U19CS076 DAA ASSIGNMENT 4

KRITHIKHA BALAMURUGAN

Q1. 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 (L_i, R_i, H_i) where L_i and R_i are the left and right (x) coordinates, respectively, of the building i ($0 < L_i < R_i$) and $H_i$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 (coor dinates) as output to assist the architect in drawing the skyline.

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

U19CS076

Skyline Problem

Incremental Approach

We convert all co-ordinates (x, y, z) to (x, h, f) through make_input function.
Here f is flag which determines whether a building start at x coord (f=1) or end at x cord (f=0).
We create the structure of (x, h, f) into an array and sort in ascending order of x.

Make_output function converts input to skyline

```
Make_output (sky [i])                                   | Cost
                                                        |
//Declare height array, set prev=0                      |
// Sort height                                          |
For i=0 to sky.size() - 1                               | C1  n
    Set tmp = sky[i].h                                  | C2  n-(n-1)
    IF (sky[i].f = 1)                                   | C3  n-1
        For j=0 to height.size()-1                      | C4  n(n-1)
            IF tmp >= height[j]                         | C5  1
                break                                   | C6  1
        Shift & insert elements temp...j                | C7  n-1
    Else                                                |
        k = find tmp in height[]                        | C8  log(n-1)
        Erase k                                         | C9  n-1
        Set max = height[0]                             | C10 n-1
    If prev ≠ max                                       | C11 n-1
        Declare temp1, temp2                            |
        Set temp1.x = sky[i].x                          | C12 n-1
```

	Cost
temp1.h = max	C_{13} n-1
temp1.f = 1	C_{14} n-1
If sky[] not empty	C_{15} n-1
Set temp2.x = sky[sky.size(i-1)].x	C_{16} n-1
temp2.h = sky[sky.size(i-1)].h	
temp2.f = sky[sky.size(i-1)].f	
If temp2.x = temp1.x	C_{17} n-1
IF (temp2.h < temp.h)	C_{18} n-1
Push back last element	C_{19} n-1
Push in struct temp3	C_{20} n-1
ELSE	
Push in temp struct in sky[]	C_{21} n-1
Else	
Push back temp3 in skyout	C_{22} n-1
Size = Size + 1	C_{23} n-1
Set prev = max	C_{24} n-1
Push in {sky[i-1], x, 0, 1}	C_{25} 1

Time complexity analysis

$$T(n) = C_1(n) + C_2(n-1) + C_3(n-1) + C_4\frac{(n-1)}{2} + C_5 + (C_5 + C_6)\left[\frac{k(n-3)}{2}\right] + C_7 n + C_8 \log n(n-1) + C_9 n(n-1) + C_{10}(n-1) + C_{11}(n-1) + [C_{12} + C_{13} + C_{14} + C_{15} + C_{16} + C_{17} + C_{18} + C_{19} + C_{20} + C_{21} + C_{22} + C_{23} + C_{24}](n-1) + C_{25}$$

