**ADVANCED PROGRAMMING LAB-2 ASSIGNMENT**

**Submitted By:**

**Name- Anshika Yadav**

**UID- 22BCS10051**

**Section-22BCS\_IOT\_605-B**

1. Median of two sorted array

class Solution {

public:

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

if (nums1.size() > nums2.size())

return findMedianSortedArrays(nums2, nums1);

int m = nums1.size(), n = nums2.size();

int left = 0, right = m, mid1, mid2;

while (left <= right) {

mid1 = left + (right - left) / 2;

mid2 = (m + n + 1) / 2 - mid1;

int maxLeft1 = (mid1 == 0) ? INT\_MIN : nums1[mid1 - 1];

int minRight1 = (mid1 == m) ? INT\_MAX : nums1[mid1];

int maxLeft2 = (mid2 == 0) ? INT\_MIN : nums2[mid2 - 1];

int minRight2 = (mid2 == n) ? INT\_MAX : nums2[mid2];

if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {

if ((m + n) % 2 == 0)

return (max(maxLeft1, maxLeft2) + min(minRight1, minRight2)) / 2.0;

else

return max(maxLeft1, maxLeft2);

}

else if (maxLeft1 > minRight2)

right = mid1 - 1;

else

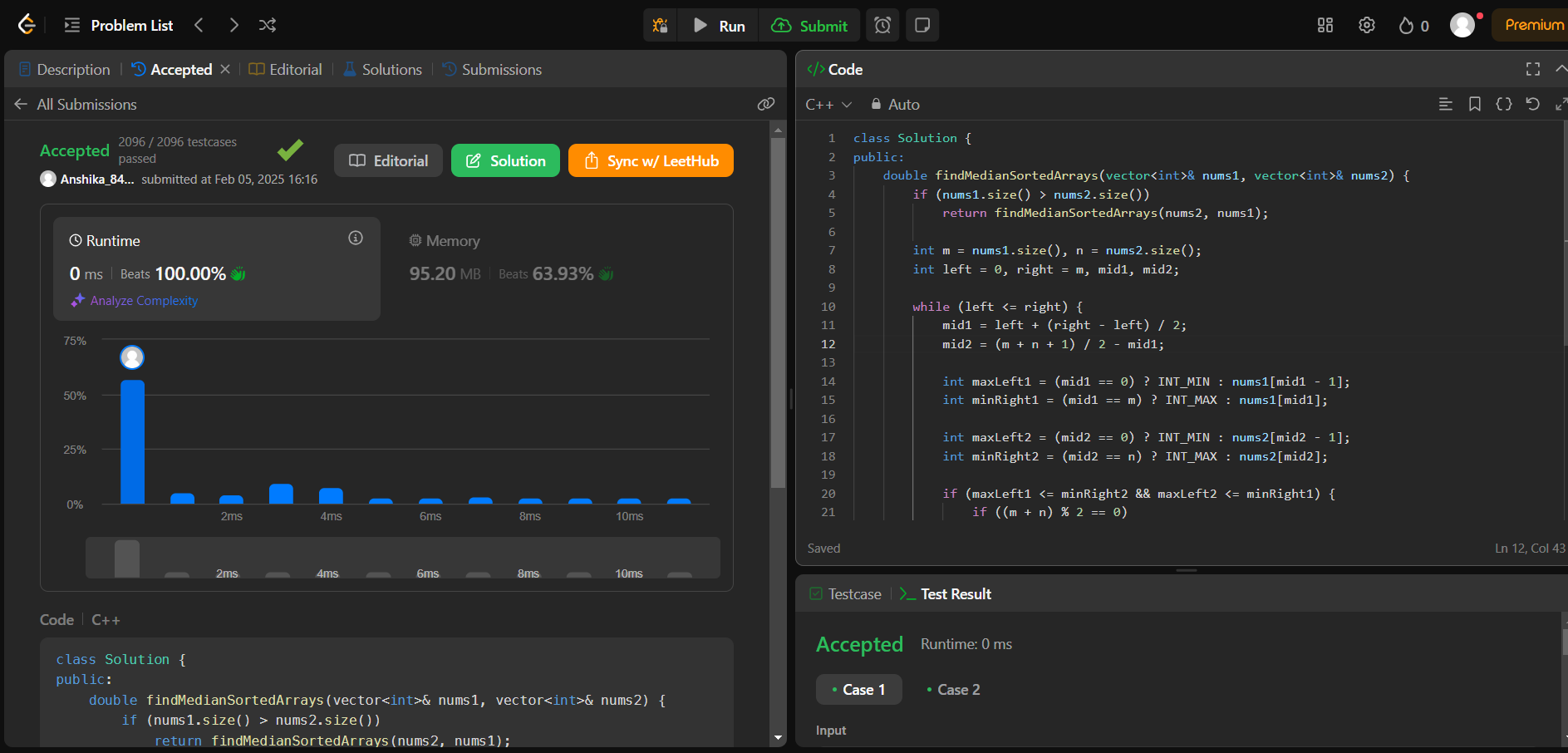
left = mid1 + 1;

}

return 0.0;

}

};



1. Kth smallest element in a sorted matrix

class Solution {

public:

    int countLessEqual(vector<vector<int>>& matrix, int mid, int n) {

        int count = 0, row = n - 1, col = 0;

        while (row >= 0 && col < n) {

            if (matrix[row][col] <= mid) {

                count += (row + 1);

                col++;

            } else {

                row--;

            }

        }

        return count;

    }

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

        int n = matrix.size();

        int left = matrix[0][0], right = matrix[n - 1][n - 1];

        while (left < right) {

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

            int count = countLessEqual(matrix, mid, n);

            if (count < k)

                left = mid + 1;

