**ASSIGNMENT -1 (ADVANCED PROGRAMMING)**

1. **Problem 1:** Longest Nice Substring
2. **Implementation/Code:**

class Solution {

public:

    string longestNiceSubstring(string s, int start = 0, int end = -1) {

        if (end == -1)

            end = s.size();

        int cnt[26][2] = {}, j = start - 1;

        for (auto i = start; i < end; ++i)

            cnt[tolower(s[i]) - 'a'][(bool)islower(s[i])] = 1;

        string res;

        for (auto i = start; i <= end; ++i) {

            int ch = i == end ? -1 : tolower(s[i]) - 'a';

            if (ch == -1 || cnt[ch][0] + cnt[ch][1] == 1) {

                if (j == -1 && ch == -1)

                    return s;

                auto res1 = longestNiceSubstring(s.substr(j + 1, i - j - 1));

                if (res1.size() > res.size())

                    res = res1;

                j = i;

            }

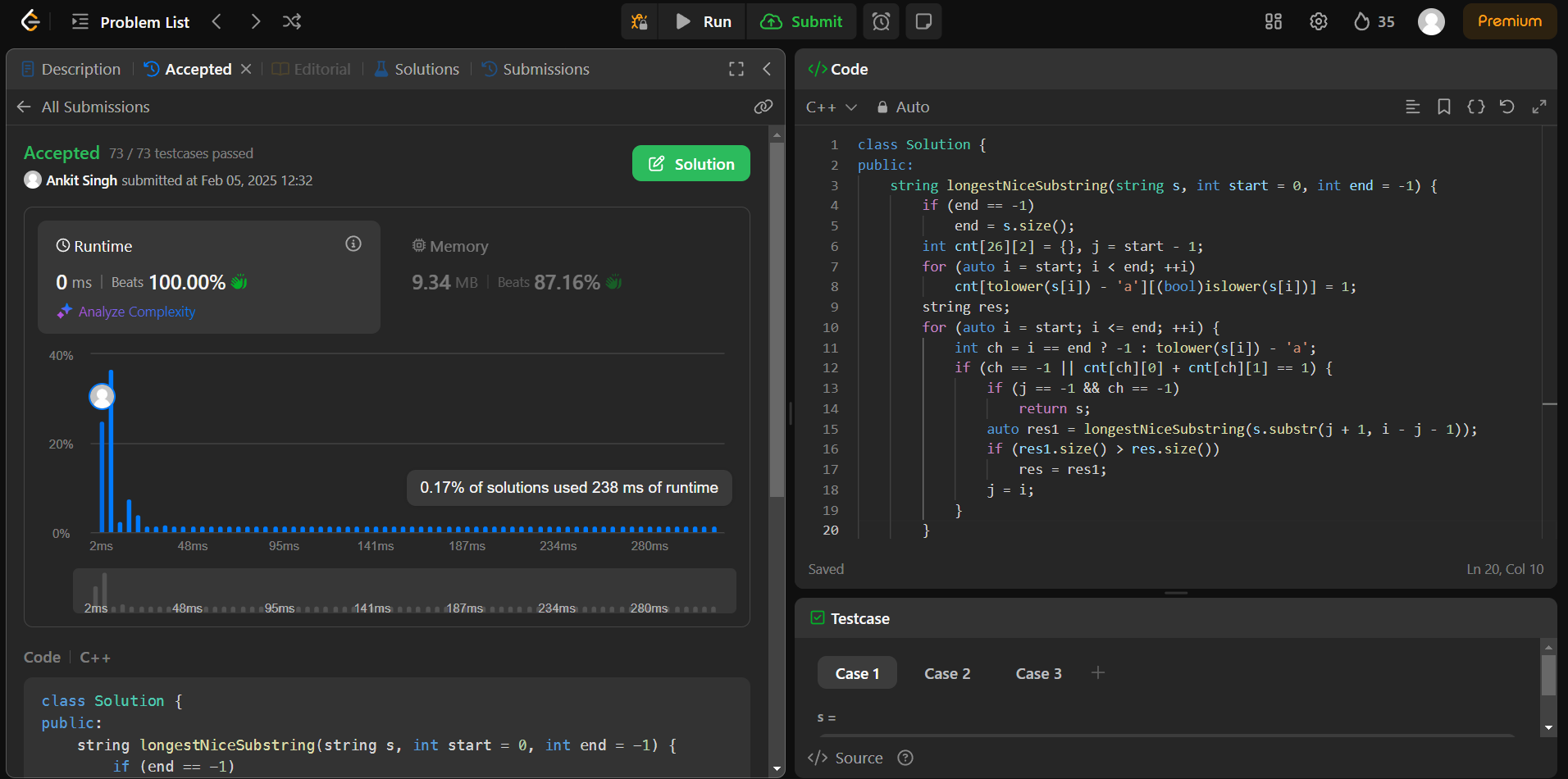
        }

        return res;

    }

};

**Output:**

****

1. **Problem 2:** Reverse Bits
2. **Implementation/Code:**

class Solution {

public:

    uint32\_t reverseBits(uint32\_t n) {

        uint32\_t result = 0;

        int shift = 31;

        while (n > 0) {

            result ^= ((n % 2) << shift);

            n >>= 1;

            --shift;

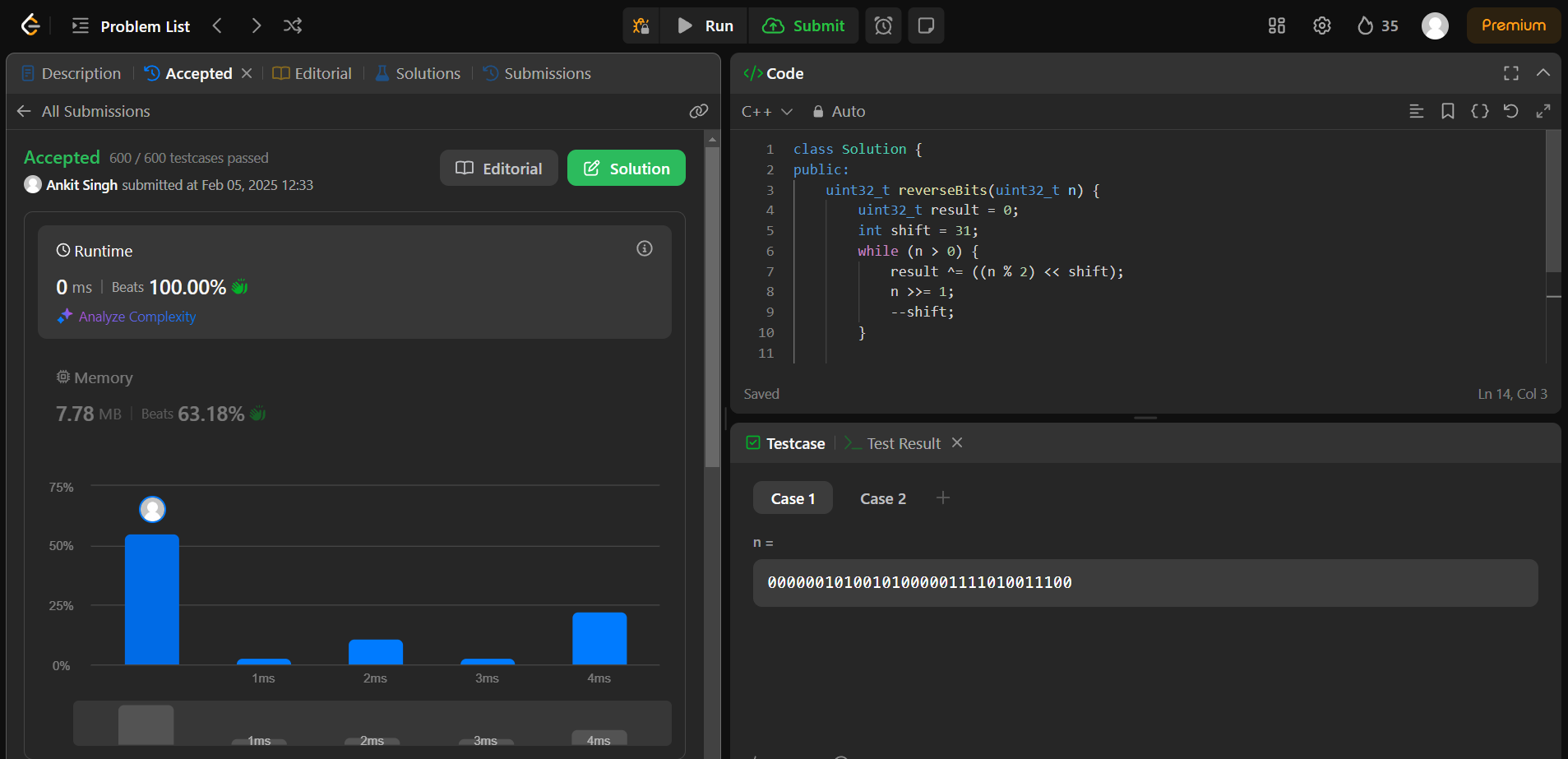
        }

        return result;

    }

};

1. **Output:**

****

1. **Problem 3:** Number of 1 bits
2. **Implementation/code:**

class Solution {

public:

    int hammingWeight(int n) {

        int cnt=0,rem;

        while(n/2!=0){

            if(n%2==1){

                cnt++;

            }

            n=n/2;

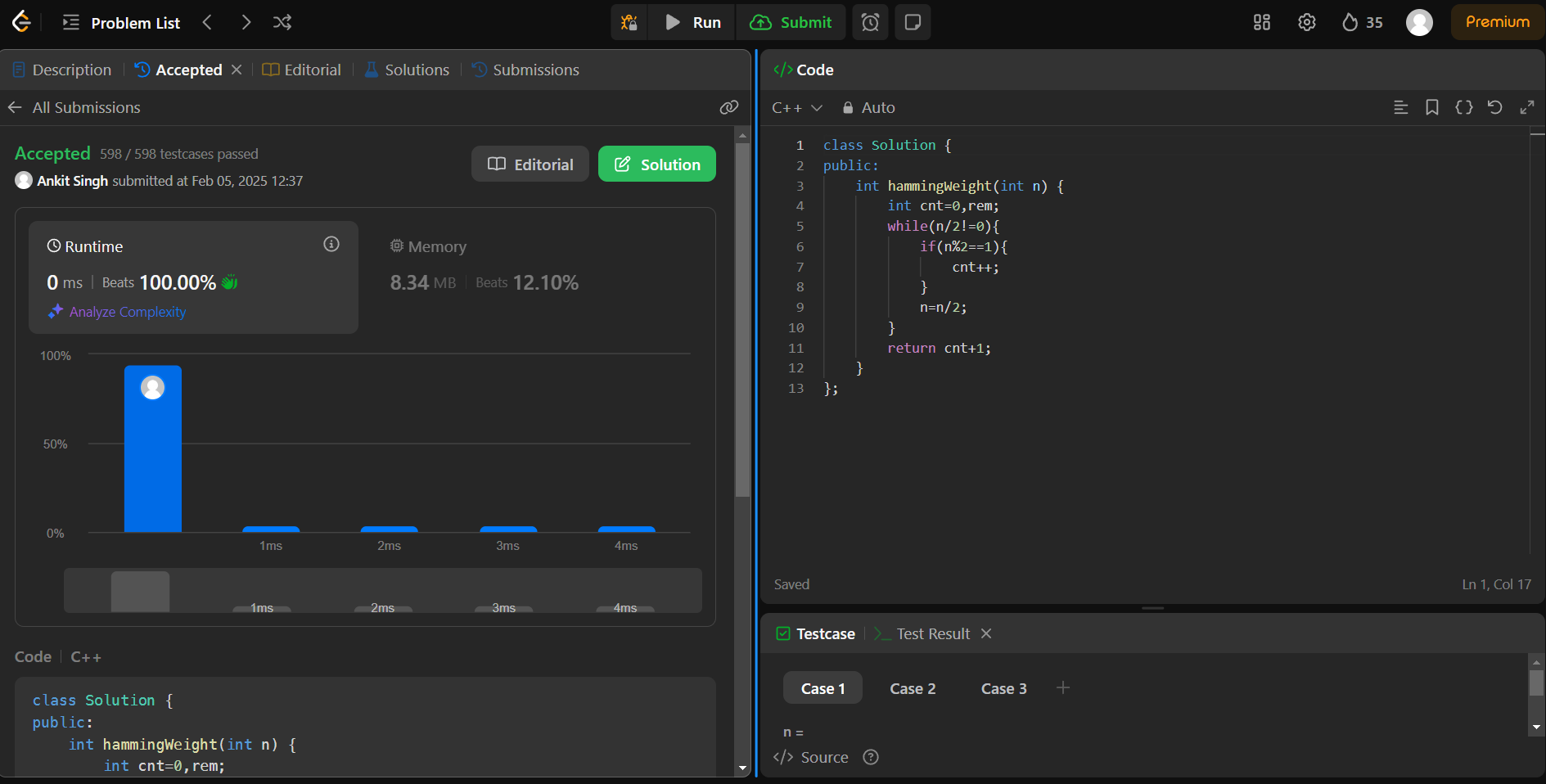
        }

        return cnt+1;

