**Name: Vanshika UID: 22BCS15478**

**Assignment-2**

1. **Longest Nice Substring**import java.util.HashSet;

import java.util.Scanner;

public class LongestNiceSubstring {

public String longestNiceSubstring(String s) {

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

HashSet<Character> set = new HashSet<>();

for (char c : s.toCharArray()) {

set.add(c);

}

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

char c = s.charAt(i);

if (set.contains(Character.toUpperCase(c)) && set.contains(Character.toLowerCase(c))) continue;

String left = longestNiceSubstring(s.substring(0, i));

String right = longestNiceSubstring(s.substring(i + 1));

return left.length() >= right.length() ? left : right;

}

return s;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string: ");

String input = scanner.nextLine();

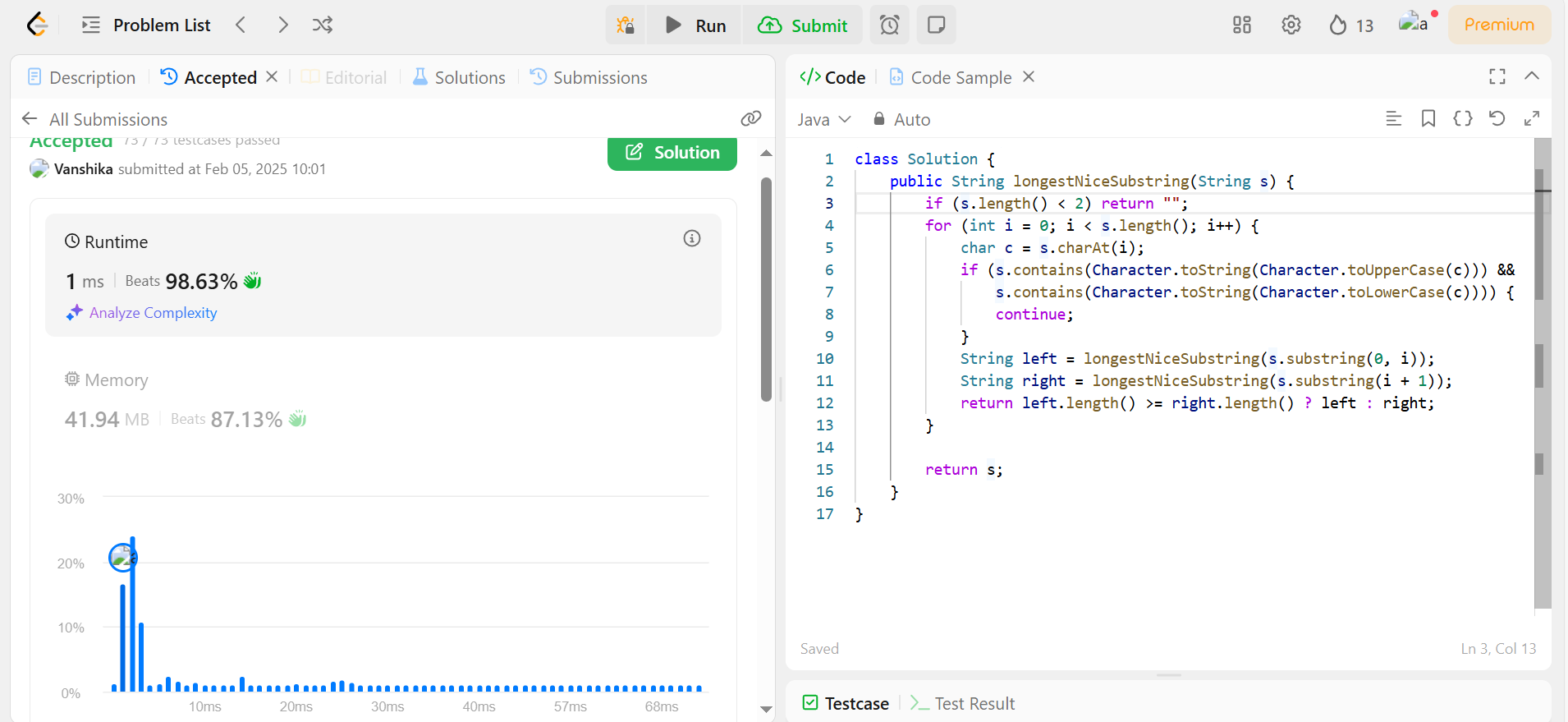
scanner.close();

LongestNiceSubstring solution = new LongestNiceSubstring();

System.out.println("Longest nice substring: " + solution.longestNiceSubstring(input));

}

}



1. **Reverse Bits**

import java.util.Scanner;

public class ReverseBits {

public int reverseBits(int n) {

int result = 0;

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

result <<= 1;

result |= (n & 1);

n >>= 1;

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter an integer: ");

int n = scanner.nextInt();