$$T(n) = A(n) + B(1) + C_8(n \log n) + C_9 n^2 + (C_5 + C_6)(kn/2)$$

$$= An^2 + Bn + C \rightarrow \text{for best and worst case}$$

$$= \Theta(n^2)$$

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

```
#include<bits/stdc++.h>
using namespace std;
struct initial{
   int x;
   int y;
   int h;
};
struct skyline{
int x;
int h;
int f; //flag if 1 then start of building if 0 then end of building
};
struct initial c[]={ //input
                                        {31, 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}
                                };
vector<struct skyline> in;
vector<struct skyline> out;
int size=0;
void make_input(vector<struct skyline> &in);
void sort(int n);
void make_output(vector<struct skyline> &out);
int main()
{
   int i;
   make_input(in);
   sort(in.size());
   /*cout<<"\n\nBuilding coords are : "<<endl;
   for(i=0;i<in.size();i++)
   {
   cout<<"( "<<in[i].x<<" , "<<in[i].h<<" )"<<endl;
   }
   */
   make_output(out); //skyline function
}
```

```
void make_input(vector<struct skyline> &in)
{
   int i;
   cout<<"\n\nEntered input  was : "<<endl;
   for(i=0;i<11;i++)
   {
            cout<<"( "<<c[i].x<<" , "<<c[i].h<<" , "<<c[i].y<<" ) "<<"\t";
   }
   struct skyline s;
   for(i=0;i<11;i++)
   {
            s.x=c[i].x;
            s.h=c[i].h;
            s.f=1;
            in.push_back(s);
            s.x=c[i].y;
            s.h=c[i].h;
            s.f=0;
            in.push_back(s);
   }
            /*cout<<"\n\nNow modified input  is : "<<endl;
   for(i=0;i<in.size();i++)
   {
            cout<<"( "<<in[i].x<<" , "<<in[i].h<<" ) "<<endl;
   }*/
}
void sort(int n) //for sorting
{
   int i;
   for(int i = 0 ; i < in.size() ; i++)
   {
            bool swap = false;
            for (int j = 0 ; j < in.size()-i-1 ; j++)
            {
                     if(in[j].x > in[j+1].x)
                     {
                              int x1, h1, f1;                    //swapping both
                              x1 = in[j].x;
                              h1 = in[j].h;
                              f1 = in[j].f;
                              in[j].x = in[j+1].x;
                              in[j].h = in[j+1].h;
                              in[j].f = in[j+1].f;
                              in[j+1].x = x1;
                              in[j+1].h = h1;
                              in[j+1].f = f1;
```

```
                        swap = true;
                }
        }
        if(!swap)
                break;
  }
}
void make_output(vector<struct skyline> &out)
{
  vector<int> height; //height vector is made such that it is sorted in //descending order
  int temp,prev=0,max;
  struct skyline s1,s2,s3;
  int i,j,l;
  for(i=0;i<in.size();i++)
  {
        temp=in[i].h;
        if(in[i].f==1) //start of building
        {
                if(height.empty()==true) //no height in the vector
                {
                        if(i!=0) //to append the points when a skyline starts
                        {
                                s1.x=in[i-1].x;
                                s1.h=0;
                                s1.f=0; //flags not relavent
                                out.push_back(s1);
                                size++;
                        }
                        else
                        {                                               //for first case initalise
x and h with 0
                                s1.x=0;
                                s1.h=0;
                                s1.f=1;
                                out.push_back(s1);
                                size++;
                        }
                                height.push_back(temp);
                }
                else //height vector not empty
                {
                        for(j=0;j<height.size();j++)
                        {
                                if(temp>=height[j])
                                        break;
                        }
```

```
                        height.push_back(0);
                        for(l=height.size()-1; l>j; l--) //shifting right
                        {
                                height[l]=height[l-1];
                        }
                                height[j]=temp; //height added to vector
                }
                max=height[0]; //since its  in desc order
        }
        else //f==0 i.e. end of building
        {
                auto k=find(height.begin(),height.end(),temp);
                height.erase(k);
                max=height[0];
        }
        if(prev!=max)
        {
                s3.x=in[i].x;
                s3.h=max;
                s3.f=1;
                if(size!=0) //size here is size of out
                {
                        s2.x=out[size-1].x;
                        s2.h=out[size-1].h;
                        s2.f=out[size-1].f;
                        if(s2.x==s3.x)
                        {
                                if(s2.h<s3.h)
                                {
                                out.pop_back(); //to remove repeated pts
                                out.push_back(s3);
                                }
                        }
                        else
                        {
                                out.push_back(s3);
                        }
                }
                else
                {
                        out.push_back(s1); //appending the original point
                }
                size++;
                prev=max; //changing previous max height
        }
}
```

```
   s3.x=in[i-1].x; //appending the last point
   s3.h=0;
   s3.f=1;
   out.push_back(s3);
   //displaying
   cout<<"\n\nSkyline output is : "<<endl;
   for(i=0;i<out.size();i++)
   {
            cout<<"( "<<out[i].x<<" , "<<out[i].h<<" ) "<<endl;
   }
}
```

Result;

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

U19CS076

Divide and Conquer Approach

There are 2 structures → building (x, y, h) and point coordinates of (x, y)

```
Skyline (int l, int h, vector <bulding> sky)
Declare vector <struct points> skyline
    IF  l > h
    |   return skyline vector
    ELSE IF  l = h
    |   Push back skyline {sky[l].x, sky[l].h},
    |                      {sky[l].y, 0}
    ELSE
    |   set mid = l + [(h-1)/2]
    |   Set vector <struct points> sky_l = skyline(l, mid, sky)
    |   Set vector <struct points> sky_h = skyline(mid+1, h, sky)
    |
    Return merge_sky(sky_l, sky_h)

Merge_sky(vector <points> sky_l, vector <points> sky_h)
{Set h_l=0, h_h=0
set vector <points> merged_sky
While i < sky_l.size() & j < sky_h.size()
    IF sky_l or sky_h is empty
    |   break
    Set temp of <struct points>
    IF sky_l[i].x < sky_h[j].x
    |   temp.x = sky_l[i].x
    |   temp.y = sky_l[i].y
        IF sky_l[i].y < h_h
```

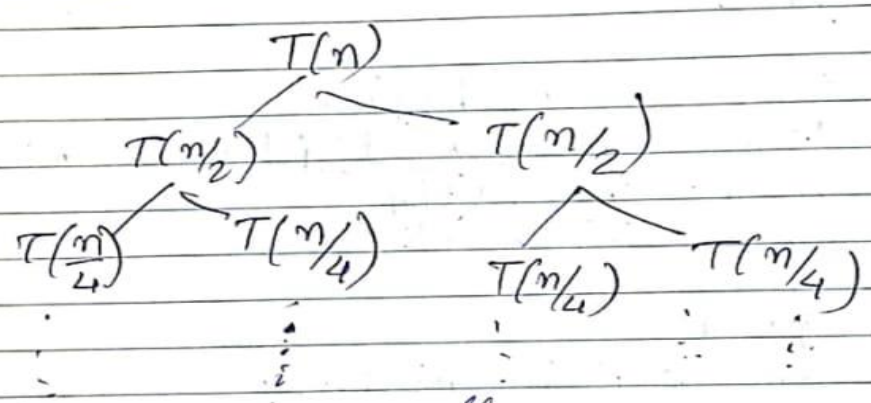
```
        temp.y = h - h
        hl = sky_l[i].y
        i += 1
    ELSE IF sky_l[i].x > sky_h[j].x
        temp.x = sky_h[j].x
        temp.y = sky_h[i].y
        IF sky_h[i].y < hl
            temp.y = h - l
        h_h = sky_h[i].y
        j += 1
    ELSE
        temp_x = sky_h[j].x
        temp_y = max [sky_l[j].y, sky_h[j].y]
        h_l = sky_l[j].y
        h_h = sky_h[j].y
        i += 1
        j += 1
    Push in 'temp' in merged_sky
IF i >= sky_l.size()
    While j < sky_h.size()
        push_back sky_h[j] in merged_sky
        i += 1
IF j >= sky_h.size()
    While i < sky_l.size()
        push back sky_l[i] in merged_sky
Set inx = 1, push back 0 in reductant vector
While (inx < merged_sky.size())
    IF merged_sky[inx].y = merged_sky[inx - 1].y
        redundant pushback(1) into redundant
    ELSE push 0 into redundant
    inx += 1
```

