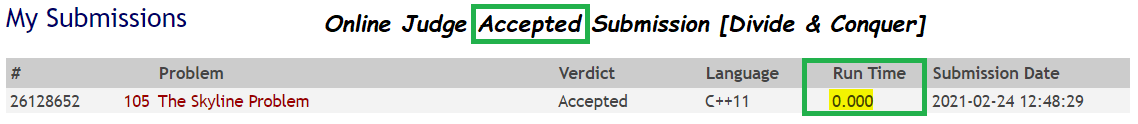
        j++;

    }

*return* ans;

}

Only Change in UVa Problem 105 is {Li,Hi,Ri} Instead if {Li,Ri,Hi} [As Mentioned in this Assignment]



[Run-Time = 0.000 seconds as Compared to **Brute Force** Solution whose Run-Time was 0.010 seconds.]

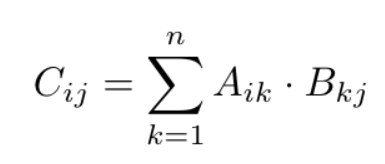
--------------------------------------------------------------------------------------------

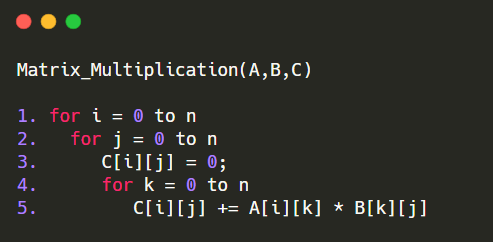
2. Given two matrices *A* and *B*, answer the following questions.

2.1. (T) Write a pseudocode (using an incremental/conventional approach) to multiply the given matrices. Analyze the time complexity.

Suppose we are multiplying 2 matrices A and B and both of them have dimensions n x n.

The resulting matrix C after multiplication in the naive algorithm is obtained by the formula:





In this algorithm, the statement “C[i][j] += A[i][k] \* B[k][j]” executes n³ times as evident from the three nested for loops and is the most costly operation in the algorithm.

Time Complexity of above method is O(N3).

2.2. (L) Write a program using an incremental (conventional) approach to multiply the given matrices.

*#include* <bits/stdc++.h>

using namespace std;

*// Change this According to Your Requirement [Constraints on N]*

*#define* MAX\_N 100

*// Dimensions of A[n1\*m1] & B[n2\*m2]*

int n1, n2, m1, m2;

*// 2 Dimensions Matrix [A,B & C]*

vector<vector<int>> A(MAX\_N, vector<int>(MAX\_N, 0));

vector<vector<int>> B(MAX\_N, vector<int>(MAX\_N, 0));

vector<vector<int>> C(MAX\_N, vector<int>(MAX\_N, 0));

*// Brute Force Method to Multiply Two Matrices 0(N^3)*

void matrix\_multiply()

{

*for* (int i = 0; i < n1; i++)

    {

*for* (int j = 0; j < m2; j++)

        {

            C[i][j] = 0;

*for* (int k = 0; k < m1; k++)

            {

                C[i][j] += A[i][k] \* B[k][j];

            }

        }

    }

*return*;

}

int main()

{

*// Dimensions of Matrix A*

    cout << "Enter Dimensions of Matrix 1 [row col]: " << endl;

    cin >> n1 >> m1;

    cout << "Enter the Values in Matrix 1:" << endl;

*for* (int i = 0; i < n1; i++)

    {

*for* (int j = 0; j < m1; j++)

        {

            cout << "A[" << i << "][" << j << "] = ";

            cin >> A[i][j];

        }

    }

    cout << "Enter Dimensions of Matrix 2 [row col]: " << endl;

*// Dimensions of Matrix B*

    cin >> n2 >> m2;

    cout << "Enter the Values in Matrix 2:" << endl;

*for* (int i = 0; i < n2; i++)

    {

*for* (int j = 0; j < m2; j++)

        {

            cout << "B[" << i << "][" << j << "] = ";

            cin >> B[i][j];

        }

    }

*if* (m1 != n2)

    {

        cout << "Matrix Can't Be Multiplied!" << endl;

        cout << "For Matrix Multiplication,\n No. Of Columns [Matrix-1] Must be Equal No. Of Rows [Matrix-2]!" << endl;

    }

*else*

    {

*// n1 \* m2 = Dimensions of C*

        matrix\_multiply();

        cout << "MATRIX A:" << endl;

