NAME :- SAHIL PRAVIN THAKUR

PRN : - 2020BTECS00042

BATCH : - T7

SUB :- DAA LAB

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7

Assignment:- 3

Q1) Implement algorithm to Find the maximum element in an array which is first increasing and then decreasing, with Time Complexity *O(Logn)*.

Algorithm: -

Step1:- start with l = 0 and r = n-1

Step2:- find mid element each time and since it is first increasing and then decresing max element is present if both its left side and right value is greater than mid value

Step3:- If it is not max them

If arr[mid]>arr[mid+1] discard right part and sech on left part

Else discard left part and search om right part

```
#include<iostream>
using namespace std;
int getMax(int arr[],int n)
{
    int l = 0, r = n - 1;
    int ans = 0 ;
    while(l<=r)
    {   if(l==r)return arr[l];
        int mid = (l + r) / 2;
        if(arr[mid]>arr[mid+1]&&arr[mid]>arr[mid-1])
            return arr[mid];

        else if(arr[mid]>arr[mid+1])
            r = mid - 1;
            else
                l = mid + 1;
    }

    return 0;
}
int main()
{
    int arr[] = { 10, 20, 80, 100, 200, 400, 500, 3, 2, 1};
    int n = sizeof(arr) / sizeof(int);
    cout<<"Maximum value in array = "<<getMax(arr,n) ;
    return 0 ;
}
```

Explanation :-

since array monotonically inceases in first half and then decreses we can apply binery search with some modification shown in fig .

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7

TimeComplexity :-

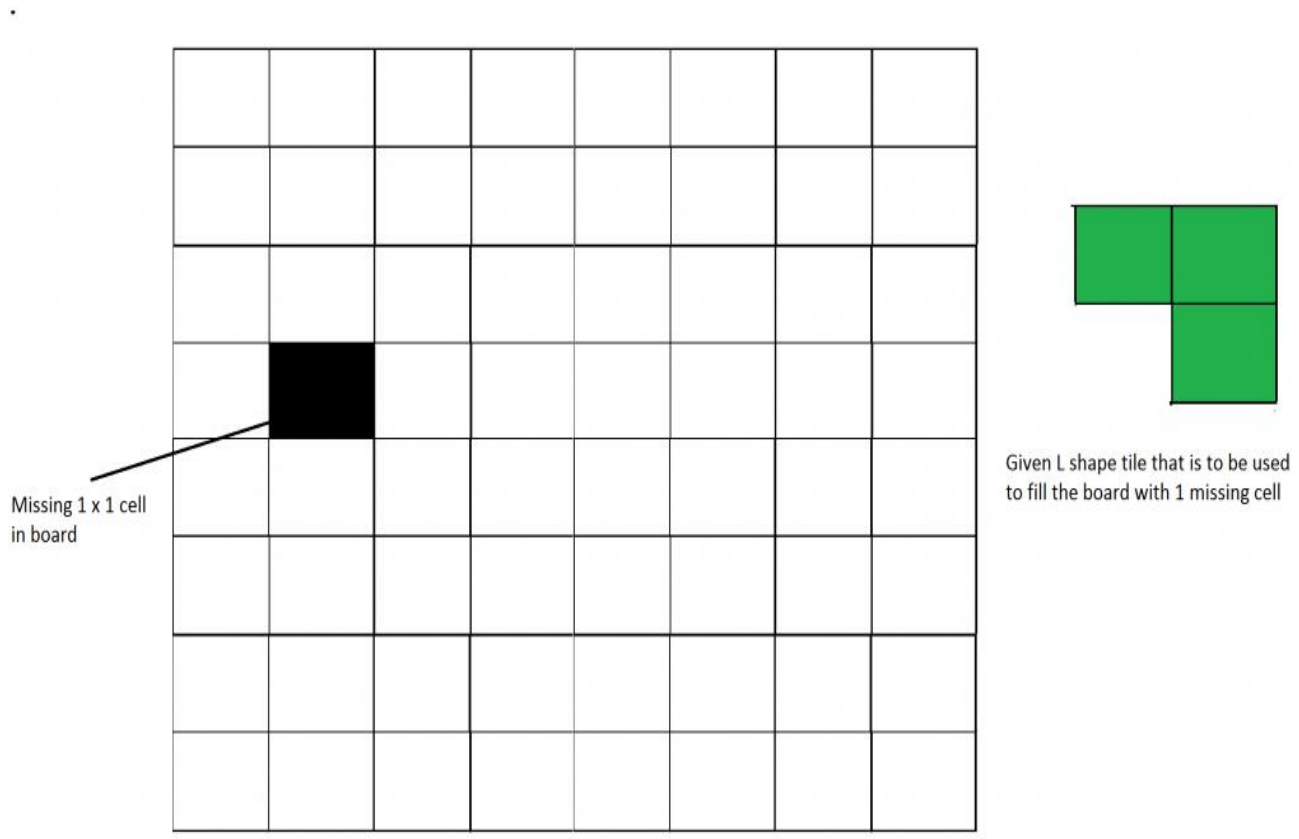
Since here binery search is used it’s time complexity O(NlogN)