            else

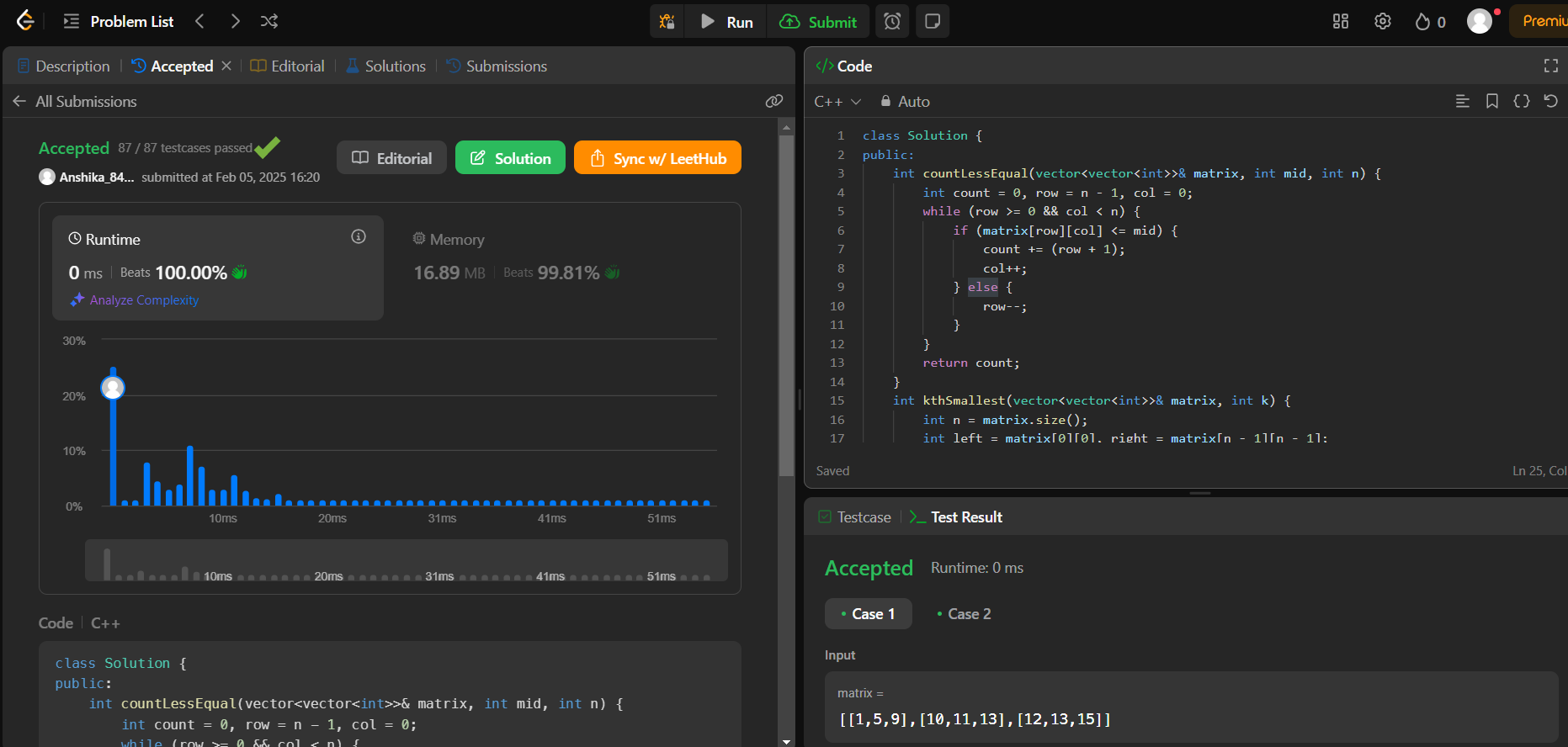
                right = mid;

        }

        return left;

    }

};



1. Search in a rotated sorted array

class Solution {

public:

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

int low = 0;

int high = nums.size() - 1;

while (low <= high) {

int mid = (low+high) / 2;

if (nums[mid] == target) {

return mid;

}

if (nums[low] <= nums[mid]) {

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

high = mid - 1;

} else {

low = mid + 1;

}

} else {

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

low = mid + 1;

} else {

high = mid - 1;

}

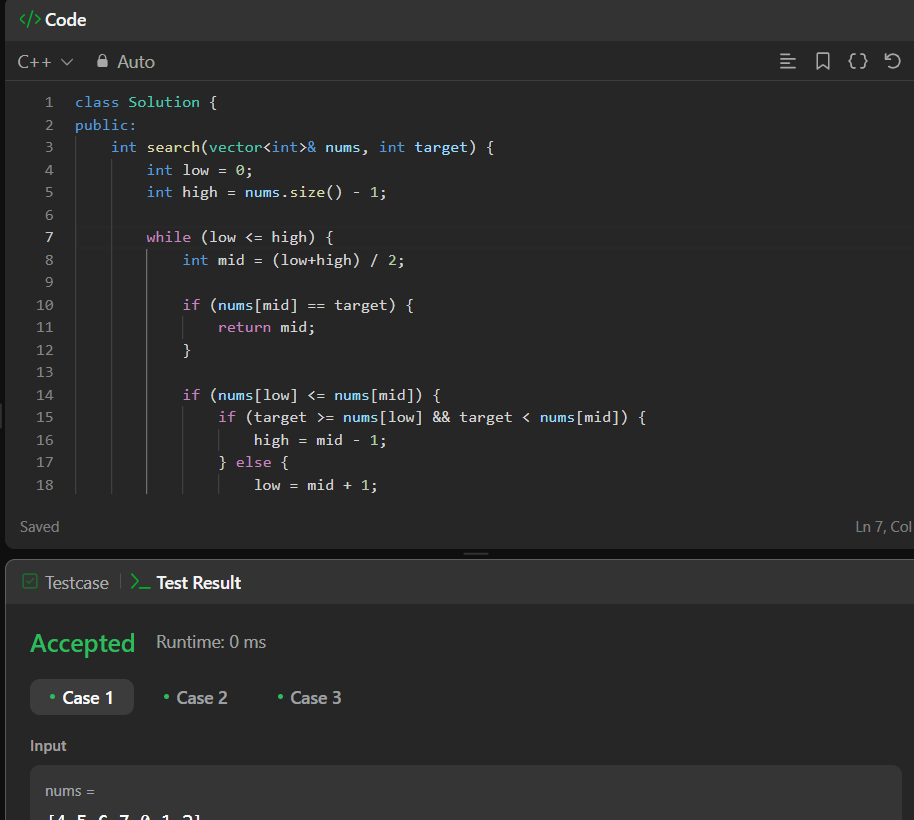
}

}

return -1;

}

};



1. Merge sorted array

class Solution {

public:

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

int i = m - 1, j = n - 1, k = m + n - 1;