U19CS076

```
For i=0 to redundant.size()
    IF redundant[i] = 1
        merged_sky[i].x = -1
return merged_sky
```

Time complexity analysis

Recursion Tree Method



$T(n)$ can be written as

$T(n) = 2T(n/2) + \Theta(n)$ → Merge will take $\Theta(n)$

also $\Theta(n) = cn$

$T(n) = 2T(n/2) + cn$

and $T(n/2) = 2T(n/4) + cn/2$

$T(n) = 4T(n/4) + cn + cn$

IIIly $T(n) = 2^h T\left(\frac{n}{2^h}\right) + cn\,2^h$

$2^h = n \rightarrow h = \log n$

$T(n) = nT(1) + cn(\log n)$

$T(n) = cn\log n + n + 1 = \Theta(n\log n)$

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

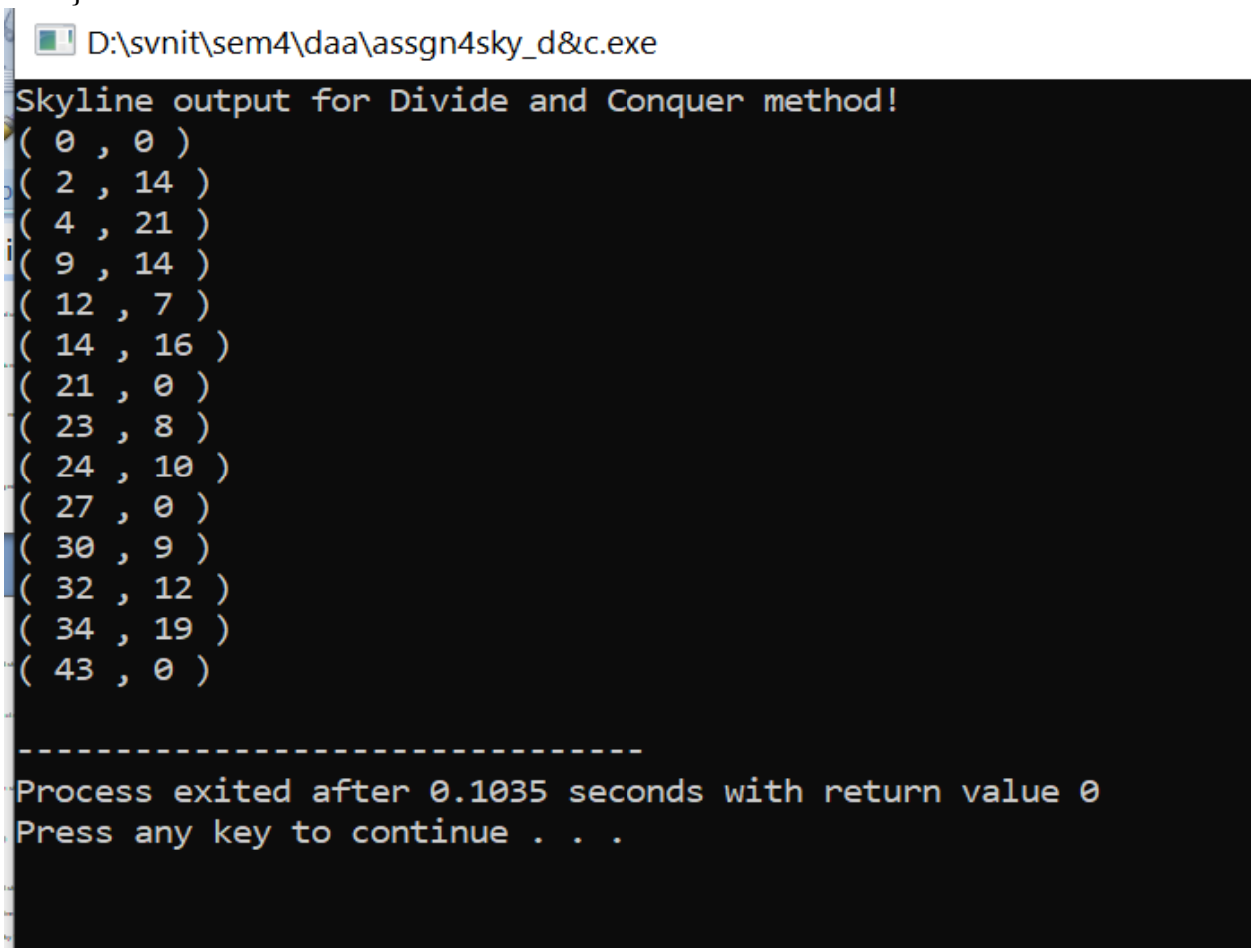
```
#include <bits/stdc++.h>
using namespace std;
struct input{
   int x;
   int y;
   int h;
};
struct sky{
   int x;
   int y;
};
vector<struct input> in{
                                                {31, 41, 5},
                                                {4, 9, 21},
                                                {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}
                                };
vector<struct sky> out;
vector<struct sky> mergesky(vector<sky> sky_l,vector<struct sky> sky_h);
vector<struct sky> skyline(int l, int h,vector<struct input> sky1);

int main()
{
   out=skyline(0,in.size()-1,in);
   out.insert(out.begin(),{0,0});
   int i;
   cout<<"Skyline output for Divide and Conquer method!\n";
   for(i=0;i<out.size();i++)
   {
            if(out[i].x!=-1)                        //we made it to remove redundant pts
            {
                     cout<<"( "<<out[i].x<<" , "<<out[i].y<<" ) ";
                     cout<<endl;
            }
   }
   return 0;
}
```

```
vector<struct sky> merge(vector<struct sky> sky_l,vector<struct sky> sky_h)
{
  int h_l=0, h_h=0;
  int ind=0;
  int i=0,j=0;
  vector<struct sky> merged_sky;
  while(i < sky_l.size() && j<sky_h.size())
  {
        if(sky_l.empty() || sky_h.empty())
        {
              break;
        }
        struct sky temp;
        if(sky_l[i].x < sky_h[j].x)
        {
              temp.x=sky_l[i].x;
              temp.y=sky_l[i].y;
        //if the height of selected coordinate is less than previous height of other skyline
then update height of the coordinate of point to be inserted
              if(sky_l[i].y < h_h)
              {
                    temp.y=h_h;
              }
              h_l=sky_l[i].y; //changing h_l which is previous height of sky_l vector
              i++;
        }
        else if(sky_l[i].x > sky_h[j].x) //similar with sky_h vector
        {
              temp.x=sky_h[j].x;
              temp.y=sky_h[j].y;
              if(sky_h[j].y < h_l)
              {
                    temp.y=h_l;
              }
              h_h = sky_h[j].y;
              j++;
        }
        else                                    //if both are equal then point would be the x
coordinate of either one y would be max height
        {
              temp.x = sky_h[j].x;
              temp.y = max(sky_l[i].y,sky_h[j].y);
              h_l = sky_l[i].y;
              h_h = sky_h[j].y;
              i++;
              j++;
```