Output:-

```
PS D:\sem5\DAA Lab\Assignmet3> cd "d:\sem5\
Maximum value in array = 500
PS D:\sem5\DAA Lab\Assignmet3>
```

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7

Q2) Implement algorithm for Tiling problem: Given an *n by n* board where n is of form 2^k where *k >= 1* (Basically n is a power of *2* with minimum value as *2*). The board has one missing cell (of size *1 x 1*). Fill the board using L shaped tiles. An *L* shaped tile is a *2 x 2* square with one cell of size *1×1* missing



Algorithm:-

// n is size of given square, p is location of missing cell

Tile(int n, Point p)

Step1 :-

Base case: n = 2, A 2 x 2 square with one cell missing is nothing but a tile and can be filled with a single tile.

Step2:-

Place a L shaped tile at the center such that it does not coverthe n/2 * n/2 subsquare that has a missing square. **Now all four**

subsquares of size n/2 x n/2 have a missing cell (a cell that doesn't need to be filled). See figure 2 below.

Step3:-

Solve the problem recursively for following four. Let p1, p2, p3 and p4 be positions of the 4 missing cells in 4 squares.

a) Tile(n/2, p1)
b) Tile(n/2, p2)
c) Tile(n/2, p3)
d) Tile(n/2, p3)

```
#include <iostream>
using namespace std;
int arr[1001][1001];
```

