ASSIGNMENT 3

Title of assignment: Divide and Conquer Strategy

Name: Mayur Savale

PRN: 21520010

1. Implement algorithm to find the maximum element in an array which is first increasing and then decreasing, with Time Complexity O(logn).

Ans:

a) **Algorithm: (Pseudocode)**

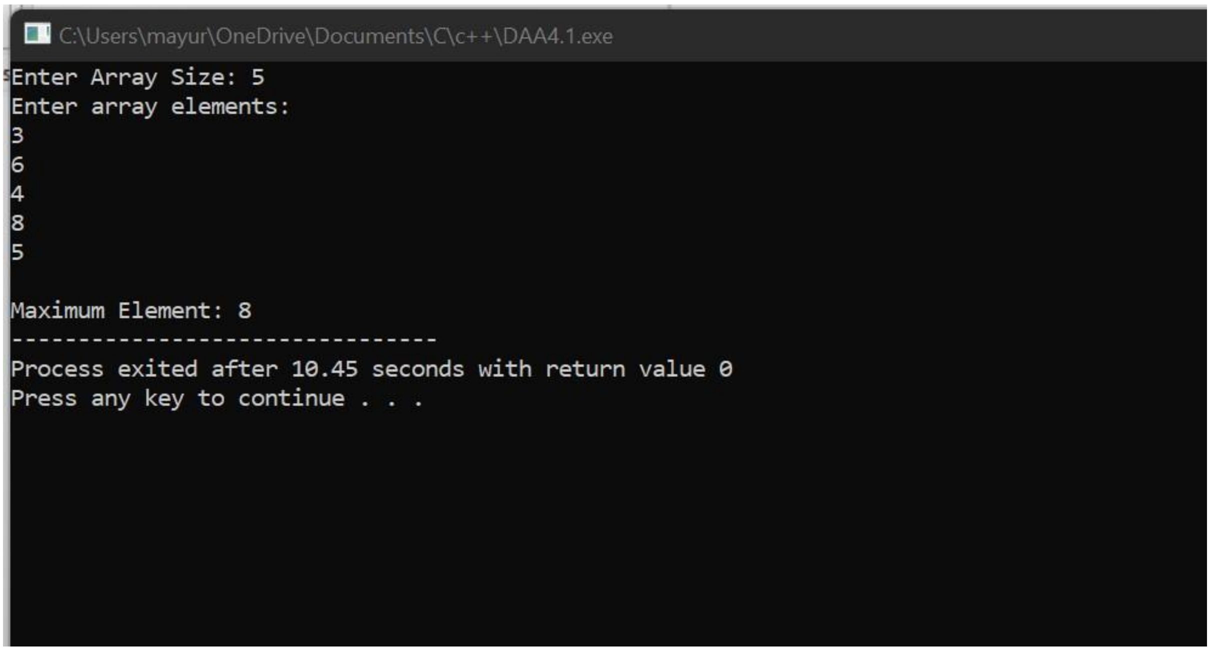
- We use binary search algorithm with some modification.
- If the mid element is greater than both of its adjacent elements, then mid is the maximum.
- If mid element is greater than its next element and smaller than the previous element then maximum lies on left side of mid.
- If mid element is smaller than its next element and greater than the previous element then maximum lies on right side of mid.

b) **Code snapshots of implementation**

```
#include<bits/stdc++.h>
using namespace std;

int findMax(vector<int> arr,int low,int high)
{
        if(low==high)
                return arr[low];
        if((high==low+1) && arr[low]>=arr[high])
                return arr[low];
        if((high==low+1) && arr[low]<arr[high])
                return arr[high];
        int mid=(low+high)/2;
        if(arr[mid]>arr[mid+1] && arr[mid]> arr[mid-1])
                return arr[mid];
        if(arr[mid]>arr[mid+1] && arr[mid]< arr[mid-1])
```

```
                return findMax(arr,low,mid-1);
        else
                return findMax(arr,mid+1,high);
}

int main()
{
        int n;
        cout<<"Enter Array Size: ";
        cin>>n;
        vector<int> arr(n);
        cout<<"Enter array elements:\n";
        for(int i=0;i<n;i++)
                cin>>arr[i];
        cout<<"\nMaximum Element: "<<findMax(arr,0,n-1);
}
```

