Advanced Programming Assignment 4

Name- Vivek Singh UID-22BCS11457 Section- 607-B

Qs 1. Longest Nice Substring:

Code: class Solution {

public:

string longestNiceSubstring(string s) {

if (s.size() < 2) return "";

unordered_set<char> st(begin(s), end(s));

for (int i = 0; i < s.size(); i++) {

if (st.find((char) toupper(s[i])) == end(st) || st.find((char) tolower(s[i])) == end(st)) {

string s1 = longestNiceSubstring(s.substr(0, i));

string s2 = longestNiceSubstring(s.substr(i + 1));

return s1.size() >= s2.size() ? s1 : s2;

}

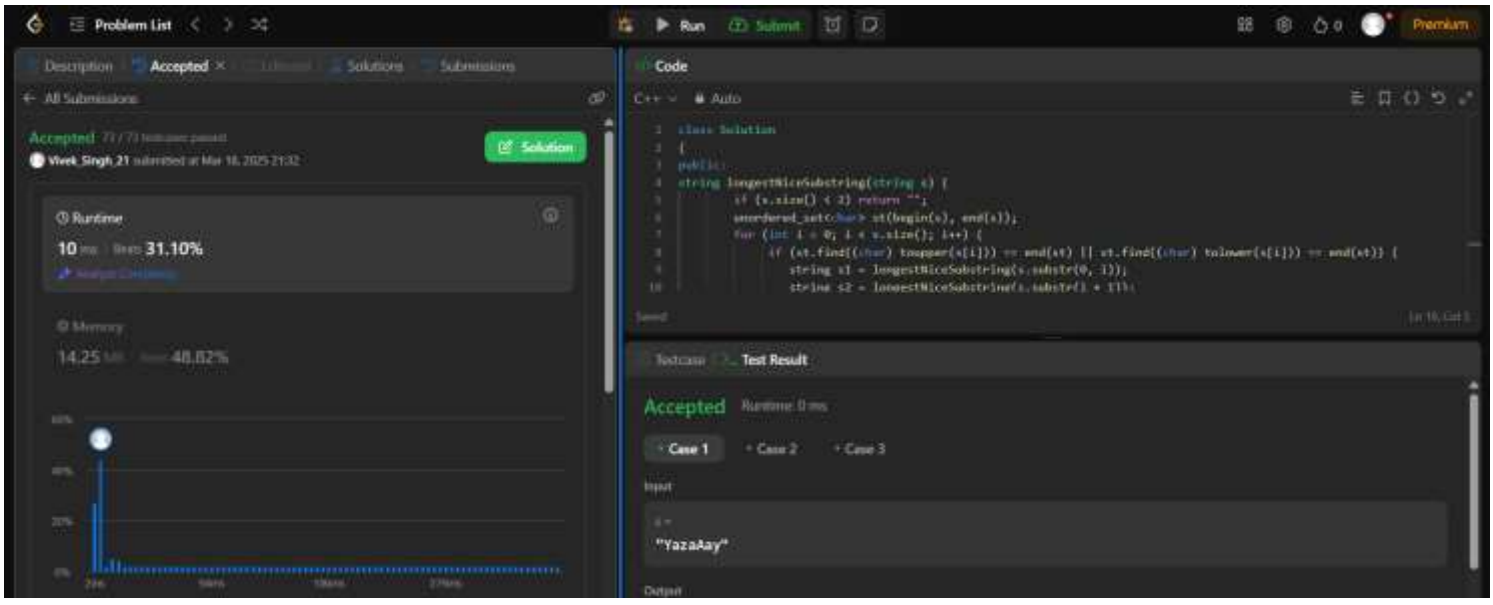
}

return s;

}

};

Submission:

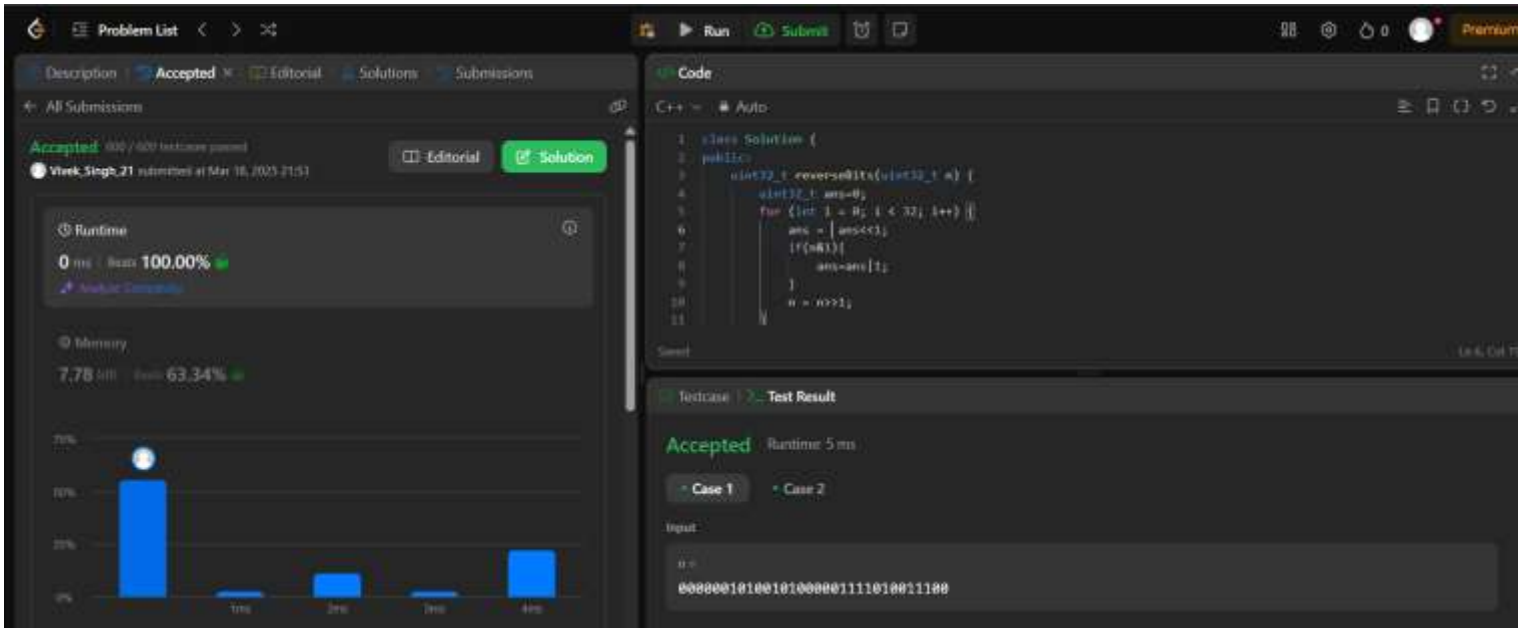


Qs 2. Reverse Bits:

Code:

```
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;
    }
};
```

Submission:



Qs 3. Number of 1 bits:

Code: class Solution {

public:

int hammingWeight(uint32_t n) {

int res = 0;

for (int i = 0; i < 32; i++) {

if ((n >> i) & 1) {

res += 1;