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7

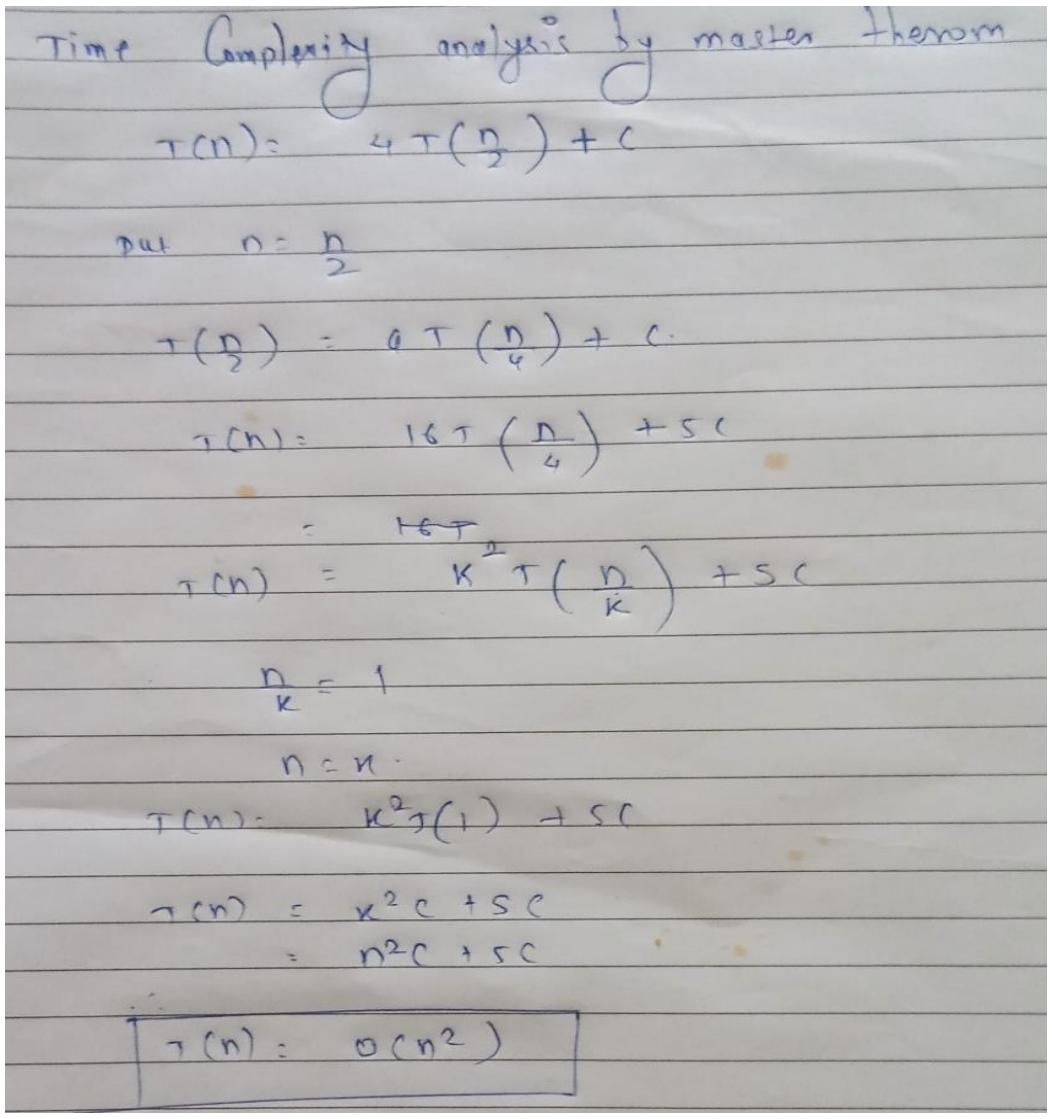
```
int cnt = 0;
int placeLShape(int x1, int y1, int x2, int y2, int x3, int y3)
{   cnt++;
    arr[x1][y1] = cnt;
    arr[x2][y2] = cnt;
    arr[x3][y3] = cnt;
}
void completeGrid(int x, int y, int n)
{
    int dx, dy;
    if (n == 2)
    {
        cnt++;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (arr[x + i][y + j] == 0)
                {
                    arr[x + i][y + j] = cnt;
                }
            }
        }
        return ;
    }
    for (int i = x; i < x + n; i++)
    {
        for (int j = y; j < y + n; j++)
        {
            if (arr[i][j] != 0)
            {
                dx = i, dy = j;
                break;
            }
        }
    }
    if (dx < (x + n / 2) && dy < (y + n / 2)) // first Q
    {                                        //     2                    //4
        //3
        placeLShape(x + n / 2 - 1, y + n / 2, x + n / 2, y + n / 2, x + n / 2, y + n
/ 2 - 1);
    }
    else if (dx < (x + n / 2) && dy >= (y + n / 2)) // second Q
    {
        placeLShape(x + n / 2 - 1, y + n / 2 - 1, x + n / 2, y + n / 2, x + n / 2, y
+ n / 2 - 1);
    }
    else if (dx >= (x + n / 2) && dy < (y + n / 2)) //  third Q
    {
        placeLShape(x + n / 2 - 1, y + n / 2, x + n / 2, y + n / 2, x + n / 2 - 1, y
+ n / 2 - 1);
```

```
    }
    else if (dx >= (x + n / 2) && dy >= (y + n / 2)) // fourth Q
    {
        placeLShape(x + n / 2 - 1, y + n / 2, x + n / 2, y + n / 2, x + n / 2 - 1, y
+ n / 2 - 1);
    }

    completeGrid(x, y, n / 2);
    completeGrid(x + n / 2, y, n / 2);
    completeGrid(x, y + n / 2, n / 2);
    completeGrid(x + n / 2, y + n / 2, n / 2);
}
int main()
{
    int n = 8;          // size of grid
    int a = 3, b = 1; // cell that don't require to be fill
    arr[a][b] = -1;