    }

};

1. **Output:**

****

1. **Problem 4: Maximum Sub array**
2. **Implementation/code:**

class Solution {

public:

    int maxSubArray(vector<int>& nums) {

        vector<vector<int>> dp(2, vector<int>(size(nums), -1));

        return solve(nums, 0, false, dp);

    }

    int solve(vector<int>& A, int i, bool mustPick, vector<vector<int>>& dp) {

        if(i >= size(A)) return mustPick ? 0 : -1e5;

        if(dp[mustPick][i] != -1) return dp[mustPick][i];

        if(mustPick)

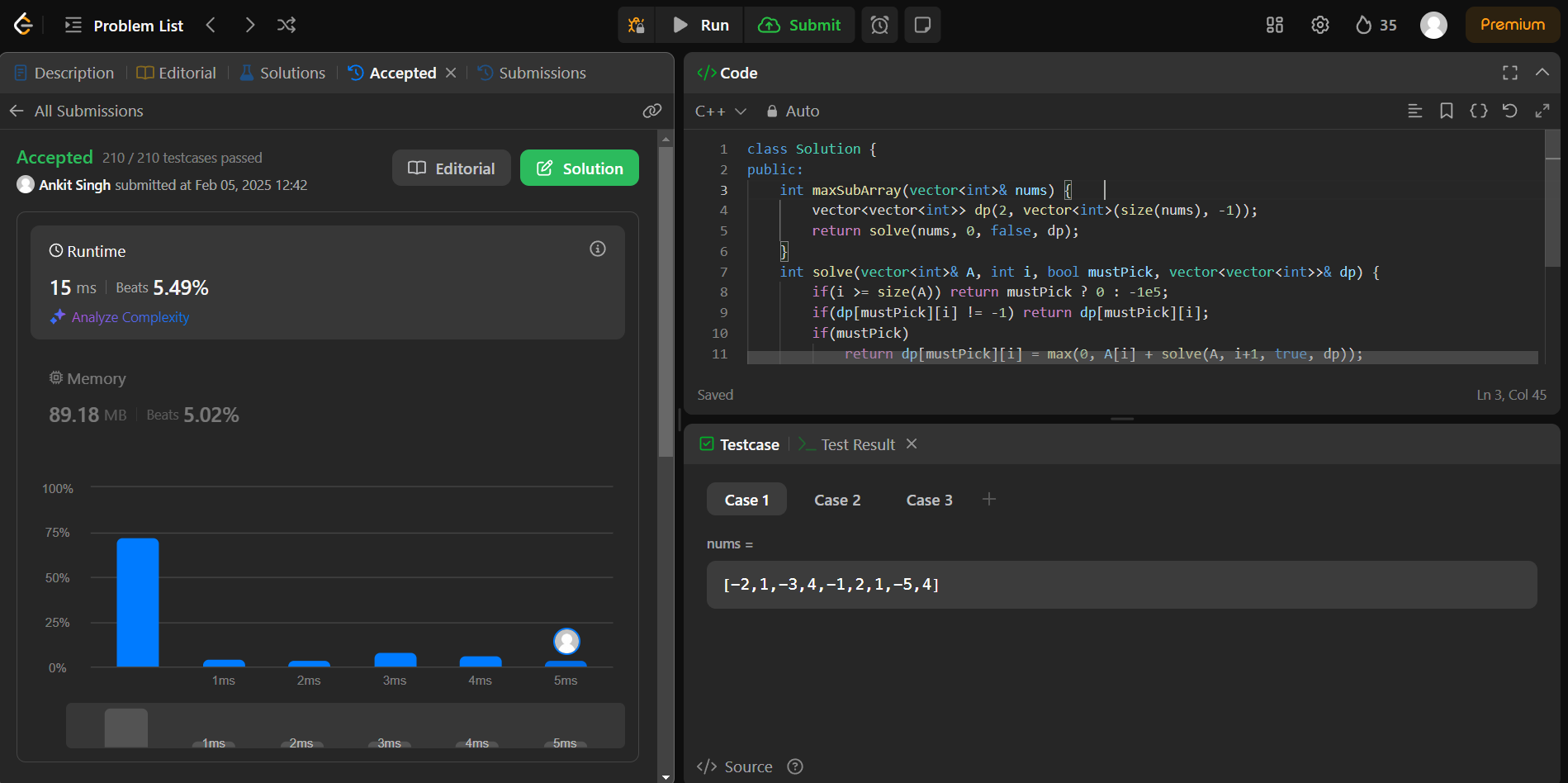
            return dp[mustPick][i] = max(0, A[i] + solve(A, i+1, true, dp));

        return dp[mustPick][i] = max(solve(A, i+1, false, dp), A[i] + solve(A, i+1, true, dp));

    }

};

1. **Output:**

****

1. **Problem 5: Search a 2D Matrix II**
2. **Implementation/Code:**

class Solution {

public:

    bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int cols = matrix[0].size() - 1;

        int n = matrix.size() - 1;

        int rows = 0;

        while(rows <= n && cols >= 0){

            int toCompare = matrix[rows][cols];

            if(toCompare > target){

                cols--;

            }else if(toCompare < target){

                rows++;

            }else{

                return true;

            }

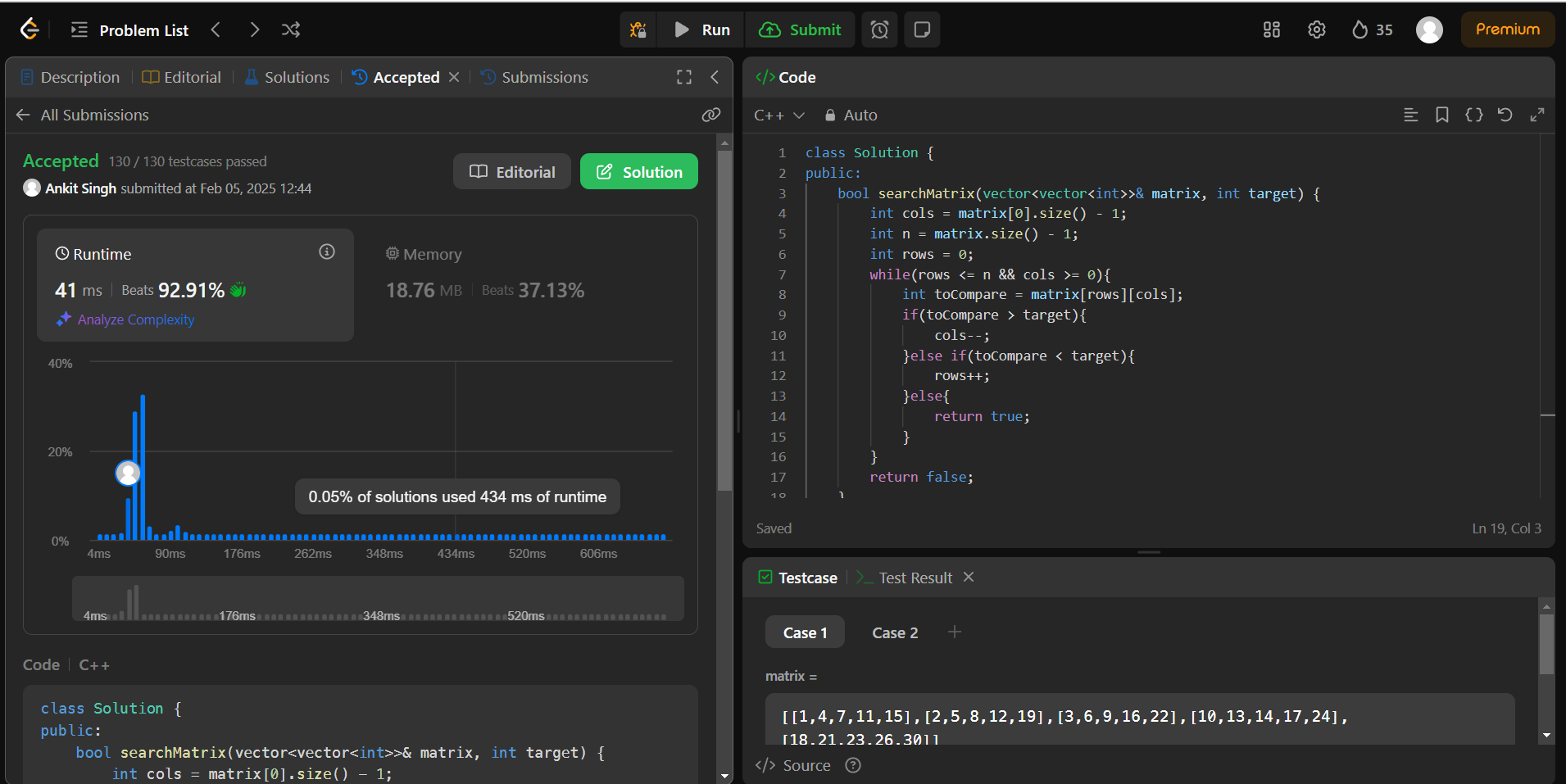
        }

        return false;

    }

};

1. **Output:**

****

1. **Problem 6: Super Pow**
2. **Implementation/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();

