Name-Suman kumar

Uid-22BCS15488

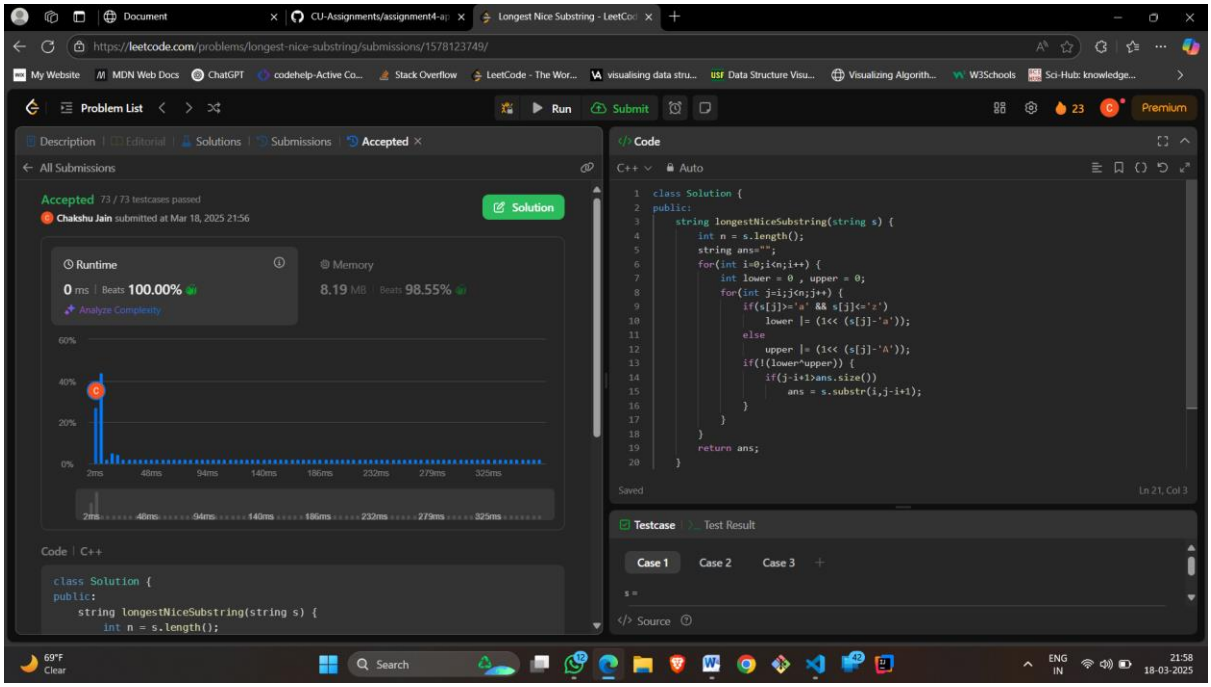
Section-608/B

AP ASSIGNMENT

1. Longest Nice substring :

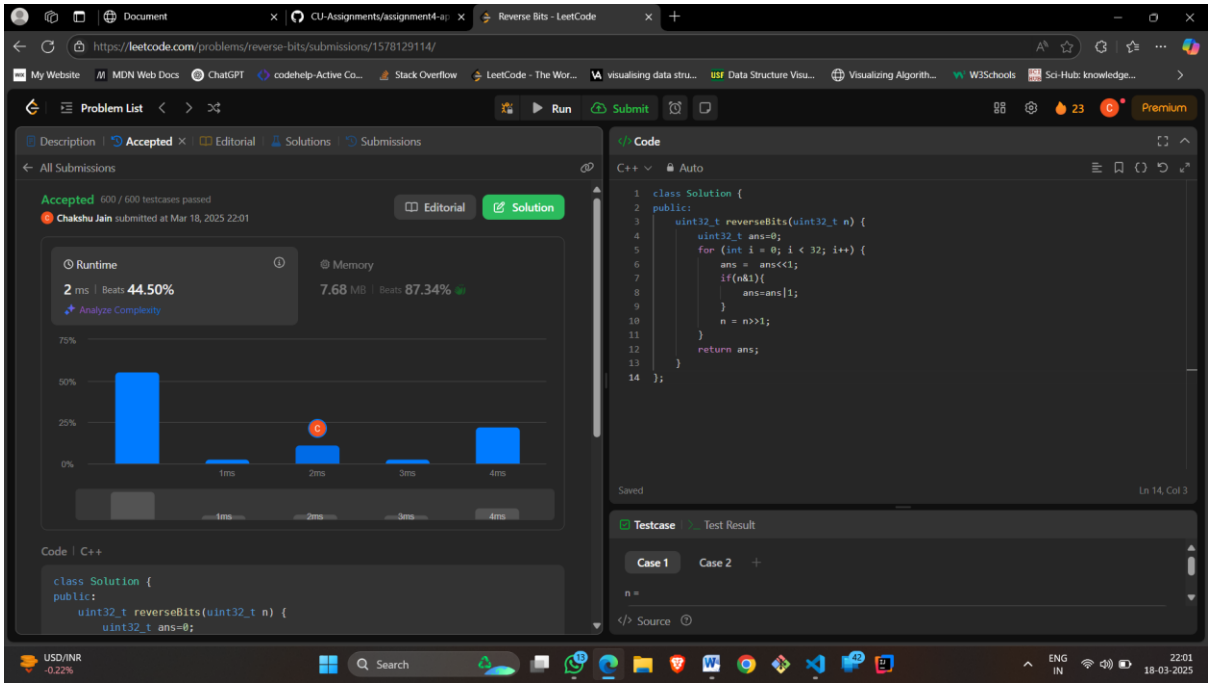
```
class Solution {
public:
    string longestNiceSubstring(string s) {
        int n = s.length();
        string ans="";
        for(int i=0;i<n;i++) {
            int lower = 0 , upper = 0;
            for(int j=i;j<n;j++) {