Output:



```
Enter Array Size: 5
Enter array elements:
3
6
4
8
5

Maximum Element: 8
--------------------------------
Process exited after 10.45 seconds with return value 0
Press any key to continue . . .
```

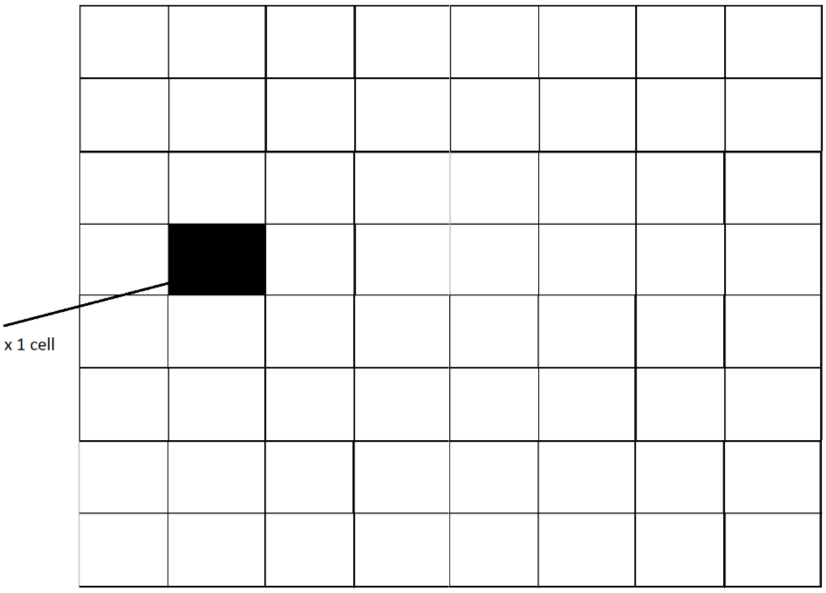
c) **Complexity of proposed algorithm (Time & Space)**

- Time Complexity: O(logn)
- Space Complexity: O(1)

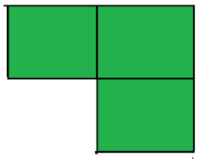
d) **Your comment (How your solution is optimal?)**

- The proposed algorithm is space efficient also needs O(logn) time to solve it.

2. 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 x 1 missing.



Missing 1 x 1 cell in board



Given L shape tile that is to be used to fill the board with 1 missing cell

Ans:

a) **Algorithm: (Pseudocode)**

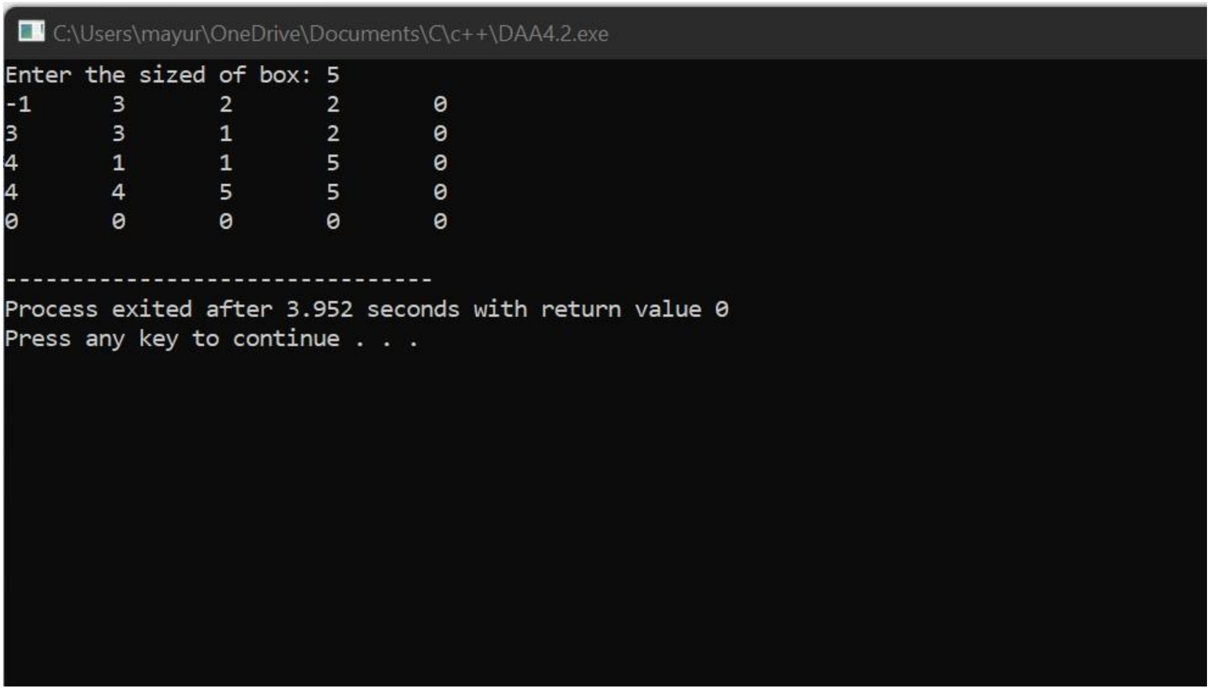
- Base case: n= 2, A 2 x 2 square with one cell missing is nothing but a tile and can be filled with a single tile.
- Place a L shaped tile at the center such that it does not cover the n/2 * n/2 subsequence that has a missing square. Now all four subsequence of size n/2 x n/2 have missing cell (a cell that doesn't need to be filled).
- 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, p4)

b) Code snapshots of implementation

```
#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()
{
    cout<<"Enter the sized of box: ";
    cin>>size_of_grid;
    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";
    }
    return 0;
}
```

Output:



C:\Users\mayur\OneDrive\Documents\C\c++\DAA4.2.exe

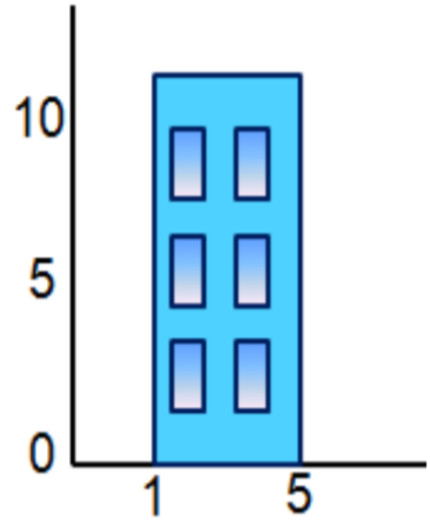
```
Enter the sized of box: 5
-1      3       2       2       0
3       3       1       2       0
4       1       1       5       0
4       4       5       5       0
0       0       0       0       0

--------------------------------
Process exited after 3.952 seconds with return value 0
Press any key to continue . . .
```

c) **Complexity of proposed algorithm (Time & Space)**

- Time Complexity:
  Recurrence relation for above recursive algorithm can be written as T(n)=4T(n/2)+C, where C is constant.
  By applying Masters Method to above recursion the Time Complexity is O(n^2)
- Space Complexity: O(n^2)

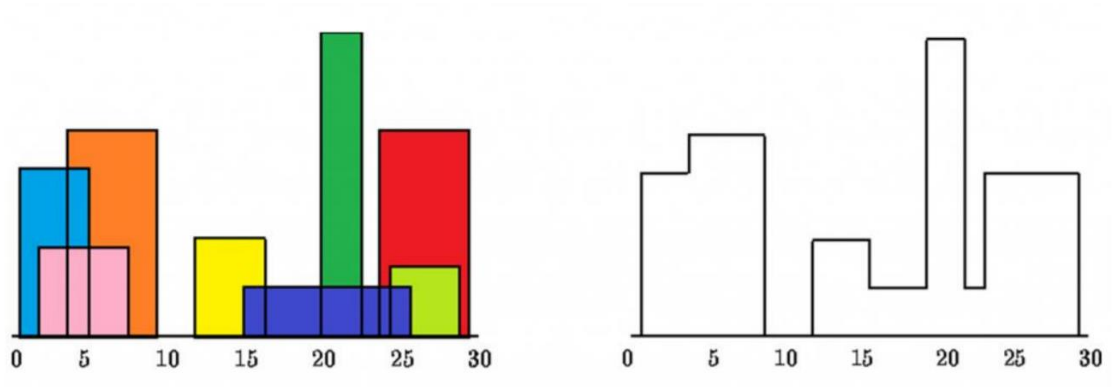
ASSIGNMENT 3

3. 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 coordinate of left side (or wall).
   'right': is x coordinate of right side
   'ht': is the 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.