while (i >= 0 && j >= 0) {

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

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

else

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

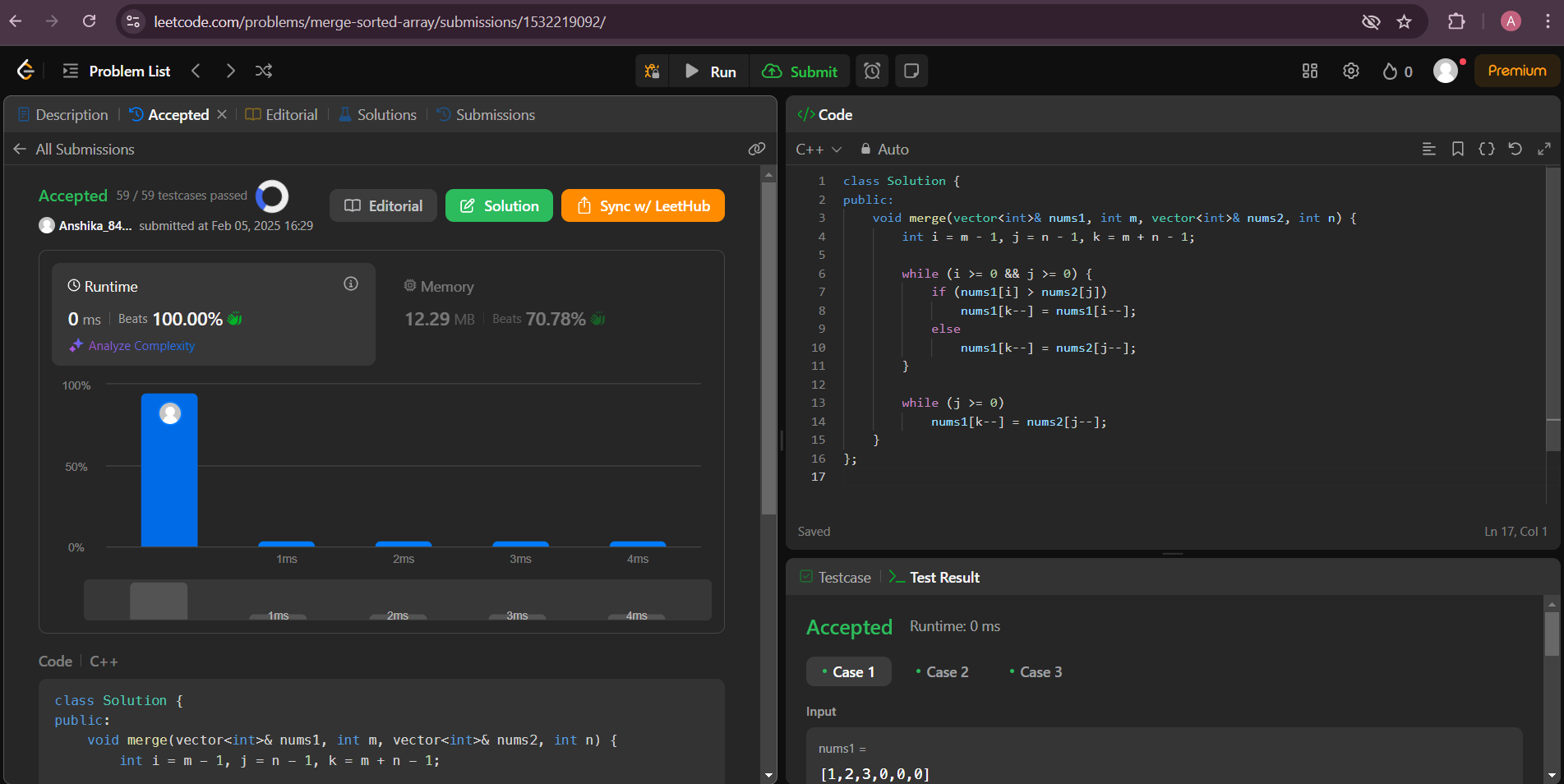
}

while (j >= 0)

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

}

};



1. Sort Color

class Solution {

public:

void sortColors(vector<int>& nums) {

int low = 0, mid = 0, high = nums.size() - 1;

while (mid <= high) {

if (nums[mid] == 0) {

swap(nums[low++], nums[mid++]);

} else if (nums[mid] == 1) {

mid++;

} else {

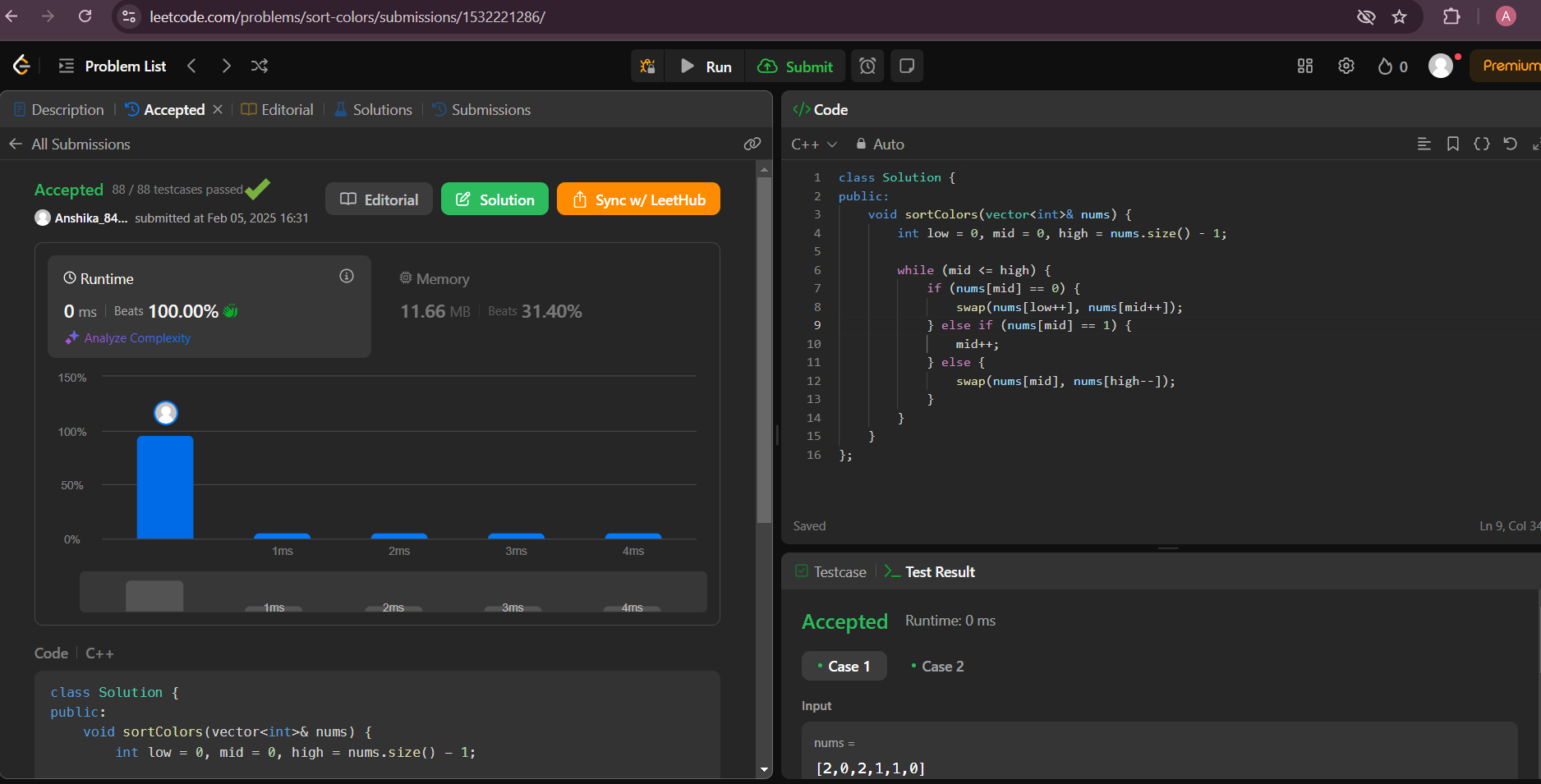
swap(nums[mid], nums[high--]);

}

}

}

};