                if(s[j]>='a' && s[j]<='z')
                    lower |= (1<< (s[j]-'a'));
                else
                    upper |= (1<< (s[j]-'A'));
                if(!(lower^upper)) {
                    if(j-i+1>ans.size())
                        ans = s.substr(i,j-i+1);
                }
            }
        }
        return ans;
    }
```

};



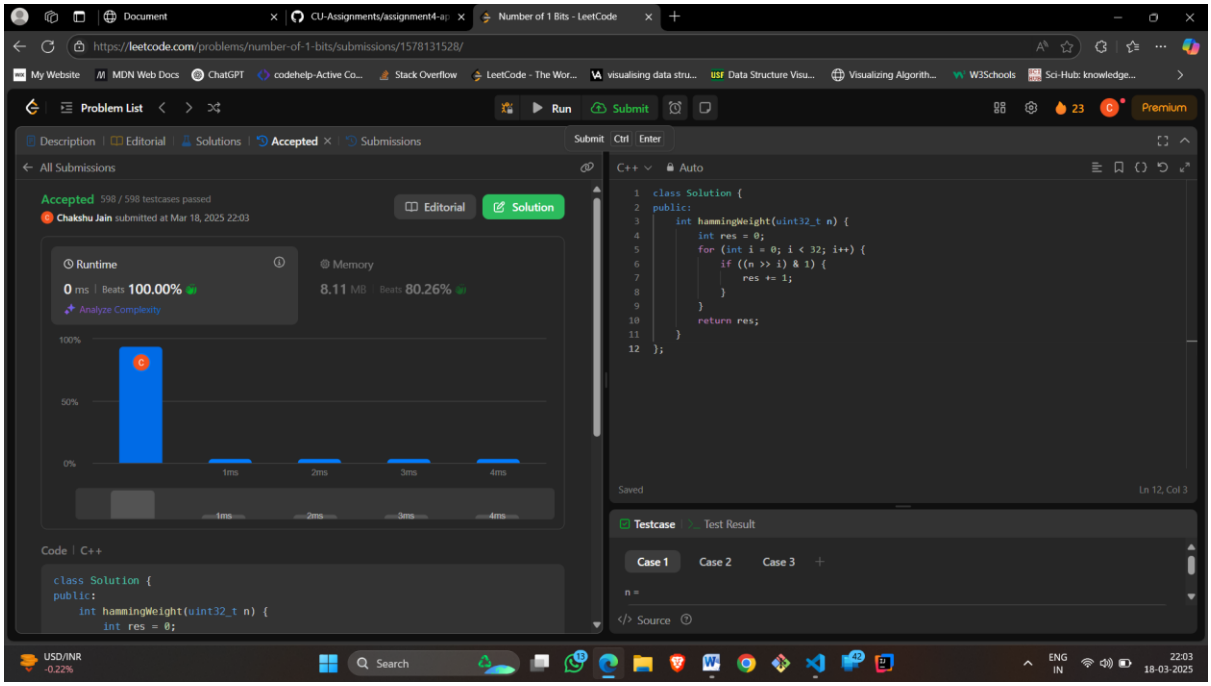
2.Reverse bits :

