1763. Longest Nice Substring

class Solution {

public:

    bool check(string s){

        vector<int> t1(26, 0), t2(26, 0);

        for(int i=0; i<s.length(); i++) if(s[i] >= 'a' && s[i] <= 'z') t1[s[i] - 'a']++;

        for(int i=0; i<s.length(); i++) if(s[i] >= 'A' && s[i] <= 'Z') t2[s[i] - 'A']++;

         for(int i=0; i<26; i++){

            if(t1[i] == 0 && t2[i] == 0) continue;

            else if(t1[i] == 0 && t2[i] > 0) return false;

            else if(t1[i] > 0 && t2[i] == 0) return false;

        }

        return true;

    }

    string longestNiceSubstring(string s) {

        if(s == "") return "";

        if(check(s)) return s;

        unordered\_map<char,int> m;

        vector<int> t1(26, 0), t2(26, 0);

        for(int i=0; i<s.length(); i++) if(s[i] >= 'a' && s[i] <= 'z') t1[s[i] - 'a']++;

        for(int i=0; i<s.length(); i++) if(s[i] >= 'A' && s[i] <= 'Z') t2[s[i] - 'A']++;

        for(int i=0; i<26; i++){

            if(t1[i] == 0 && t2[i] == 0) continue;

            else if(t1[i] == 0 && t2[i] > 0) m[i + 'A']++;

            else if(t1[i] > 0 && t2[i] == 0) m[i + 'a']++;

        }

        vector<string> ans; string temp = "";

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

            if(m.find(s[i]) == m.end()) temp += s[i];

            else {

                ans.push\_back(temp); temp = "";

            }

        }

        if(temp != "") ans.push\_back(temp);

        string kk = "";

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

            if(ans[i] == "") continue;

            string t = longestNiceSubstring(ans[i]);

            if(t == "") continue;

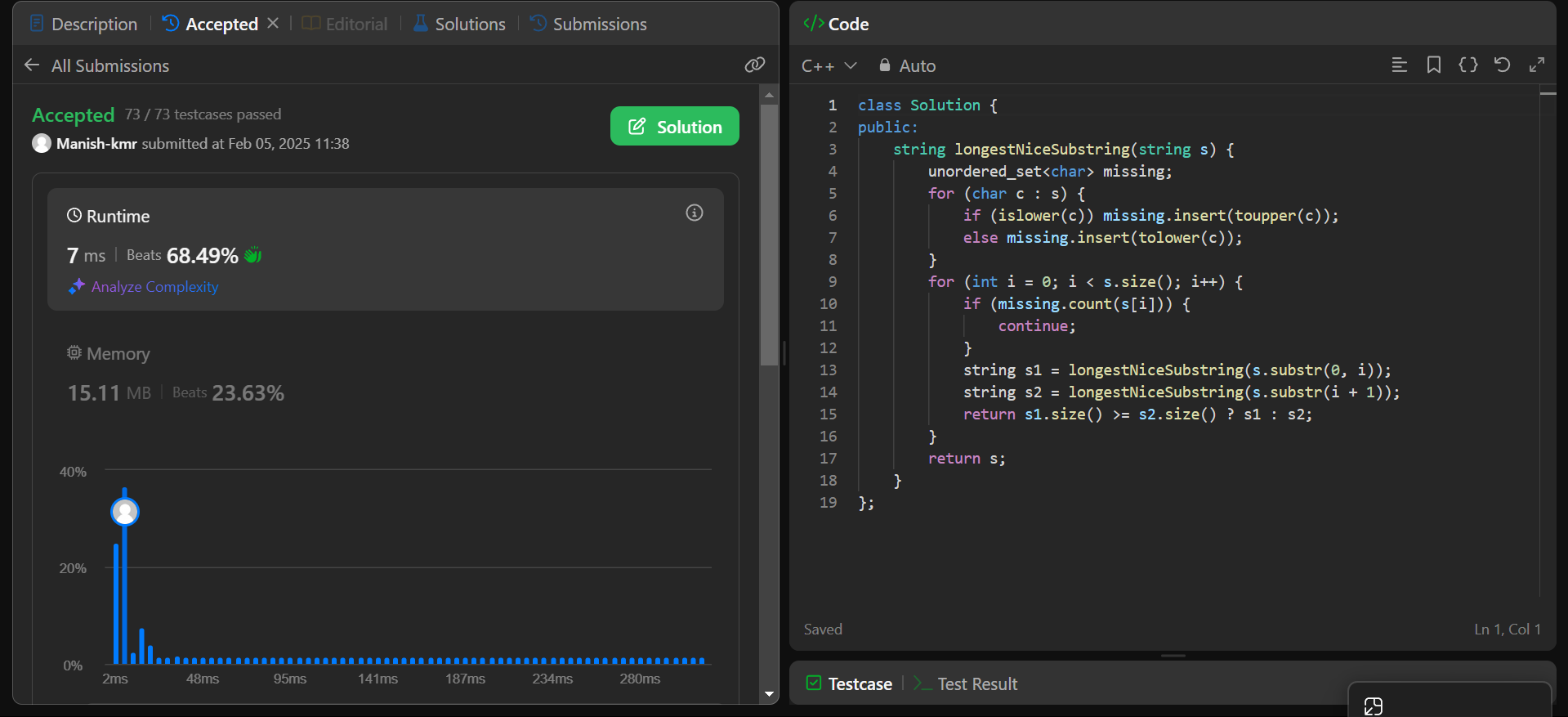
            if(kk == "" || kk.length() < t.length()) kk = t;

        }

        return kk;

    }

};



2

class Solution {

public:

    uint32\_t reverseBits(uint32\_t n) {

        uint32\_t result = 0;

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

            int bit = n & 1;       // Extract the least significant bit

            result = (result << 1) | bit; // Append the bit to the result

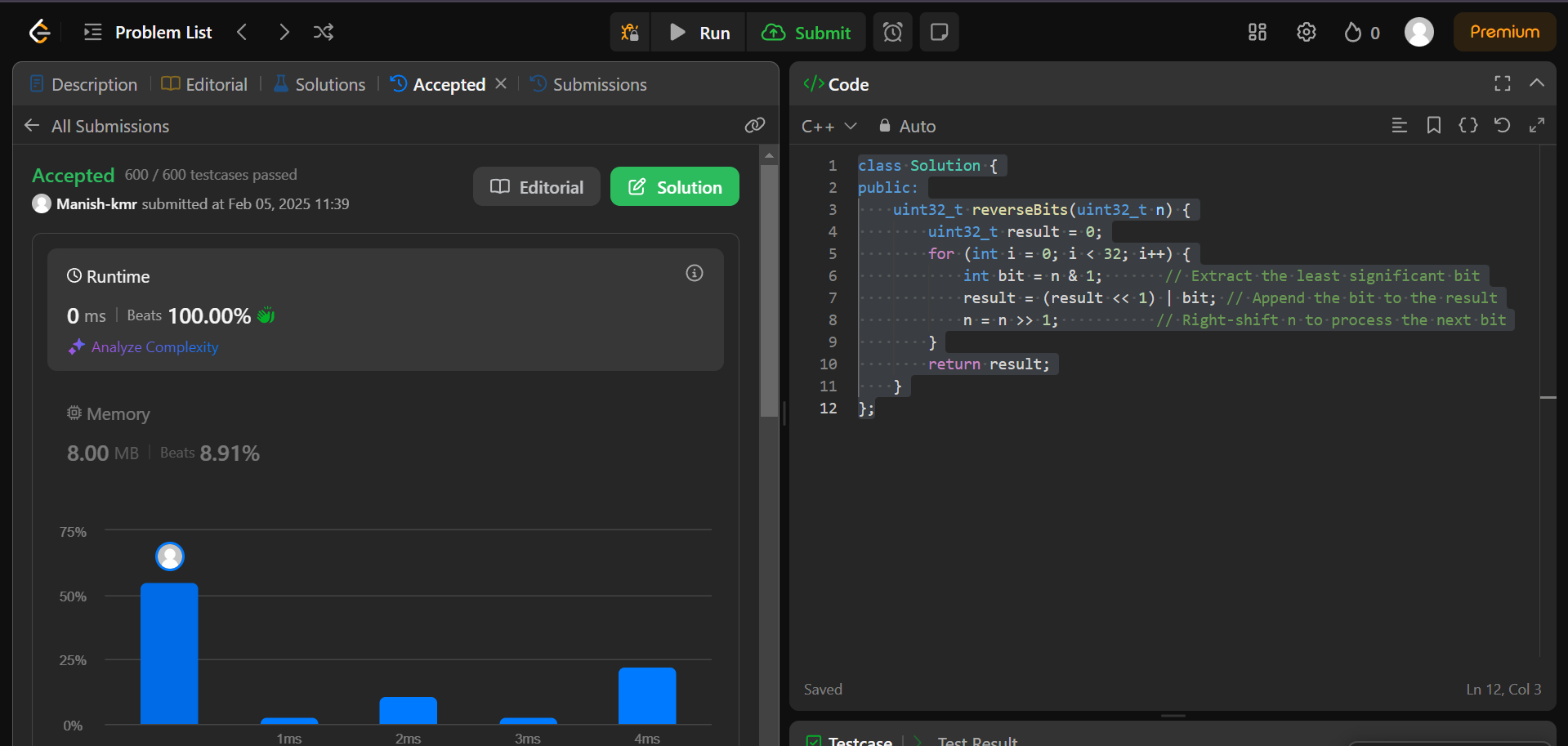
            n = n >> 1;           // Right-shift n to process the next bit

        }

        return result;

    }

};



3

class Solution {

public:

    int hammingWeight(uint32\_t n) {

        int res = 0;

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

            if ((n >> i) & 1) {

                res += 1;