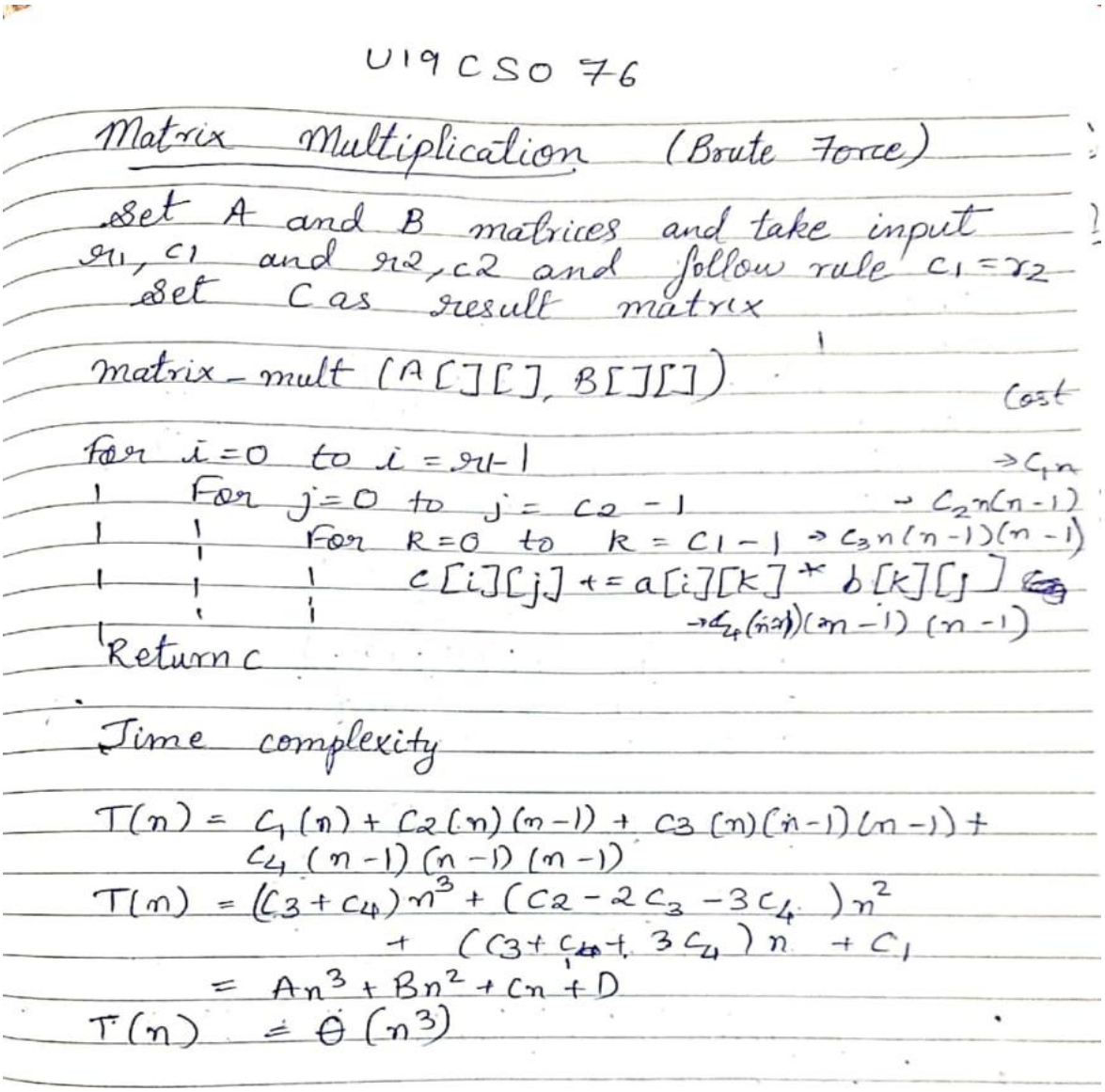
```
            }
            merged_sky.push_back(temp);
    }
    if(i>= sky_l.size()) //adding the remaining points of sky_l
    {
            while(j< sky_h.size())
            {
                    merged_sky.push_back(sky_h[j]);
                    j++;
            }
    }
    if(j>= sky_h.size()) //adding the remaining points of sky_h
    {
            while(i< sky_l.size())
            {
                    merged_sky.push_back(sky_l[i]);
                    i++;
            }
    }
    int inx = 1;
    vector<int> redundant;
    redundant.push_back(0);
    while(inx < merged_sky.size())
    {
            if(merged_sky[inx].y == merged_sky[inx-1].y)
            {
                    redundant.push_back(1);
            }
            else
            {
            redundant.push_back(0);
            }
            inx++;
    }
    for(i=0;i<redundant.size();i++)
    {
            if(redundant[i]==1)
            {
                    merged_sky[i].x=-1;
            }
    }
    return merged_sky;
}
vector<struct sky> skyline(int l,int h,vector<struct input> sky1)
{
    vector<struct sky> sky_line;
```

```
    if(l > h)
    {
        return sky_line;
    }
    else if(l==h)
    {
        struct sky p={sky1[l].x,sky1[l].h};
        sky_line.push_back(p);
        p.x=sky1[l].y;
        p.y=0;
        sky_line.push_back(p);
        return sky_line;
    }
    else
    {
        int mid=l+((h-l)/2);
        vector<struct sky> sky_l=skyline(l,mid,sky1);
        vector<struct sky> sky_h=skyline(mid+1,h,sky1);
        return merge(sky_l,sky_h);
    }
}
```