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t ans=0;
        for (int i = 0; i < 32; i++) {
            ans =  ans<<1;
            if(n&1){
                ans=ans|1;
            }
            n = n>>1;
        }
        return ans;
    }
};
```



3.Number of 1-bits:

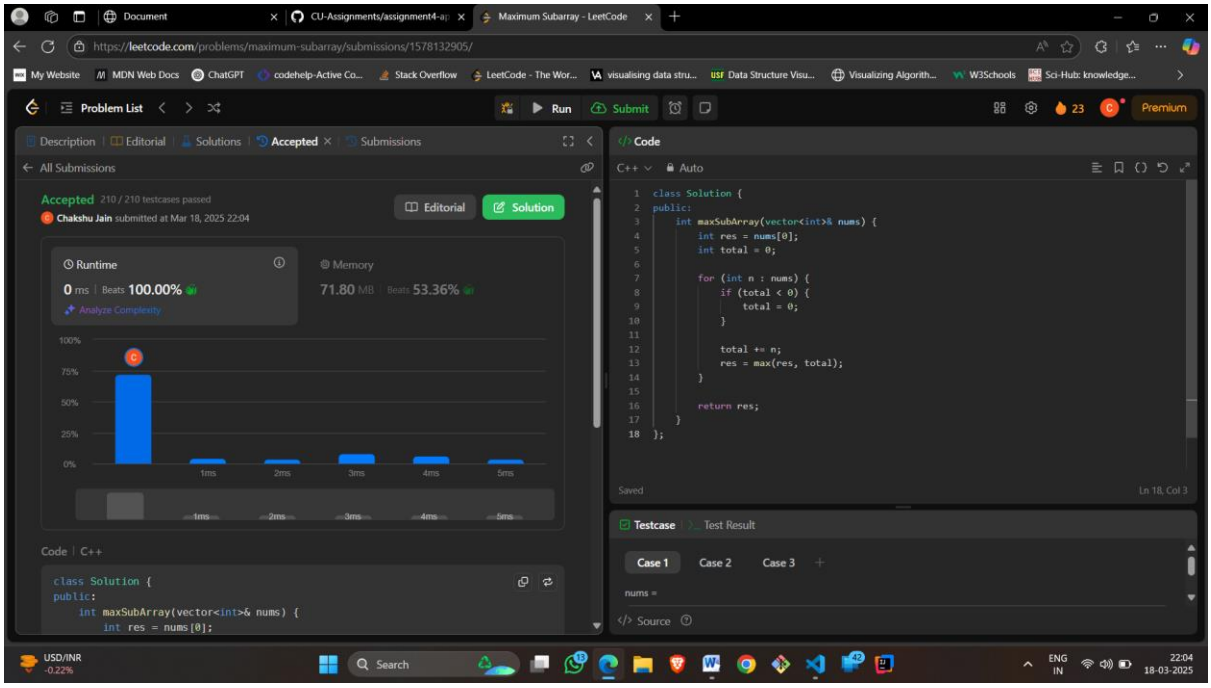
```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int res = 0;
        for (int i = 0; i < 32; i++) {
            if ((n >> i) & 1) {
                res += 1;
            }
        }
        return res;
    }
};
```



4.maximum of subbarray:

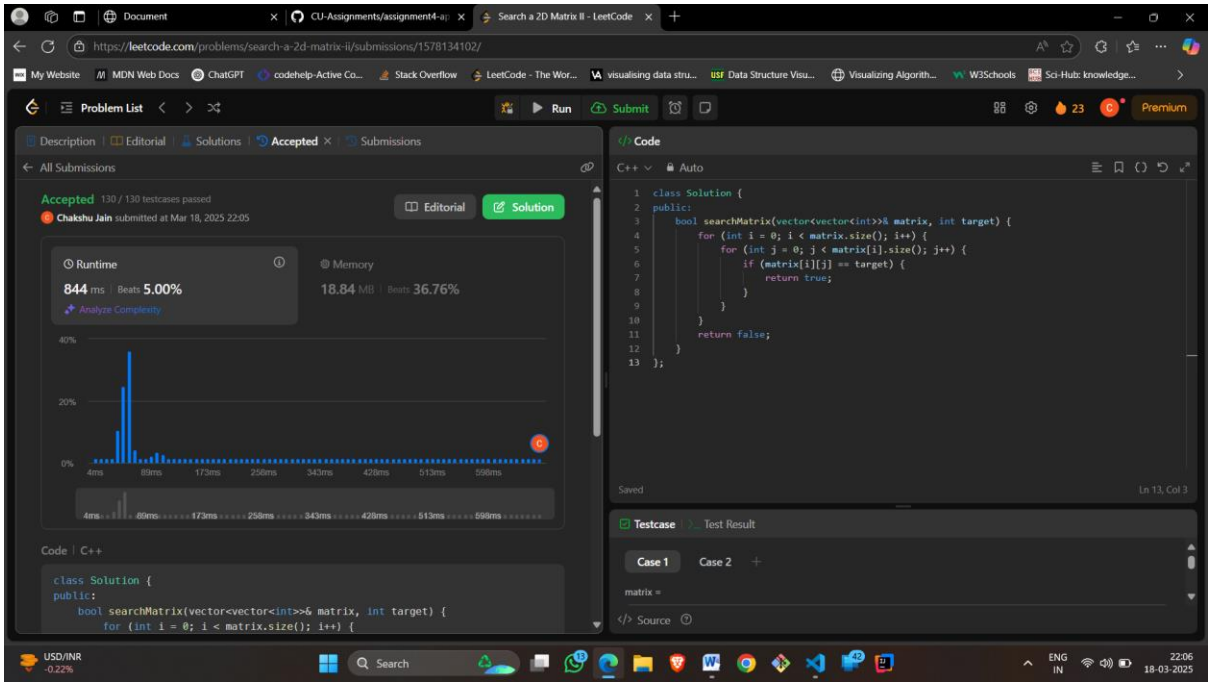
```
class Solution {