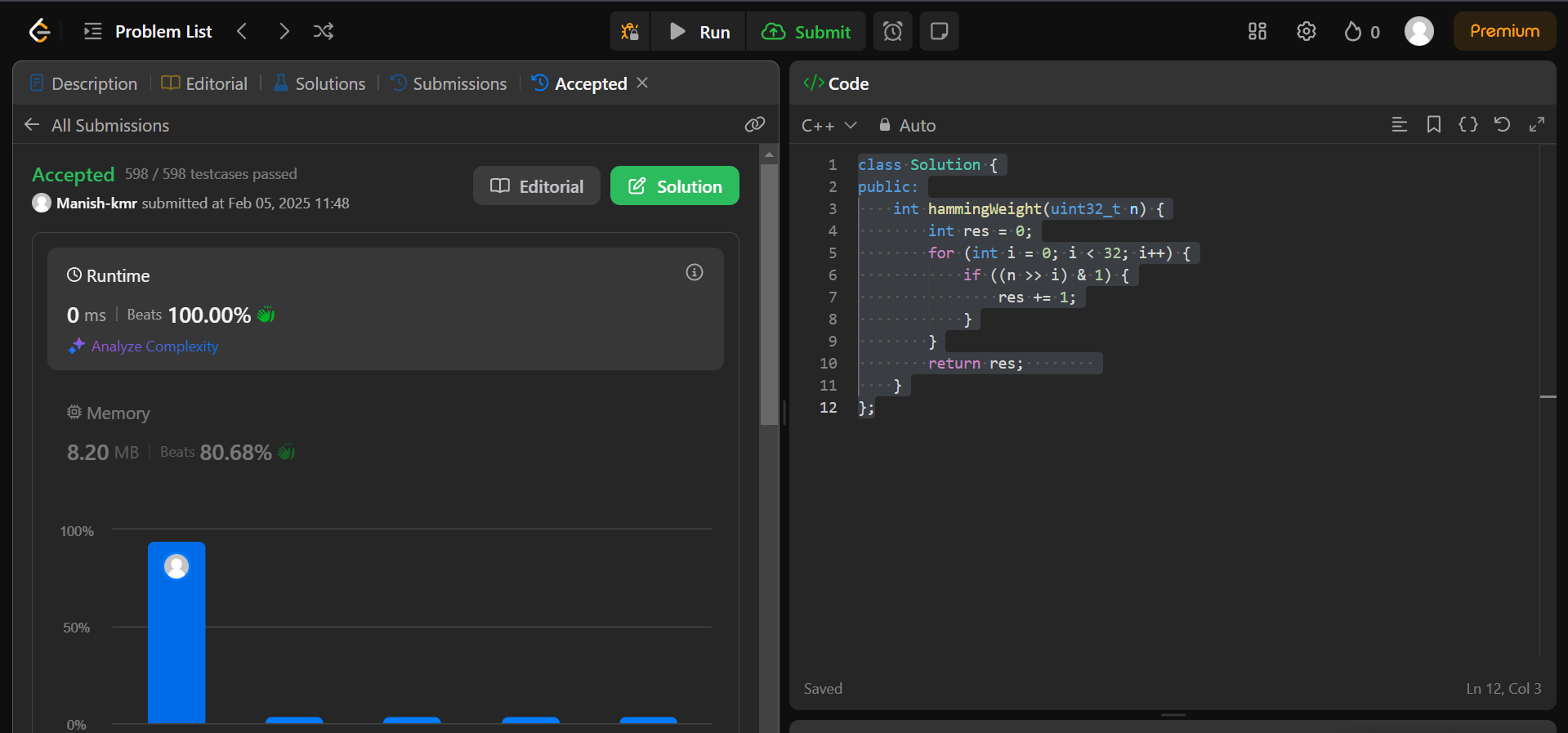
            }

        }

        return res;

    }

};



4

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;

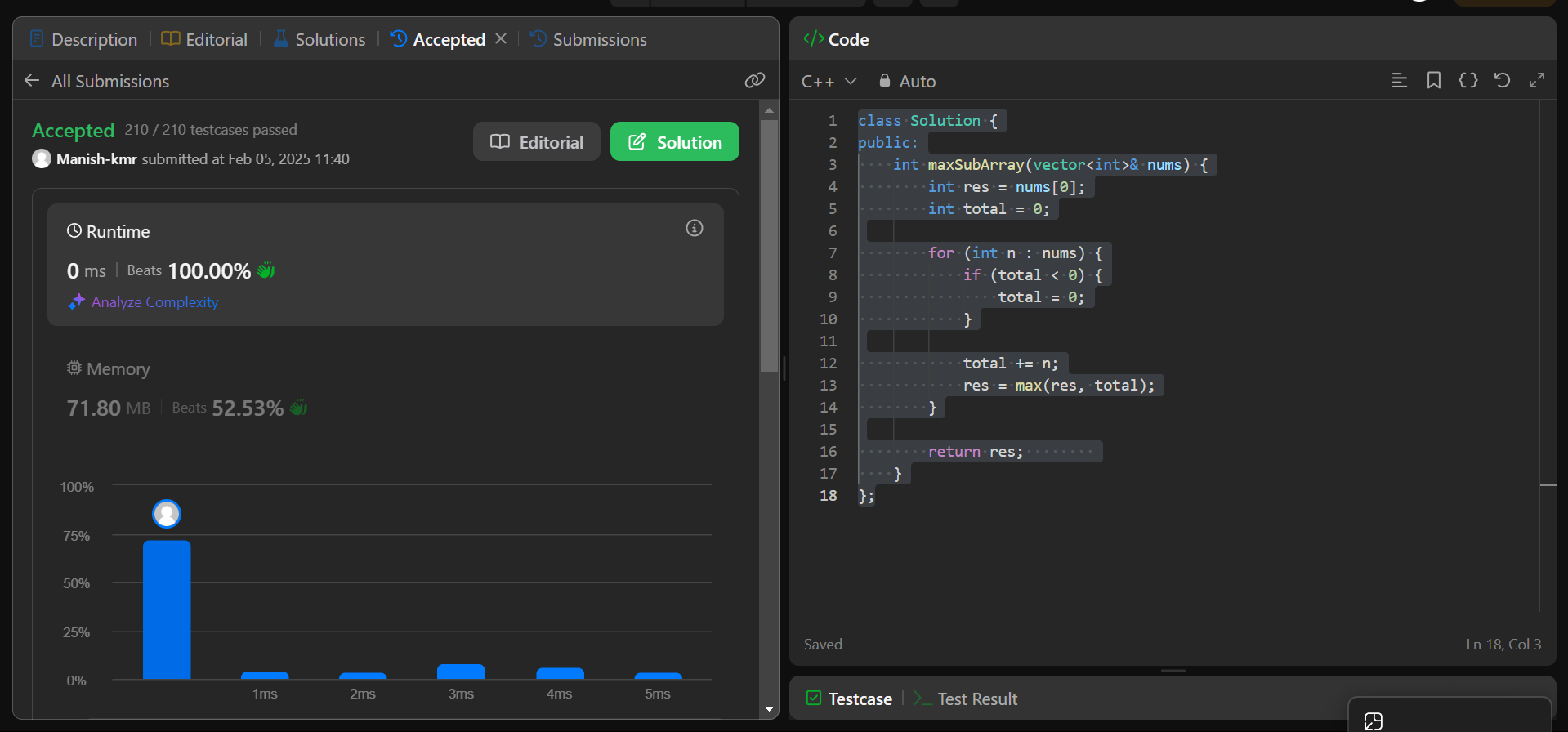
            res = max(res, total);

        }

        return res;

    }

};



5

class Solution {

public:

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

        int n = matrix.size(), m = matrix[0].size();

        int row = 0, col = m - 1;

        while (row < n && col >= 0) {

            if (matrix[row][col] == target) return true;

            else if (matrix[row][col] < target) row++;

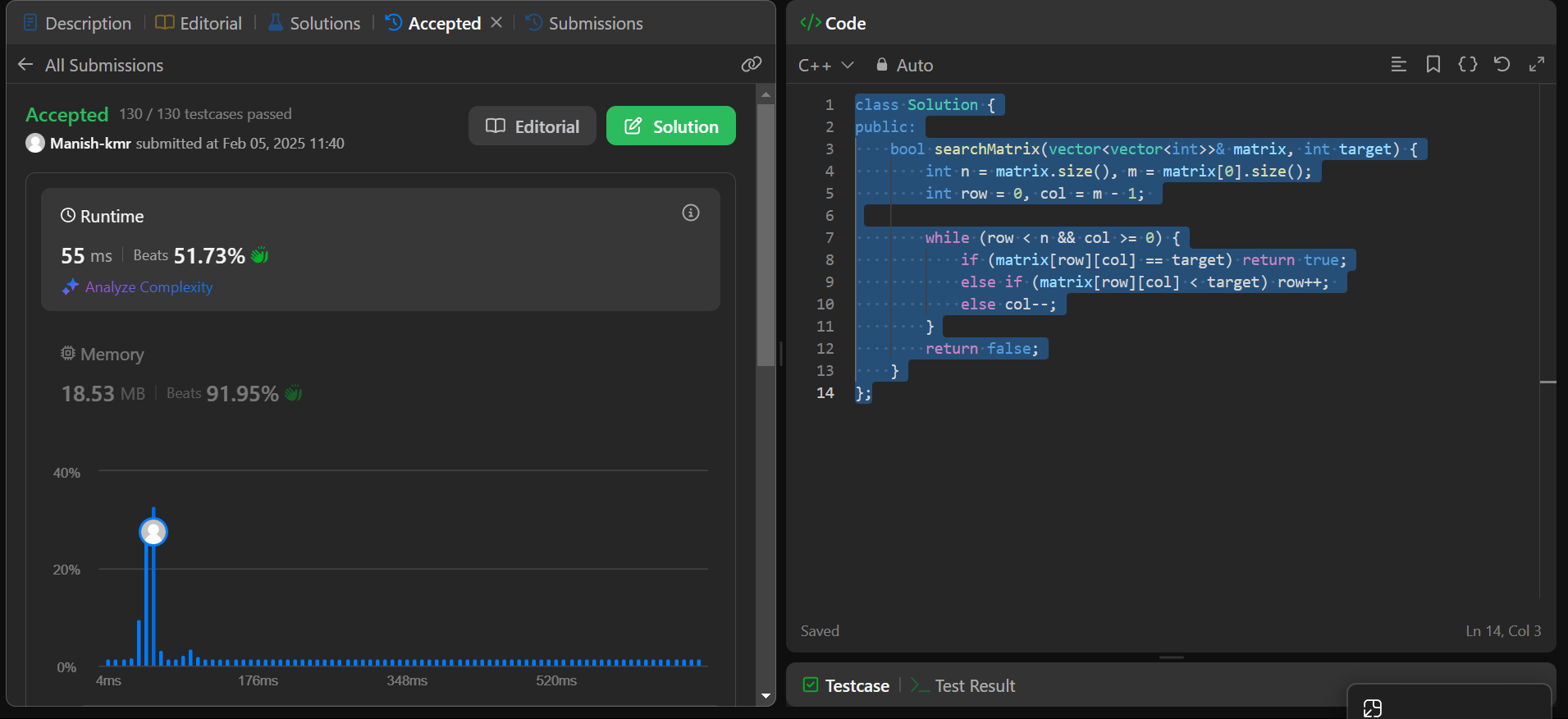
            else col--;