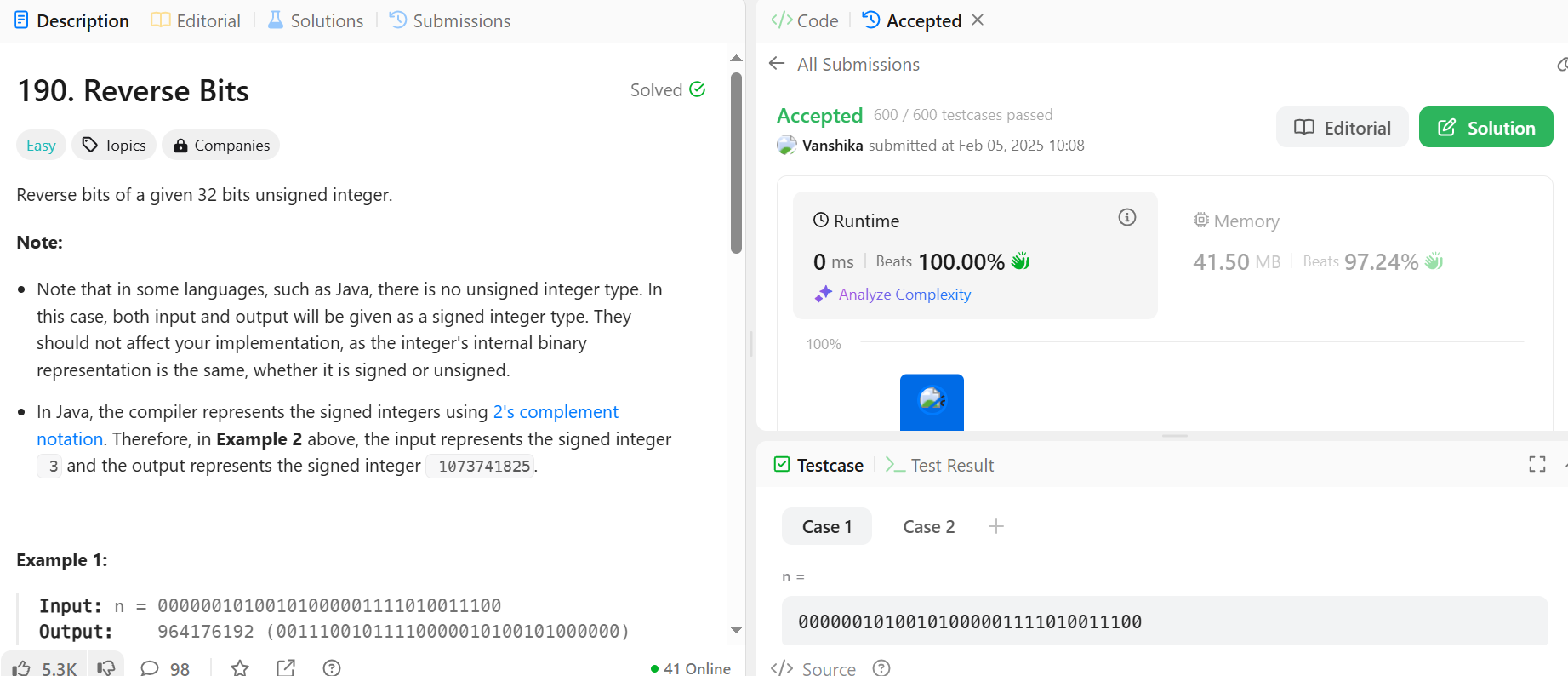
scanner.close();

ReverseBits solution = new ReverseBits();

System.out.println("Reversed bits: " + solution.reverseBits(n));

}

}

****

1. **Number of 1 Bits**

import java.util.Scanner;

public class HammingWeight {

public int hammingWeight(int n) {

int count = 0;

while (n != 0) {

count += (n & 1);

n >>>= 1;

}

return count;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter an integer: ");

int n = scanner.nextInt();

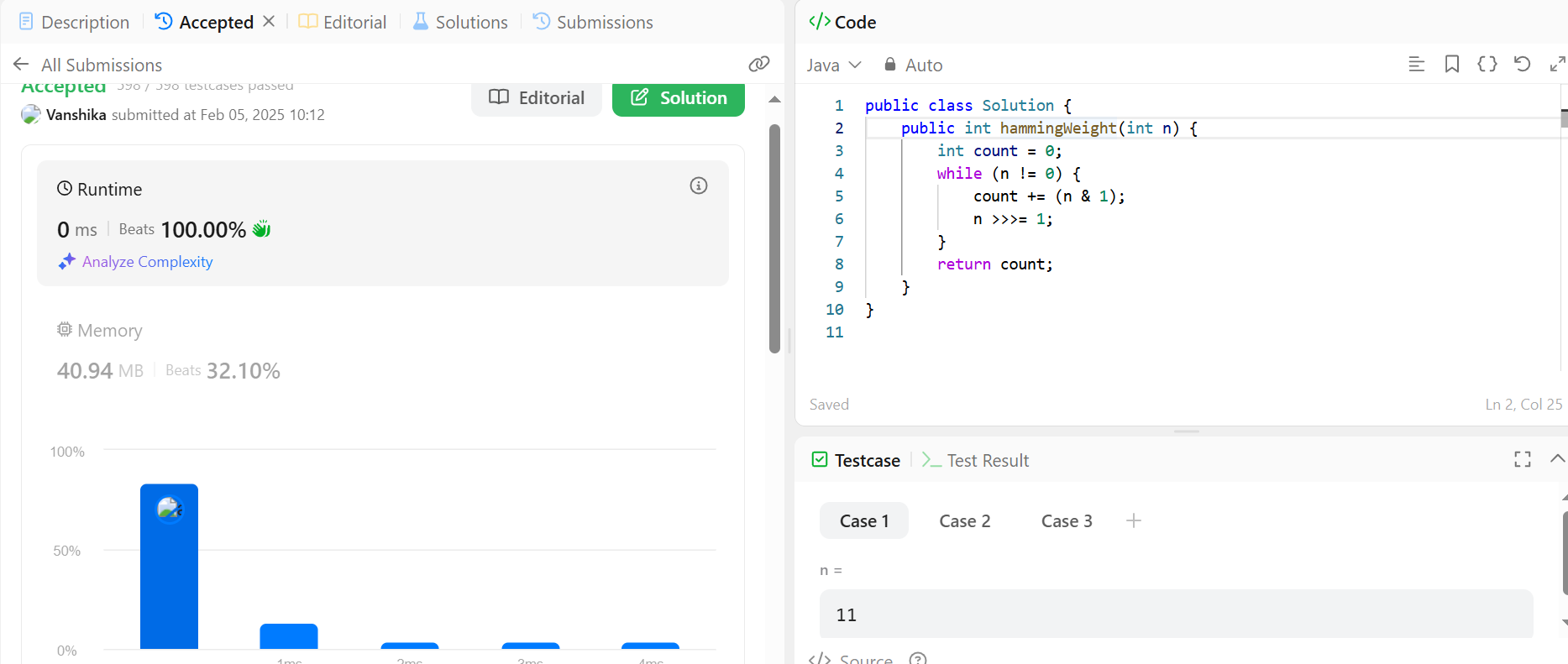
scanner.close();

HammingWeight solution = new HammingWeight();

System.out.println("Number of 1 bits: " + solution.hammingWeight(n));

}

}



1. **Maximum Subarray**

import java.util.Scanner;

public class MaxSubarray {

public int maxSubArray(int[] nums) {

int maxSum = nums[0], currentSum = nums[0];

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

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

maxSum = Math.max(maxSum, currentSum);

}

return maxSum;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the elements of the array:");

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

nums[i] = scanner.nextInt();