1. Wiggle Sort II

class Solution {

public:

void wiggleSort(vector<int>& nums) {

for (int i = 0; i < nums.size() - 1; i++) {

if ((i % 2 == 0 && nums[i] > nums[i + 1]) ||

(i % 2 == 1 && nums[i] < nums[i + 1])) {

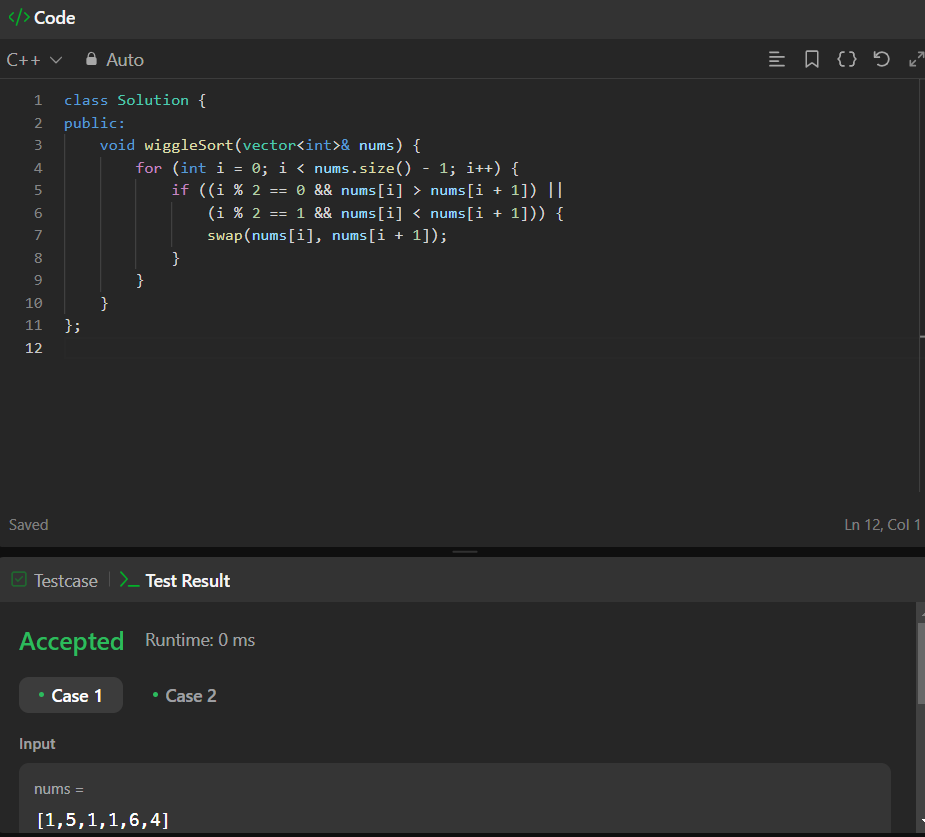
swap(nums[i], nums[i + 1]);

}

}

}

};



1. Search a 2D Matrix II

class Solution {

public:

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

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

int row = 0, col = n - 1;

while (row < m && col >= 0) {

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

else if (matrix[row][col] > target) col--;

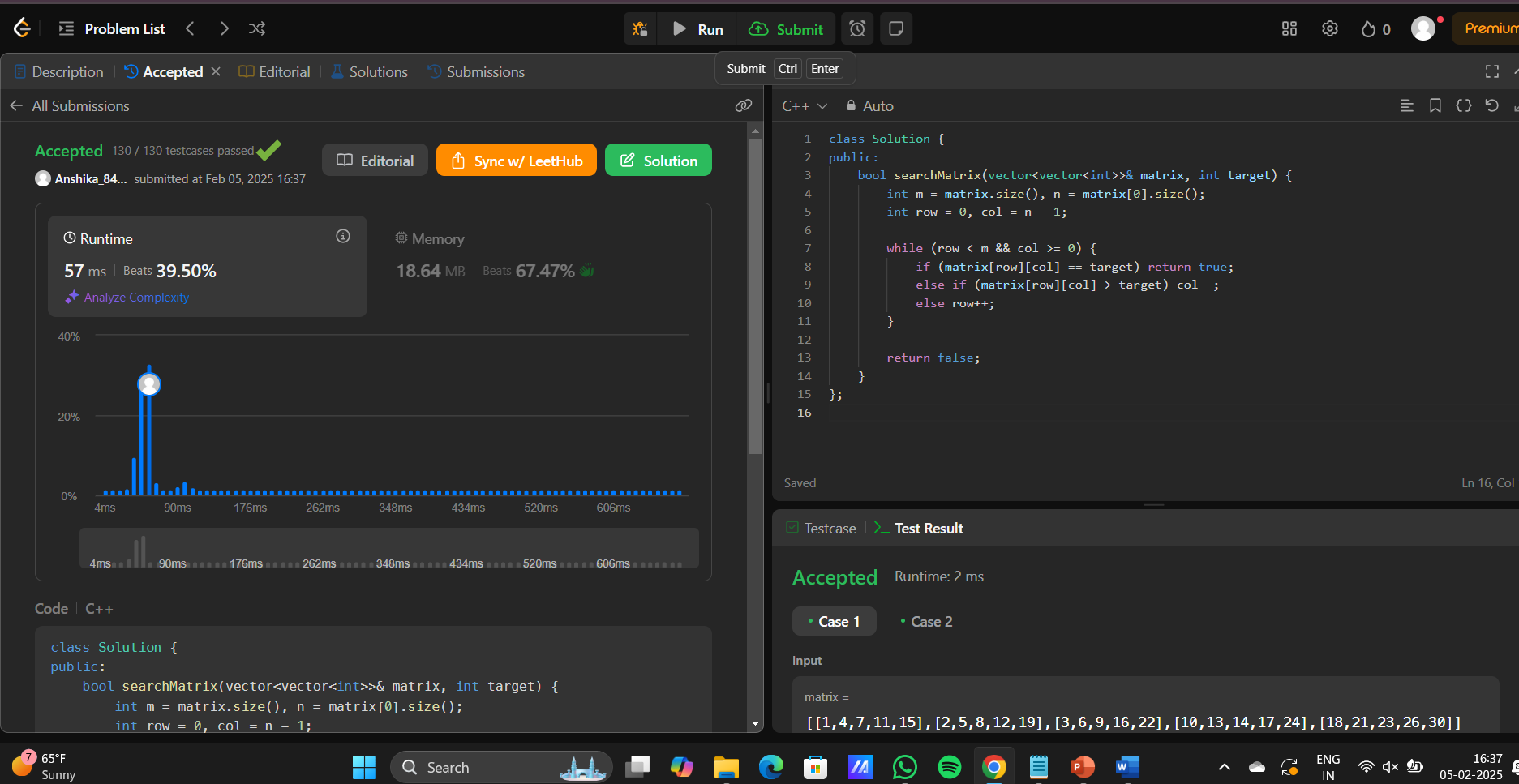
else row++;

}

return false;

}

};



1. Merge intervals

class Solution {

public:

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

if (intervals.empty()) return {};

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

vector<vector<int>> merged;

for (const auto& interval : intervals) {

if (merged.empty() || merged.back()[1] < interval[0]) {

merged.push\_back(interval);

} else {

merged.back()[1] = max(merged.back()[1], interval[1]);