        }

        return false;

    }

};



6

class Solution {

private:

    int solve(int base, int power, int mod) {

        int ans = 1;

        while (power > 0) {

            if (power & 1) {

                ans = (ans \* base) % mod;

            }

            base = (base \* base) % mod;

            power >>= 1;

        }

        return ans;

    }

public:

    int superPow(int a, vector<int>& b) {

        a%=1337;

        int n = b.size();

        int m = 1140;

        int expi = 0;

        for(int i : b){

            expi = (expi\*10+i)%m;

        }

        if (expi == 0) {

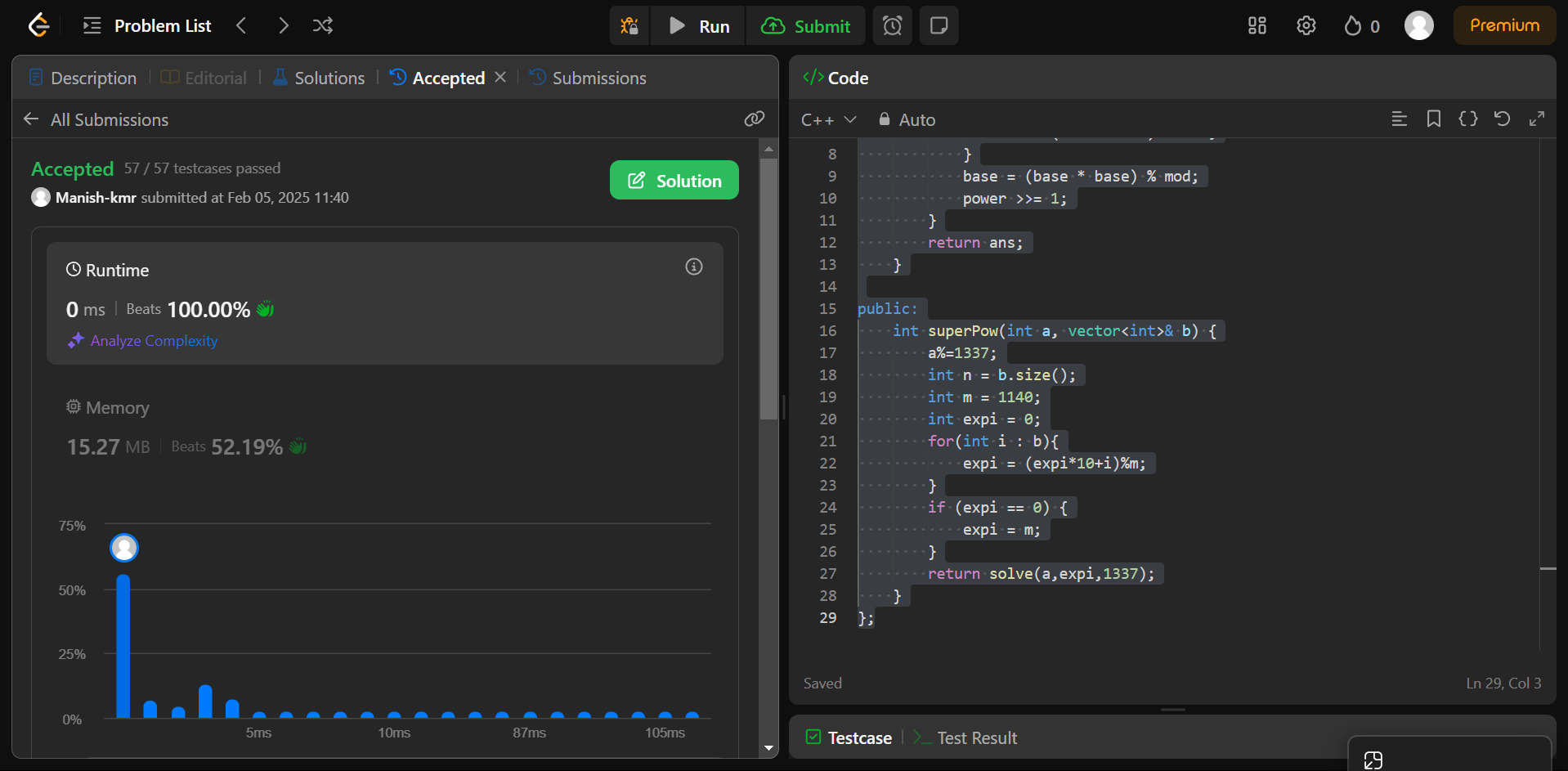
            expi = m;

        }

        return solve(a,expi,1337);

    }

};



7

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;

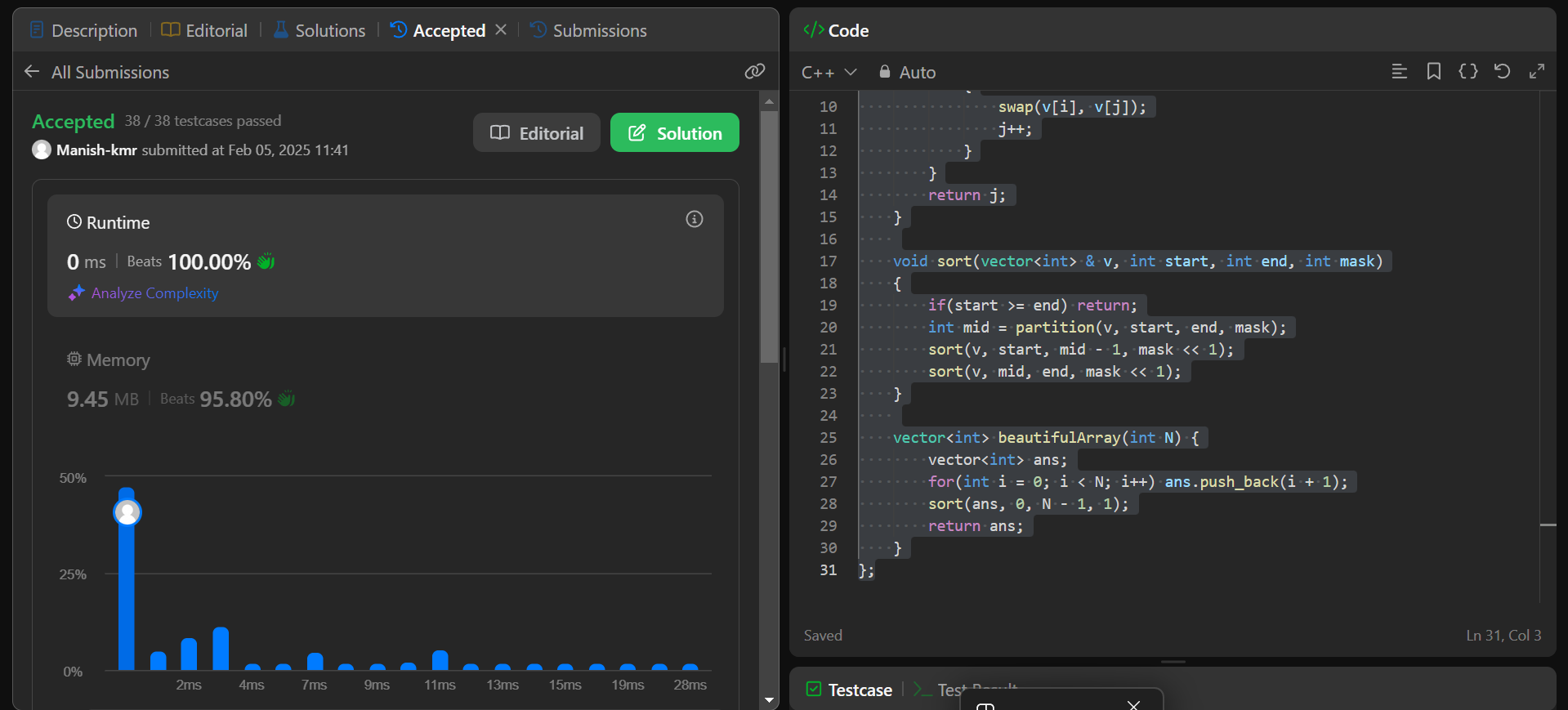
        for(int i = 0; i < N; i++) ans.push\_back(i + 1);

        sort(ans, 0, N - 1, 1);

        return ans;

    }

};



8

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)