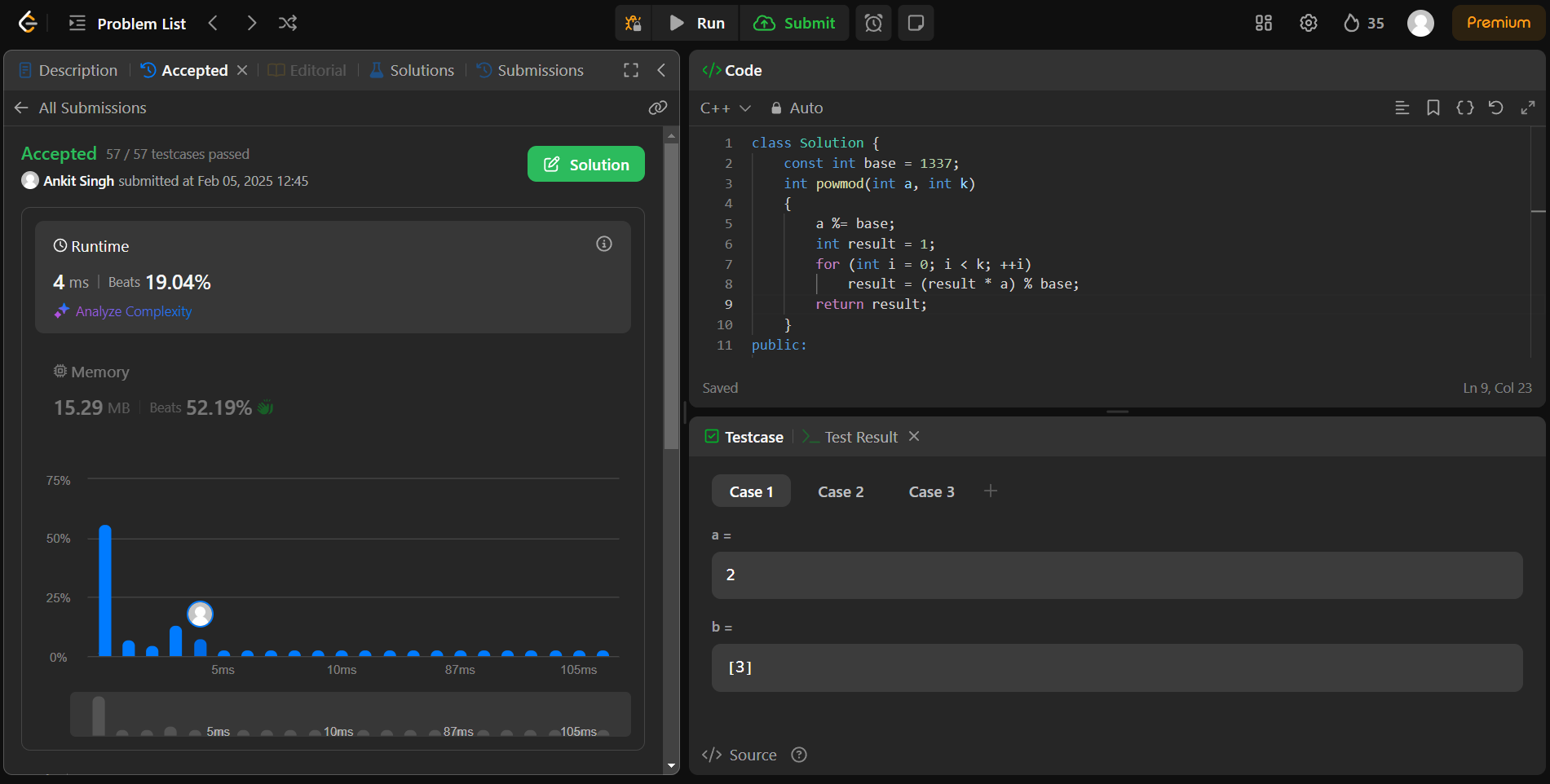
        b.pop\_back();

        return powmod(superPow(a, b), 10) \* powmod(a, last\_digit) % base;

    }

};

1. **Output:**

****

1. **Problem 7: Beautiful Array**
2. **Implementation/code:**

    vector<int> beautifulArray(int N) {

        vector<int> res = {1};

        while (res.size() < N) {

            vector<int> tmp;

            for (int i : res) if (i \* 2 - 1 <= N) tmp.push\_back(i \* 2 - 1);

            for (int i : res) if (i \* 2 <= N) tmp.push\_back(i \* 2);

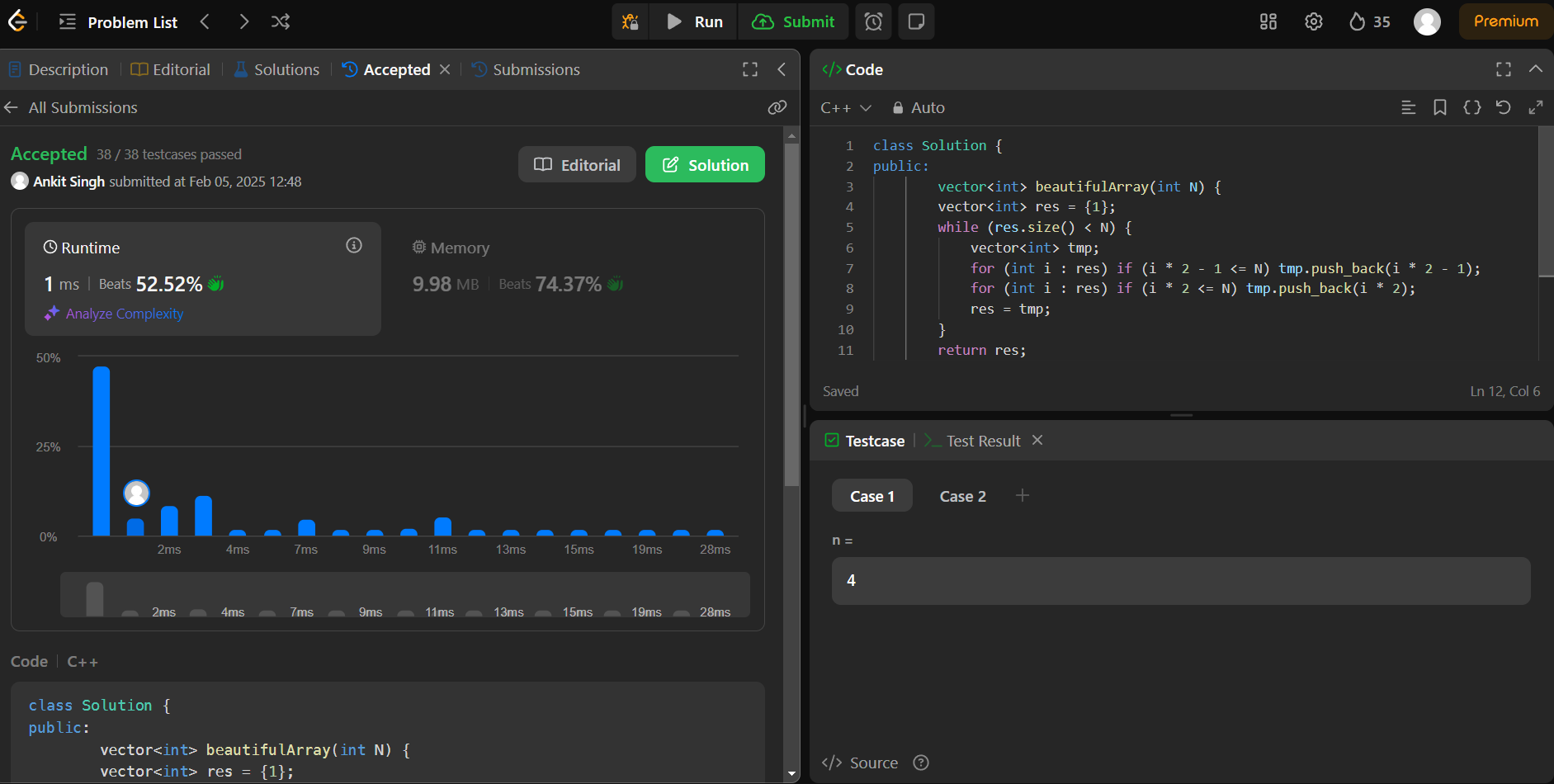
            res = tmp;

        }

        return res;

    }

1. **Output:**

****

1. **Problem 8: The Skyline Problem.**
2. **Implementation/code:**

class Solution {

public:

    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

        int edge\_idx = 0;

        vector<pair<int, int>> edges;

        priority\_queue<pair<int, int>> pq;

        vector<vector<int>> skyline;

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

            const auto &b = buildings[i];

            edges.emplace\_back(b[0], i);

            edges.emplace\_back(b[1], i);

        }

        std::sort(edges.begin(), edges.end());

        while (edge\_idx < edges.size()) {

            int curr\_height;

            const auto &[curr\_x, \_] = edges[edge\_idx];

            while (edge\_idx < edges.size() &&

                    curr\_x == edges[edge\_idx].first) {

                const auto &[\_, building\_idx] = edges[edge\_idx];

                const auto &b = buildings[building\_idx];

                if (b[0] == curr\_x)

                    pq.emplace(b[2], b[1]);

                ++edge\_idx;

            }

            while (!pq.empty() && pq.top().second <= curr\_x)

                pq.pop();

            curr\_height = pq.empty() ? 0 : pq.top().first;

            if (skyline.empty() || skyline.back()[1] != curr\_height)

                skyline.push\_back({curr\_x, curr\_height});

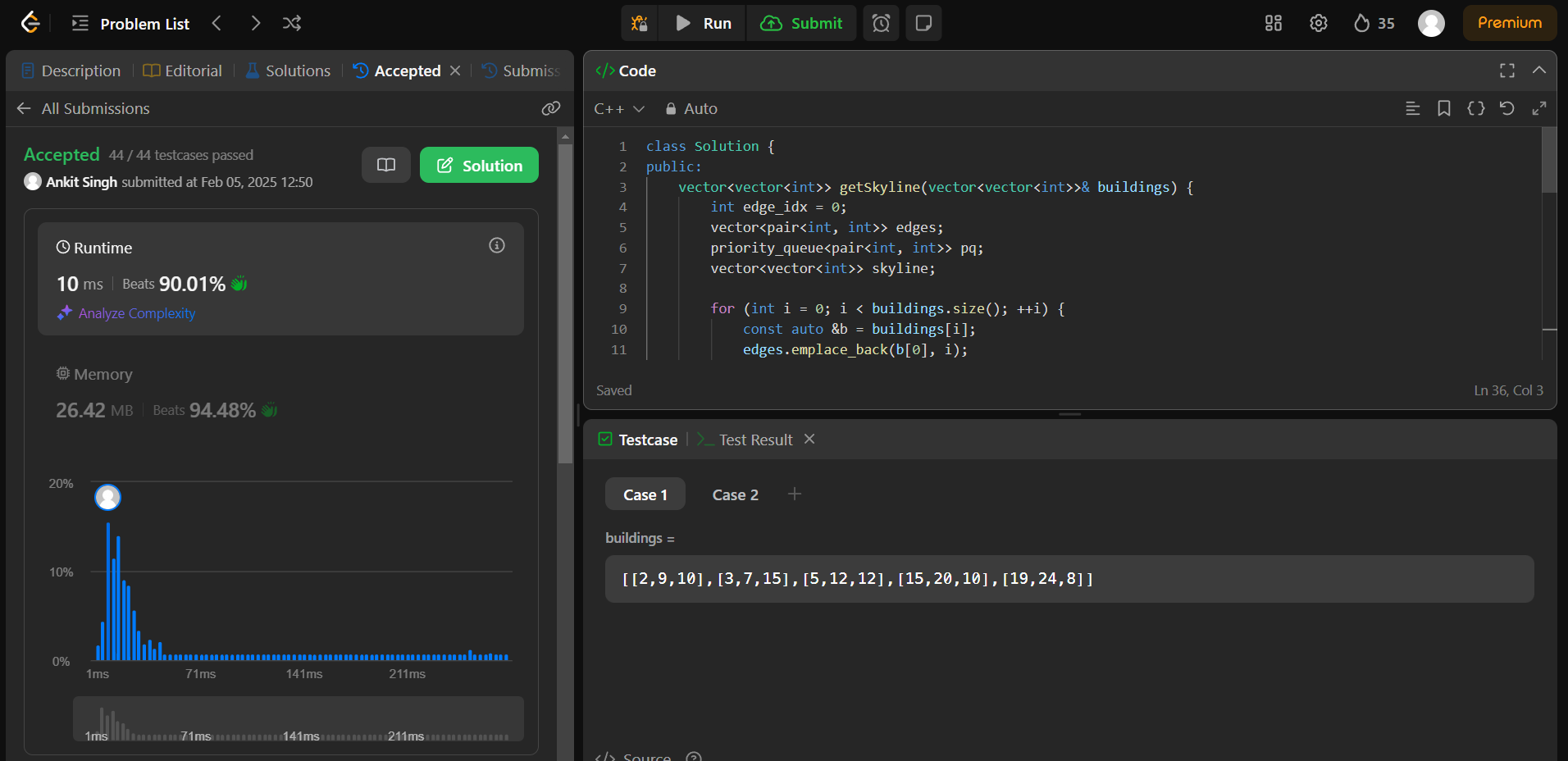
        }

        return skyline;

