Walchand College of Engineering, Sangli
Computer Science & Engineering
Third Year
Course: Design and analysis of algorithm Lab
Lab course coordinator:
Mrs A M Chimanna- Batch: - T1, T2, T3,T4

Week 4 Assignment

PRN : 21510111

Batch : T1

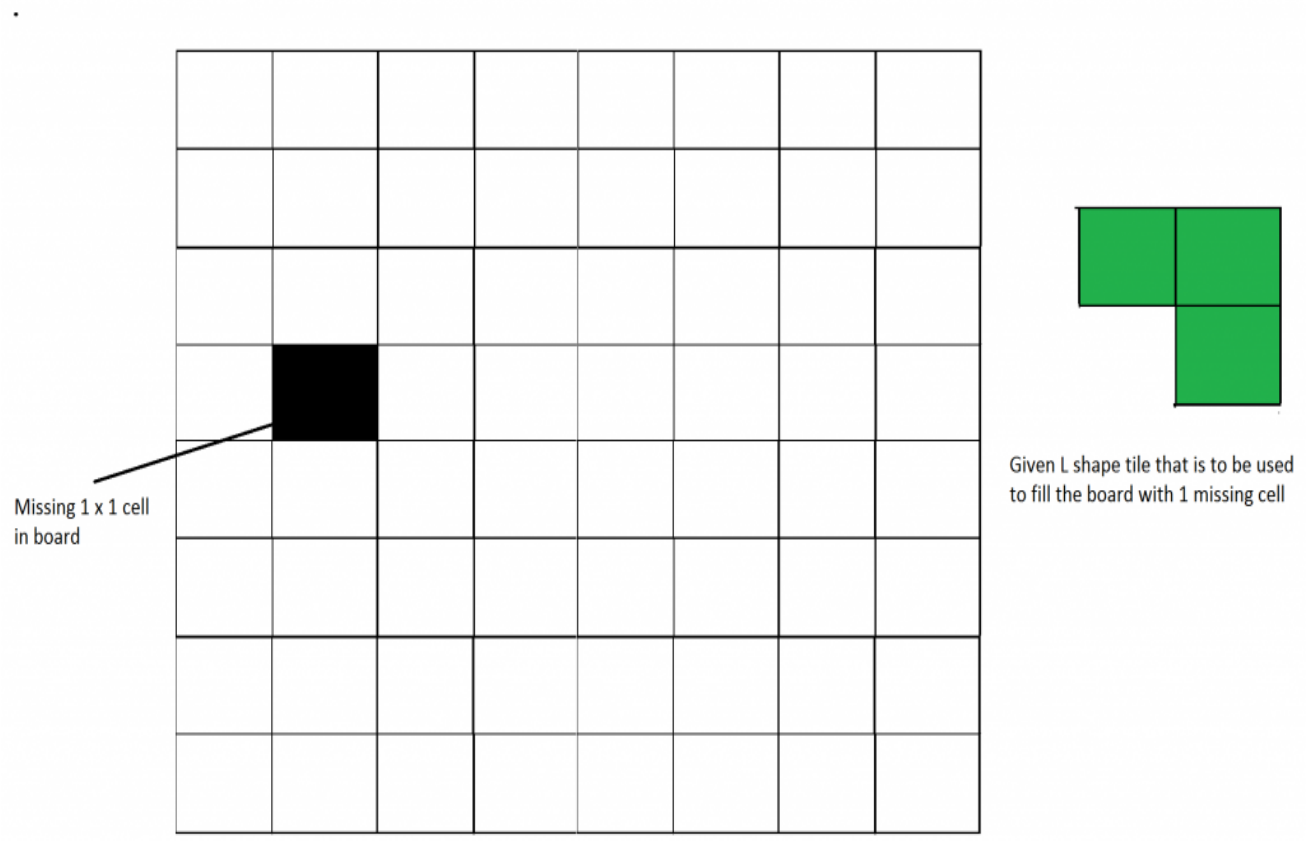
Divide and conquer strategy

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

```
#include <iostream>

using namespace std;


int maxInBitonic(int arr[], int low, int high)

{

    int n = high + 1;

    while (low <= high) {

        int mid = low + (high - low) / 2;

        if(arr[mid] > arr[mid+1] and arr[mid] > arr[mid-1])

            return arr[mid];

        else if (arr[mid] < arr[mid + 1])

            low = mid + 1;

        else

            high = mid - 1;

    }


    return arr[high];

}
```

```
int main()

{

    int arr[] = { 1, 3, 50, 10, 9, 7, 6 };

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "The maximum element is "<< maxInBitonic(arr, 0, n -
1)<<endl;

    return 0;

}
```