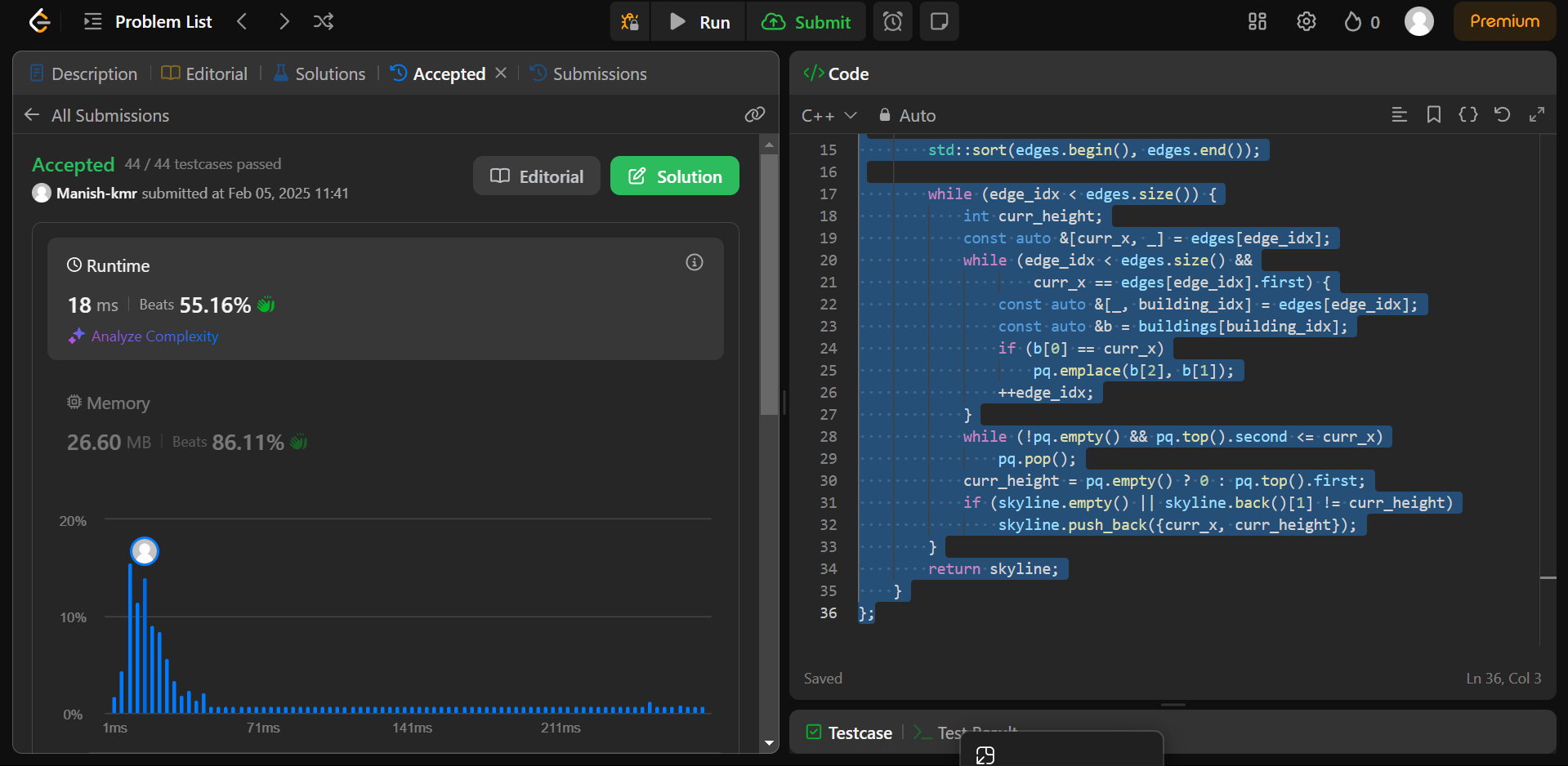
                skyline.push\_back({curr\_x, curr\_height});

        }

        return skyline;

    }

};



9

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);

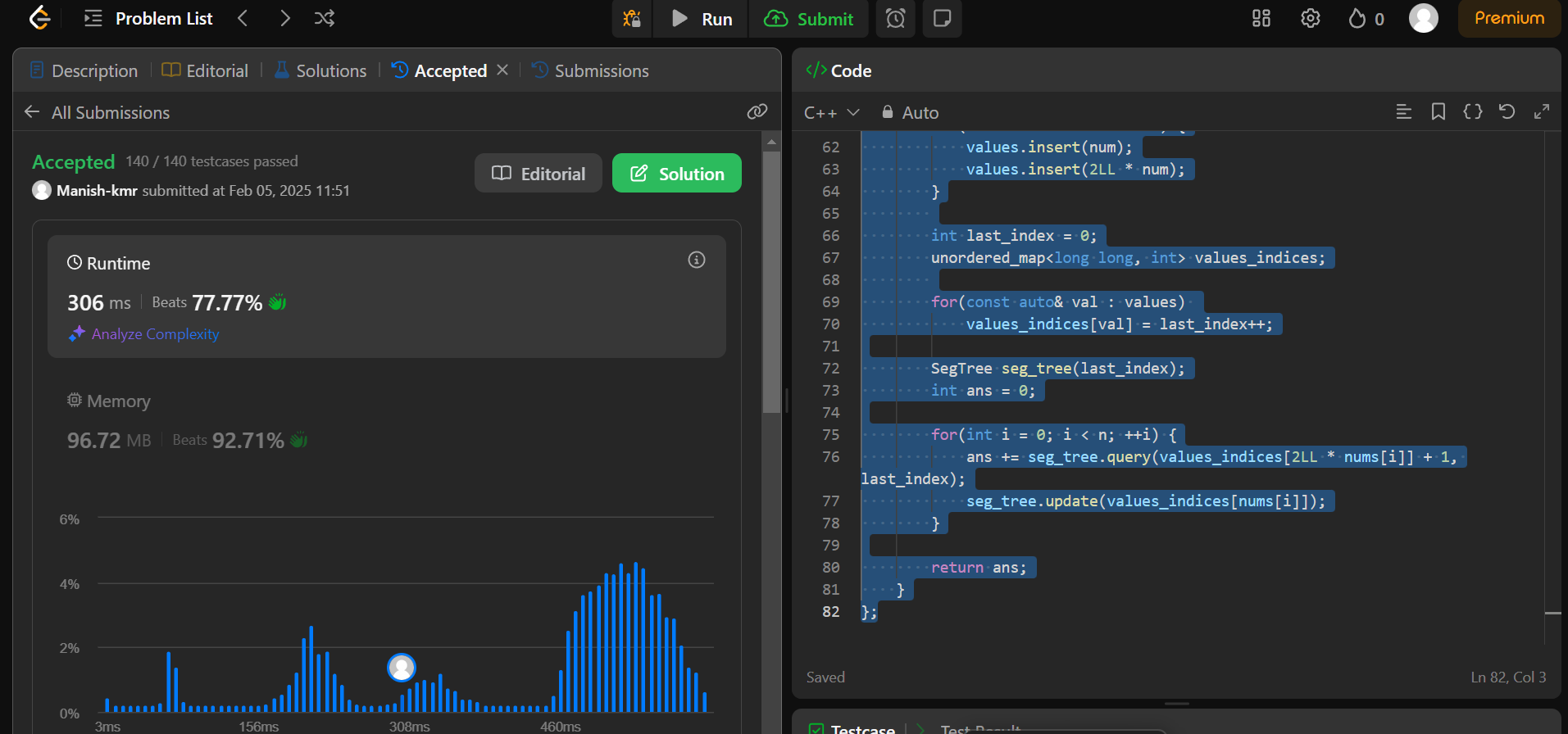
            seg\_tree.update(values\_indices[nums[i]]);

        }

        return ans;

    }

};



10

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;

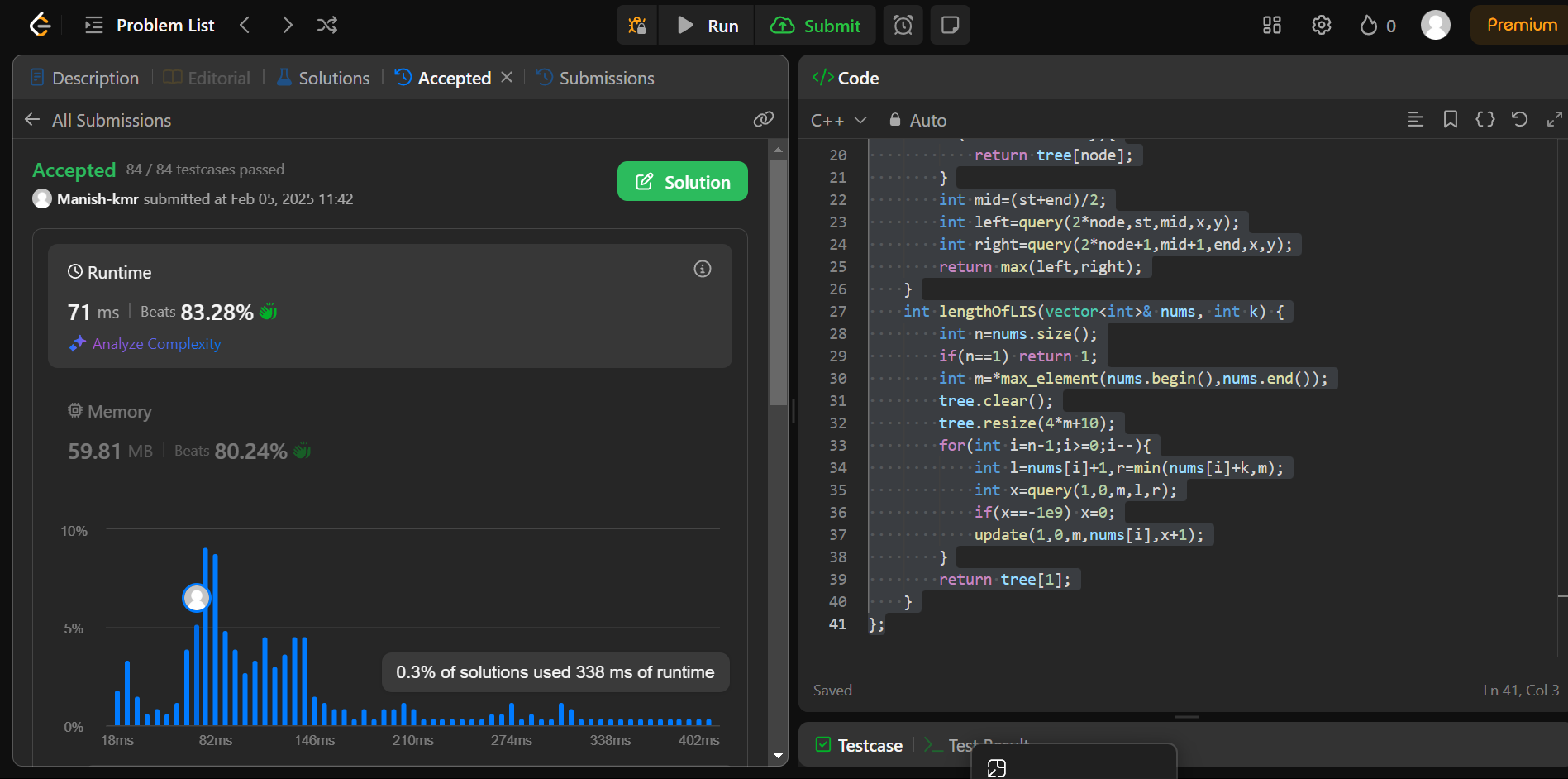
            update(1,0,m,nums[i],x+1);