    }

};

1. **Output:**

****

1. **Problem 9: Reverse Pairs**
2. **Implementation/code:**

class Solution {

public:

    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;

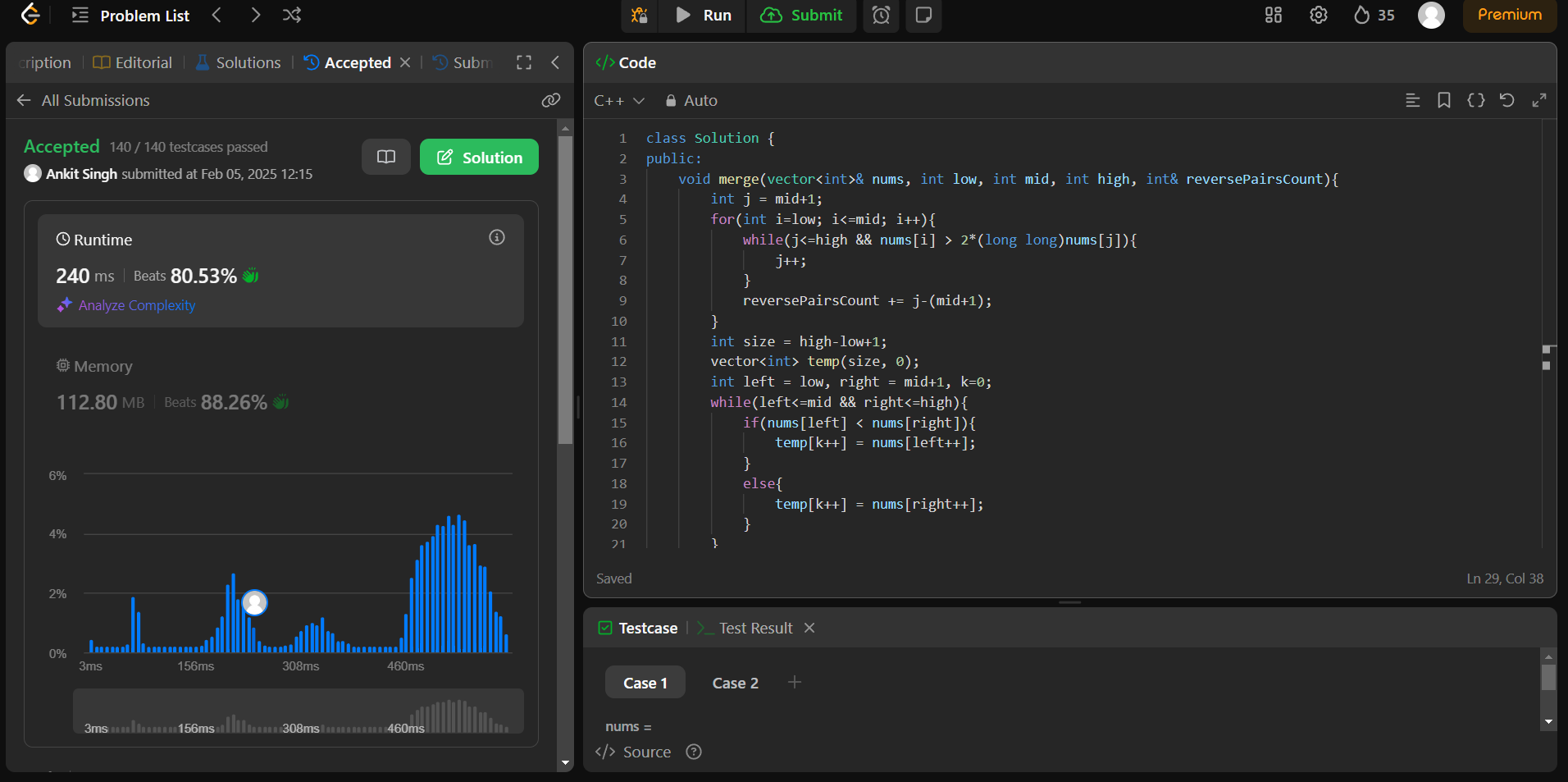
        mergeSort(nums, 0, nums.size()-1, reversePairsCount);

        return reversePairsCount;

    }

};

1. **Output:**

****

1. **Problem 10: Longest Increasing SubSequence**
2. **Code:**

class Solution {

public:

    vector<int> seg;

    void upd(int ind, int val, int x, int lx, int rx) {

        if(lx == rx) {

            seg[x] = val;

            return;

        }

        int mid = lx + (rx - lx) / 2;

        if(ind <= mid)

            upd(ind, val, 2 \* x + 1, lx, mid);

        else

            upd(ind, val, 2 \* x + 2, mid + 1, rx);

        seg[x] = max(seg[2 \* x + 1], seg[2 \* x + 2]);

    }

    int query(int l, int r, int x, int lx, int rx) {

        if(lx > r or rx < l) return 0;

        if(lx >= l and rx <= r) return seg[x];

        int mid = lx + (rx - lx) / 2;

        return max(query(l, r, 2 \* x + 1, lx, mid), query(l, r, 2 \* x + 2, mid + 1, rx));

    }

    int lengthOfLIS(vector<int>& nums, int k) {

        int x = 1;

        while(x <= 200000) x \*= 2;

        seg.resize(2 \* x, 0);

        int res = 1;

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

            int left = max(1, nums[i] - k), right = nums[i] - 1;

            int q = query(left, right, 0, 0, x - 1);

            res = max(res, q + 1);

            upd(nums[i], q + 1, 0, 0, x - 1);

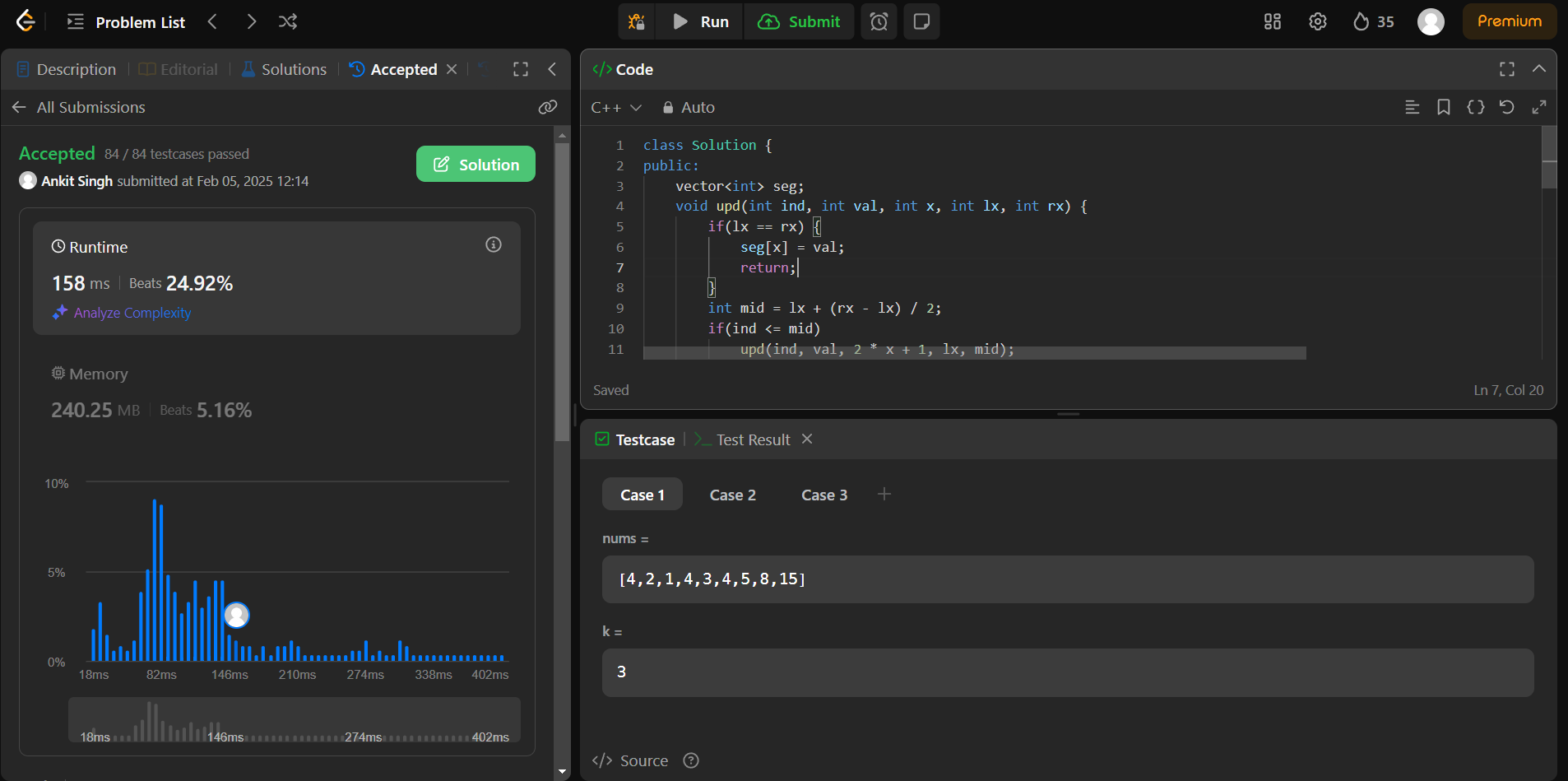
        }

        return res;

    }

};

1. **Output:**

****

1. **Problem 11: Merge Sorted Array**
2. **Code:**

class Solution {

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        int i = m - 1;

        int j = n - 1;

        int k = m + n - 1;

        while (i >= 0 && j >= 0) {

            if (nums1[i] > nums2[j]) {

                nums1[k--] = nums1[i--];

            } else {

                nums1[k--] = nums2[j--];