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

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

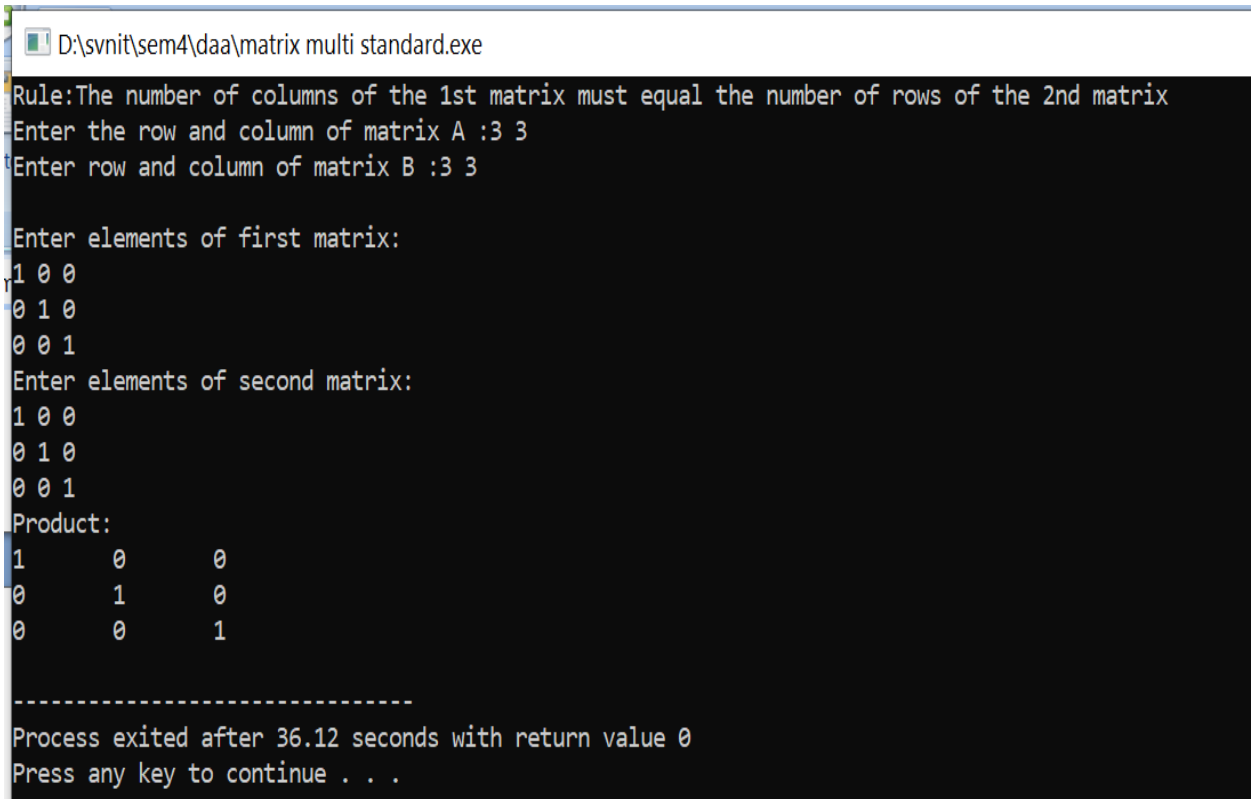


2.2. Write a program using an incremental (conventional) approach to multiply the given matrices.

```
#include <bits/stdc++.h>
using namespace std;
int ** matrix_allocate(int n)
{
        int ** mat = new int *[n];
        for (int i = 0; i < n; ++i)
```

```
        {
           mat[i] = new int[n];
        }
        for(int i=0;i<n;i++)
        {
                for(int j=0;j<n;j++)
                {
                        mat[i][j]=0;
                }
        }
        return mat;
}
int main() {
        int r1,c1,r2,c2;
        cout<<"Rule:The number of columns of the 1st matrix must equal the number of rows of
the 2nd matrix \n";
        cout<<"Enter the row and column of matrix A :";
        cin>>r1>>c1;
        cout<<"Enter row and column of matrix B :";
        cin>>r2>>c2;
        if(c1!=r2)
        {
          cout<<"Error!Refer Rule";
          return 0;}
        int n = max(r1,max(c1,max(r2,c2)));
        if(ceil(log(n))!=floor(log(n)))
        {
                n=pow(2,ceil(log2(n)));
        }
          int ** a = matrix_allocate(n);
          int ** b = matrix_allocate(n);
          int ** c = matrix_allocate(n);
          cout<<"\nEnter elements of first matrix: ";
        for (int i = 0; i < r1; i++)
        {
           for (int j = 0; j < c1; j++)
                {
              cin>>a[i][j];
           }
        }
        cout<<"Enter elements of second matrix: ";
        for (int i = 0; i < r2; i++)
        {
           for (int j = 0; j < c2; j++)
                {
              cin>>b[i][j];
```

```
            }
        }
        for (int i = 0; i < r1; i++)
   {
            for (int j = 0; j < c2; j++)
        {
                c[i][j] = 0;
                for (int k = 0; k < c1; k++)
                        {
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
          }
           cout<<"Product:\n";
        for(int i=0;i<r1;i++)
        {
                for(int j=0;j<c2;j++)
                  {
                        cout<<c[i][j]<<"\t";
                  }
                cout<<"\n";
        }
   return 0;
}
```



2.3. Write a pseudocode to multiply the given matrices using the divide and con quer approach Analyze the time

U19CS076

Matrix multiplication (Divide & Conquer)
We use strassen matrix mult technique.

```
Matrix_mult (A[][], B[][], n)
Initialize matrix C
IF n = 1
|  C[0][0] = a[0][0] * b[0][0]
Set a11[][], a12[][], a21[][], a22[][]          } θ(1)
|  b11[][], b12[][], b21[][], b22[][]
|  of row, col size n/2
FOR I = 0 to (N/2)-1
|  FOR J = 0 to J = (N/2)-1
|  |  a[i][j] = a[i][j]
|  |  a12[i][j] = a[i][j+n/2]
|  |  a21[i][j] = a[i+n/2][j]
|  |  a22[i][j] = a[i+n/2][j+n/2]
|  |
|  |  b11[i][j] = b[i][j]
|  |  b12[i][j] = b[i][j+n/2]
|  |  b21[i][j] = b[i+n/2][j]
|  |  b22[i][j] = b[i+n/2][j]
|                                              θ(n^2)
Initialize P1[][], P2[][], P3[][], P4[][], P5[][],
|  P6[][], P7[][],
   C11[][], C12[][], C21[][], C22[][]
|
P1 := Matrix_mult(a11, matrix_sub(b12, b22, n/2), n/2)
P2 = Matrix_mult(matrix_add(a11, a12, n/2), b22, n/2)
P3 = Matrix_mult(matrix_add(a22, a22, n/2), b11, n/2)
P4 = matrix_mult(a22, matrix_sub(b21, b11, n/2), n/2)
P5 = matrix_mult(matrix_add(a11, a22, n/2), matrix_add(b11, b22, n/2), n/2
P6 = matrix_mult(matrix_sub(a12, a22, n/2), matrix_add(b21, b22, n/2), n/2
```