}

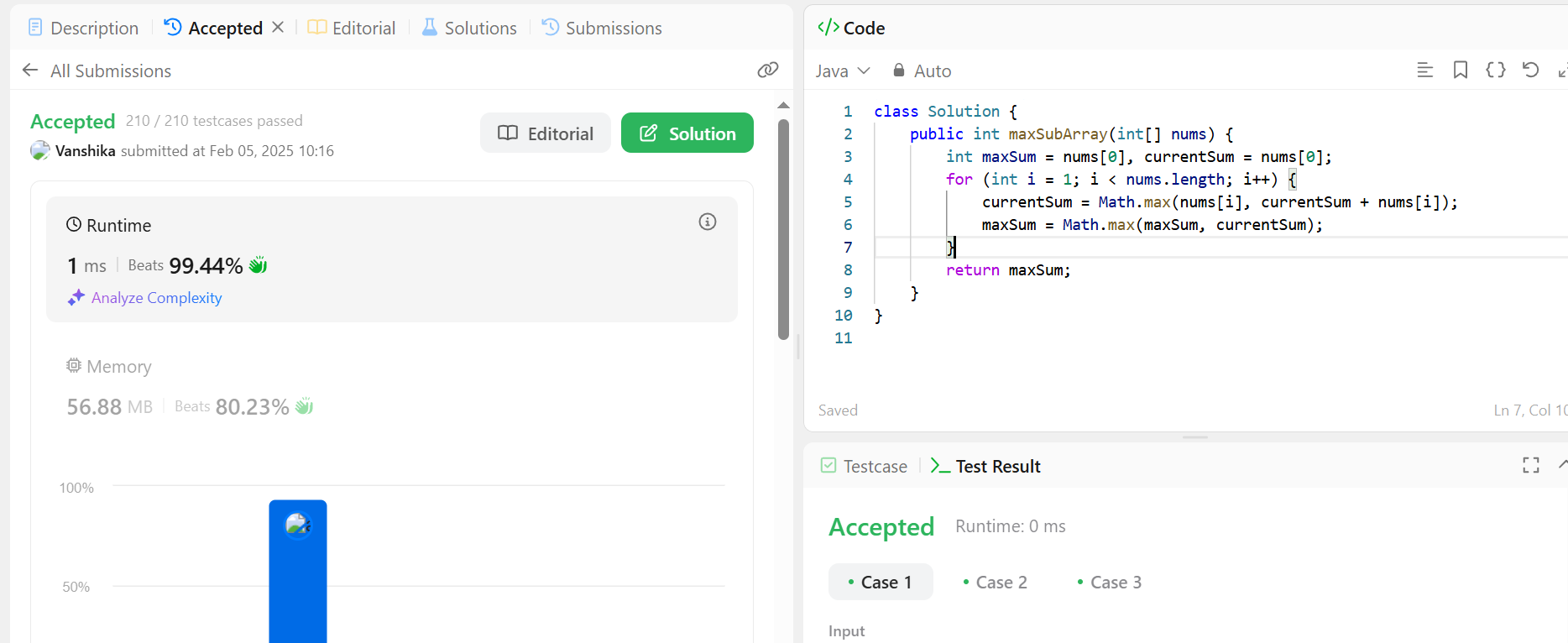
scanner.close();

MaxSubarray solution = new MaxSubarray();

System.out.println("Maximum subarray sum: " + solution.maxSubArray(nums));

}

}



1. **Search a 2D Matrix II**

import java.util.Scanner;

public class Search2DMatrix {

public boolean searchMatrix(int[][] matrix, int target) {

int row = 0, col = matrix[0].length - 1;

while (row < matrix.length && col >= 0) {

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

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

else row++;

}

return false;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of rows: ");

int rows = scanner.nextInt();

System.out.print("Enter number of columns: ");

int cols = scanner.nextInt();

int[][] matrix = new int[rows][cols];

System.out.println("Enter the elements row-wise:");

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

for (int j = 0; j < cols; j++) {

matrix[i][j] = scanner.nextInt();

}

}

System.out.print("Enter the target number: ");

int target = scanner.nextInt();

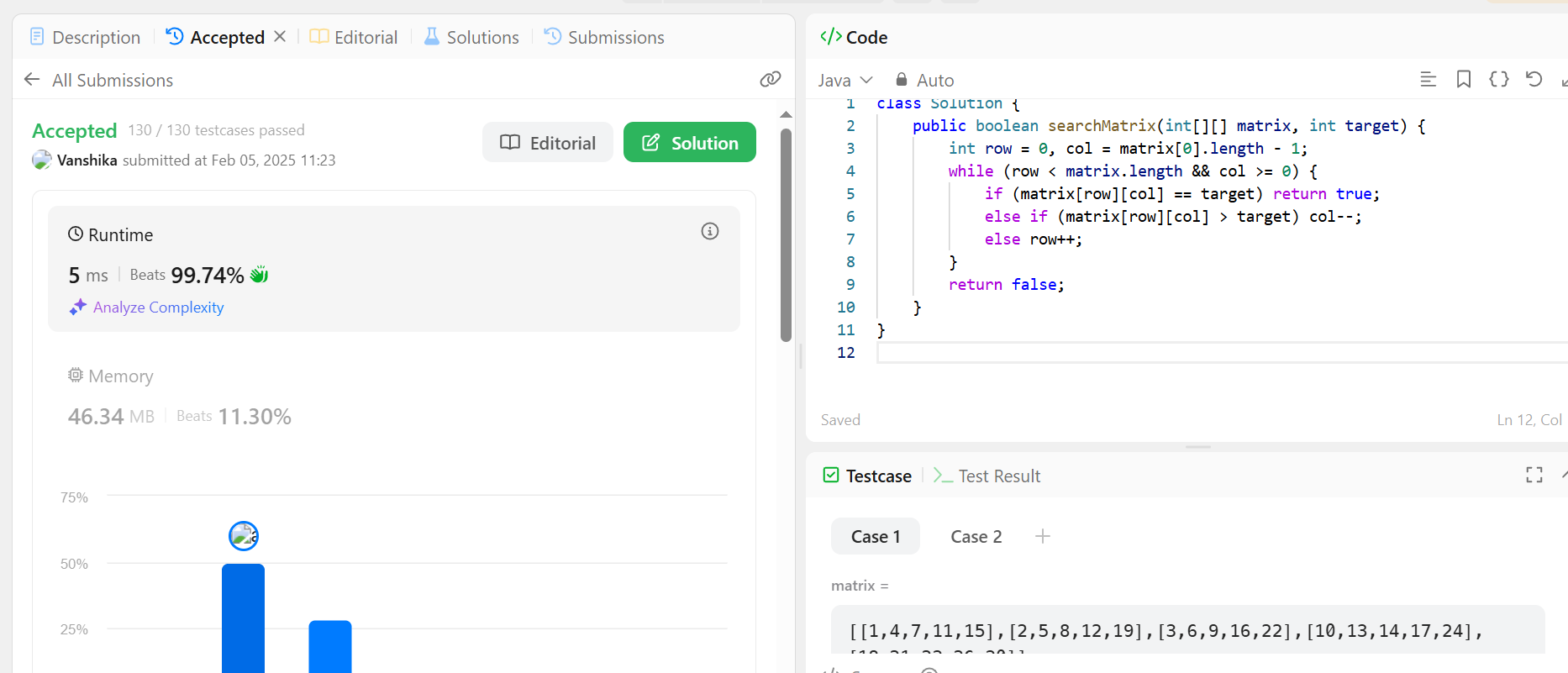
scanner.close();