    completeGrid(0, 0, n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Time Complexity analysis :-

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7



Output:-

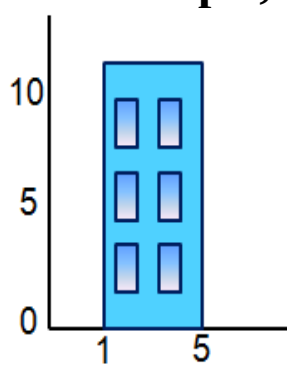
```
PS D:\sem5\DAA Lab\Assignmet3> cd "d:
3 3 5 5 13 13 15 15
3 2 2 5 13 12 12 15
4 4 2 6 14 14 12 16
4 -1 6 6 1 14 16 16
8 8 10 1 1 18 20 20
8 7 10 10 18 18 17 20
9 7 7 11 19 17 17 21
9 9 11 11 19 19 21 21
PS D:\sem5\DAA Lab\Assignmet3>
```

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7

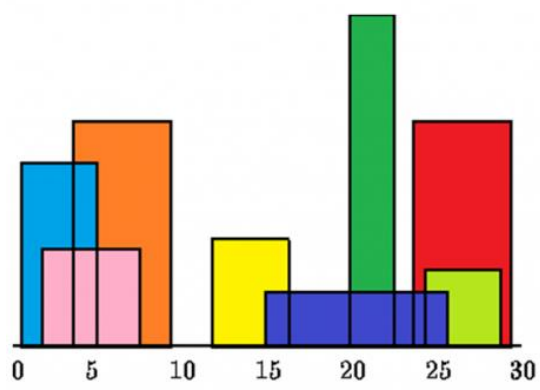
Q3) Implement algorithm for The Skyline Problem: Given n rectangular buildings in a 2-dimensional city, computes the skyline of these buildings, eliminating hidden lines. The main task is to view buildings from a side and remove all sections that are not visible.

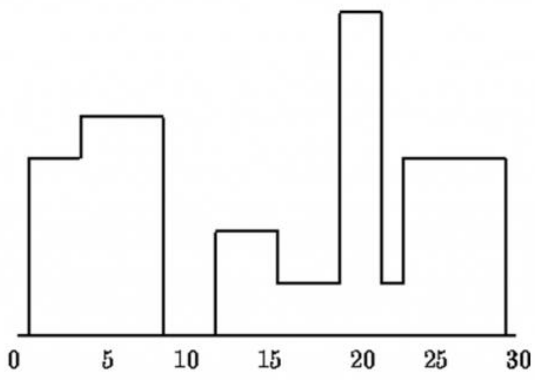
All buildings share common bottom and every building **is represented by triplet (left, ht, right)**
'left': is x coordinated of left side (or wall).
'right': is x coordinate of right side
'ht': is height of building.

For example, the building on right side is represented as *(1, 11, 5)*



A skyline is a collection of rectangular strips. A rectangular strip is represented as a pair (left, ht) where left is x coordinate of left side of strip and ht is height of strip.





With Time Complexity *O(nLogn)*

Algorithm:-

Step1:-

Insert start ,height and start,end pair in array .

Step2:-

Than scan array and if current pair is (start,height) add height to max heap and

If max heap top value changes than consider this pair and add of skyline points

Step3:-

If current pair is (end,height) then remove height value form max heap and if max heap top value changes than add it to final list of skyline

Step2:-

Code:-

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7