            }

        }

        while (j >= 0) {

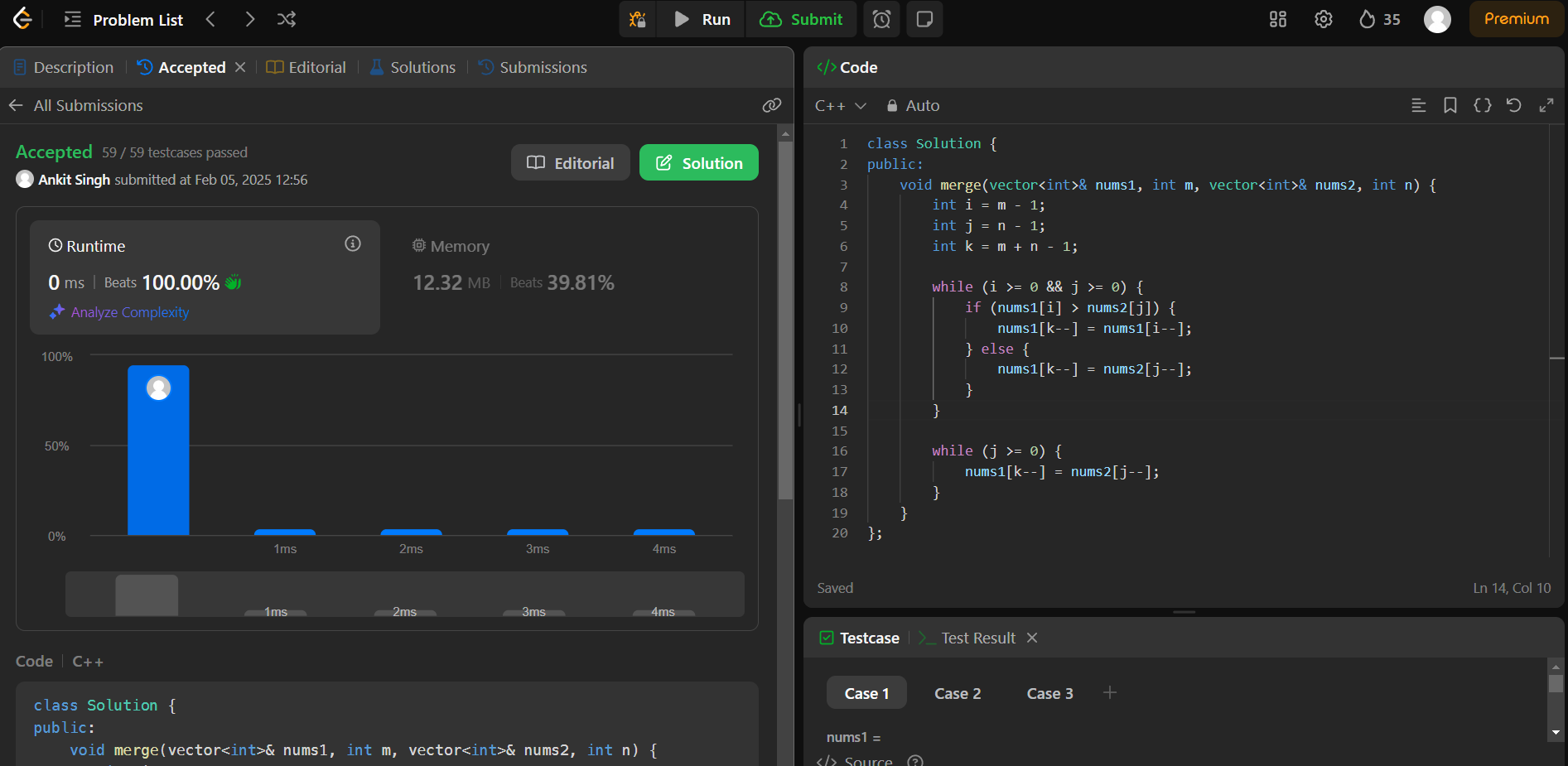
            nums1[k--] = nums2[j--];

        }

    }

};

1. **Output:**

****

1. **Problem 12: First Bad Version**
2. **Code:**

// The API isBadVersion is defined for you.

// bool isBadVersion(int version);

class Solution {

public:

    int firstBadVersion(int n) {

        if (isBadVersion(1)) {

            return 1;

        }

        long long int l = 1, up = n;

        long long int mid = (l + up) / 2;

        while (l < up) {

            if (isBadVersion(mid)) {

                up = mid;

            } else {

                l = mid + 1;

            }

            mid = (l + up) / 2;

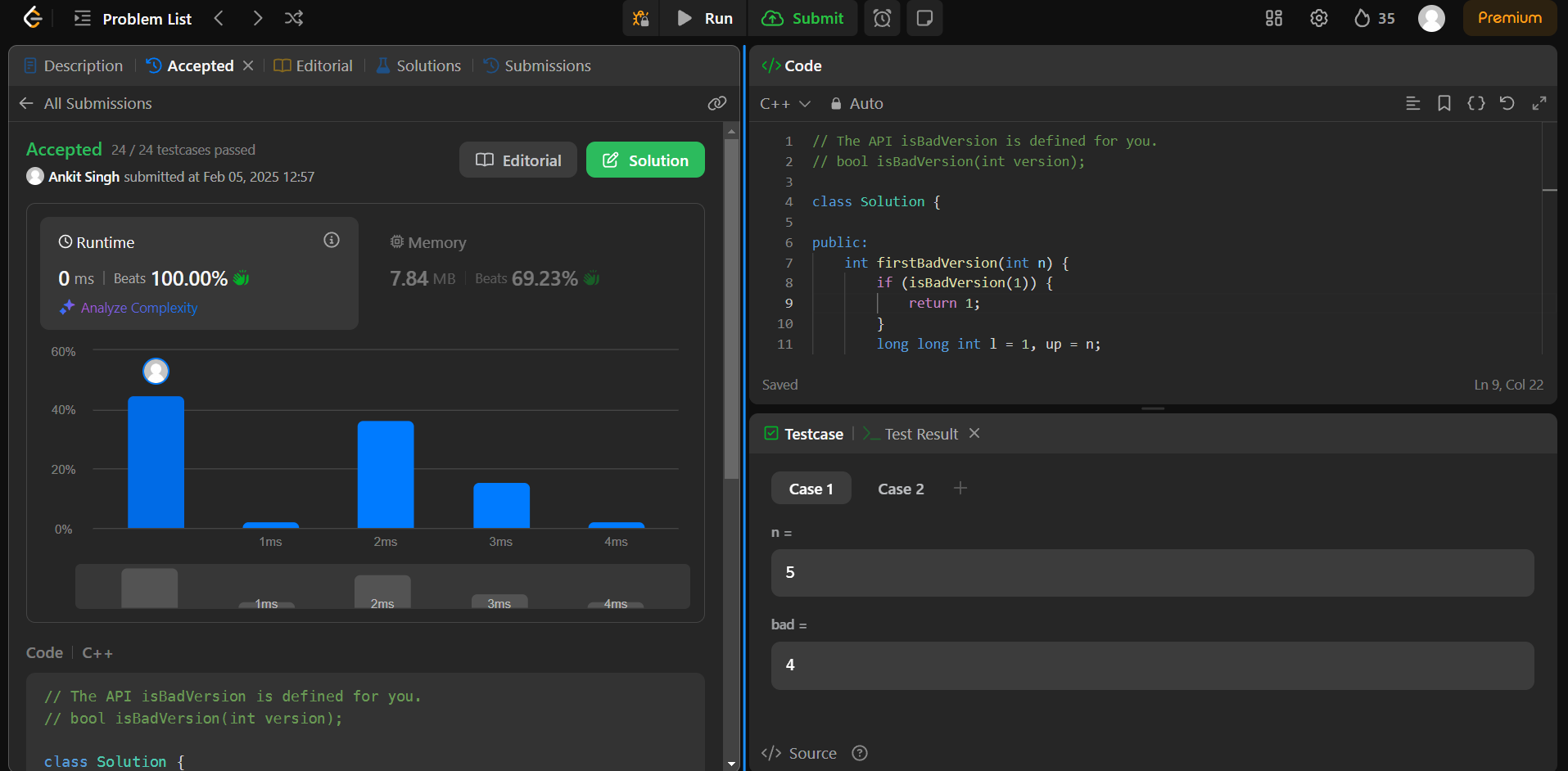
        }

        return mid;

    }

};

1. **Output:**

****

1. **Problem 13: Sort Colors**
2. **Code:**

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int i = 0, j = nums.size() - 1, k = 0;

        while (k <= j) {

            if (nums[k] == 2) {

                swap(nums[k], nums[j]);

                j--;

            }

            else if (nums[k] == 0) {

                swap(nums[k], nums[i]);

                k++;i++;

            }

            else {

                k++;

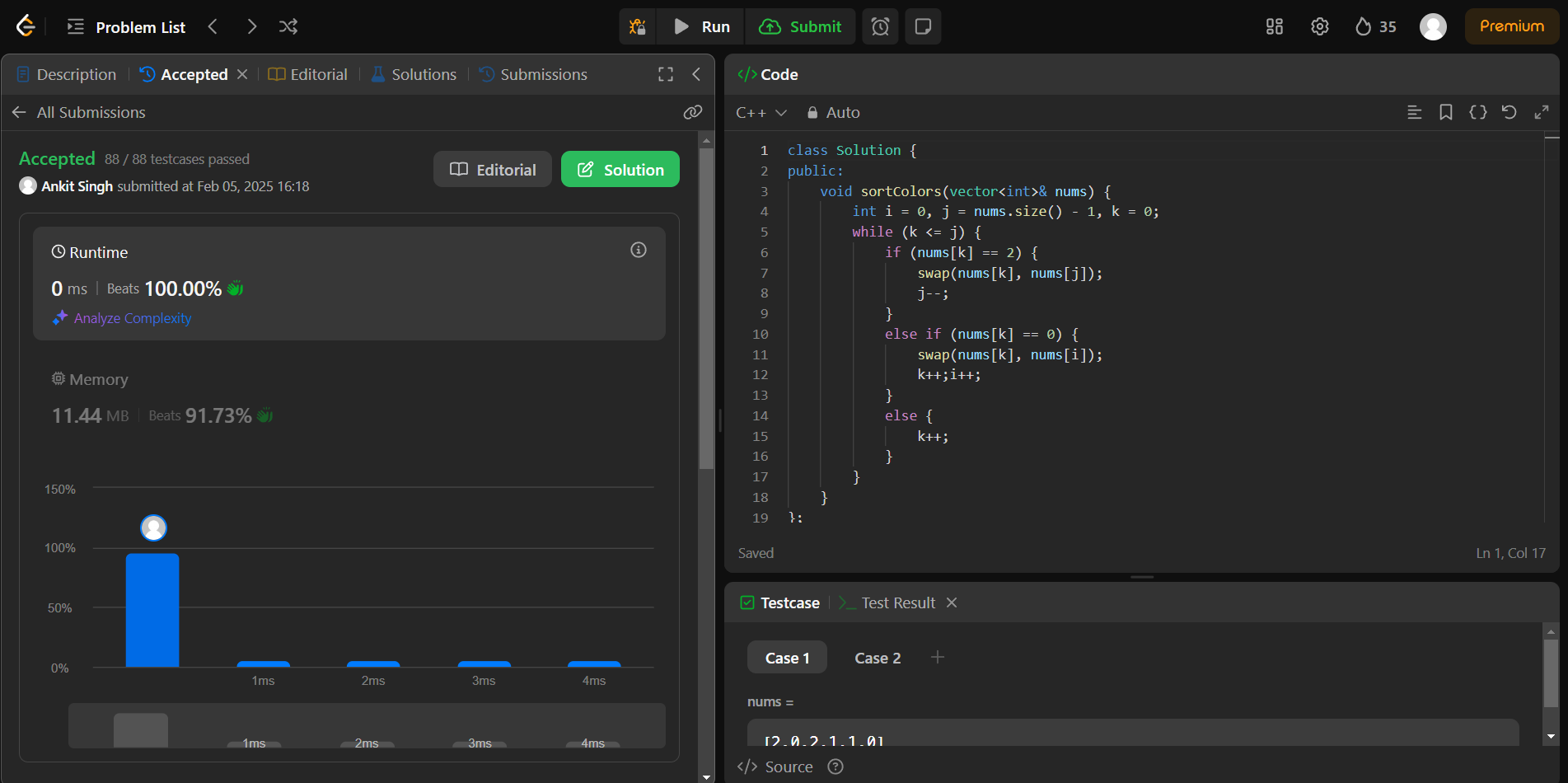
            }

        }

    }

};

1. **Output:**

****

1. **Problem 14: Top K frequent Elements**
2. **Code:**

class Solution {

public:

    vector<int> topKFrequent(vector<int>& nums, int k) {

        unordered\_map<int,int> mp;

        for(auto x:nums){

            mp[x]++;

        }

        priority\_queue<pair<int,int>> p;

        for(auto x:mp){

            p.push({x.second,x.first});

        }

        vector<int> ans;

        for(int i=0;i<k;i++){

            ans.push\_back(p.top().second);

            p.pop();