Search2DMatrix solution = new Search2DMatrix();

System.out.println("Target found: " + solution.searchMatrix(matrix, target));

}

}



1. **Super Pow**

class Solution {

private static final int MOD = 1337;

public int superPow(int a, int[] b) {

a %= MOD;

return helper(a, b, b.length);

}

private int helper(int a, int[] b, int length) {

if (length == 0) return 1;

int lastDigit = b[length - 1];

int remainingPow = helper(a, b, length - 1);

return powerMod(remainingPow, 10) \* powerMod(a, lastDigit) % MOD;

}

private int powerMod(int base, int exp) {

int result = 1;

while (exp > 0) {

if (exp % 2 == 1) {

result = result \* base % MOD;

}

base = base \* base % MOD;

exp /= 2;

}

return result;

}

public static void main(String[] args) {

Solution solution = new Solution();

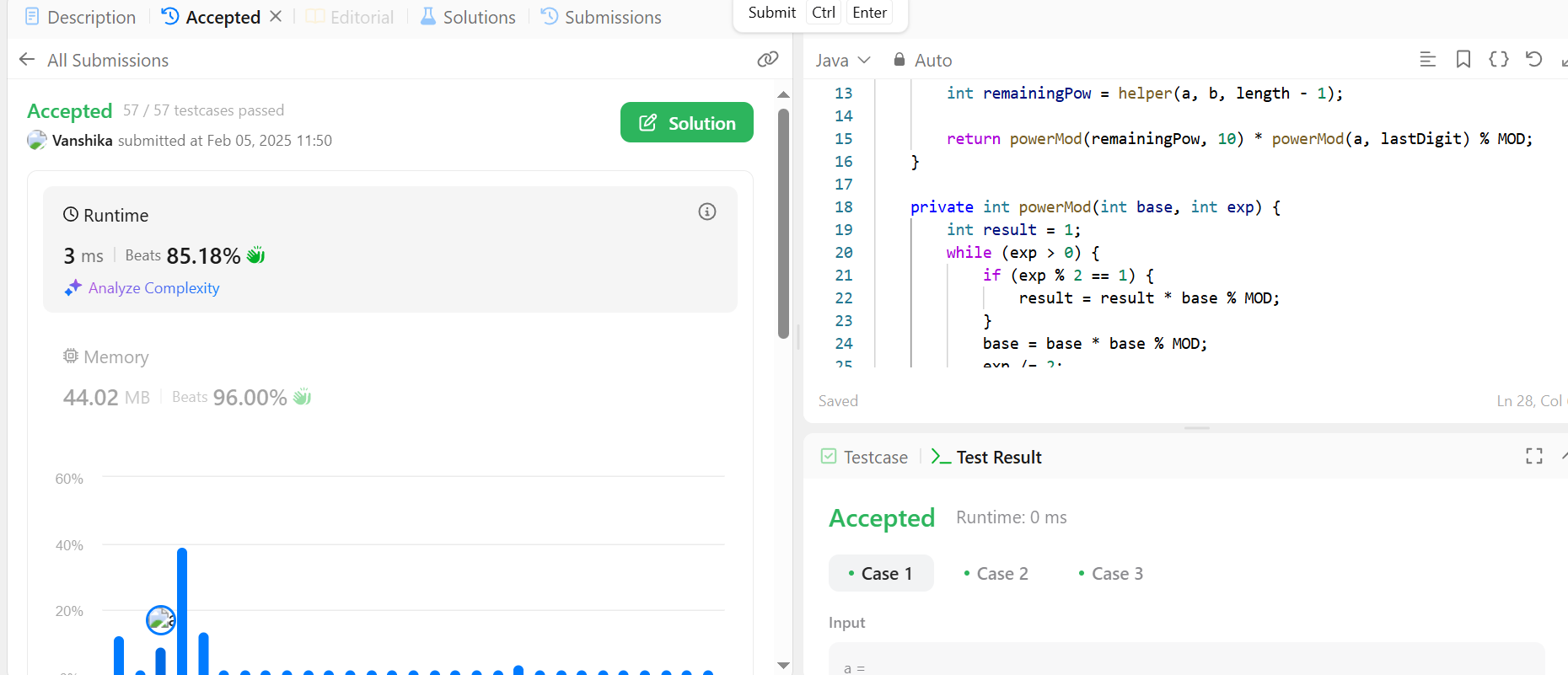
int a = 2;

int[] b = {1, 0}; // 2^10 mod 1337 = 1024

System.out.println("Result: " + solution.superPow(a, b));

}

}



1. **Beautiful Array**

import java.util.ArrayList;

import java.util.List;

class Solution {

public static void test(int start , int increment , ArrayList<Integer> ans , int n){

if(start + increment > n){

ans.add(start);

return;

}

test(start , 2 \* increment , ans , n);

test(start + increment , 2 \* increment , ans , n);

}

public int[] beautifulArray(int n) {

ArrayList<Integer> ans = new ArrayList<>();

test(1,1,ans, n);

int arr[] = new int[n];

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

arr[i] = ans.get(i);

}

return arr;

}

public static void main(String[] args) {

Solution solution = new Solution();

int n = 5; // Example input

int[] beautifulArr = solution.beautifulArray(n);

System.out.print("Beautiful Array: ");