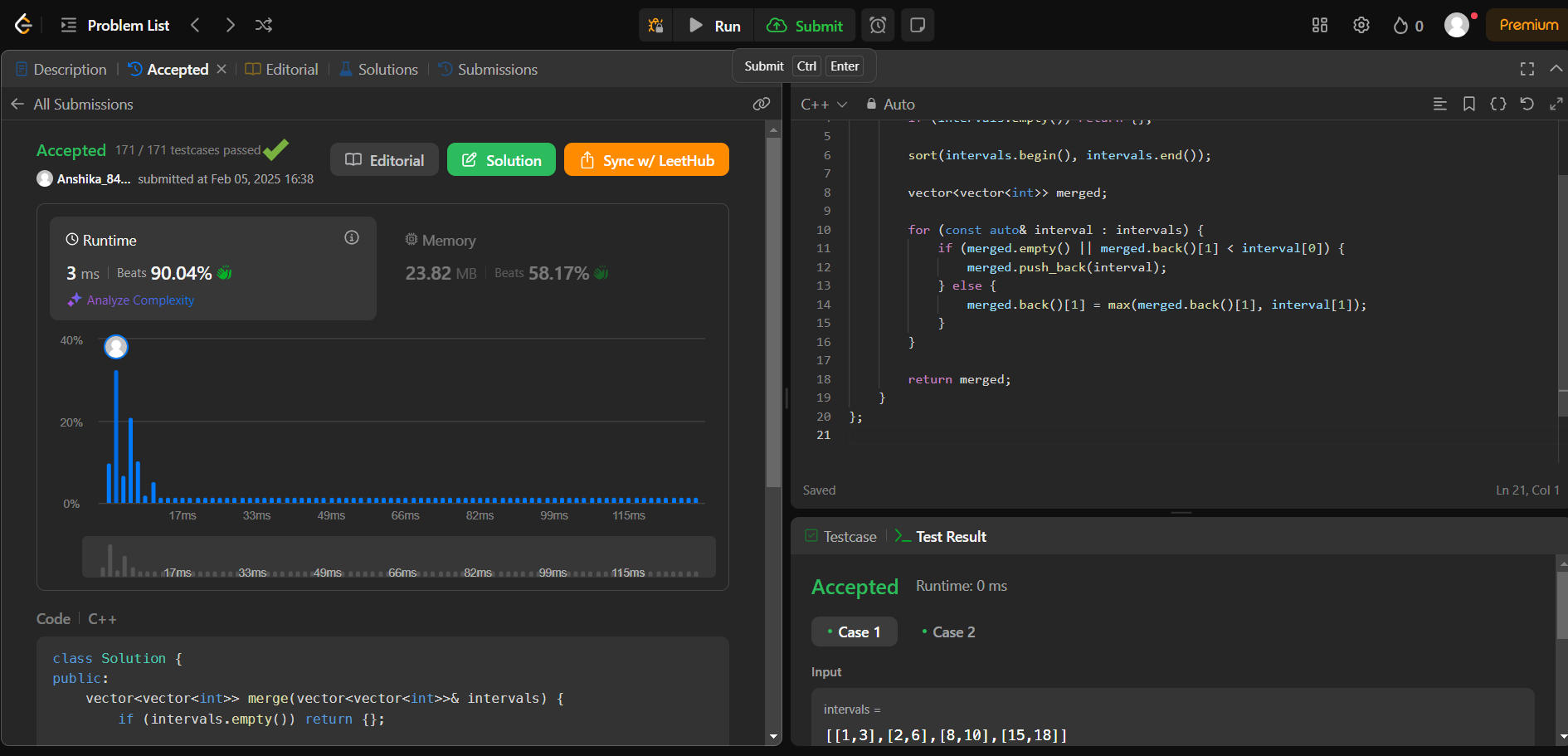
}

}

return merged;

}

};



1. Find Peak Element

class Solution {

public:

int findPeakElement(vector<int>& nums) {

int left = 0, 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;

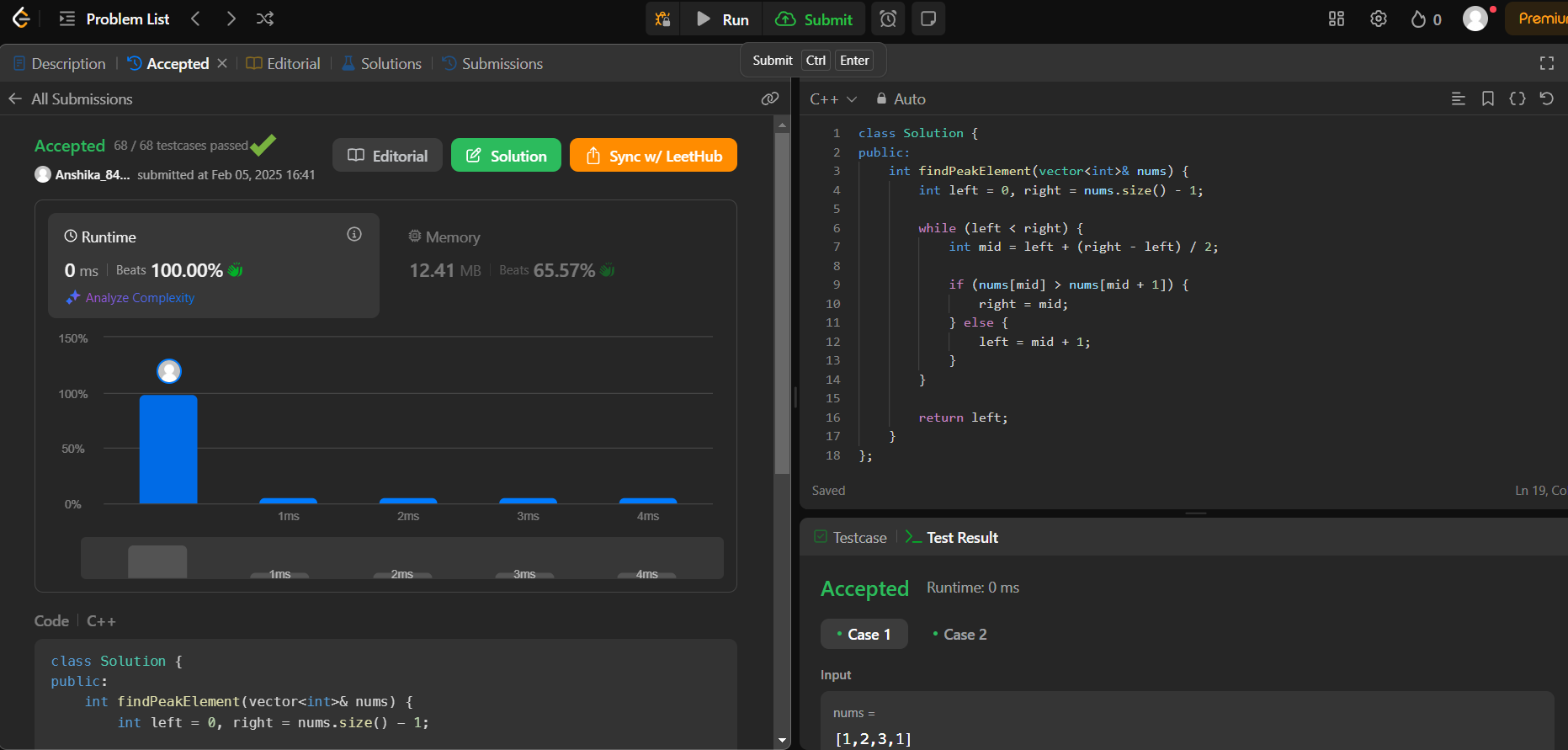
}

}

return left;