```
#include <iostream>
using namespace std;

struct Building {
 int left;

 int ht;

 int right;
};

class Strip {
 int left;

 int ht;

public:
 Strip(int l = 0, int h = 0)
 {
  left = l;
  ht = h;
 }
 friend class SkyLine;
};

class SkyLine {
 Strip* arr;

 int capacity;

 int n;

public:
 ~SkyLine() { delete[] arr; }
 int count() { return n; }

 SkyLine* Merge(SkyLine* other);

SkyLine(int cap)
{
 capacity = cap;
 arr = new Strip[cap];
 n = 0;
}

void append(Strip* st)
{
  if (n > 0 && arr[n - 1].ht == st->ht)
  return;
 if (n > 0 && arr[n - 1].left == st->left) {
  arr[n - 1].ht = max(arr[n - 1].ht, st->ht);
```

```
   return;
  }

  arr[n] = *st;
  n++;
 }

  void print()
 {
  for (int i = 0; i < n; i++) {
   cout << " (" << arr[i].left << ", "
    << arr[i].ht << "), ";
  }
 }
};

SkyLine* findSkyline(Building arr[], int l, int h)
{
 if (l == h) {
  SkyLine* res = new SkyLine(2);
  res->append(
   new Strip(
    arr[l].left, arr[l].ht));
  res->append(
   new Strip(
    arr[l].right, 0));
  return res;
 }

 int mid = (l + h) / 2;

  SkyLine* sl = findSkyline(
  arr, l, mid);
 SkyLine* sr = findSkyline(
  arr, mid + 1, h);
 SkyLine* res = sl->Merge(sr);

 delete sl;
 delete sr;
 return res;
}

SkyLine* SkyLine::Merge(SkyLine* other)
{

 SkyLine* res = new SkyLine(
  this->n + other->n);

 int h1 = 0, h2 = 0;

 int i = 0, j = 0;
```

```
while (i < this->n && j < other->n) {
   if (this->arr[i].left < other->arr[j].left) {
  int x1 = this->arr[i].left;
  h1 = this->arr[i].ht;

  int maxh = max(h1, h2);

  res->append(new Strip(x1, maxh));
  i++;
 }
 else {
  int x2 = other->arr[j].left;
  h2 = other->arr[j].ht;
  int maxh = max(h1, h2);
  res->append(new Strip(x2, maxh));
  j++;
  }
 }


 while (i < this->n) {
  res->append(&arr[i]);
  i++;
 }
 while (j < other->n) {
  res->append(&other->arr[j]);
  j++;
 }
 return res;
}

int main()
{
 Building arr[] = {
  { 1, 11, 5 }, { 2, 6, 7 }, { 3, 13, 9 }, { 12, 7, 16 }, { 14, 3, 25 }, { 19, 18,
22 }, { 23, 13, 29 }, { 24, 4, 28 }
 };
 int n = sizeof(arr) / sizeof(arr[0]);

 SkyLine* ptr = findSkyline(arr, 0, n - 1);
 cout << " Skyline for given buildings is " << endl
      << endl;
 ptr->print();
 return 0;
}
```

Explanation:-

The idea is similar to merge of merge sort, start from first strips of two skylines, compare x coordinates. Pick the strip with smaller x coordinate and add it to result. The height of added strip is considered as maximum of current heights from skyline1 and skyline2.

Time Complexity:-

Since we travers all n elements and each time we insert or delete from Max heap which take log(n) time so it's time complexity is O(nlogn);

Output:-

```
Skyline for given buildings is

(1, 11), (3, 13), (9, 0), (12, 7), (16, 3), (19, 18), (22, 3), (23, 13), (29, 0),
PS D:\sem5\DAA Lab\Assignment3>
```