T(n/2) x 7

U19CS076

```
P7 = matrix_mult(matrix_sub(a11,a21,n/2), matrix_add(b11,b12,n/2), n/2)

θ(n²) {
  C11 = matrix_sub(matrix_add(matrix_add(P5,P4,n/2),P6,n/2),P2,n/2)
  C12 = matrix_add(P1, P2, n/2)
  C21 = matrix_add(P3, P4, n/2)
  C22 = matrix_sub(matrix_sub(matrix_add(P5,P1,n/2),P3,n/2),P7,n/2)
}

For i = 0 to n/2                                          } θ(n²)
    For j = 0 to n/2                                      }
        C[i][j] = C11[i][j]                               }
        C[i][j+n/2] = C12[i][j]                           }
        C[i+n/2][j] = C21[i][j]                           }
        C[i+n/2][j+n/2] = C22[i][j]                       } → θ(1)
Return C
```

The functions of add, subtract, ~~join and s~~ have been declared. They have complexity of $\theta(n^2)$

$$T(n) = 7\,T(n/2) + \theta(n^2)$$

By Master method

$$n^{\log_b a} = n^{\log_2 7} \quad f(n) = cn^2$$

$$n^{\log_2 7} = n^{2.807} > f(n)$$

$$T(n) = \theta(n^{2.807})$$

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

```
#include <bits/stdc++.h>
using namespace std;
int **matrix_allocate(int n)
{
        int ** mat = new int *[n];
        int i,j;
        for (i = 0; i < n; ++i)
        {
           mat[i] = new int[n];
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        mat[i][j]=0;
                }
        }
        return mat;
}
int max(int a,int b)
{
        if(a>=b)
        {
                return a;
        }
        else
        {
                return b;
        }
}
void matrix_free(int ** m, int n)
{
  int i;
  for (i = 0; i < n; i++)
  {
    free(m[i]);
  }
  free(m);
}
int **matrix_add(int ** a, int ** b, int n)
{
  int i, j;
  int ** c = matrix_allocate(n);
  for (i = 0; i < n; i++)
```