}

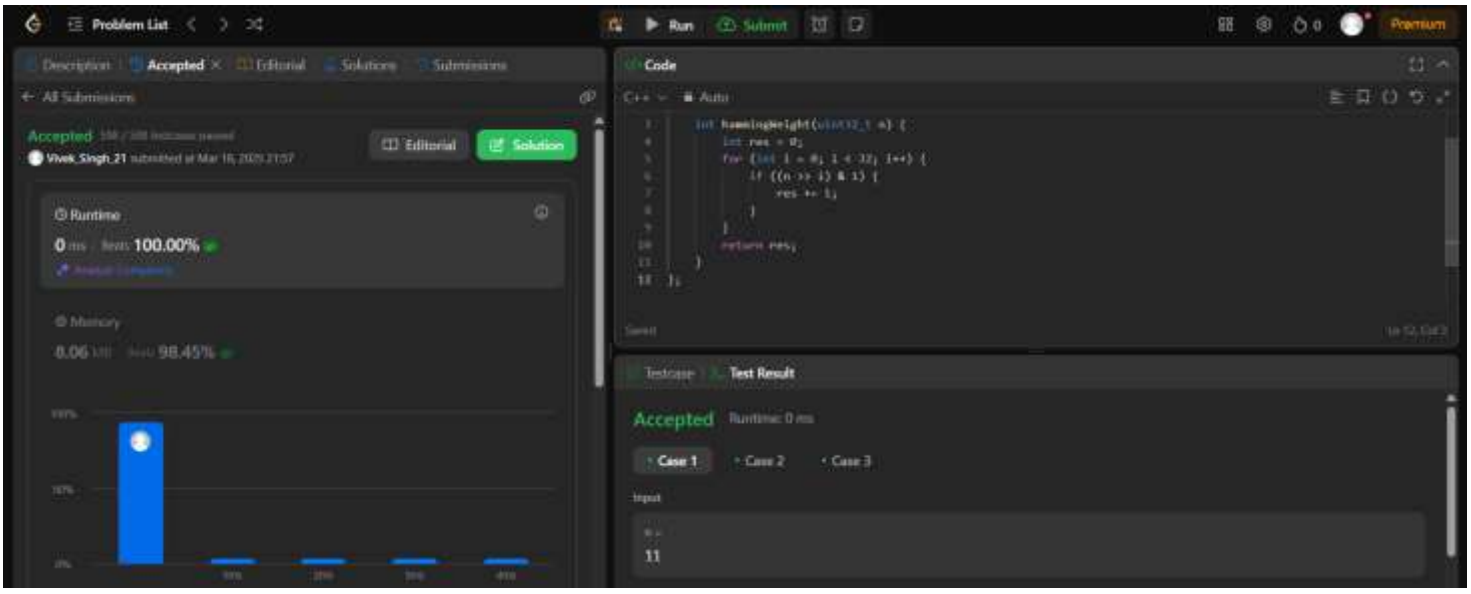
}

return res;

}

};

Submission:



Qs 4. Maximum subarray:

Code: class Solution {

public:

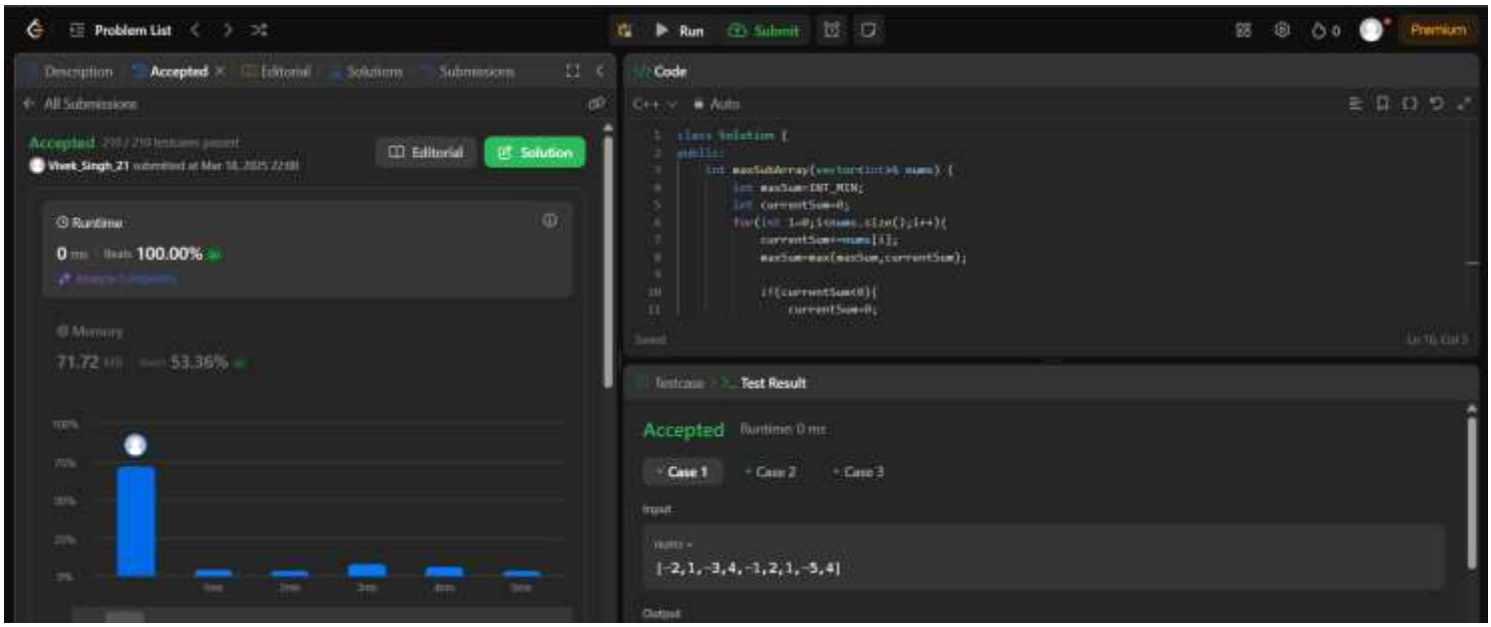
int maxSubArray(vector<int>& nums) {

int maxSum=INT_MIN;

int currentSum=0;

```
        for(int i=0;i<nums.size();i++){
            currentSum+=nums[i];
            maxSum=max(maxSum,currentSum);

            if(currentSum<0){
                currentSum=0;
            }
        }
        return maxSum;
    }
};
```

Submission:



Qs 5. Search in 2D Matrix ll:

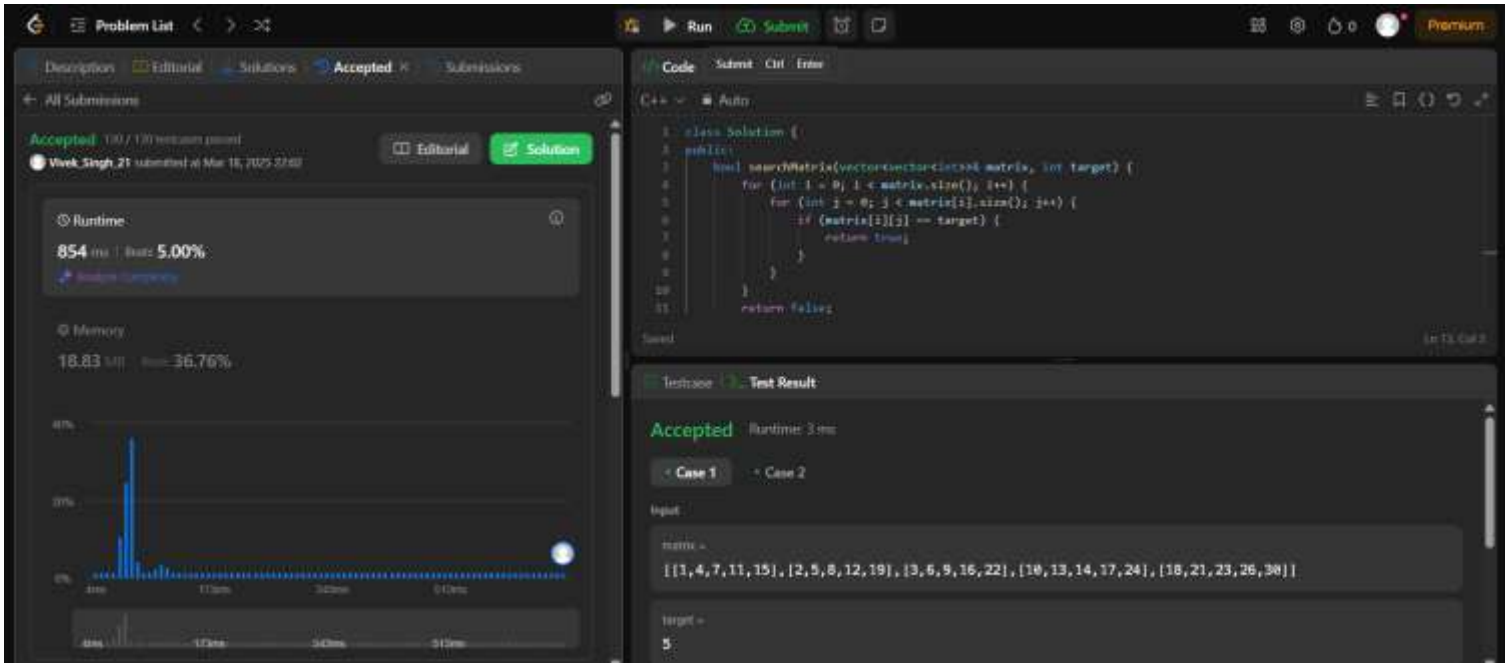
Code: class Solution {

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;
    }
};
```

Submission:



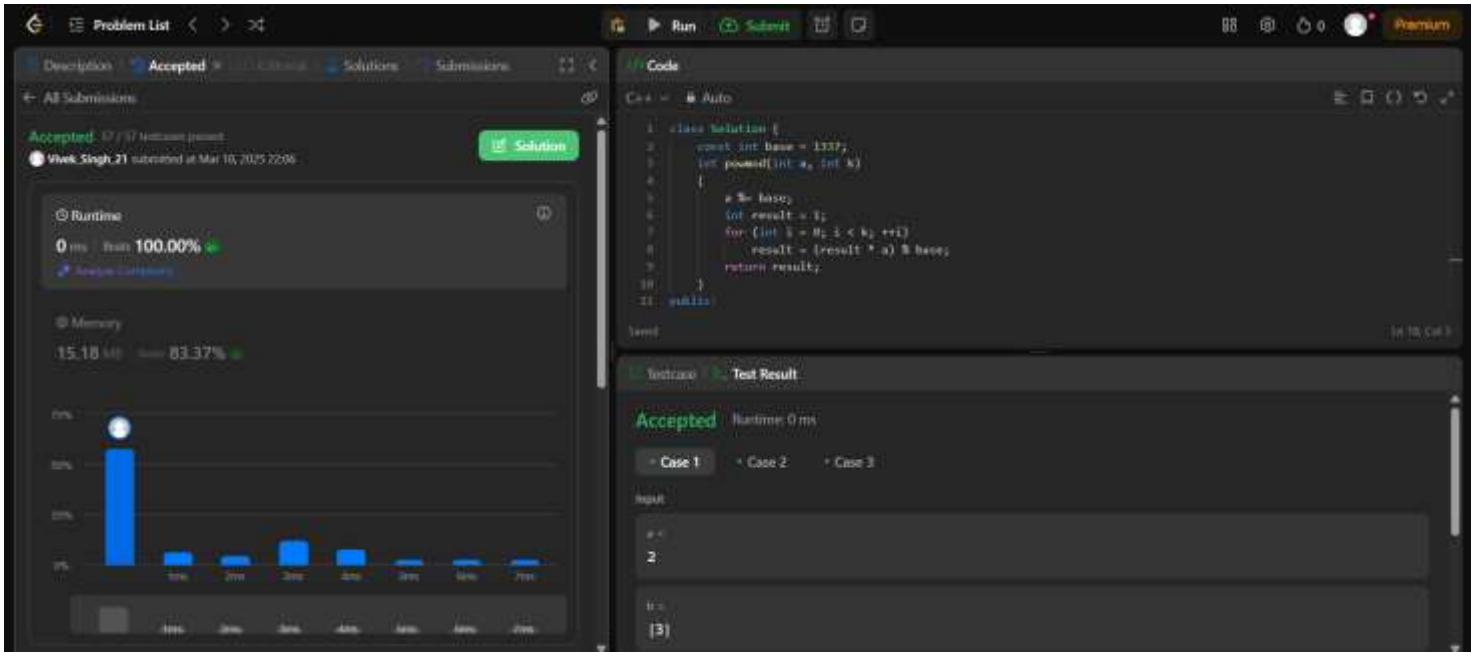
Qs 6. Super Pow:

Code: class Solution {