public:

    int maxSubArray(vector<int>& nums) {

        int res = nums[0];

        int total = 0;


        for (int n : nums) {

            if (total < 0) {

                total = 0;

            }


            total += n;

            res = max(res, total);

        }


        return res;
```

```
    }
};
```



5.Search a 2D matrix :

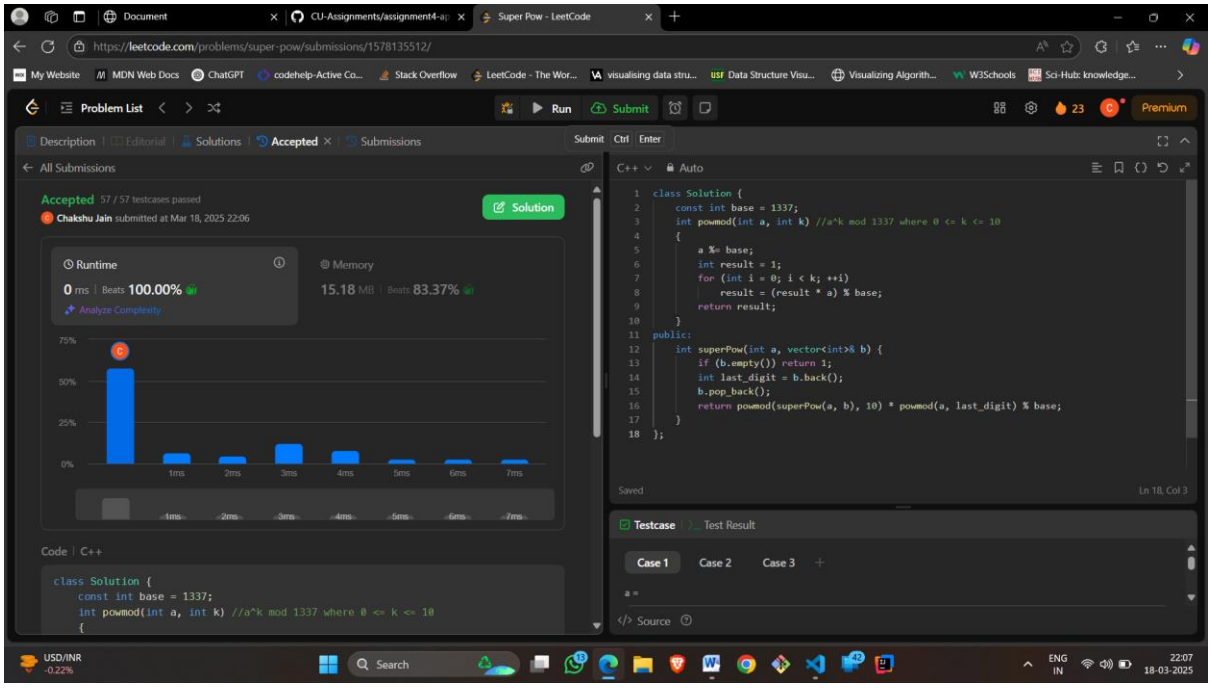
```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        for (int i = 0; i < matrix.size(); i++) {
            for (int j = 0; j < matrix[i].size(); j++) {
                if (matrix[i][j] == target) {
                    return true;
                }
            }
        }
        return false;
    }
};
```



6.super pow:

```
class Solution {
    const int base = 1337;
    int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10
    {
        a %= base;
        int result = 1;
        for (int i = 0; i < k; ++i)
            result = (result * a) % base;
        return result;
    }
public:
    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1;
        int last_digit = b.back();
        b.pop_back();
        return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;
```

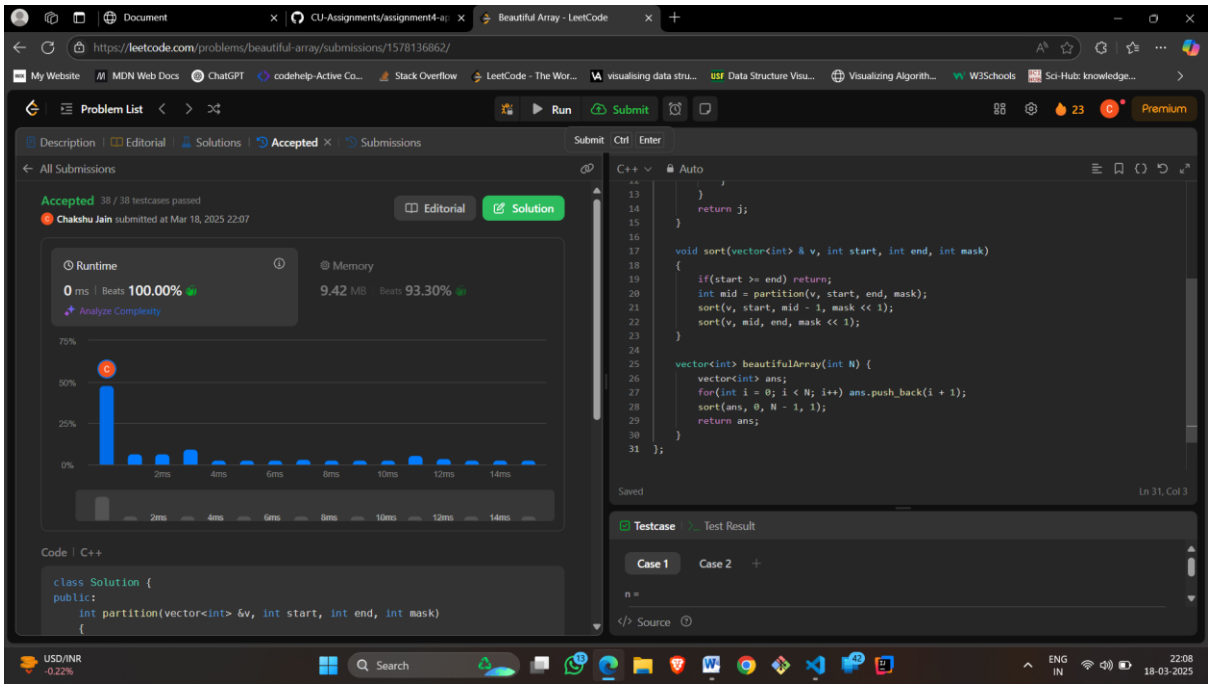
```
    }
};
```



7.beautiful array :

```
class Solution {
public:
    int partition(vector<int> &v, int start, int end, int mask)
    {
        int j = start;
        for(int i = start; i <= end; i++)
        {
            if((v[i] & mask) != 0)
            {
                swap(v[i], v[j]);
                j++;
            }
        }
        return j;
```

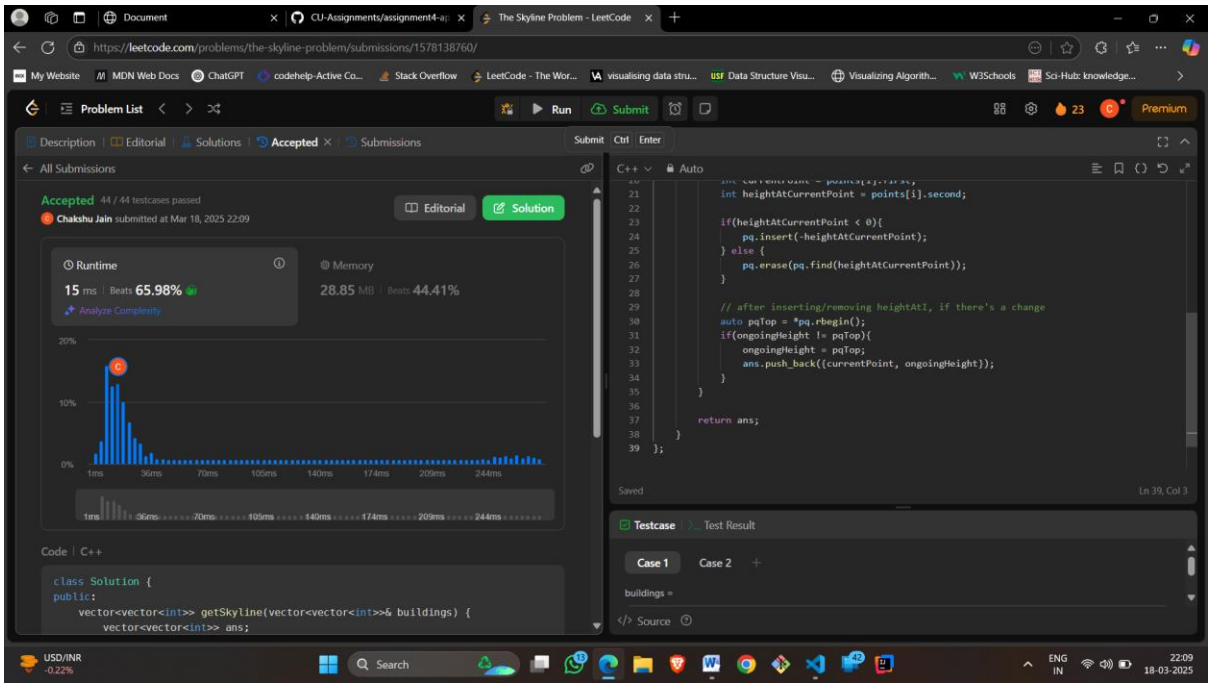
```
    }