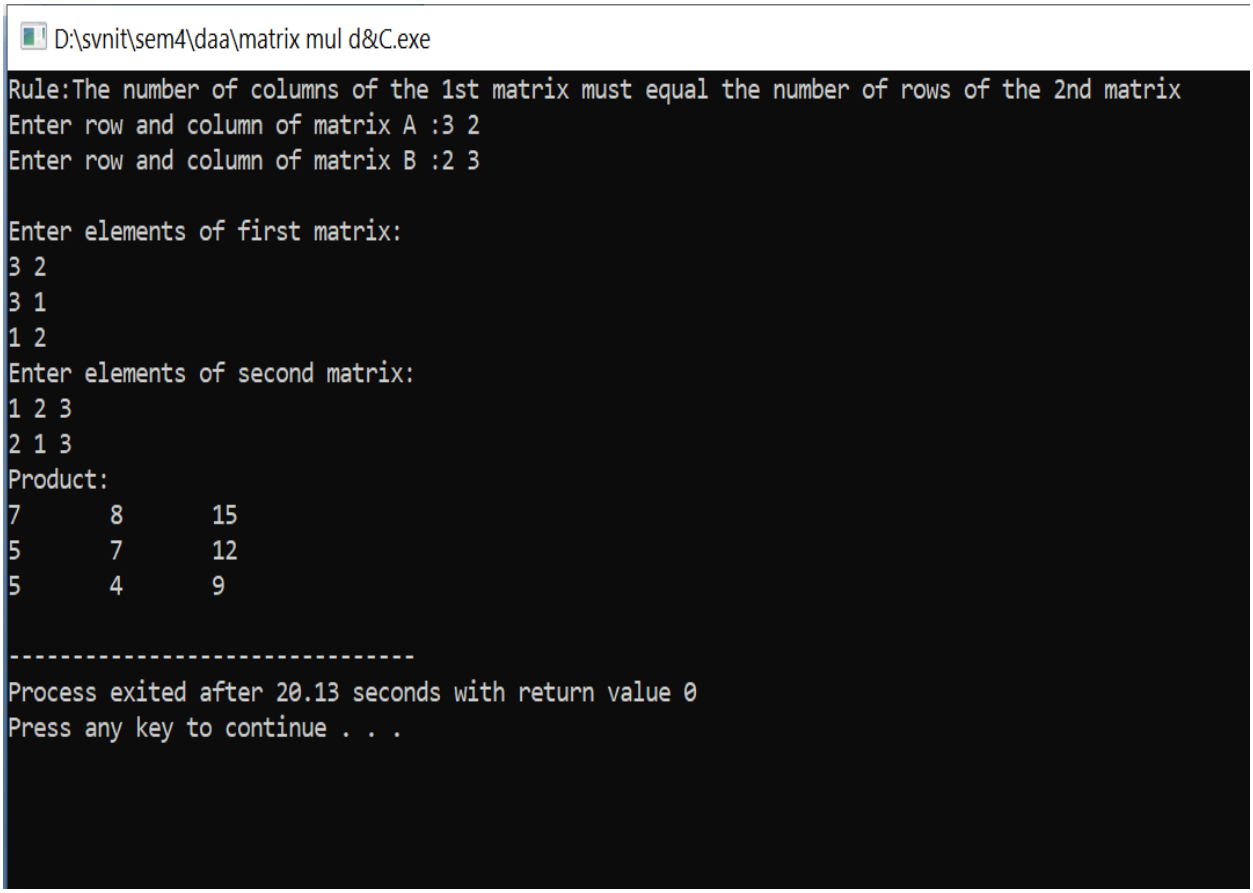
```
            {
            for (j = 0; j < n; j++)
                    {
             c[i][j] = a[i][j] + b[i][j];
            }
            }
  return c;
}
int **matrix_sub(int ** a, int ** b, int n)
{
  int i, j;
  int ** c = matrix_allocate(n);
  for (i = 0; i < n; i++)
  {
    for (j = 0; j < n; j++)
            {
         c[i][j] = a[i][j] - b[i][j];
    }
  }
  return c;
}
int **mul_strassen(int ** a, int ** b, int n)
{
  int i, j;
  int **c = matrix_allocate(n);
  if (n == 1)
  {
     c[0][0] = a[0][0] * b[0][0];
  }
  else
  {
     int ** a11 = matrix_allocate(n / 2);
     int ** a12 = matrix_allocate(n / 2);
     int ** a21 = matrix_allocate(n / 2);
     int ** a22 = matrix_allocate(n / 2);
     int ** b11 = matrix_allocate(n / 2);
     int ** b12 = matrix_allocate(n / 2);
     int ** b21 = matrix_allocate(n / 2);
     int ** b22 = matrix_allocate(n / 2);
     for (i = 0; i < n / 2; i++)
             {
        for (j = 0; j < n / 2; j++)
                    {
            a11[i][j] = a[i][j];
            a12[i][j] = a[i][j + n / 2];
            a21[i][j] = a[i + n / 2][j];
```

```
            a22[i][j] = a[i + n / 2][j + n / 2];
            b11[i][j] = b[i][j];
            b12[i][j] = b[i][j + n / 2];
            b21[i][j] = b[i + n / 2][j];
            b22[i][j] = b[i + n / 2][j + n / 2];
        }
    }
    int** P1 = mul_strassen(a11, matrix_sub(b12, b22, n/2), n/2);
    int** P2 = mul_strassen(matrix_add(a11, a12, n/2), b22, n/2);
    int** P3 = mul_strassen(matrix_add(a21, a22, n/2), b11, n/2);
    int** P4 = mul_strassen(a22, matrix_sub(b21, b11, n/2), n/2);
    int** P5 = mul_strassen(matrix_add(a11, a22, n/2), matrix_add(b11, b22, n/2), n/2);
    int** P6 = mul_strassen(matrix_sub(a12, a22, n/2), matrix_add(b21, b22, n/2), n/2);
    int** P7 = mul_strassen(matrix_sub(a11, a21, n/2), matrix_add(b11, b12, n/2), n/2);

    int** c11 = matrix_sub(matrix_add(matrix_add(P5, P4, n/2), P6, n/2), P2, n/2);
    int** c12 = matrix_add(P1, P2, n/2);
    int** c21 = matrix_add(P3, P4, n/2);
    int** c22 = matrix_sub(matrix_sub(matrix_add(P5, P1, n/2), P3, n/2), P7, n/2);