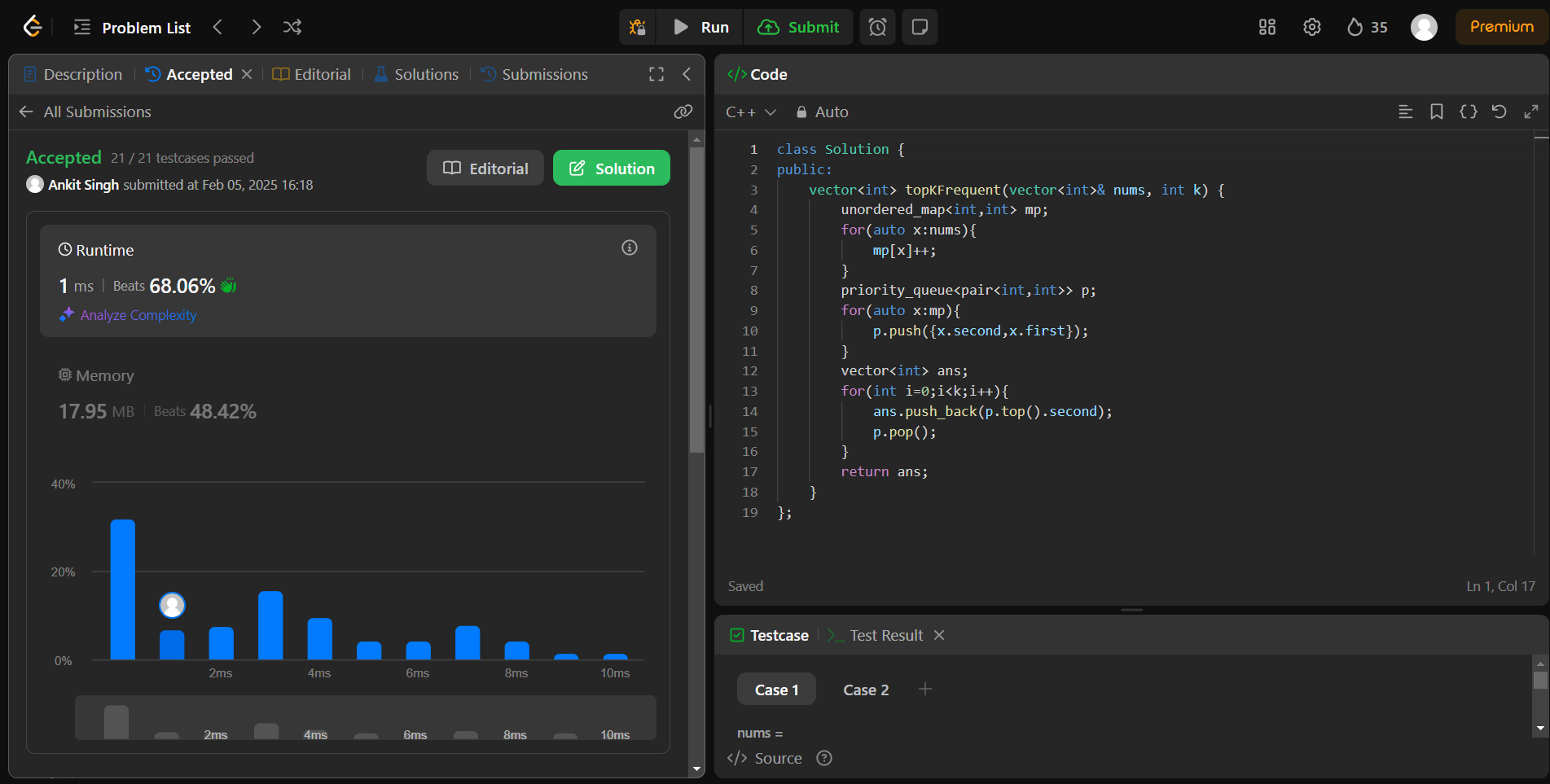
        }

        return ans;

    }

};

1. **Output:**

****

1. **Problem 15:**
2. **Code:**

class Solution {

public:

    int findKthLargest(vector<int>& nums, int k) {

        priority\_queue<int> q;

        for(auto n:nums){

            q.push(n);

        }

        for(int i=1;i<k;i++){

            q.pop();

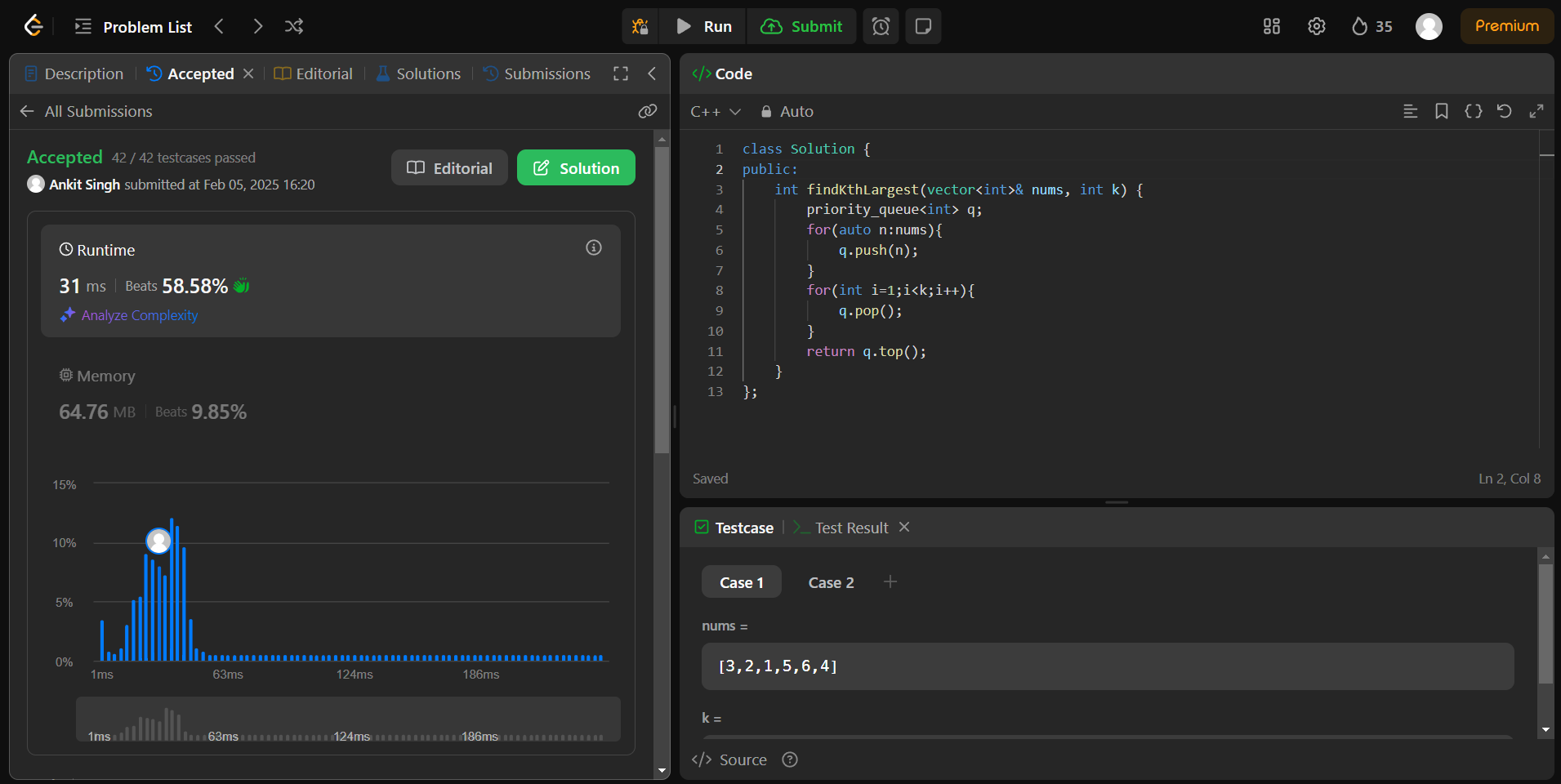
        }

        return q.top();

    }

};

1. **Output:**

****

1. **Problem 16: Find Peak Element**
2. **Code:**

class Solution {

public:

    int findPeakElement(vector<int>& nums) {

        int left = 0;

        int right = nums.size() - 1;

        while (left < right) {

            int mid = left + (right - left) / 2;

            if (nums[mid] > nums[mid + 1]) {

                right = mid;

            } else {

                left = mid + 1;

            }

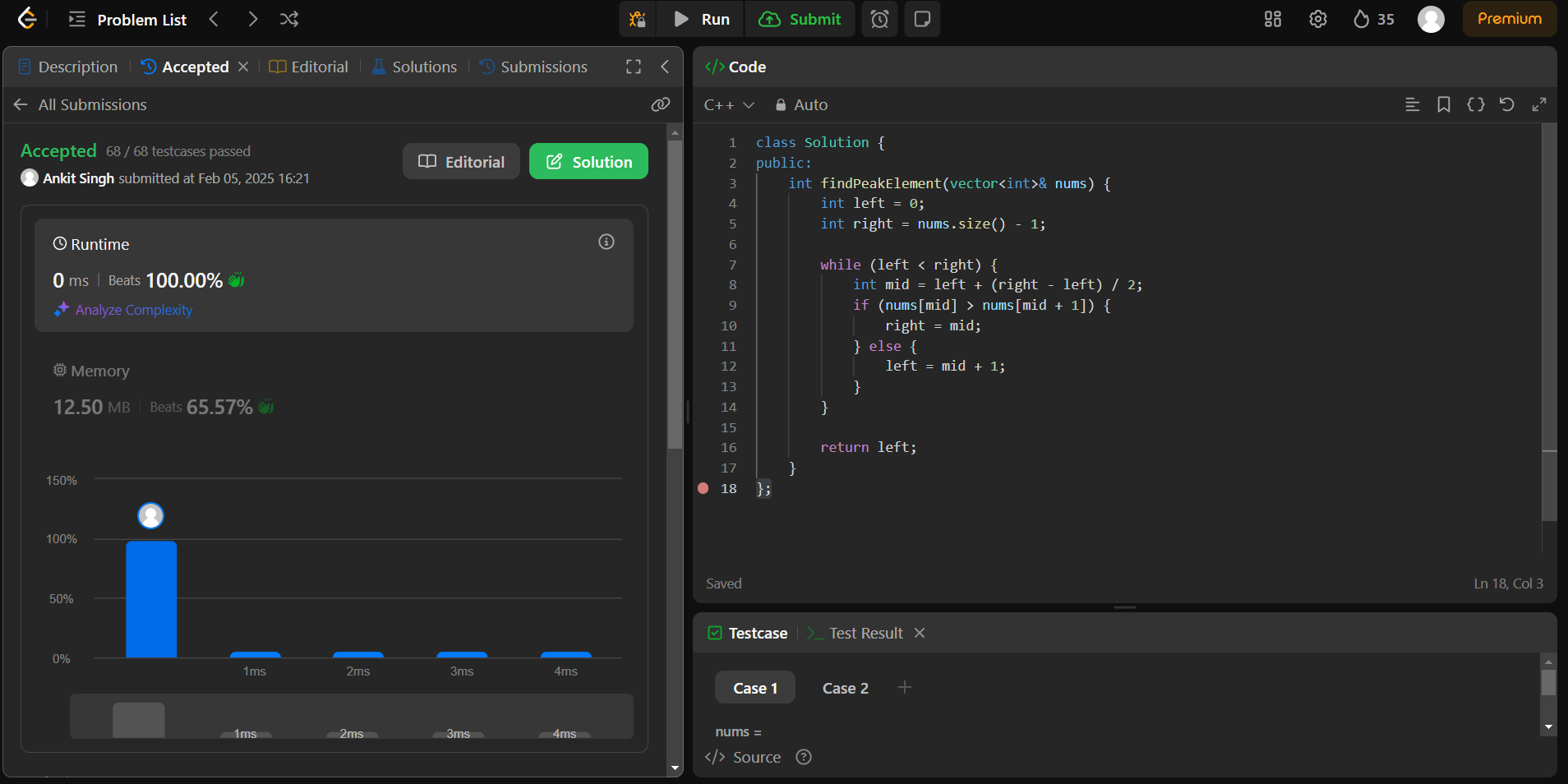
        }

        return left;