    void sort(vector<int> & v, int start, int end, int mask)
    {
        if(start >= end) return;
        int mid = partition(v, start, end, mask);
        sort(v, start, mid - 1, mask << 1);
        sort(v, mid, end, mask << 1);
    }

    vector<int> beautifulArray(int N) {
        vector<int> ans;
        for(int i = 0; i < N; i++) ans.push_back(i + 1);
        sort(ans, 0, N - 1, 1);
        return ans;
    }
};
```



8.the skyline problem:

```
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;

        // points.first = x coordinate, points.second = height
        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));
```

```
        }

        // after inserting/removing heightAtI, if there's a change
        auto pqTop = *pq.rbegin();
        if(ongoingHeight != pqTop){
            ongoingHeight = pqTop;
            ans.push_back({currentPoint, ongoingHeight});
        }
      }

      return ans;
   }
};
```



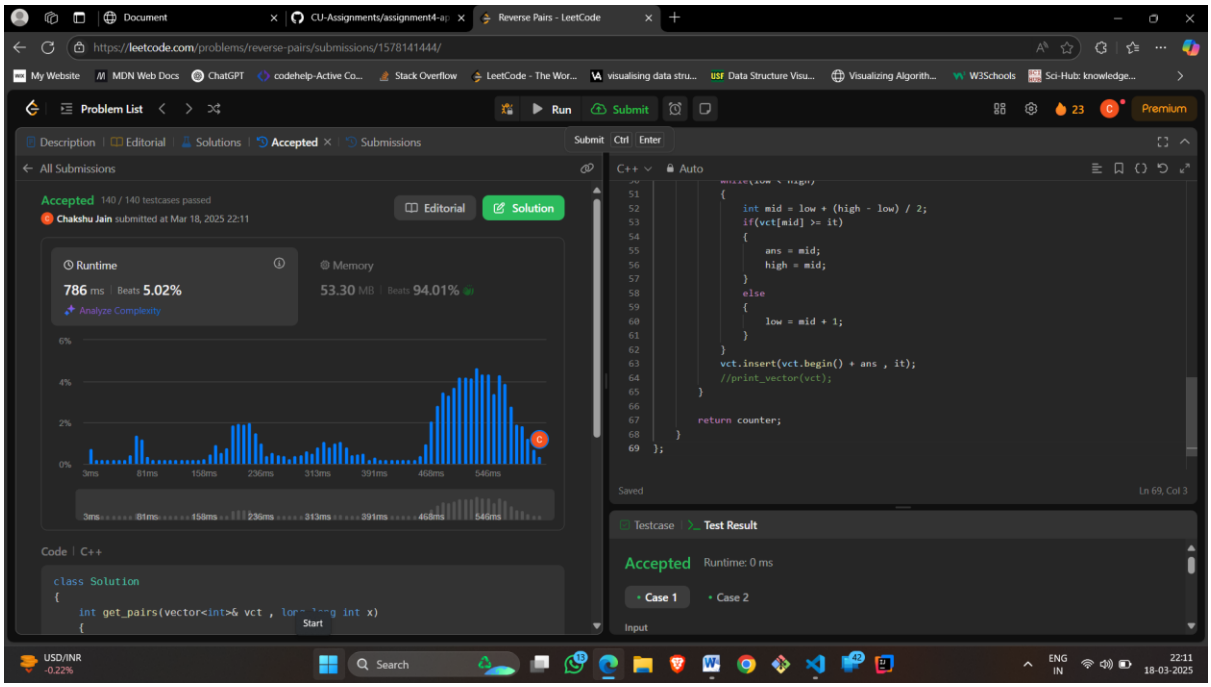
9.Reverse pairs :