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




```
#include <bits/stdc++.h>

using namespace std;


int size_of_grid, b, a, cnt = 0;

int arr[128][128];
```

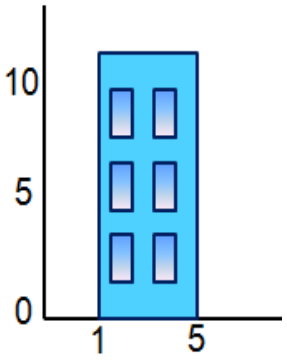
```
void place(int x1, int y1, int x2,
        int y2, int x3, int y3)
{
    cnt++;
    arr[x1][y1] = cnt;
    arr[x2][y2] = cnt;
    arr[x3][y3] = cnt;
}
int tile(int n, int x, int y)
{
    int r, c;
    if (n == 2) {
        cnt++;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (arr[x + i][y + j] == 0) {
                    arr[x + i][y + j] = cnt;
                }
            }
        }
        return 0;
    }
    for (int i = x; i < x + n; i++) {
        for (int j = y; j < y + n; j++) {
            if (arr[i][j] != 0)
                r = i, c = j;
        }
    }

    if (r < x + n / 2 && c < y + n / 2)
        place(x + n / 2, y + (n / 2) - 1, x + n / 2,
            y + n / 2, x + n / 2 - 1, y + n / 2);
```

```
    else if (r >= x + n / 2 && c < y + n / 2)
        place(x + (n / 2) - 1, y + (n / 2), x + (n / 2),
            y + n / 2, x + (n / 2) - 1, y + (n / 2) - 1);
    else if (r < x + n / 2 && c >= y + n / 2)
        place(x + n / 2, y + (n / 2) - 1, x + n / 2,
            y + n / 2, x + n / 2 - 1, y + n / 2 - 1);
    else if (r >= x + n / 2 && c >= y + n / 2)
        place(x + (n / 2) - 1, y + (n / 2), x + (n / 2),
            y + (n / 2) - 1, x + (n / 2) - 1,
            y + (n / 2) - 1);
    tile(n / 2, x, y + n / 2);
    tile(n / 2, x, y);
    tile(n / 2, x + n / 2, y);
    tile(n / 2, x + n / 2, y + n / 2);

    return 0;
}
int main()
{
    size_of_grid = 8;
    memset(arr, 0, sizeof(arr));
    a = 0, b = 0;
    arr[a][b] = -1;
    tile(size_of_grid, 0, 0);
    for (int i = 0; i < size_of_grid; i++) {
        for (int j = 0; j < size_of_grid; j++)
            cout << arr[i][j] << " \t";
        cout << "\n";
    }
}
```

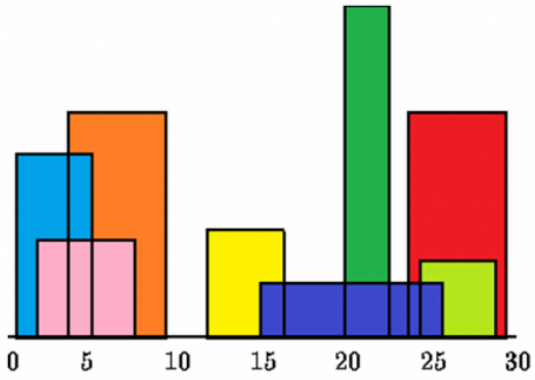
Q3) Implement algorithm for The Skyline Problem: Given n rectangular buildings in a 2-dimensional city, computes the skyline of these buildings, eliminating hidden lines. The main task is to view buildings from a side and remove all sections that are not visible.

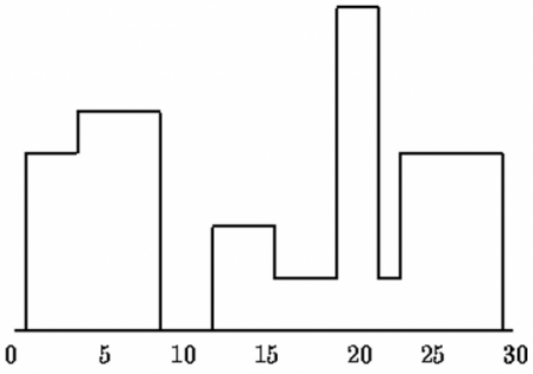
All buildings share common bottom and every **building** is represented by triplet (left, ht, right)
'left': is x coordinated of left side (or wall).
'right': is x coordinate of right side
'ht': is height of building.

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)*

```
#include <bits/stdc++.h>

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 \n";
    ptr->print();
    return 0;
}
```