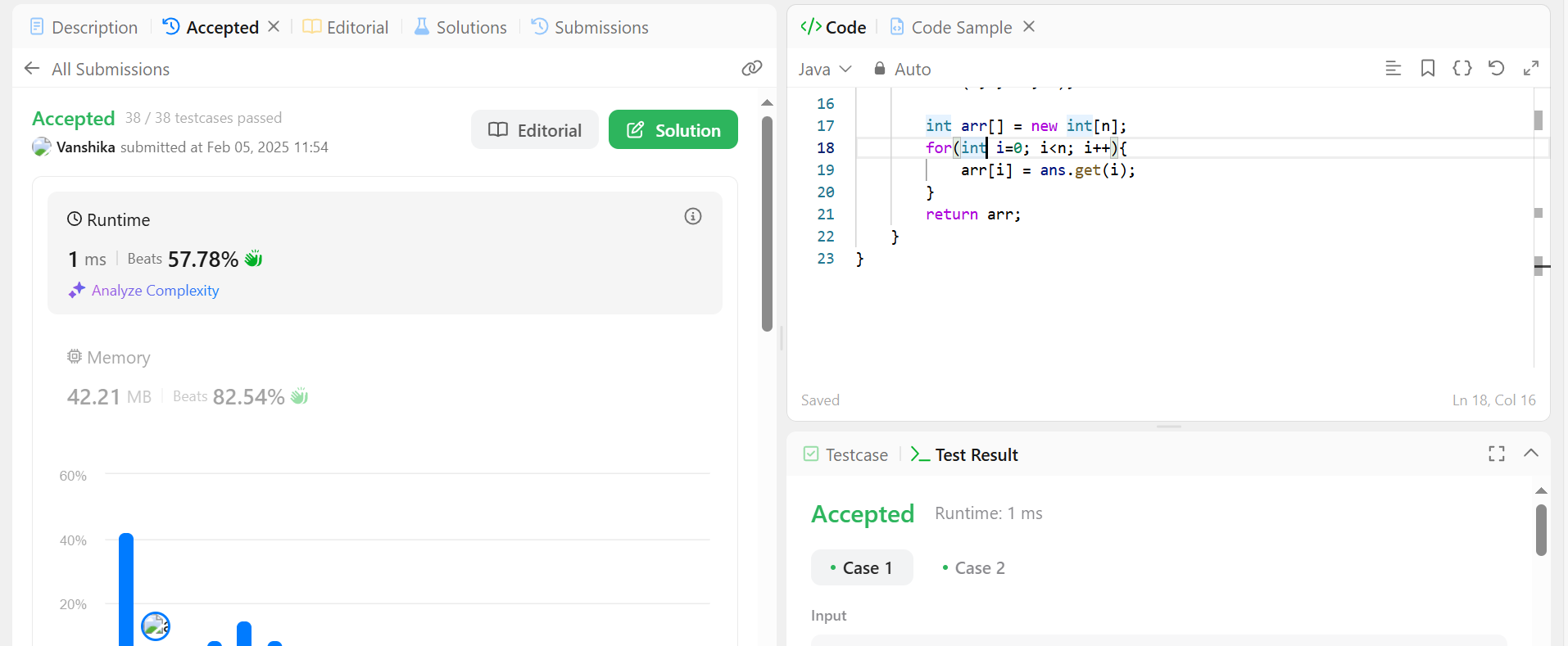
for (int num : beautifulArr) {

System.out.print(num + " ");

}

}

}



1. **The Skyline Problem**

import java.util.\*;

class Solution {

public List<List<Integer>> getSkyline(int[][] buildings) {

return new AbstractList<List<Integer>>() {

private List<List<Integer>> resList;

private void onload() {

resList = new ArrayList<>();

List<int[]> heights = new ArrayList<>();

for (int[] building : buildings) {

heights.add(new int[]{building[0], -building[2]});

heights.add(new int[]{building[1], building[2]});

}

heights.sort((a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));

PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

pq.offer(0);

int prev = 0;

for (int[] height : heights) {

if (height[1] < 0) {

pq.offer(-height[1]);

} else {

pq.remove(height[1]);

}

int cur = pq.peek();

if (prev != cur) {

resList.add(Arrays.asList(height[0], cur));

prev = cur;

}

}

}

private void init() {

if (resList == null) {

onload();

}

}

@Override

public List<Integer> get(int index) {

init();

return resList.get(index);

}

@Override

public int size() {

init();

return resList.size();

}

};

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[][] buildings = new int[n][3];

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

buildings[i][0] = scanner.nextInt();

buildings[i][1] = scanner.nextInt();

buildings[i][2] = scanner.nextInt();

}

scanner.close();

Solution solution = new Solution();

List<List<Integer>> skyline = solution.getSkyline(buildings);

