Name: Khushi Nitinkumar Patel
PRN: 2020BTECS00037
Batch: T5

Design and analysis of algorithm Lab

Week 3 Assignment

Part 1: 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).

- Algorithm:

The brute force approach is doing Linear Search which takes O(n) time.

The optimized approach is using Binary Search.

Step:

1. Find the middle element, if it is greater than both of its adjacent elements then it is the maximum element.
2. If middle element is smaller than its next element, search in right half of array i.e., l=mid+1
3. If middle element is greater than its next element, search in left half of array i.e.

r=mid-1

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;
int maxElement(int arr[], int n, int l, int r)
{
    while (l <= r)
    {
        int mid = l + (r - l) / 2;
        if (arr[mid] > arr[mid - 1] && arr[mid] > arr[mid + 1])
        {
            return arr[mid];
        }
        else if (arr[mid] < arr[mid + 1])
        {
            l = mid + 1;
        }
        else
        {
            r = mid - 1;
        }
    }
    return arr[r];
}

int main()
{
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    int l = 0, r = n - 1;
    int ans = maxElement(arr, n, l, r);
    cout << "Max Element in array that is first increasing and then decreasing :
"<<ans<<endl;
}
```

Output:

```
PS C:\Users\khush\Desktop\acads\5th sem\lab\daa\assignments\week3> cd "c
g++ q1.cpp -o q1 } ; if ($?) { .\q1 }
4
22 6 8 5
Max Element in array that is first increasing and then decreasing : 8
```

Time Complexity: O(logn)

Space Complexity: O(1)

Q2) Implement algorithm for Tiling problem: Given an n by n board where n is of form 2k 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*l missing.

>Algorithm:

The given n*n board is divided into (n/2)*(n/2) board repeatedly which produces 4 (n/2)*(n/2) non-identical boards. To make these boards identical by removing one cell from other three boards, place the L-shaped tile at the center.

1. Declare variable r, c to store index of missing tile and cnt to fill the tiles.
2. The base for this problem if 2*2 board, fill the board such that it covers all three cells which are not filled.
3. Find the index of missing cell.
4. If the missing cell is in 1st quadrant, call the place function which places the L-shape tile at center making all the boards identical i.e., the 2nd ,3rd ,4th quadrant now contains a missing cell.
5. If the missing cell is in 3rd quadrant, call the place function which places the L-shape tile at center making all the boards identical i.e., the 1st ,2nd ,4th quadrant now contains a missing cell.

6. If the missing cell is in 2nd quadrant, call the place function which places the L-shape tile at center making all the boards identical i.e., the 1st ,3rd ,4th quadrant now contains a missing cell.

7. If the missing cell is in 4th quadrant, call the place function which places the L-shape tile at center making all the boards identical i.e., the 2nd ,3rd ,1st quadrant now contains a missing cell.

8. Now we have 4 sub boards, thus call the function tile for these subboards.

Code:

```
#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 = 4;
    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";
    }
}
```

Output:

```
Max Element in array that is first increasing and t
PS C:\Users\khush\Desktop\acads\5th sem\lab\daa\ass
g++ q2.cpp -o q2 } ; if ($?) { .\q2 }
-1      3       2       2
3       3       1       2
4       1       1       5
4       4       5       5
```

Time Complexity: O(n^2)

Space Complexity: O(n^2)

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.

- Algorithm:

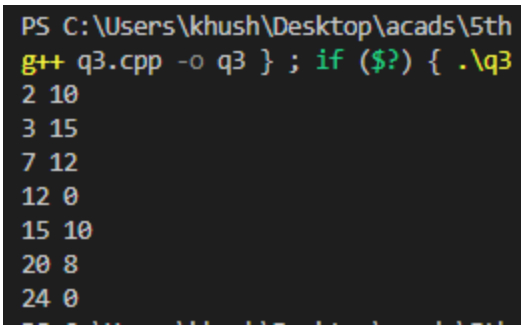
1. Store the start point of building and end point of building along with height.
2. Sort the start point, end point.
3. Traverse from left to right, if we come across start point of building store it in min heap, using height as key.
4. If we come across end point of building then remove it from heap until we reach a building whose right node is still ahead.

Code:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
vector<vector<int>> getSkyline(vector<vector<int>> &buildings)
{
    vector<vector<int>> edges;
    // push start_point,height,end_point
    for (int i = 0; i < buildings.size(); i++)
    {
        int x = edges.size();
        edges.push_back(vector<int>());
        edges[x].push_back(buildings[i][0]);
        edges[x].push_back(-buildings[i][2]);
        edges[x].push_back(buildings[i][1]);
    }

    // push end_points and their ending will be 0 and no height so 1e9
    for (int i = 0; i < buildings.size(); i++)
    {
        int x = edges.size();
        edges.push_back(vector<int>());
        edges[x].push_back(buildings[i][1]);
        edges[x].push_back(0);
        edges[x].push_back(1e9);
    }
    // sort so that start point and end point are arranged correctly
    sort(edges.begin(), edges.end());
    // min heap of pair of integers
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>>
        prevHighest;

    prevHighest.push({0, 1e9});

    vector<vector<int>> skyline;
    for (int i = 0; i < edges.size(); i++)
    {
        int start = edges[i][0];
        int currHeight = -1 * edges[i][1];
        int end = edges[i][2];
        // if end point of prev building is less than start point of next
```

```
        // building, then it will not be present in ans
        while (prevHighest.top().second <= start)
        {
            prevHighest.pop();
        }
        if (currHeight > 0)
        {
            prevHighest.push({-currHeight, end});
        }
        if (skyline.size() == 0)
        {
            skyline.push_back(vector<int>());
            skyline[0].push_back(start);
            skyline[0].push_back(-prevHighest.top().first);
        }
        else if (skyline.back()[1] != -prevHighest.top().first)
        {
            int x = skyline.size();
            skyline.push_back(vector<int>());
            skyline[x].push_back(start);
            skyline[x].push_back(-prevHighest.top().first);
        }
    }
    return skyline;
}
int main()
{
    //{start, end, height}
    vector<vector<int>>
        buildings = {{2, 9, 10}, {3, 7, 15}, {5, 12, 12}, {15, 20, 10}, {19, 24,
8}};
    //  vector<vector<int>>
    // buildings={{1,5,11},{2,7,6},{3,9,13},{12,16,7},{14,25,3},{19,22,18},
    //  {23,29,13},{24,28,4}};
    vector<vector<int>> ans = getSkyline(buildings);
    for (int i = 0; i < ans.size(); i++)
    {
        cout << ans[i][0] << " " << ans[i][1] << endl;
    }
    return 0;
}
```

Output:



Time Complexity: O(nlogn)

Space Complexity: O(n)