        }

        return tree[1];

    }

};



11

class Solution {

public:

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

        int midx = m - 1;

        int nidx = n - 1;

        int right = m + n - 1;

        while (nidx >= 0) {

            if (midx >= 0 && nums1[midx] > nums2[nidx]) {

                nums1[right] = nums1[midx];

                midx--;

            } else {

                nums1[right] = nums2[nidx];

                nidx--;

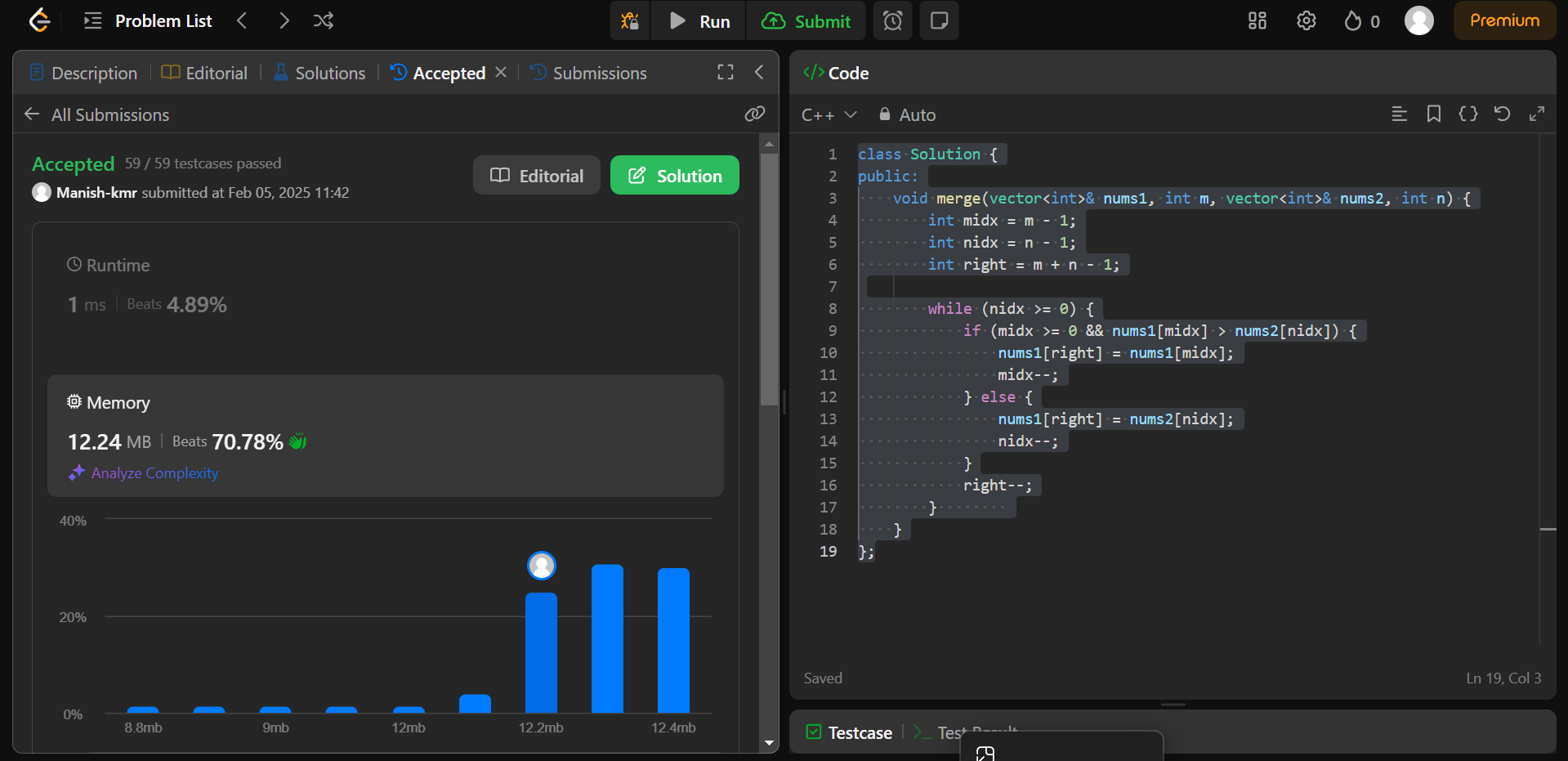
            }

            right--;

        }

    }

};



11

/\*

By   :: savetrees

Used :: Binary Search

\*/

class Solution {

public:

    int firstBadVersion(int n) {

        int low=1;

        int high=n;

        while(low<=high)

        {

            int mid=low+(high-low)/2;

            if(isBadVersion(mid))high=mid-1;

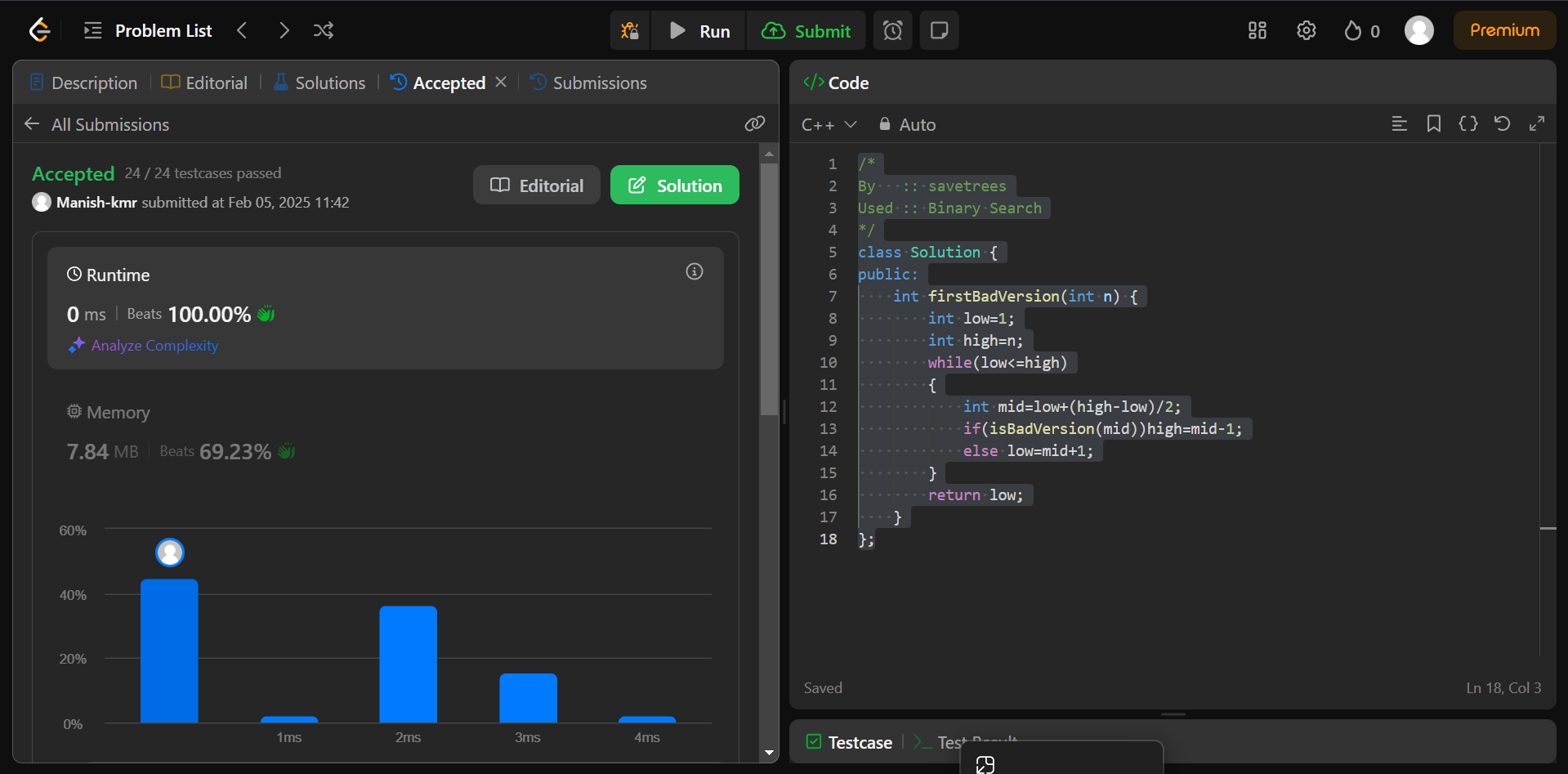
            else low=mid+1;