```
    const int base = 1337;
    int powmod(int a, int k)
    {
        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;
    }
};
```

Submission:



Qs 7. Beautiful array:

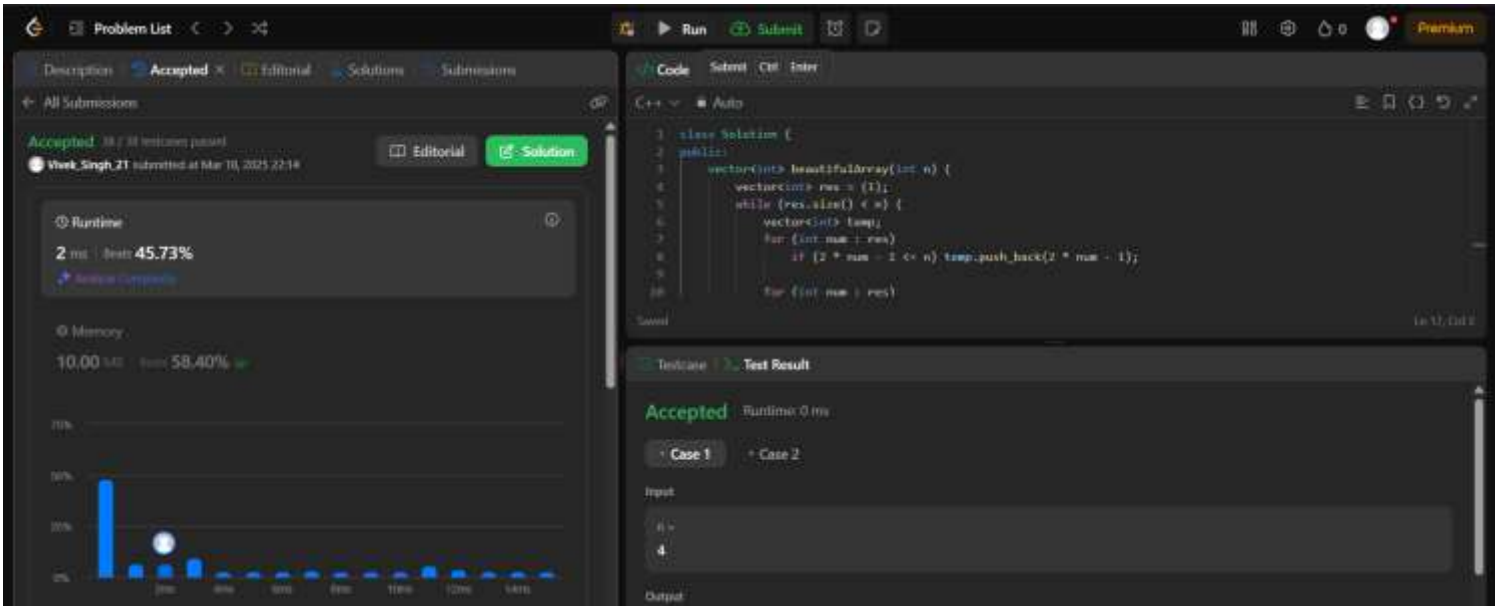
Code: class Solution {

public:

```
    vector<int> beautifulArray(int n) {
        vector<int> res = {1};
        while (res.size() < n) {
            vector<int> temp;
```

```
            for (int num : res)
                if (2 * num - 1 <= n) temp.push_back(2 * num - 1);

            for (int num : res)
                if (2 * num <= n) temp.push_back(2 * num);

            res = temp;
        }
        return res;
    }
};
```

Submission:



Qs 8. The Skyline Problem:

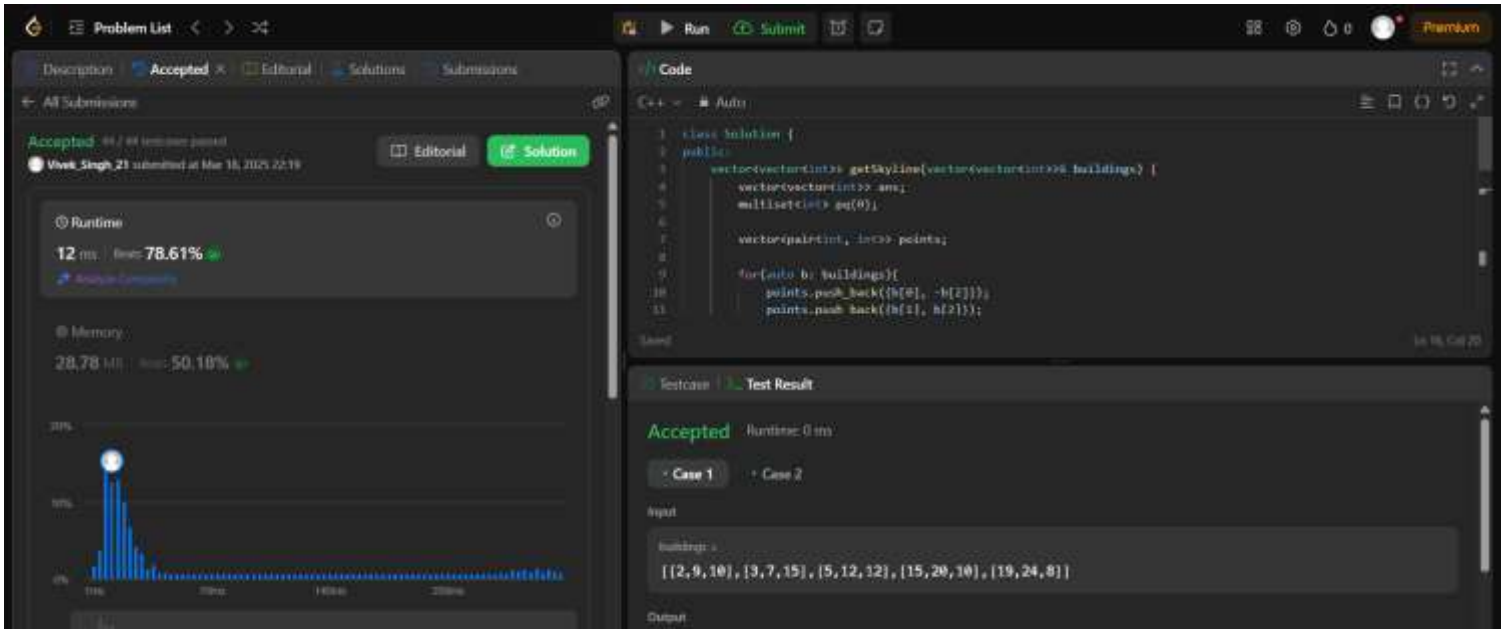
Code: class Solution {

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;
    }
};
```

Submission:



Qs 9. Reverse Pairs:

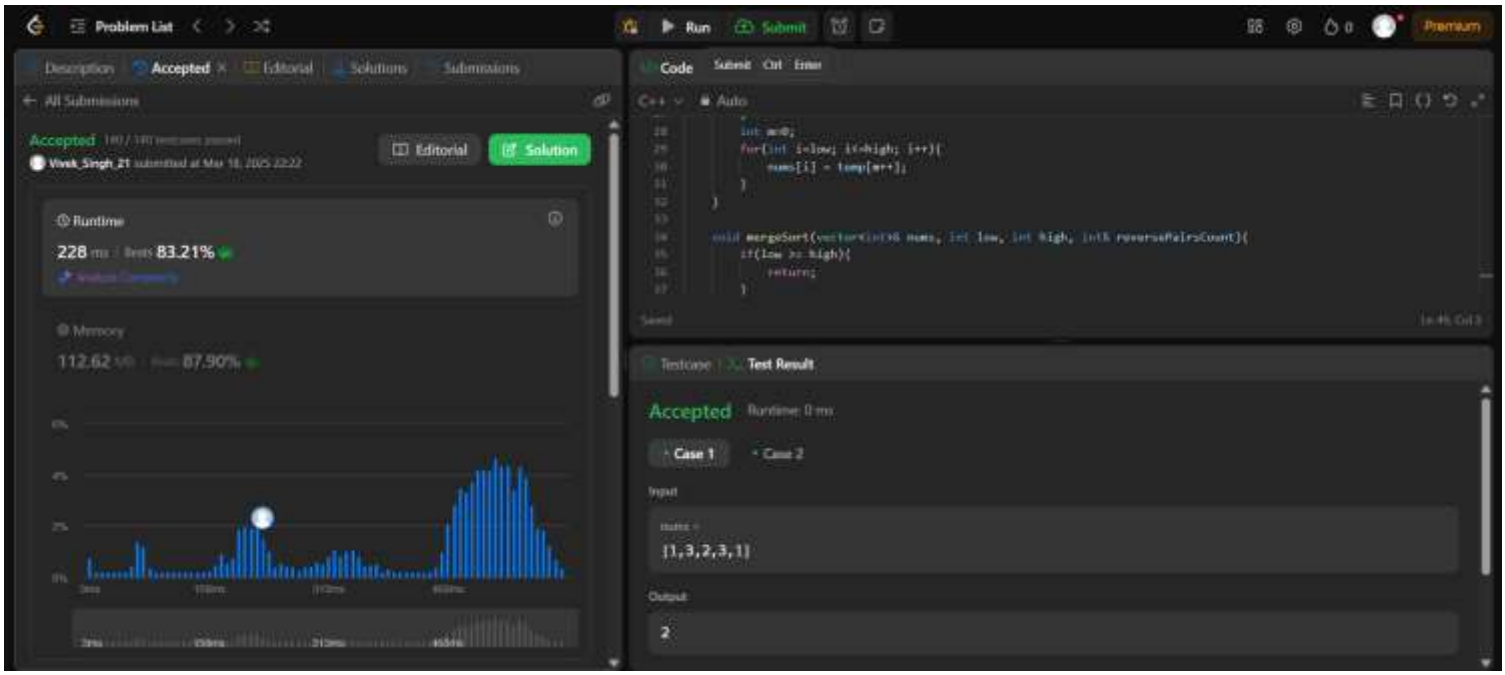
Code: class Solution {

private:

```
    void merge(vector<int>& nums, int low, int mid, int high, int& reversePairsCount){
        int j = mid+1;
        for(int i=low; i<=mid; i++){
            while(j<=high && nums[i] > 2*(long long)nums[j]){
                j++;
            }
            reversePairsCount += j-(mid+1);
        }
        int size = high-low+1;
        vector<int> temp(size, 0);
        int left = low, right = mid+1, k=0;
        while(left<=mid && right<=high){
            if(nums[left] < nums[right]){
                temp[k++] = nums[left++];
            }
```

```
            else{
                temp[k++] = nums[right++];
            }
        }
        while(left<=mid){
            temp[k++] = nums[left++];
        }
        while(right<=high){
            temp[k++] = nums[right++];
        }
        int m=0;
        for(int i=low; i<=high; i++){
            nums[i] = temp[m++];
        }
    }

    void mergeSort(vector<int>& nums, int low, int high, int& reversePairsCount){
        if(low >= high){
            return;
        }
        int mid = (low + high) >> 1;
        mergeSort(nums, low, mid, reversePairsCount);
        mergeSort(nums, mid+1, high, reversePairsCount);
        merge(nums, low, mid, high, reversePairsCount);
    }
public:
    int reversePairs(vector<int>& nums) {
        int reversePairsCount = 0;
        mergeSort(nums, 0, nums.size()-1, reversePairsCount);
        return reversePairsCount;  } };
```

Submission:



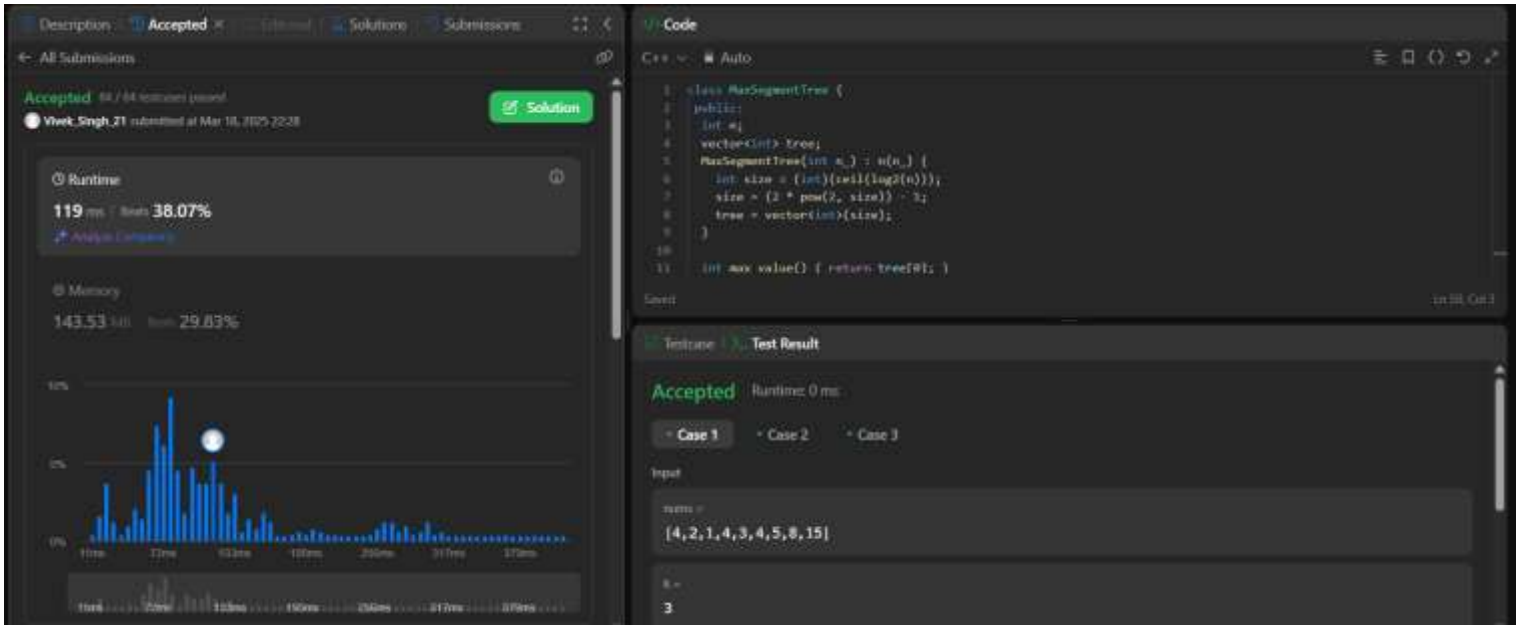
Qs 10. Longest Increasing Subsequence II:

Code: class MaxSegmentTree {

```
 public:
  int n;
  vector<int> tree;
  MaxSegmentTree(int n_) : n(n_) {
    int size = (int)(ceil(log2(n)));
    size = (2 * pow(2, size)) - 1;
    tree = vector<int>(size);
  }

  int max_value() { return tree[0]; }
  int query(int l, int r) { return query_util(0, l, r, 0, n - 1); }
  int query_util(int i, int qL, int qR, int l, int r) {
    if (l >= qL && r <= qR) return tree[i];
    if (l > qR || r < qL) return INT_MIN;
```

```
    int m = (l + r) / 2;
    return max(query_util(2 * i + 1, qL, qR, l, m), query_util(2 * i + 2, qL, qR, m + 1, r));
  }

  void update(int i, int val) { update_util(0, 0, n - 1, i, val); }
  void update_util(int i, int l, int r, int pos, int val) {
    if (pos < l || pos > r) return;
    if (l == r) {
      tree[i] = max(val, tree[i]);
      return;
    }
    int m = (l + r) / 2;
    update_util(2 * i + 1, l, m, pos, val);
    update_util(2 * i + 2, m + 1, r, pos, val);
    tree[i] = max(tree[2 * i + 1], tree[2 * i + 2]);
  }
};

class Solution {
 public:
  int lengthOfLIS(vector<int>& nums, int k) {
    MaxSegmentTree tree(1e5 + 1);
    for (int i : nums) {
      int lower = max(0, i - k);
      int cur = 1 + tree.query(lower, i - 1);
      tree.update(i, cur);
    }

    return tree.max_value();
  }};
```

Submission:


Description
Accepted
Solutions
Submissions
All Submissions
Accepted
Vivek_Singh_21
Solution
Runtime
119 ms
Beats 38.07%
Memory
143.53 MB
Beats 29.83%
Code
C++
Auto
class MaxSegmentTree {
public:
int n;
vector<int> tree;
MaxSegmentTree(int n_) : n(n_) {
int size = (int)(ceil(log2(n)));
size = (2 * pow(2, size)) - 1;
tree = vector<int>(size);
}
int max_value() { return tree[0]; }
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[4,2,1,4,3,4,5,8,15]
k =
3