```
class Solution
{
   int get_pairs(vector<int>& vct , long long int x)
```

```
{
    //sort(vct.begin() , vct.end());
    int size = vct.size();
    int low = 0;
    int high = size - 1;
    int ans = -1;
    while(low <= high)
    {
        int mid = high - (high - low) / 2;
        int ele = vct[mid];
        if(ele > x)
        {
            ans = mid;
            high = mid - 1;
        }
        else
        {
            low = mid  + 1;
        }
    }
    if(ans == -1) return 0;
    return vct.size() - ans;
}

// void print_vector(vector<int>& nums)
// {
//     cout<<endl;
```

```
    //     for(auto it : nums)
    //     {
    //         cout<<" "<<it;
    //     }
    //     cout<<endl;
    // }

public:
    int reversePairs(vector<int>& nums)
    {
        vector<int> vct;
        int counter = 0;
        for(auto it : nums)
        {
            long long int x = 1LL * 2 * it;
            counter += get_pairs(vct , x);
            int low = 0;
            int high = vct.size();
            int ans = vct.size();
            while(low < high)
            {
                int mid = low + (high - low) / 2;
                if(vct[mid] >= it)
                {
                    ans = mid;
                    high = mid;
                }
```

```
            else
            {
                low = mid + 1;
            }
        }
        vct.insert(vct.begin() + ans , it);
        //print_vector(vct);
    }

    return counter;
  }
};
```



10.longest increasing substring :

```
class Solution {
public:
    vector<int>tree;
```

```
void update(int node,int st,int end,int i,int val){
    if(st==end){
        tree[node]=max(tree[node],val);
        return;
    }
    int mid=(st+end)/2;
    if(i<=mid){
        update(node*2,st,mid,i,val);
    }else{
        update(node*2+1,mid+1,end,i,val);
    }
    tree[node]=max(tree[node*2],tree[node*2+1]);
}
int query(int node,int st,int end,int x,int y){
    if(x>end || y<st) return -1e9;
    if(st>=x && end<=y){
        return tree[node];
    }
    int mid=(st+end)/2;
    int left=query(2*node,st,mid,x,y);
    int right=query(2*node+1,mid+1,end,x,y);
    return max(left,right);
}
int lengthOfLIS(vector<int>& nums, int k) {
    int n=nums.size();
    if(n==1) return 1;
    int m=*max_element(nums.begin(),nums.end());
```

```
        tree.clear();
        tree.resize(4*m+10);
        for(int i=n-1;i>=0;i--){
            int l=nums[i]+1,r=min(nums[i]+k,m);
            int x=query(1,0,m,l,r);
            if(x==-1e9) x=0;
            update(1,0,m,nums[i],x+1);
        }
        return tree[1];
    }
};
```