}

};



1. Kth largest element in an array

class Solution {

public:

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

return quickselect(nums, 0, nums.size() - 1, nums.size() - k);

}

private:

int quickselect(vector<int>& nums, int left, int right, int k) {

int pivot = partition(nums, left, right);

if (pivot == k) {

return nums[pivot];

} else if (pivot < k) {

return quickselect(nums, pivot + 1, right, k);

} else {

return quickselect(nums, left, pivot - 1, k);

}

}

int partition(vector<int>& nums, int left, int right) {

int pivot = nums[right];

int i = left;

for (int j = left; j < right; ++j) {

if (nums[j] <= pivot) {

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

i++;

}

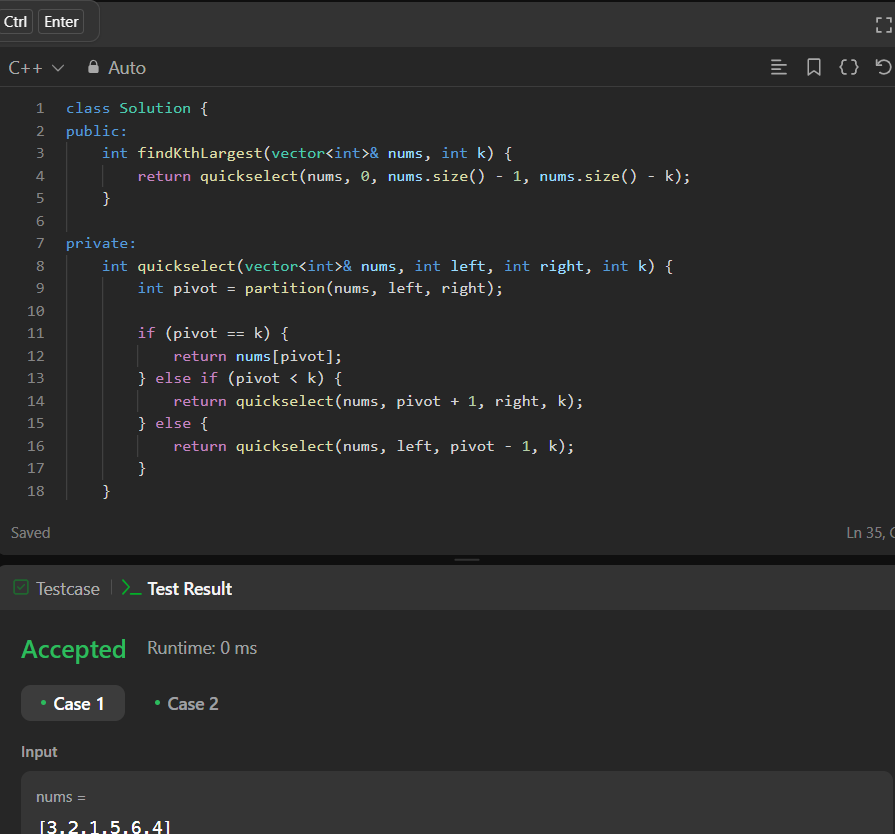
}

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

return i;

}

};



1. Top k frequent elements

class Solution {

public:

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

unordered\_map<int, int> freq;

for (int num : nums) {

freq[num]++;

}

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> minHeap;

for (auto& entry : freq) {

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

if (minHeap.size() > k) {

minHeap.pop();

}

}

vector<int> result;

while (!minHeap.empty()) {

result.push\_back(minHeap.top().second);

minHeap.pop();