        }

        return low;

    }

};



12

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int l=nums.size();

        int count0=0;

        int count1=0;

        int count2=0;

        for(int i=0;i<l;i++){

            if(nums[i]==0) count0++;

            else if(nums[i]==1) count1++;

            else count2++;

        }

        count1=count0+count1;

        count2=count0+count1+count2;

        for(int i=0;i<l;i++){

            if(i<count0) nums[i]=0;

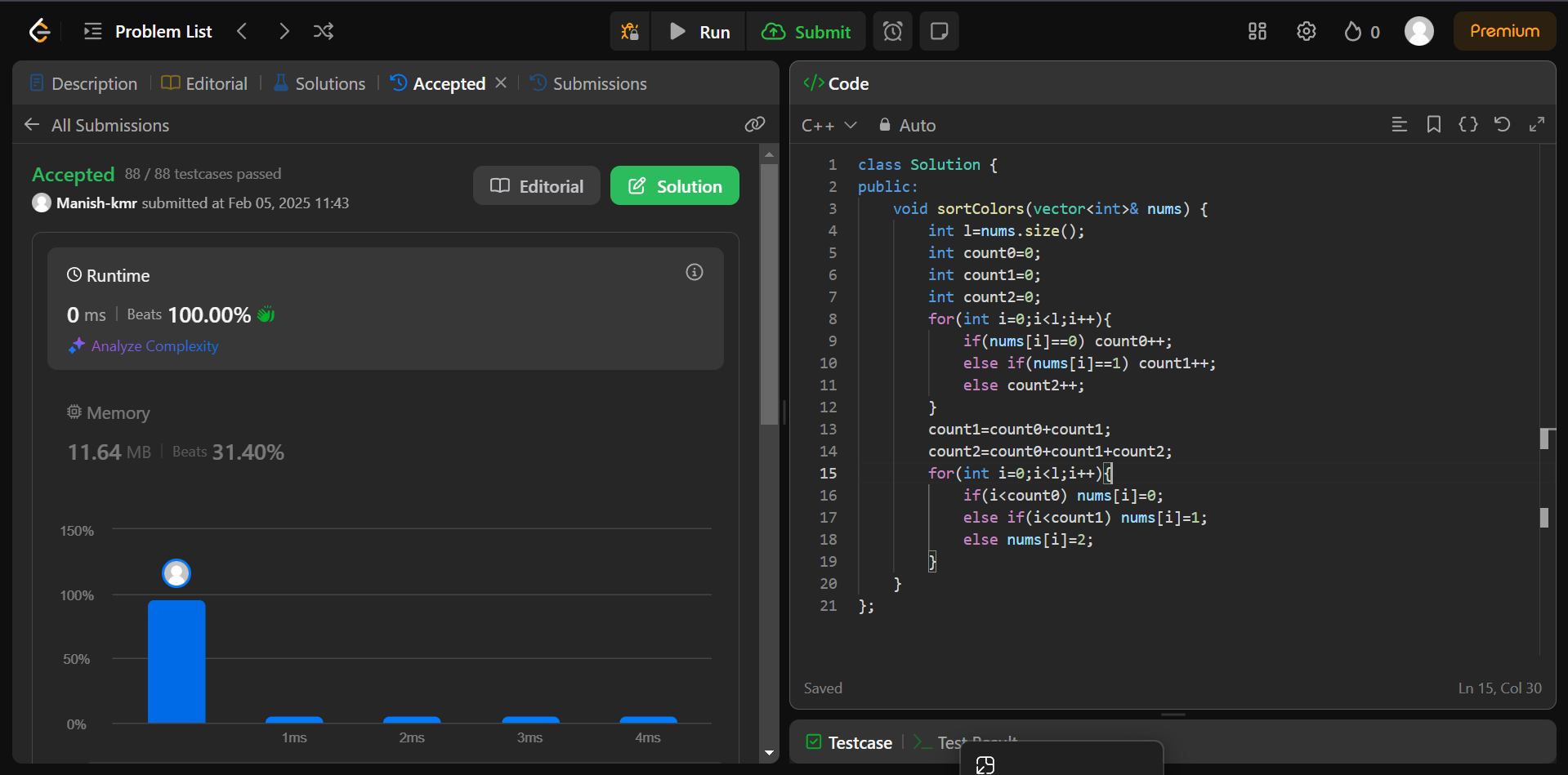
            else if(i<count1) nums[i]=1;

            else nums[i]=2;

        }

    }

};



13

class Solution {

public:

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

        unordered\_map<int, int> counter;

        for (int n : nums) {

            counter[n]++;

        }

        auto comp = [](pair<int, int>& a, pair<int, int>& b) {

            return a.second < b.second;

        };

        priority\_queue<pair<int, int>, vector<pair<int, int>>, decltype(comp)> heap(comp);

        for (auto& entry : counter) {

            heap.push({entry.first, entry.second});

        }

        vector<int> res;

        while (k-- > 0) {

            res.push\_back(heap.top().first);

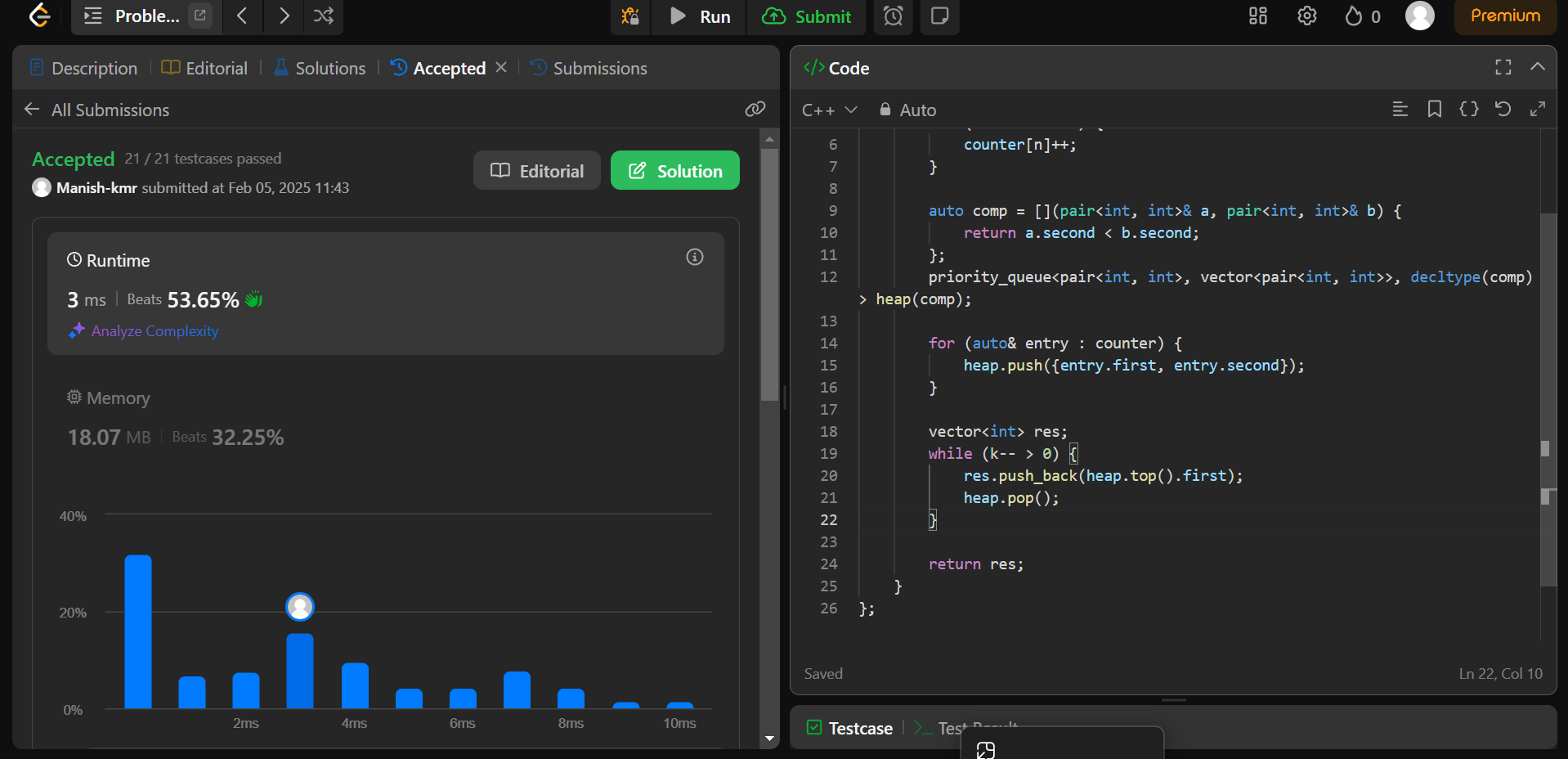
            heap.pop();

        }

        return res;

    }

};



14

class Solution {

public:

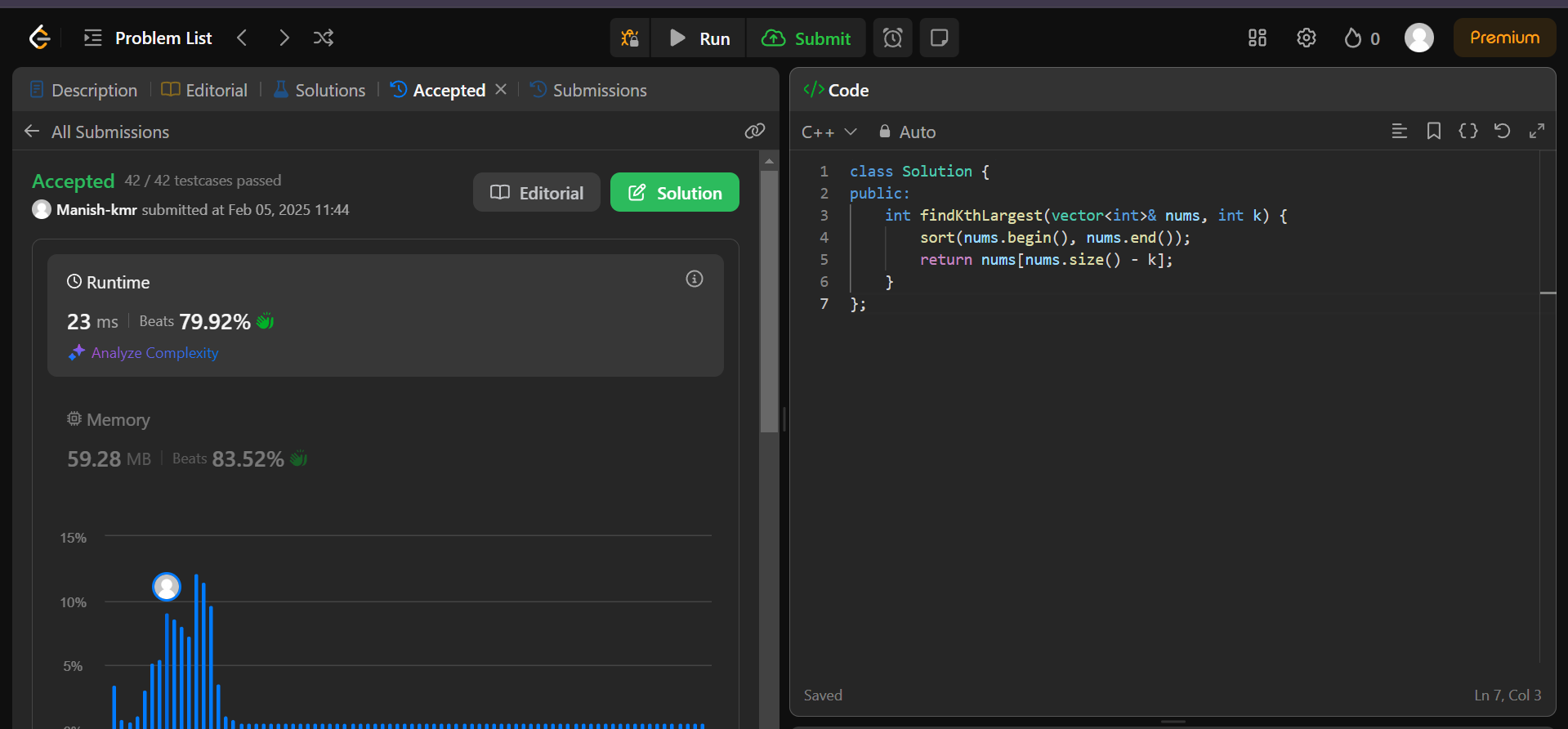
    int findKthLargest(vector<int>& nums, int k) {

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

        return nums[nums.size() - k];

    }

};



15

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;

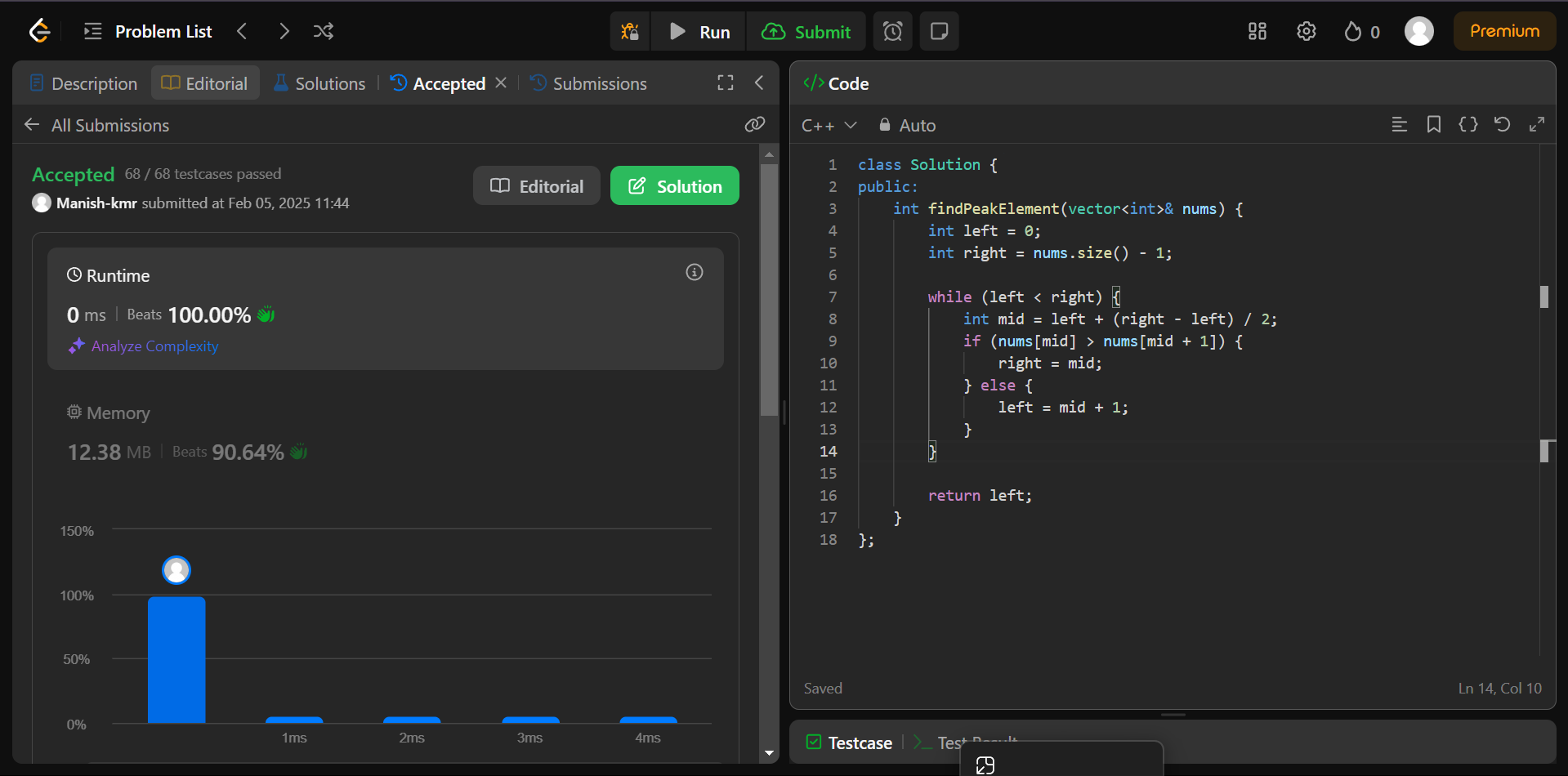
            }

        }

        return left;

    }

};



16

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;

                }

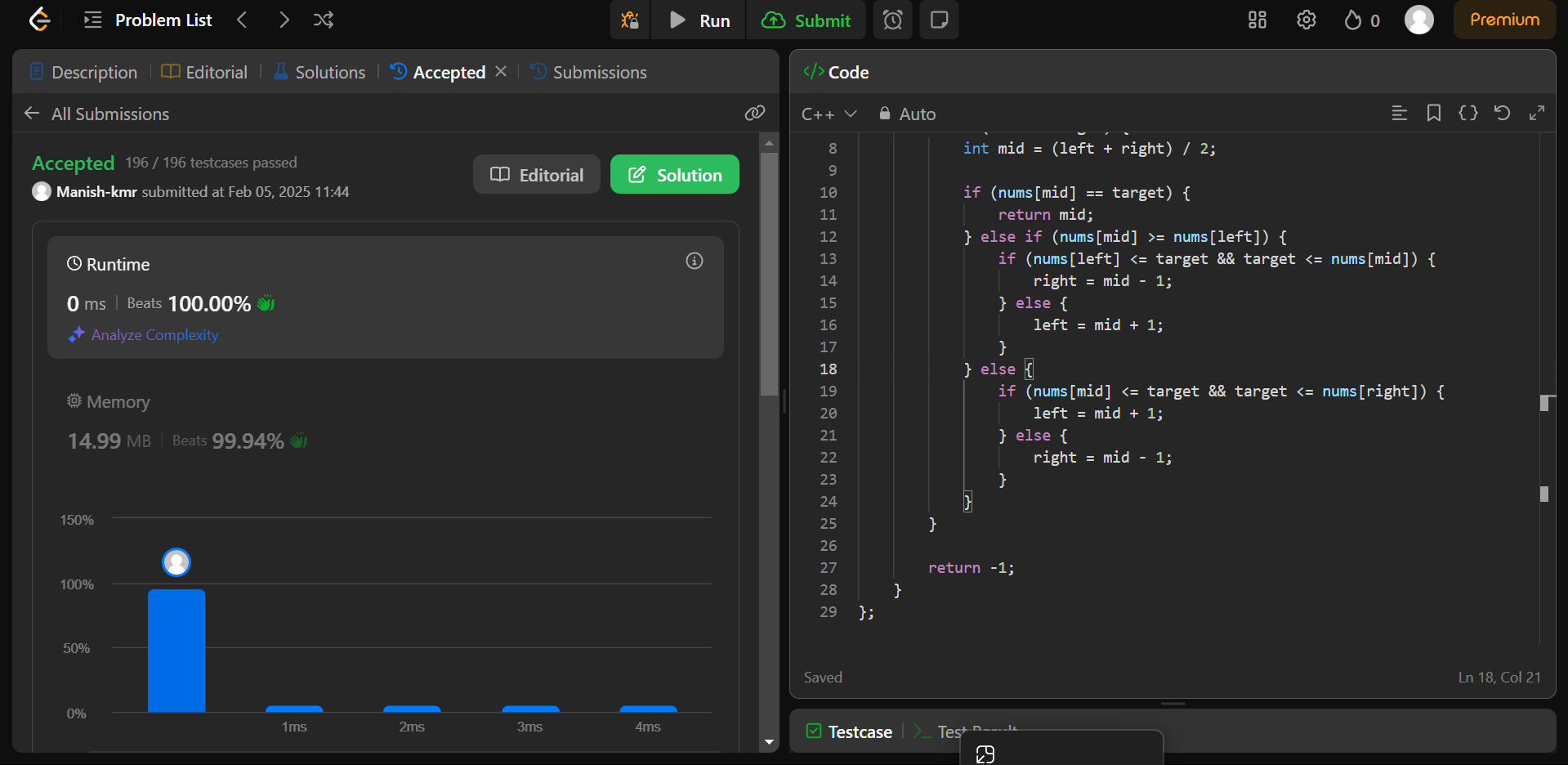
            }

        }

        return -1;