ASSIGNMENT 3



With Time Complexity (nLogn)

Ans:

a) Algorithm: (Pseudocode)

- We divide the given set of buildings in two subsets having only their left point and the height. We then sort them.
- 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.

b) Code snapshots of implementation

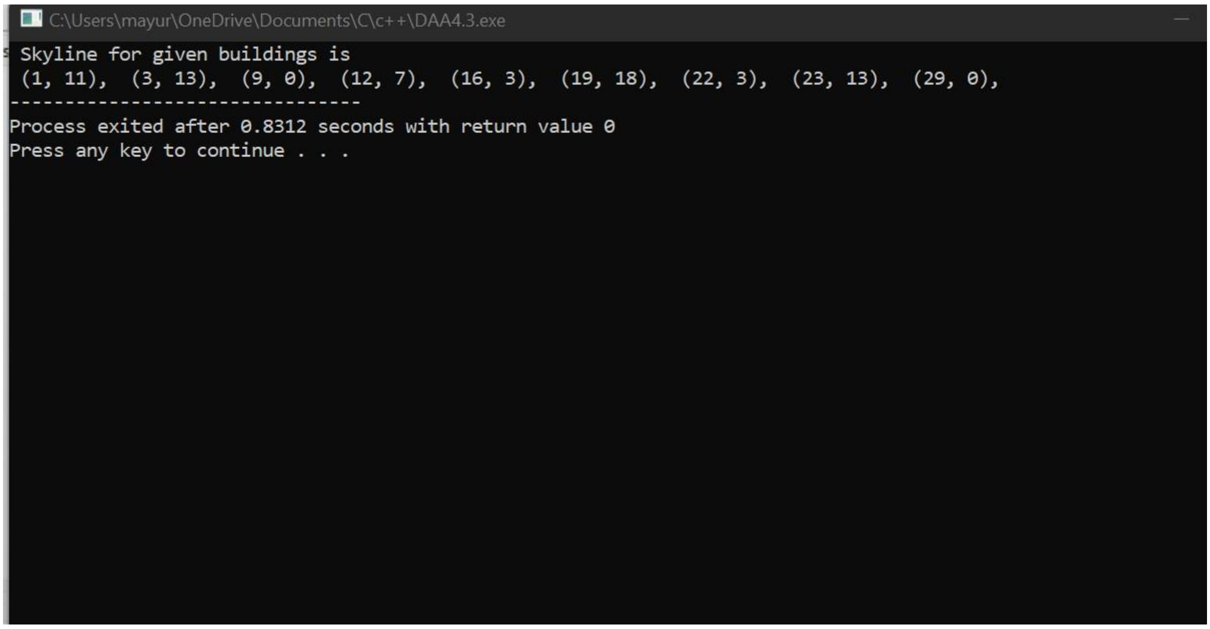
```
#include<bits/stdc++.h>
#include <iostream>
using namespace std;

struct Building {
   int left;

   int ht;

   int right;
};
```

ASSIGNMENT 3

```
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;
}
```

Output:

ASSIGNMENT 3



```
 Skyline for given buildings is
 (1, 11),  (3, 13),  (9, 0),  (12, 7),  (16, 3),  (19, 18),  (22, 3),  (23, 13),  (29, 0),
--------------------------------
Process exited after 0.8312 seconds with return value 0
Press any key to continue . . .
```

c) Complexity of proposed algorithm (Time & Space)

- Time Complexity: O(nlogn)

d) Your comment (How your solution is optimal?)

- We used divide and conquer to implement it in O(nlogn) time.