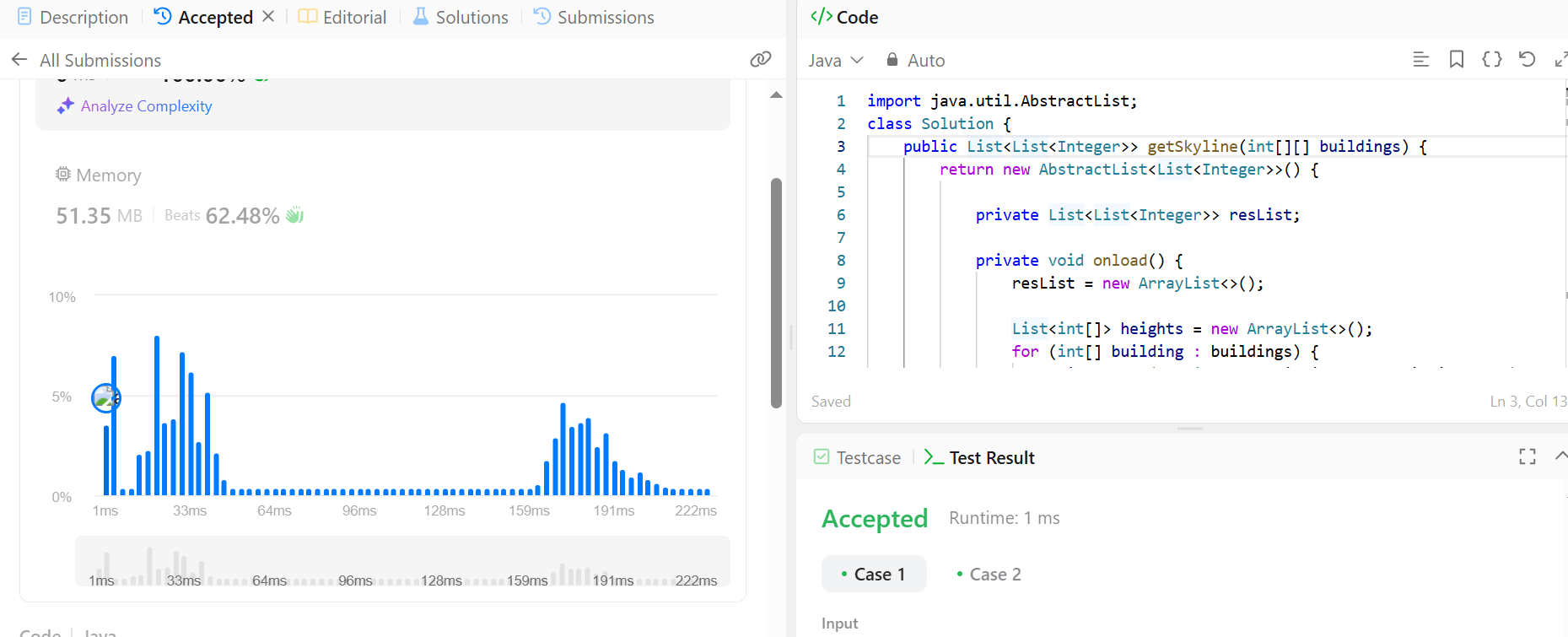
for (List<Integer> point : skyline) {

System.out.println(point.get(0) + " " + point.get(1));

}

}

}



1. **Reverse Pairs**

import java.util.\*;

class Solution {

public int reversePairs(int[] nums) {

return mergeSort(nums, 0, nums.length - 1);

}

private int mergeSort(int[] nums, int left, int right) {

if (left >= right) return 0;

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

int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);

int j = mid + 1;

for (int i = left; i <= mid; i++) {

while (j <= right && nums[i] > 2L \* nums[j]) j++;

count += j - (mid + 1);

}

merge(nums, left, mid, right);

return count;

}

private void merge(int[] nums, int left, int mid, int right) {

int[] temp = new int[right - left + 1];

int i = left, j = mid + 1, k = 0;

while (i <= mid && j <= right) {

if (nums[i] <= nums[j]) {

temp[k++] = nums[i++];

} else {

temp[k++] = nums[j++];

}

}

while (i <= mid) temp[k++] = nums[i++];

while (j <= right) temp[k++] = nums[j++];

System.arraycopy(temp, 0, nums, left, temp.length);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] nums = new int[n];

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

nums[i] = scanner.nextInt();

}

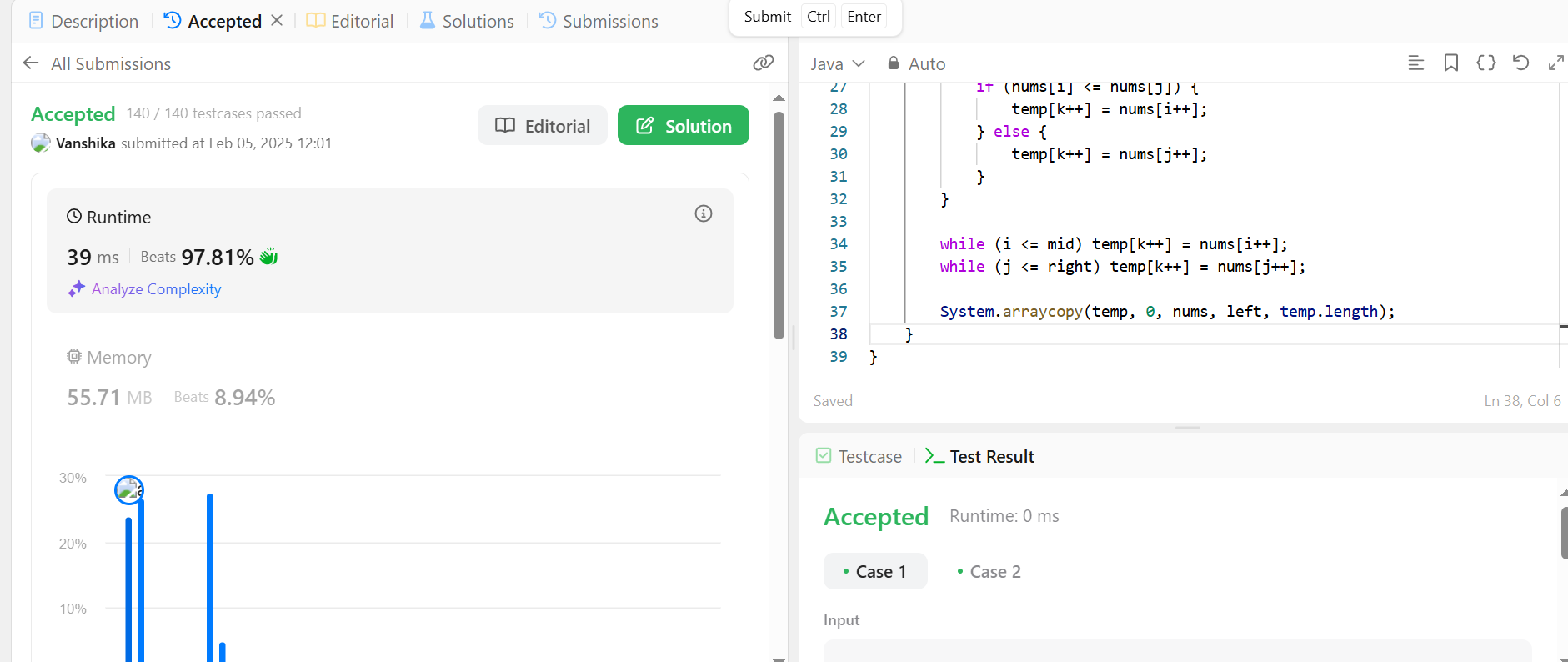
scanner.close();

Solution solution = new Solution();

System.out.println("Reverse Pairs: " + solution.reversePairs(nums));

}

}



1. **Longest Increasing Subsequence II**

import java.util.\*;

class SegmentTree {

int[] tree;

int size;

public SegmentTree(int maxVal) {

size = maxVal + 1;

tree = new int[size \* 4];

}

private int query(int node, int start, int end, int left, int right) {

if (right < start || end < left) return 0;

if (left <= start && end <= right) return tree[node];

int mid = (start + end) / 2;

return Math.max(query(node \* 2, start, mid, left, right), query(node \* 2 + 1, mid + 1, end, left, right));

}

public int query(int left, int right) {

return query(1, 0, size - 1, left, right);