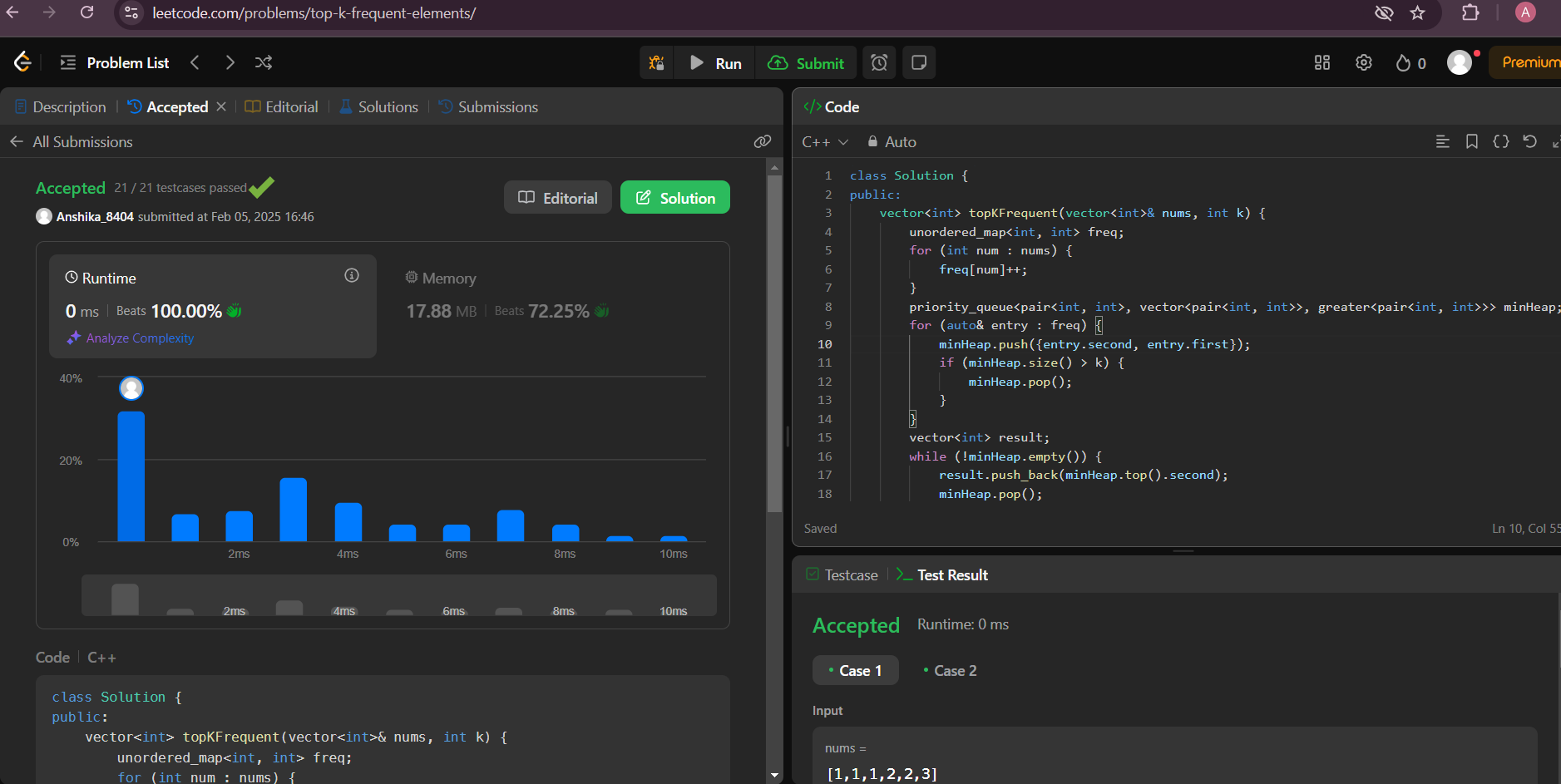
}

reverse(result.begin(), result.end());

return result;

}

};



1. First Bad Version

class Solution {

public:

    int firstBadVersion(int n) {

        int left = 1, right = n;

        while (left < right) {

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

            if (isBadVersion(mid)) {

                right = mid;

            } else {

                left = mid + 1;

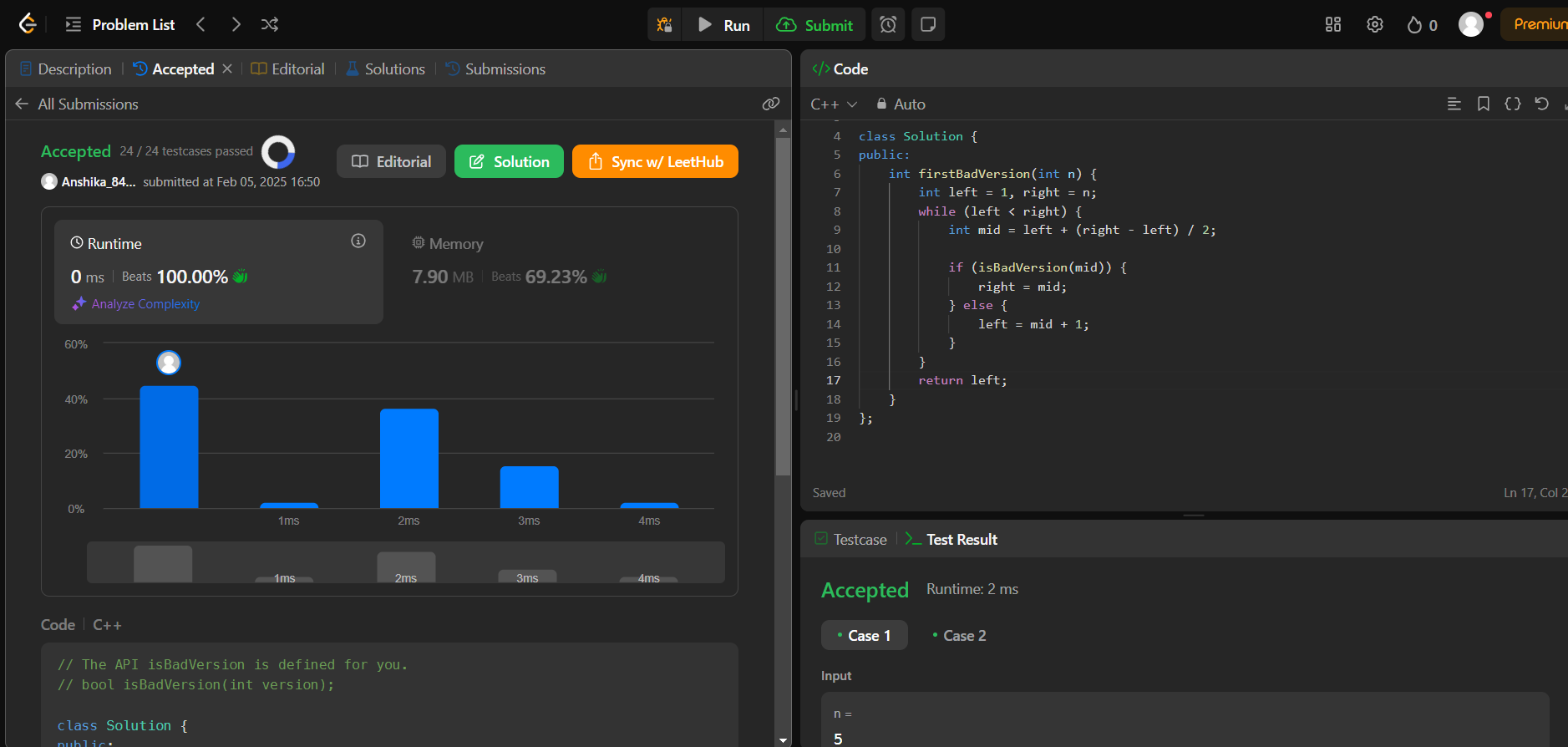
            }

        }

        return left;

    }

};



1. **Beautiful Array**

class Solution {

public:

    vector<int> beautifulArray(int n) {

        vector<int> result = {1};

        while (result.size() < n) {

            vector<int> next;

            for (int num : result) {

                if (num \* 2 - 1 <= n) {

                    next.push\_back(num \* 2 - 1);

                }

            }

            for (int num : result) {

                if (num \* 2 <= n) {

                    next.push\_back(num \* 2);