    }

};

1. **Output:**

****

1. **Problem 17: Merge Intervals**
2. **Code:**

class Solution {

public:

    vector<vector<int>> merge(vector<vector<int>>& intervals) {

        sort(intervals.begin(), intervals.end());

        vector<vector<int>> ans;

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

            if(ans.empty() || (ans.back()[1]<intervals[i][0])){

                ans.push\_back(intervals[i]);

            }else{

                ans.back()[1]=max(ans.back()[1],intervals[i][1]);

            }

        }

        return ans;

        /\*\*sort(intervals.begin(), intervals.end());

        for(int i=1;i<intervals.size();i++){

            if(intervals[i-1][1]>=intervals[i][0]){

                intervals[i-1][1]=max(intervals[i][1],intervals[i-1][1]) ;

                intervals.erase(intervals.begin()+i);

                i--;

            }

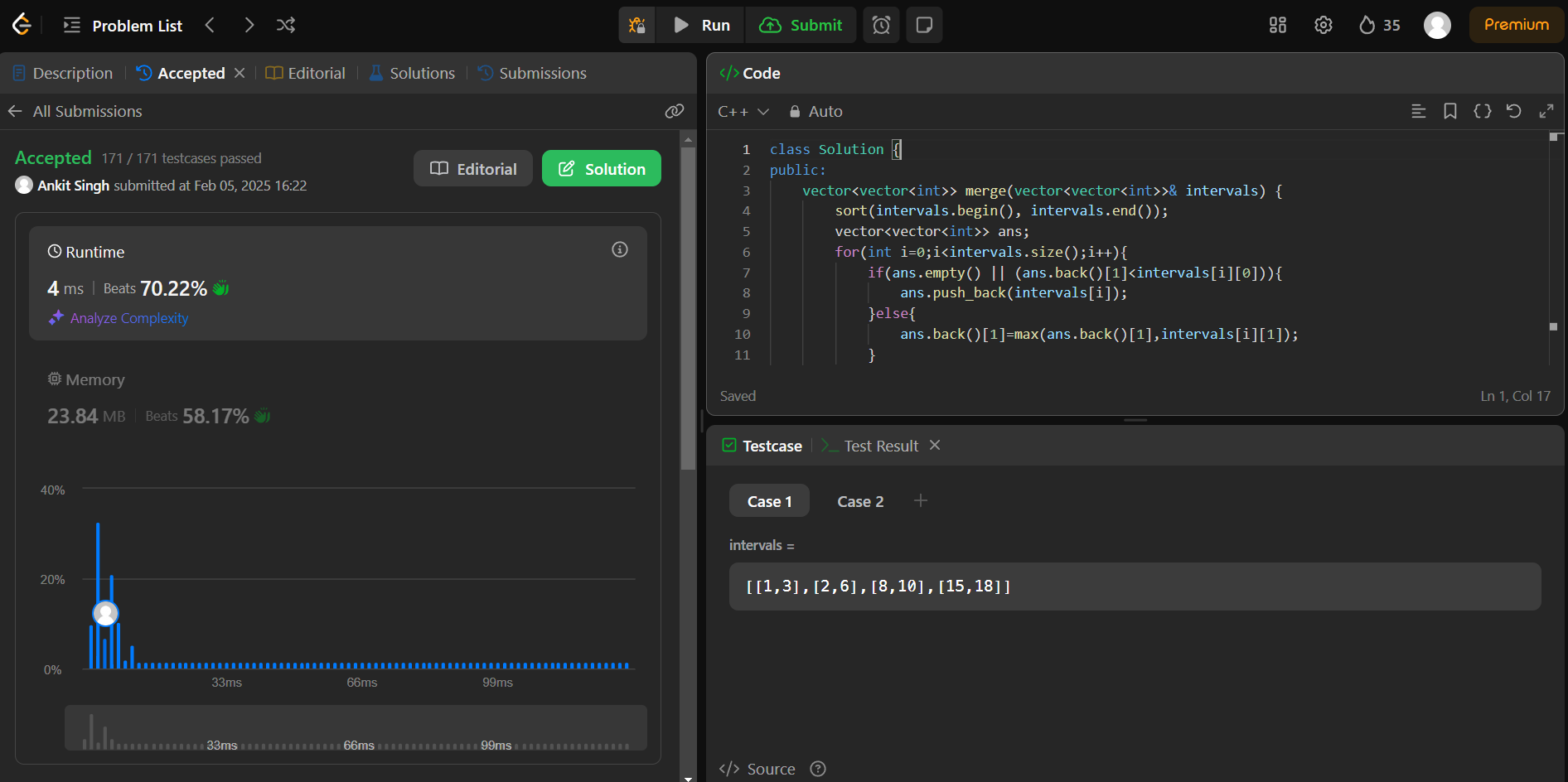
        }

        return intervals;\*\*/

    }

};

1. **Output:**

****

1. **Problem 18: Search in rotated Sorted Array**
2. **Code:**

class Solution {

public:

    int search(vector<int>& nums, int target) {

        int left = 0;

        int right = nums.size() - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

            if (nums[mid] == target) {

                return mid;

            } else if (nums[mid] >= nums[left]) {

                if (nums[left] <= target && target <= nums[mid]) {

                    right = mid - 1;

                } else {

                    left = mid + 1;

                }

            } else {

                if (nums[mid] <= target && target <= nums[right]) {

                    left = mid + 1;

                } else {

                    right = mid - 1;

                }

            }

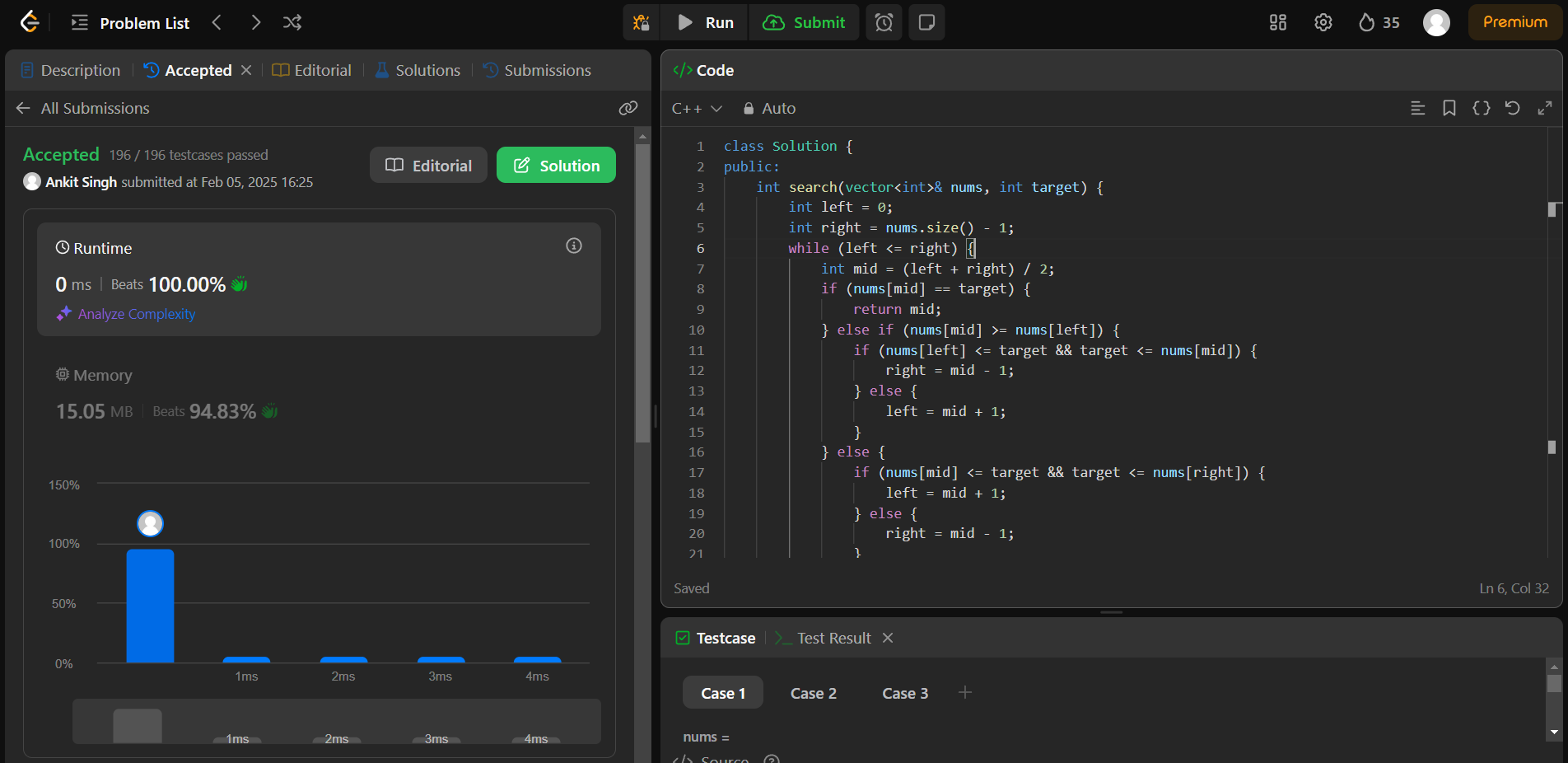
        }

        return -1;

    }

};

1. **Output:**

****

1. **Problem 19: Wiggle Sort**
2. **Code:**

class Solution {

public:

    void wiggleSort(vector<int>& nums) {

        int n= nums.size();

        priority\_queue<int> q(nums.begin(), nums.end());

        for(int i=1; i<n; i += 2){

            nums[i] = q.top();

            q.pop();

        }

        for(int i=0; i<n; i += 2){

            nums[i] = q.top();