    }

};



18

class Solution {

public:

    void wiggleSort(vector<int>& nums) {

        int n = nums.size();

        vector<int> nums1(nums);

        sort(nums1.begin(), nums1.end());

        int i = n-1;

        int j = 0;

        int k = i/2 + 1;

        while(i >= 0)

        {

            if(i % 2 == 1)

            {

                nums[i] = nums1[k];

                k++;

            }

            else

            {

                nums[i] = nums1[j];

                j++;

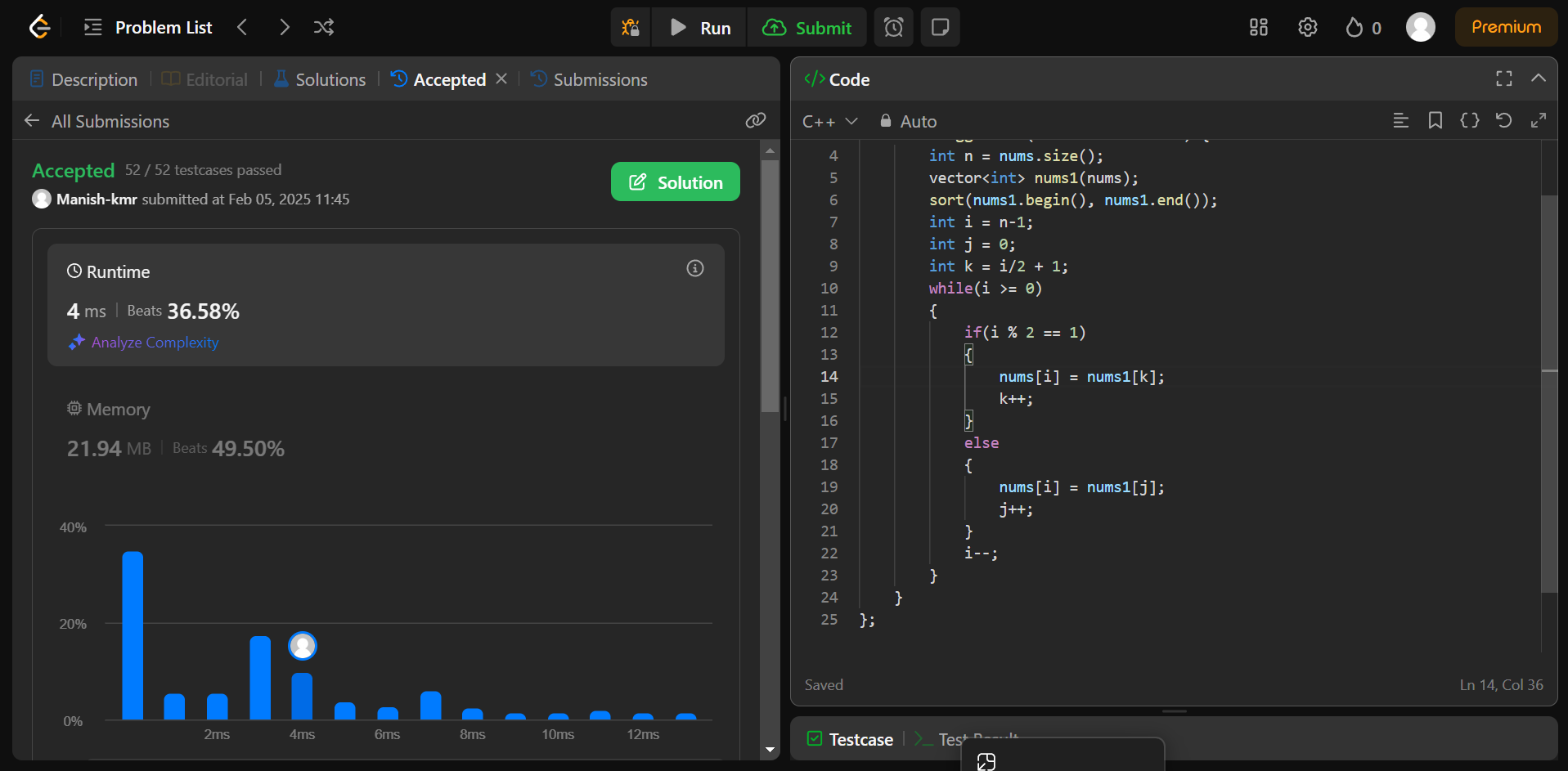
            }

            i--;

        }

    }

};



19

class Solution {

public:

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

        int n = matrix.size();

        int m = matrix[0].size();

        vector<int> p;

        // Convert 2D matrix to 1D vector

        for (int i = 0; i < n \* m; i++) {

            p.push\_back(matrix[i / m][i % m]);

        }

        // Sort the 1D vector

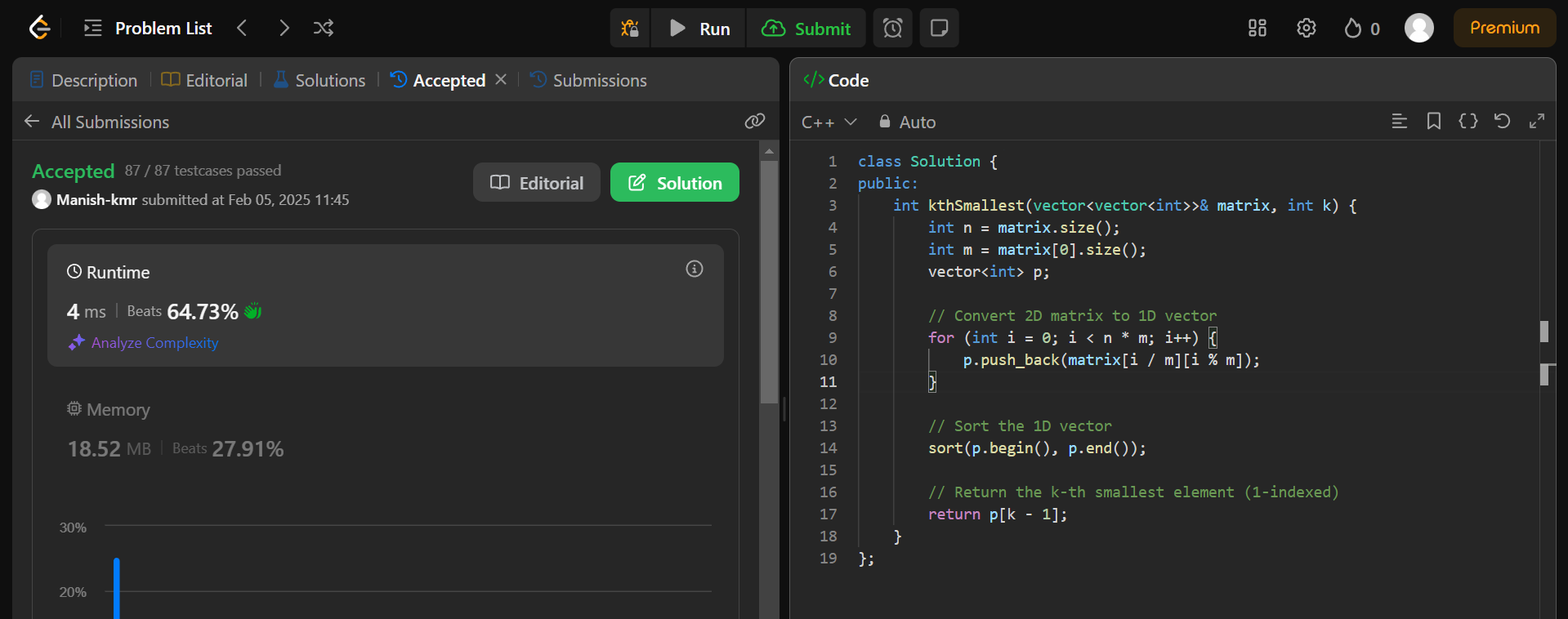
        sort(p.begin(), p.end());

        // Return the k-th smallest element (1-indexed)

        return p[k - 1];

    }

};



20

class Solution {

public:

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

        // Get the sizes of both input arrays.

        int n = nums1.size();

        int m = nums2.size();

        // Merge the arrays into a single sorted array.

        vector<int> merged;

        for (int i = 0; i < n; i++) {

            merged.push\_back(nums1[i]);

        }

        for (int i = 0; i < m; i++) {

            merged.push\_back(nums2[i]);

        }

        // Sort the merged array.

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

        // Calculate the total number of elements in the merged array.

        int total = merged.size();

        if (total % 2 == 1) {

            // If the total number of elements is odd, return the middle element as the median.

            return static\_cast<double>(merged[total / 2]);

        } else {

            // If the total number of elements is even, calculate the average of the two middle elements as the median.

            int middle1 = merged[total / 2 - 1];

            int middle2 = merged[total / 2];

            return (static\_cast<double>(middle1) + static\_cast<double>(middle2)) / 2.0;

        }

    }

};