                }

            }

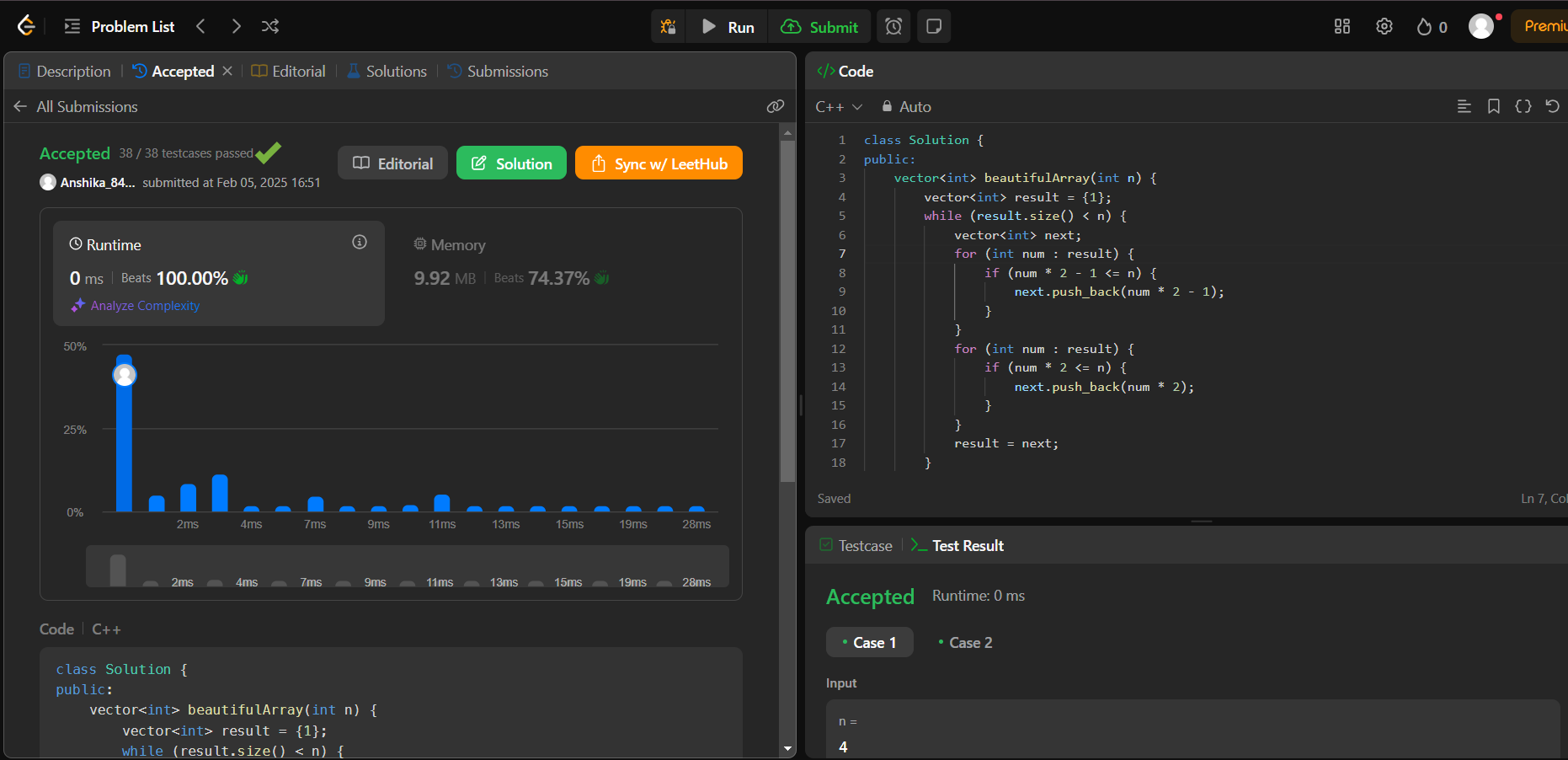
            result = next;

        }

        return result;

    }

};



1. Super Pow

class Solution {

public:

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

int mod = 1337;

a = a % mod;

int result = 1;

for (int i = b.size() - 1; i >= 0; --i) {

result = (result \* modPow(a, b[i], mod)) % mod;

a = modPow(a, 10, mod);

}

return result;

}

int modPow(int x, int y, int mod) {

int res = 1;

x = x % mod;

while (y > 0) {

if (y % 2 == 1) {

res = (res \* x) % mod;

}

x = (x \* x) % mod;

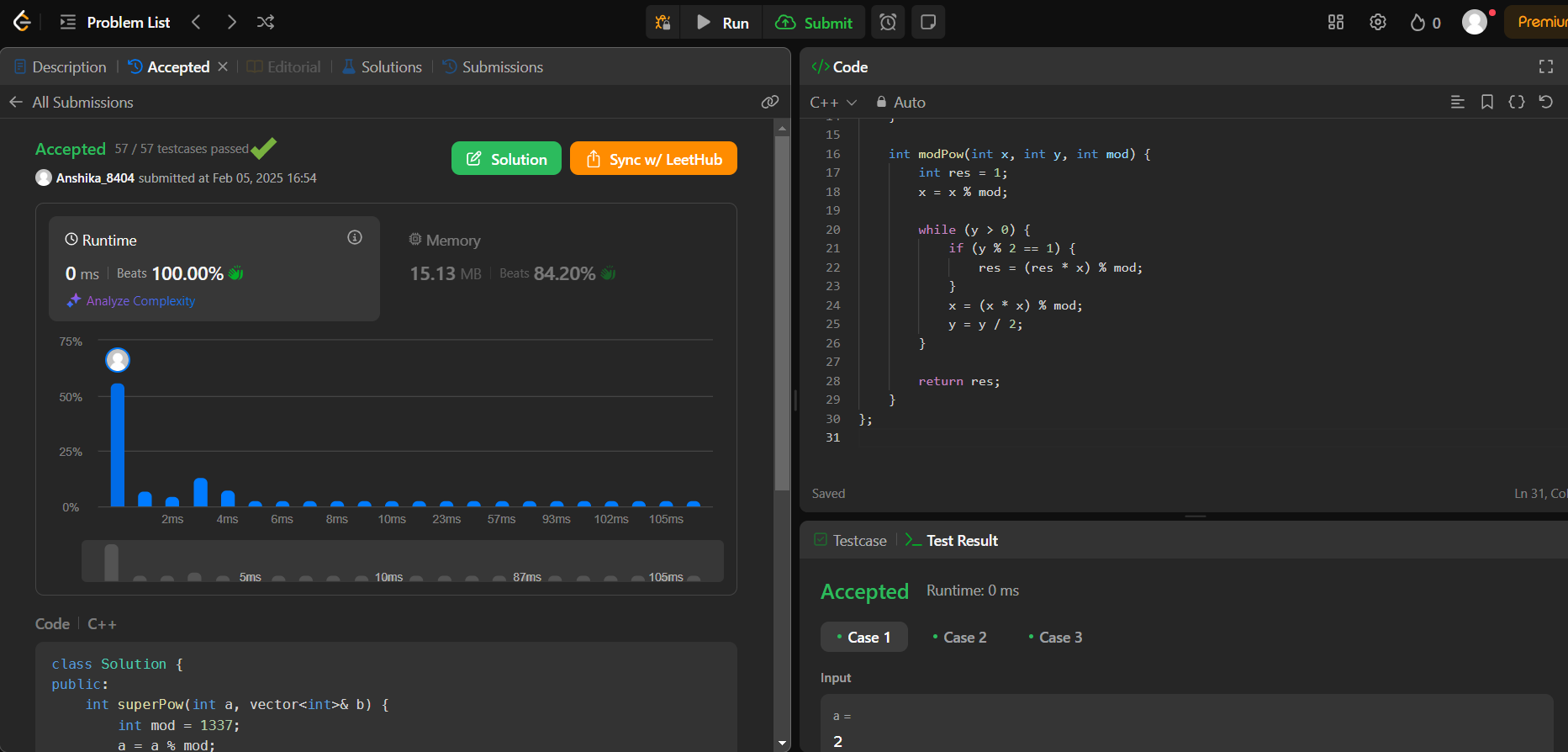
y = y / 2;

}

return res;

}

};



1. Maximum Subarray

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int currentSum = nums[0];

int maxSum = nums[0];

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

currentSum = max(nums[i], currentSum + nums[i]);

maxSum = max(maxSum, currentSum);

}

return maxSum;

}

};