}

private void update(int node, int start, int end, int index, int value) {

if (start == end) {

tree[node] = value;

return;

}

int mid = (start + end) / 2;

if (index <= mid) update(node \* 2, start, mid, index, value);

else update(node \* 2 + 1, mid + 1, end, index, value);

tree[node] = Math.max(tree[node \* 2], tree[node \* 2 + 1]);

}

public void update(int index, int value) {

update(1, 0, size - 1, index, value);

}

}

class Solution {

public int lengthOfLIS(int[] nums, int k) {

int maxVal = Arrays.stream(nums).max().getAsInt();

SegmentTree segmentTree = new SegmentTree(maxVal);

int maxLen = 0;

for (int num : nums) {

int bestPrev = segmentTree.query(Math.max(0, num - k), num - 1);

int newLen = bestPrev + 1;

segmentTree.update(num, newLen);

maxLen = Math.max(maxLen, newLen);

}

return maxLen;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int k = scanner.nextInt();

int[] nums = new int[n];

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

nums[i] = scanner.nextInt();

}

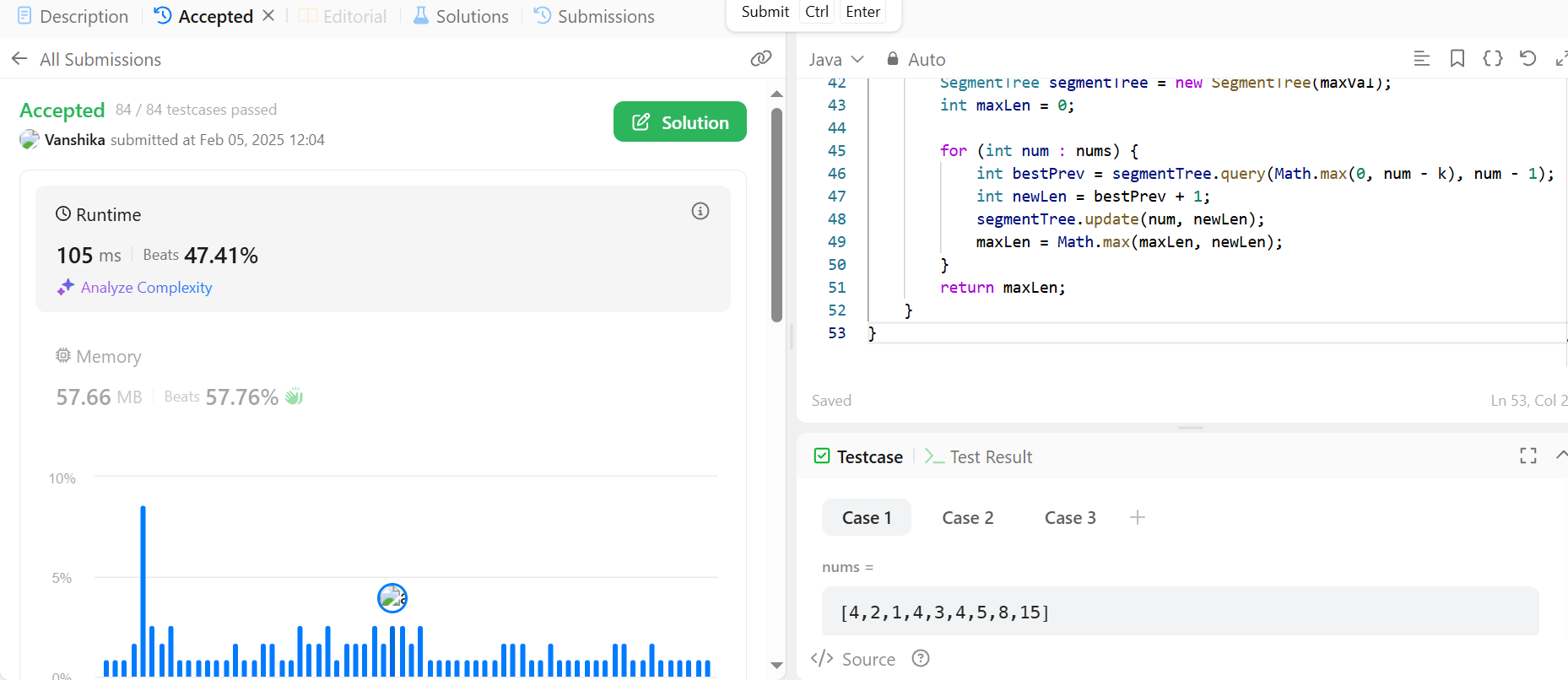
scanner.close();

Solution solution = new Solution();

System.out.println(solution.lengthOfLIS(nums, k));

}

}



1. **Merge Sorted Array**

import java.util.\*;

class Solution {

public void merge(int[] nums1, int m, 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--];

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int m = scanner.nextInt();

int n = scanner.nextInt();

int[] nums1 = new int[m + n];

int[] nums2 = new int[n];

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

nums1[i] = scanner.nextInt();

}

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

nums2[i] = scanner.nextInt();

}

scanner.close();