*for* (int i = 0; i < n1; i++)

        {

*for* (int j = 0; j < m1; j++)

            {

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

            }

            cout << endl;

        }

        cout << "MATRIX B:" << endl;

*for* (int i = 0; i < n2; i++)

        {

*for* (int j = 0; j < m2; j++)

            {

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

            }

            cout << endl;

        }

        cout << "MATRIX C [AXB]:" << endl;

*for* (int i = 0; i < n1; i++)

        {

*for* (int j = 0; j < m2; j++)

            {

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

            }

            cout << endl;

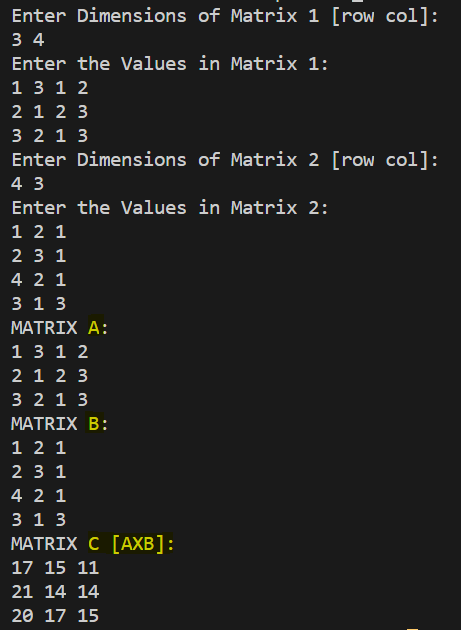
        }

    }

*return* 0;