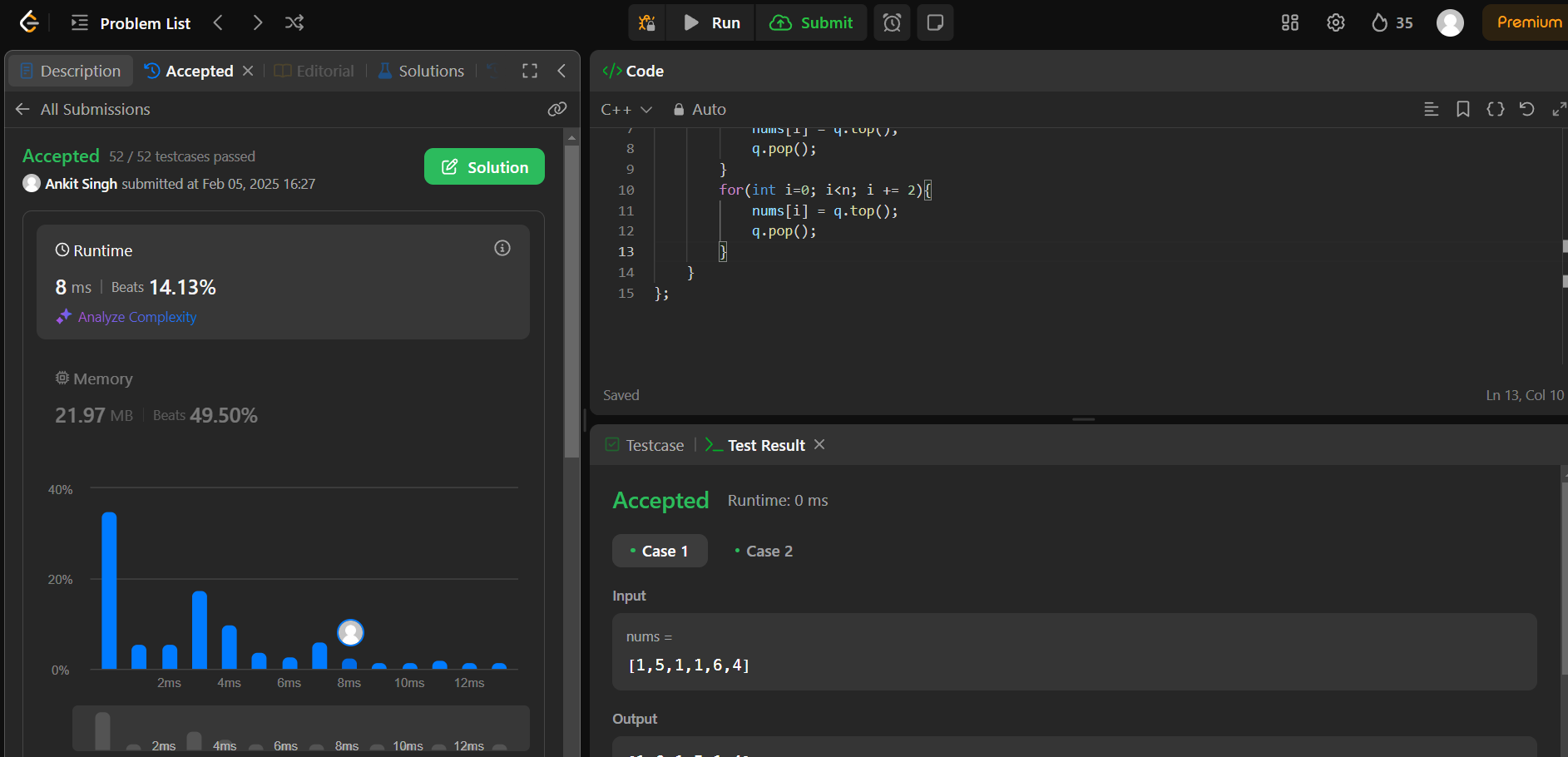
            q.pop();

        }

    }

};

1. **Output:**

****

1. **Problem 20: Kth Smallest Element**
2. **Code:**

class Solution {

public:

    int kthSmallest(vector<vector<int>>& matrix, int k) {

        int m=matrix.size();

        int n=matrix[0].size();

        priority\_queue<int> p;

        for(int i=0;i<m;i++){

            for(int j=0;j<n;j++){

                p.push(matrix[i][j]);

            }

        }

        int po=m\*n-k;

        for(int i=0;i<po;i++){

            p.pop();

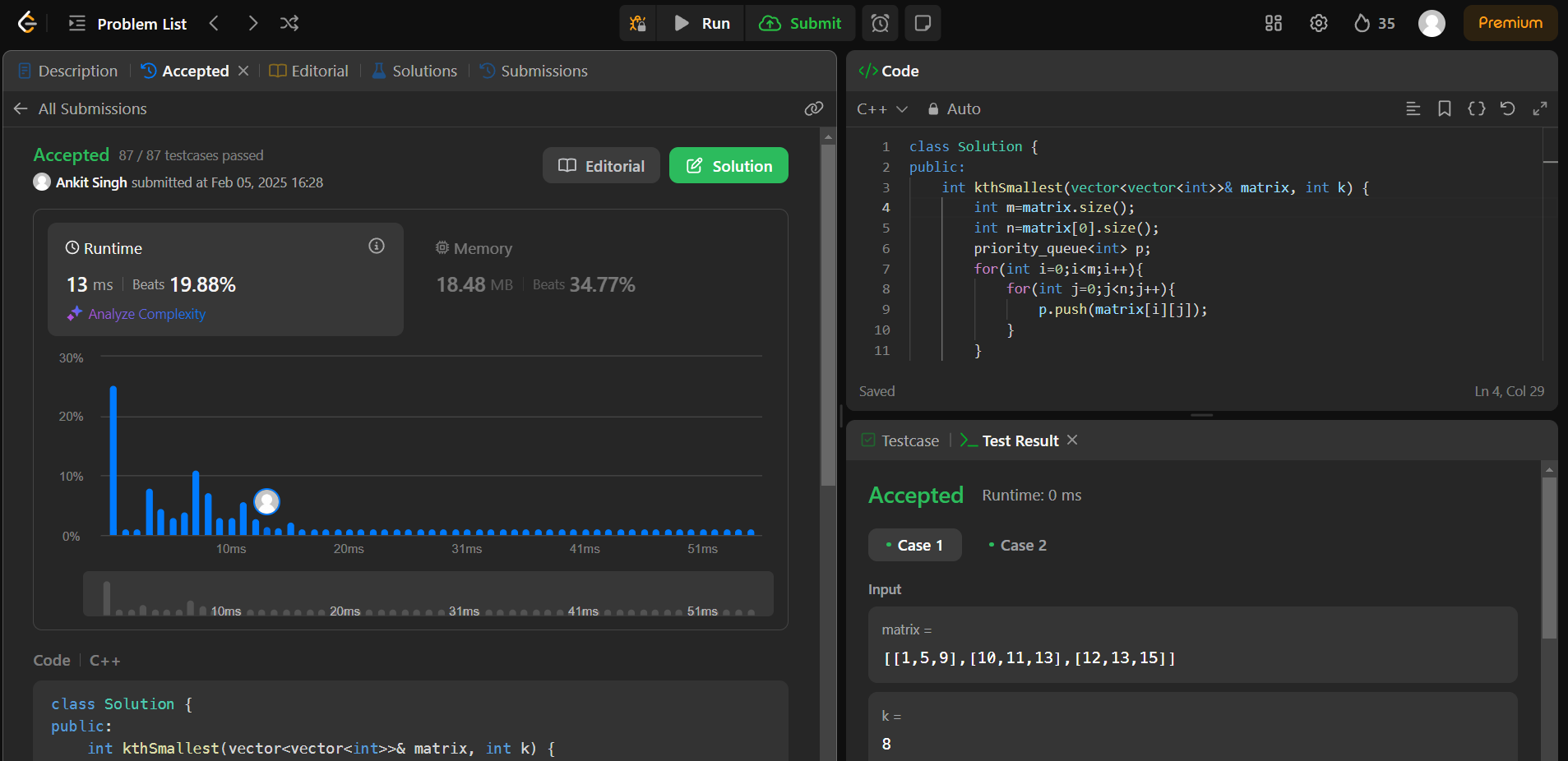
        }

        return p.top();

    }

};

1. **Output:**

****

1. **Problem 21: Median of Two Sorted Arrays.**
2. **Code:**

class Solution {

public:

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

        vector<int> nums3;

        nums3.resize(nums1.size() + nums2.size());

        int i = 0, j = 0, k = 0;

        while (i < nums1.size() && j < nums2.size()) {

            if (nums1[i] < nums2[j]) {

                nums3[k] = nums1[i];

                i++;

            } else {

                nums3[k] = nums2[j];

                j++;

            }

            k++;

        }

        while (i < nums1.size()) {

            nums3[k] = nums1[i];

            i++;

            k++;

        }

        while (j < nums2.size()) {

            nums3[k] = nums2[j];

            j++;

            k++;

        }

        int n = nums3.size();

        if (n % 2 == 1) {

            return static\_cast<double>(nums3[n / 2]);

        } else {

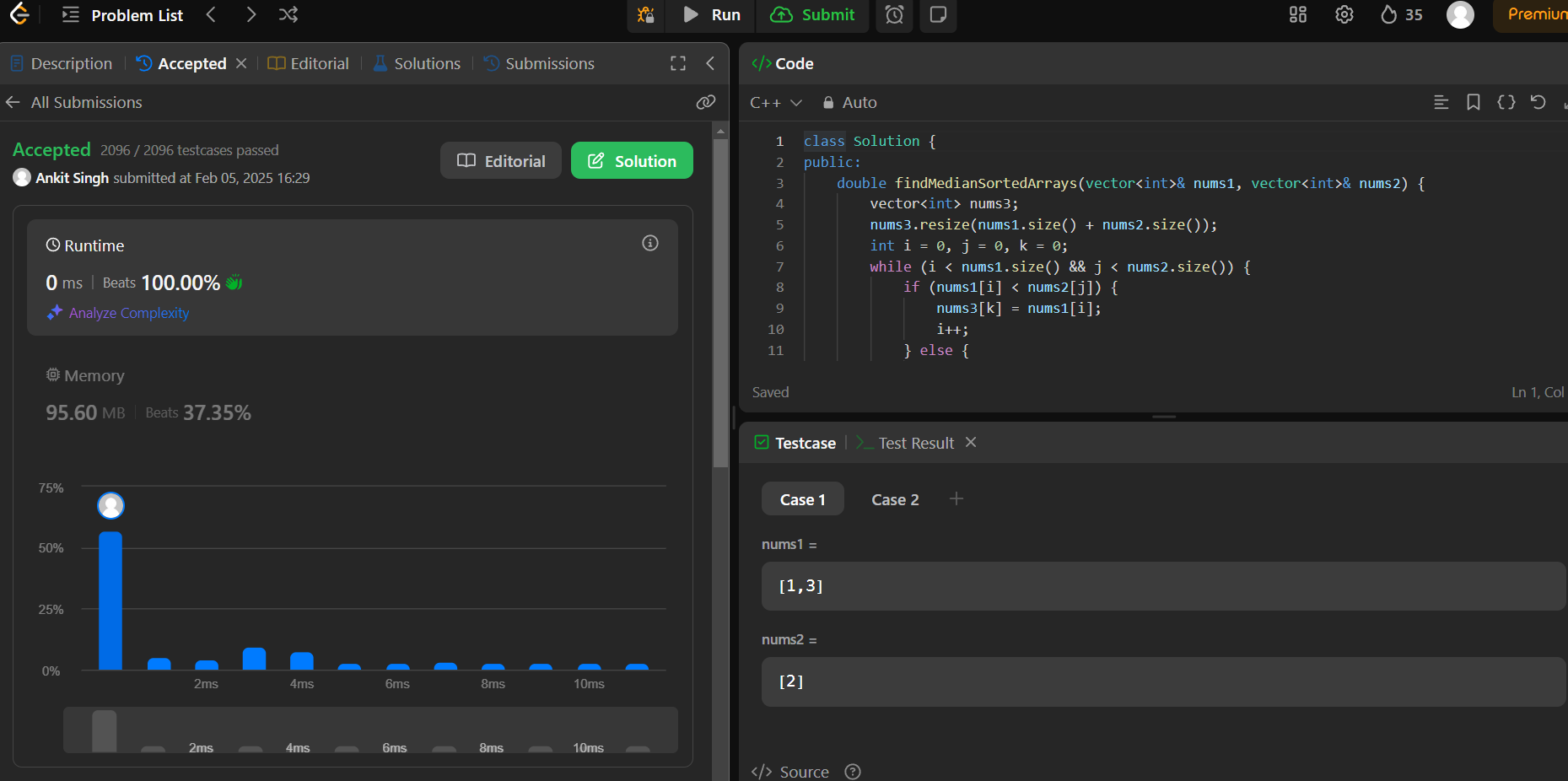
            return (static\_cast<double>(nums3[n / 2 - 1]) + static\_cast<double>(nums3[n / 2])) / 2.0;

        }

    }

};

1. **Output:**

****