    for (i = 0; i < n / 2; i++)
          {
      for (j = 0; j < n / 2; j++)
                {
            c[i][j] = c11[i][j];
            c[i][j + n / 2] = c12[i][j];
            c[i + n / 2][j] = c21[i][j];
            c[i + n / 2][j + n / 2] = c22[i][j];
        }
    }
    matrix_free(c11, n / 2);
    matrix_free(c12, n / 2);
    matrix_free(c21, n / 2);
    matrix_free(c22, n / 2);
   }
    return c;
}
int main()
{
  int i, j;
  int r1,c1,r2,c2;
cout<<"Rule:The number of columns of the 1st matrix must equal the number of rows of the 2nd
matrix \n";
  cout<<"Enter row and column of matrix A :";
  cin>>r1>>c1;
  cout<<"Enter row and column of matrix B :";
```

```
 cin>>r2>>c2;
 if(c1!=r2)
        {
                cout<<"Error!Refer Rule";
                return 0;
        }
        int n=max(r1,max(c1,max(r2,c2)));
        if(ceil(log(n))!=floor(log(n)))
        {
                n=pow(2,ceil(log2(n)));
        }
          int ** a = matrix_allocate(n);
          int ** b = matrix_allocate(n);
          cout<<"\nEnter elements of first matrix: ";
        for (i = 0; i < r1; i++)
        {
        for (j = 0; j < c1; j++)
                {
         cin>>a[i][j];
        }
        }
        printf("Enter elements of second matrix: ");
        for (i = 0; i < r2; i++)
        {
        for (j = 0; j < c2; j++)
                {
      cin>>b[i][j];
        }
        }
        int **c=mul_strassen(a, b, n);
        cout<<"Product:\n";
        for(i=0;i<r1;i++)
        {
                for(j=0;j<c2;j++)
                {
                        cout<<c[i][j]<<"\t";
                }
        cout<<"\n";
        }
        return 0;
}
```


D:\svnit\sem4\daa\matrix mul d&C.exe
Rule:The number of columns of the 1st matrix must equal the number of rows of the 2nd matrix
Enter row and column of matrix A :3 2
Enter row and column of matrix B :2 3
Enter elements of first matrix:
3 2
3 1
1 2
Enter elements of second matrix:
1 2 3
2 1 3
Product:
7 8 15
5 7 12
5 4 9
Process exited after 20.13 seconds with return value 0
Press any key to continue . . .