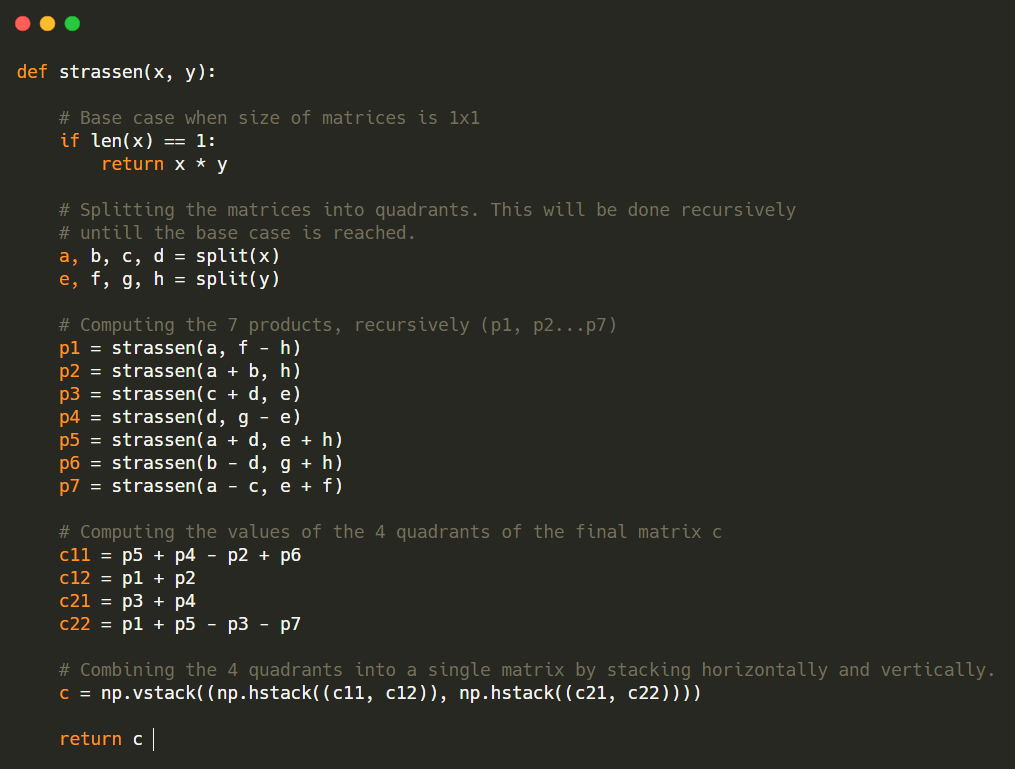
}

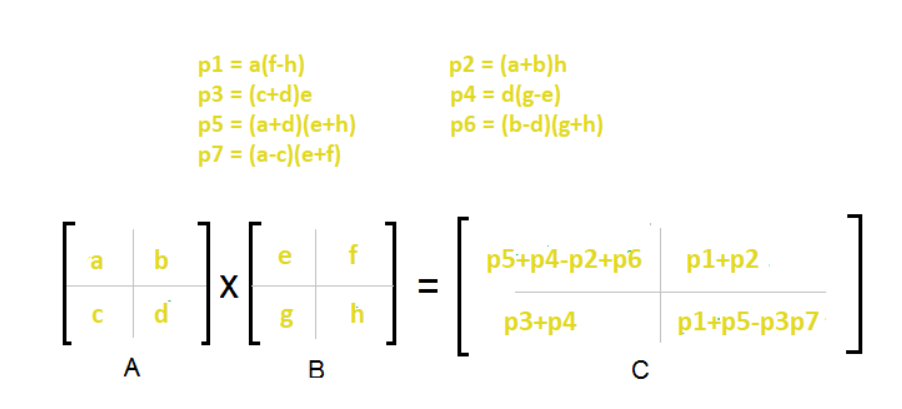
**Sample Test Case:**



2.3. (T) Write a pseudocode to multiply the given matrices using the divide and conquer approach. Analyze the time complexity.

[Note: I have Implemented in C++, But Explaining Pseudo-Code in Python was Easy]



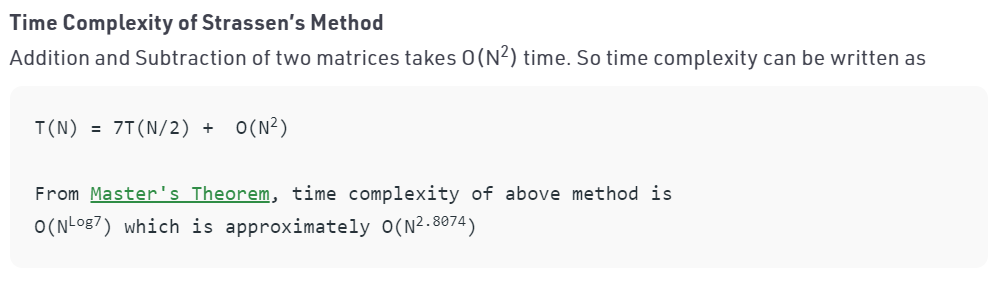


A.) Divide matrices A and B in 4 sub-matrices of size **N/2 x N/2** as shown in the above diagram.

B.) Calculate the **7 matrix multiplications** recursively.

C.) Compute the submatrices of C.

D.) Combine these submatrices into our new matrix C



2.4. (L) Write a program using the divide-and-conquer approach to multiply the given matrices.

*#include* <bits/stdc++.h>

using namespace std;

*// Matrix Operations*

*// To Intialise the Matrix*

int \*\*init\_matrix(int n);

*// To Take Input to Matrix*

void input(int \*\*M, int n);

*// To Print the Matrix*

void print\_matrix(int \*\*M, int n);

*// To Add Two Matrices of Size( n X n )*

int \*\*add(int \*\*M1, int \*\*M2, int n);

*// To Subtract Two Matrices of Size( n X n )*

int \*\*subtract(int \*\*M1, int \*\*M2, int n);

*// Strassen Multiplication Function*

int \*\*Strassen\_Multiply(int \*\*A, int \*\*B, int n);

*// Checks if n is Power of 2 or Not*

bool check(int x)

{

*return* x && (!(x & (x - 1)));

}

int main()

{

    cout << "Enter Size of the Matrix (Power of 2): ";

    int n;

    cin >> n;

*if* (check(n))

    {

        int \*\*A = init\_matrix(n);

        input(A, n);

        int \*\*B = init\_matrix(n);

        input(B, n);

        cout << "Matrix A:" << endl;

        print\_matrix(A, n);

        cout << "Matrix B:" << endl;

        print\_matrix(B, n);

        int \*\*C = init\_matrix(n);

        C = Strassen\_Multiply(A, B, n);

        cout << "MATRIX C [AXB]:" << endl;

        print\_matrix(C, n);

    }

*else*

    {

        cout << "Matrix Can't Be Multiplied!" << endl;

        cout << "Strassian Multiplication => Only Works on Square Matrices whose Dimension is a Power of 2!\n";

    }

*return* 0;

}

*// --------------------------------MATRIX\_OPERATIONS------------------------------------*

*// To Intialise the Matrix*

int \*\*init\_matrix(int n)

