Name: Ranit Pal **UID: 22BCS14455**

Semester: 6th **Date of Performance:17/2/25**

Section/Group: 617/A **Branch: CSE**

Subject: AP Lab_II **Subject code: 22CSP-351**

Problem 1

Question: The Skyline Problem

Code:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> ans;
        multiset<int> pq{0};

        vector<pair<int, int>> points;

        for(auto b: buildings){
            points.push_back({b[0], -b[2]});
            points.push_back({b[1], b[2]});
        }

        sort(points.begin(), points.end());
```

```
int ongoingHeight = 0;

        for(int i = 0; i < points.size(); i++){
            int currentPoint = points[i].first;
            int heightAtCurrentPoint = points[i].second;

            if(heightAtCurrentPoint < 0){
                pq.insert(-heightAtCurrentPoint);
            } else {
                pq.erase(pq.find(heightAtCurrentPoint));
            }


            auto pqTop = *pq.rbegin();
            if(ongoingHeight != pqTop){
                ongoingHeight = pqTop;
                ans.push_back({currentPoint, ongoingHeight});
            }
        }

        return ans;
    }
};
```

Problem 2

Aim: Reverse Pairs

Code:

```
class SegTree {
private:
    int tree_size;
    vector<int> tree;

    void update(int lx, int rx, int ni, int idx) {

        if (rx - lx == 1) {
            tree[ni]++;
            return;
        }

        int m = (lx + rx) >> 1;

        if (idx < m)
            update(lx, m, ni * 2 + 1, idx);
        else
            update(m, rx, ni * 2 + 2, idx);

        tree[ni] = tree[ni * 2 + 1] + tree[ni * 2 + 2];
    }

    int query(int l, int r, int lx, int rx, int ni) {
```

```
        if (l >= rx || r <= lx)
            return 0;

        if (l <= lx && r >= rx)
            return tree[ni];

        int m = (lx + rx) >> 1;
        return query(l, r, lx, m, ni * 2 + 1) + query(l, r, m, rx, ni * 2 + 2);
    }

public:
    SegTree(int n) {
        tree_size = 1;

        while (tree_size < n)
            tree_size <<= 1;

        tree = vector<int>(tree_size * 2);
    }

    void update(int idx) {
        update(0, tree_size, 0, idx);
    }

    int query(int l, int r) {
        return query(l, r + 1, 0, tree_size, 0);
```

```
}
};

class Solution {
public:
    int reversePairs(vector<int>& nums) {

        int n = nums.size();
        set<long long> values;

        for(const auto& num : nums) {
            values.insert(num);
            values.insert(2LL * num);
        }

        int last_index = 0;
        unordered_map<long long, int> values_indices;

        for(const auto& val : values)
            values_indices[val] = last_index++;

        SegTree seg_tree(last_index);
        int ans = 0;

        for(int i = 0; i < n; ++i) {
            ans += seg_tree.query(values_indices[2LL * nums[i]] + 1, last_index);
```

```
            seg_tree.update(values_indices[nums[i]]);

        }

        return ans;
    }
};
```

Problem 3

Aim: Longest increasing subsequence 2

Code:

```
#define ll int
class Solution {
public:
    void build(vector<ll>& seg,vector<ll>& a,ll low,ll high,ll ind){
        if(low==high){
            seg[ind]=a[low];
            return;
        }
        ll mid=(low + high)/2;
        build(seg,a,low,mid,2*ind + 1);
        build(seg,a,mid+1,high,2*ind + 2);
        seg[ind]=max(seg[2*ind + 1],seg[2*ind + 2]);
    }
    ll query(vector<ll>& seg,ll low,ll high,ll x,ll y,ll ind){
        if(x>high || y<low){
            return INT_MIN;
```

```
}
    if(low>=x && high<=y){
        return seg[ind];
    }
    ll mid=(low + high)/2;
    ll left=query(seg,low,mid,x,y,2*ind + 1);
    ll right=query(seg,mid+1,high,x,y,2*ind + 2);
    return max(left,right);
}
void update(vector<ll>& seg,ll low,ll high,ll i,ll val,ll ind){
    if(low==high){
        seg[ind]=max(seg[ind],val);
        return;
    }
    int mid=(low+high)/2;
    if(i<=mid){
        update(seg,low,mid,i,val,2*ind + 1);
    }else{
        update(seg,mid+1,high,i,val,2*ind+2);
    }
    seg[ind]=max(seg[2*ind + 1],seg[2*ind + 2]);
}
int lengthOfLIS(vector<int>& nums, int k) {
    int n=nums.size();
    if(n==1){
        return 1;
```

```
int mx=0,ans=0;
    for(int i=0;i<n;i++){
        mx=max(mx,nums[i]);
    }
    int sz=mx+1;
    vector<int> a(sz,0);
    vector<ll> seg(4*mx + 10);
    build(seg,a,0,mx,0);
    update(seg,0,sz-1,nums[n-1],1,0);
    for(int i=n-2;i>=0;i--){
        if(nums[i]==mx){
            update(seg,0,sz-1,nums[i],1,0);
            continue;
        }
        int l=nums[i]+1,r=min(mx,nums[i]+k);
        int some=query(seg,0,sz-1,l,r,0);
        if(some==INT_MIN){
            some=0;
        }
        update(seg,0,sz-1,nums[i],some+1,0);
    }
    ans=query(seg,0,sz-1,1,mx,0);

    return ans;
}
```