Gatla Amulya Reddy.
U20CS103.

Tutorial-5

1.1 Skyline problem:

Incremental approach:

Pseudo Code:

Consider three arrays L[n], R[n], H[n].

L[n] is left most bound of array.

R[n] is right most bound of building.

H[n] is height of building.

Lets define size as max. of array R[n]

```
Defining Ans[size] = {0}
for(int i=0 to i=n)
    for(int j=0; j< R[i]-L[i], j++)
        if(Ans[L[i]+j] < H[i])
            Ans[L[i]+j] = H[i]
set prev = -1
for i from 0 to size
    if(Ans[i] != prev)
        print[i+1, Ans[i])
        prev = Ans[i]
```

Time Complexity Analysis:

For each input we are checking its width.

$\therefore$ Time Complexity = $\sum$ width of towers

$= O(\text{size} \times n)$

$= O(n^2)$.

U20CS108

1.2 <u>Using Divide and Conquer approach</u>:

Defining a function skyline Sort which takes array of L, R, H as input and returns skyline.

```
Skyline Sort ( arr[][], low, high):
    if low = high
        Skyline.add [arr[low][0], arr[low][2]]
        Skyline.add (arr[low][1], 0)
        return Skyline
    if high < low
        return Skyline  //empty
    mid = (low + high)/2
    Sky 1 = Skyline Sort (arr, low, mid)
    Sky 2 = skyline Sort (arr, mid+1, high)
    final Skyline = Merge Sky (sky1, sky2)

        return finalSkyline
```

Defining MergeSky which merges two skylines

```
MergeSky ( Sky 1 [][], sky 2 [][])
    Set h1=0, h2=0, i=0, j=0, k=0
    While i< sky1.length and j< sky2.length
        if Sky 1 [i][0] < Sky[j][0]
            h1 = Sky1[i][1]
            result.add ( sky[1][0], max(h1, sky[1][1])
            i = i+1
```

U20CS103

```
else if  sky1[i][0] = sky2[j][0]
        h1 = sky1[i][1]
        h2 = sky2[j][1]
        result.add (sky1[i][0], max(h1, h2))
        i=i+1,  j=j+1

else
    h2 = sky2[j][1]
    result.add (sky2[j][0], max(h1; sky2[j][1])
    j=j+1

While i< sky1.length
    result.add [sky1[i][0], sky1[i][1])
    i=i+1

While j< sky2.length
    result.add (sky2[j][0], sky2[j][1])
    j=j+1

For i=0 till i< result.length
    top = result[i][1]
    answer.add (result[i][0], result[i][1])
    While i< result.length and top = result[i][1]
        i= i+1

return answer.
```

MergeSky function has no nested loops.
So, it's complexity is $O(n)$.

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

Let $O(n) = C\cdot n$ $\quad$ C is a constant

$$T(n/2) = 2T(n/4) + Cn/2$$

$$\Rightarrow T(n) = 4T(n/4) + Cn + Cn$$

Similarly $T(n) = 2^n\, T(n/2^n) + hCn$

$$\Rightarrow 2^h = n \Rightarrow h = \log_2 n$$

$$\therefore T(n) = nT(1) + \log_2 n\, Cn$$

$$\boxed{T(n) = O(n\log n)}$$

2.1 <u>Matrix multiplication</u>:

Using Incremental approach:

Consider two Matrices mat1[ ][ ], mat2[ ][ ] of R_1, C_1 and R_2, C_2 row and column respectively.

<u>Pseudo code</u>:

```
Declare Result[ ][ ] of R1, C2
for ( i=0; i< R1; i++)
  {
    for (j=0; j< C2; j++)
      {
        for (K=0; k< R2; K++)
```

```
        {
          Result[i][j] += mat1[i][k] * mat2[k][j] ;
        }
      }
    }
```

Print Result

Time Complexity Analysis :

We are iterating for R_1 rows for each row.
we are iterating C_2 columns and for each column.
we are iterating R_2 rows.

$$So,\ T(n) = O(R_1 * C_2 * R_2) = O(n^3).$$

2.2 Using Divide and Conquer approach :

Assumption : Both are square matrix of power 2.

Suppose we partition A, B and C.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C = A \cdot B$$

$$\Rightarrow C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

Defining a function

Matrix Recursion (A, B) :

$n = A.rows$

let C be a new matrix $(n \times n)$

if $n = 1$

$c_{11} = a_{11} \cdot b_{11}$

else

C_{11} = Matrix Recursion (A_{11}, B_{11}) + Matrix Recursion (A_{12}, B_{21})

C_{12} = Matrix Recursion (A_{11}, B_{12}) + Matrix Recursion (A_{12}, B_{22})

C_{21} = Matrix Recursion (A_{21}, B_{11}) + Matrix Recursion (A_{22}, B_{21})

C_{22} = Matrix Recursion (A_{21}, B_{12}) + Matrix Recursion (A_{21}, B_{22})

return C.

Time Complexity Analysis:

For multiplying 2 matrices of $n \times n$, we made 8 recursive calls of subproblems of $(n/2 \times n/2)$

we add two matrices that take $O(n^2/4)$

$\therefore$ Recurrance relation

$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 8T(n/2) + \theta(n^2) & \text{if } n > 1 \end{cases}$$

By Master's Theorem,

$$T(n) \leq a\,T(n/b) + O(n^d)$$

$$a = 8,\ b = 2,\ d = 2$$

as $a > b^d$

It's case 3,

$$\therefore T(n) = O(n^{\log_b a}) = O(n^{(\log_2 8)}) = O(n^3).$$