{

    int \*\*temp = new int \*[n];

*for* (int i = 0; i < n; i++)

        temp[i] = new int[n];

*return* temp;

}

*// To Take Input to Matrix*

void input(int \*\*M, int n)

{

    cout << "Enter Matrix: " << endl;

*for* (int i = 0; i < n; i++)

*for* (int j = 0; j < n; j++)

            cin >> M[i][j];

    cout << endl;

}

*// To Print the Matrix*

void print\_matrix(int \*\*M, int n)

{

*for* (int i = 0; i < n; i++)

    {

*for* (int j = 0; j < n; j++)

            cout << M[i][j] << " ";

        cout << endl;

    }

    cout << endl;

}

*// To Add Two Matrices of Size( n X n )*

int \*\*add(int \*\*M1, int \*\*M2, int n)

{

    int \*\*temp = init\_matrix(n);

*for* (int i = 0; i < n; i++)

*for* (int j = 0; j < n; j++)

            temp[i][j] = M1[i][j] + M2[i][j];

*return* temp;

}

*// To Subtract Two Matrices of Size( n X n )*

int \*\*subtract(int \*\*M1, int \*\*M2, int n)

{

    int \*\*temp = init\_matrix(n);

*for* (int i = 0; i < n; i++)

*for* (int j = 0; j < n; j++)

            temp[i][j] = M1[i][j] - M2[i][j];

*return* temp;

}

*// Strassen Multiplication Function*

int \*\*Strassen\_Multiply(int \*\*A, int \*\*B, int n)

{

*// Base Case*

*if* (n == 1)

    {

        int \*\*C = init\_matrix(1);

        C[0][0] = A[0][0] \* B[0][0];

*return* C;

    }

    int \*\*C = init\_matrix(n);

    int k = n / 2;

    int \*\*A11 = init\_matrix(k);

    int \*\*A12 = init\_matrix(k);

    int \*\*A21 = init\_matrix(k);

    int \*\*A22 = init\_matrix(k);

    int \*\*B11 = init\_matrix(k);

    int \*\*B12 = init\_matrix(k);

    int \*\*B21 = init\_matrix(k);

    int \*\*B22 = init\_matrix(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\_Multiply(A11, subtract(B12, B22, k), k);

    int \*\*P2 = Strassen\_Multiply(add(A11, A12, k), B22, k);

    int \*\*P3 = Strassen\_Multiply(add(A21, A22, k), B11, k);

    int \*\*P4 = Strassen\_Multiply(A22, subtract(B21, B11, k), k);

    int \*\*P5 = Strassen\_Multiply(add(A11, A22, k), add(B11, B22, k), k);

    int \*\*P6 = Strassen\_Multiply(subtract(A12, A22, k), add(B21, B22, k), k);

    int \*\*P7 = Strassen\_Multiply(subtract(A11, A21, k), add(B11, B12, k), k);

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

    int \*\*C12 = add(P1, P2, k);

    int \*\*C21 = add(P3, P4, k);

    int \*\*C22 = subtract(subtract(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];

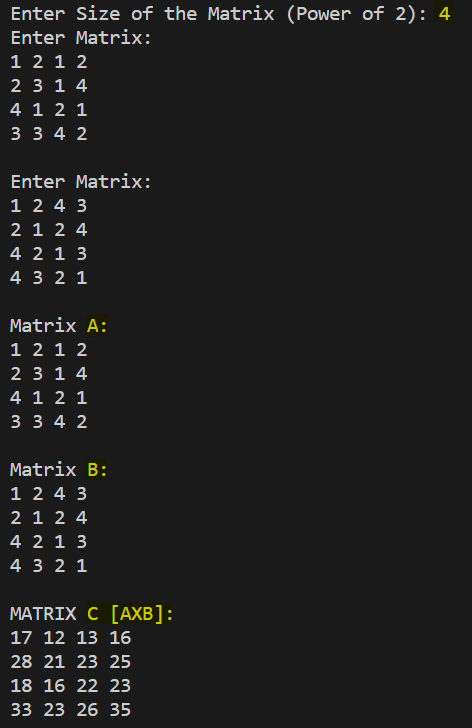
        }

    }

*return* C;

}

**SAMPLE TEST CASE:**



**SUBMITTED BY:**

**U19CS012**

**BHAGYA VINOD RANA**