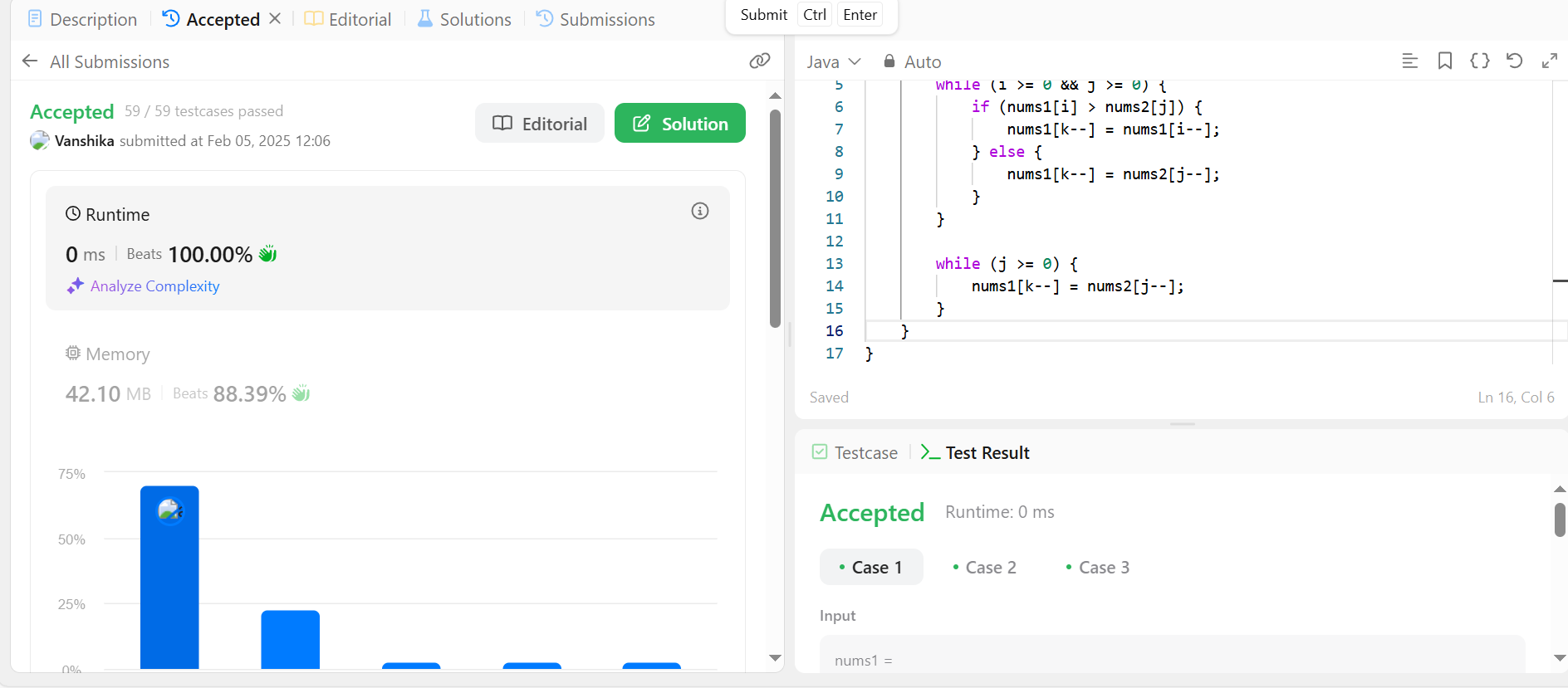
Solution solution = new Solution();

solution.merge(nums1, m, nums2, n);

System.out.println(Arrays.toString(nums1));

}

}



1. **First Bad Version**

import java.util.Scanner;

abstract class VersionControl {

abstract boolean isBadVersion(int version);

}

class Solution extends VersionControl {

public boolean isBadVersion(int version) {

return version >= 4; // Change this based on the first bad version

}

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;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

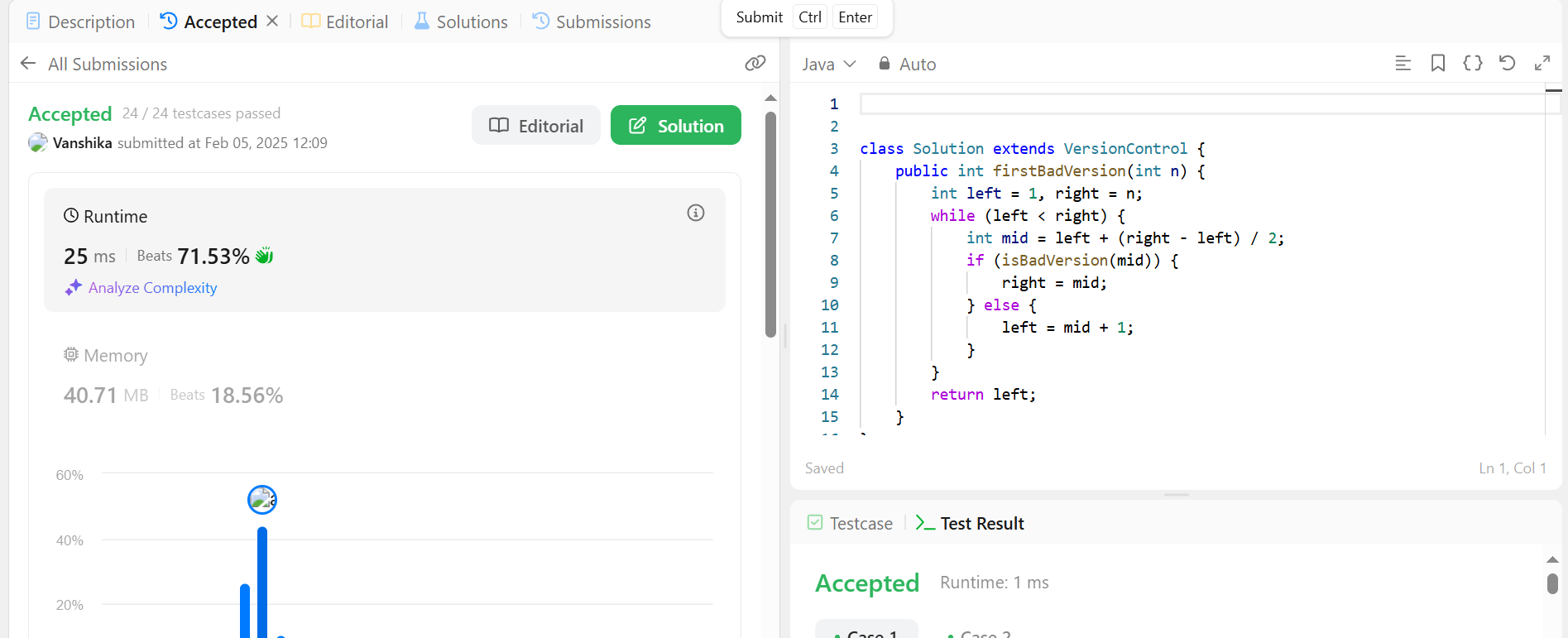
scanner.close();

Solution solution = new Solution();

System.out.println(solution.firstBadVersion(n));

}

}



1. **Sort Colors**

import java.util.\*;

class Solution {

public void sortColors(int[] nums) {

int low = 0, mid = 0, high = nums.length - 1;

while (mid <= high) {

if (nums[mid] == 0) {

int temp = nums[low];

nums[low] = nums[mid];

nums[mid] = temp;

low++;

mid++;

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

mid++;

} else {

int temp = nums[mid];

nums[mid] = nums[high];

nums[high] = temp;

high--;

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] nums = new int[n];

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

nums[i] = scanner.nextInt();

}

scanner.close();

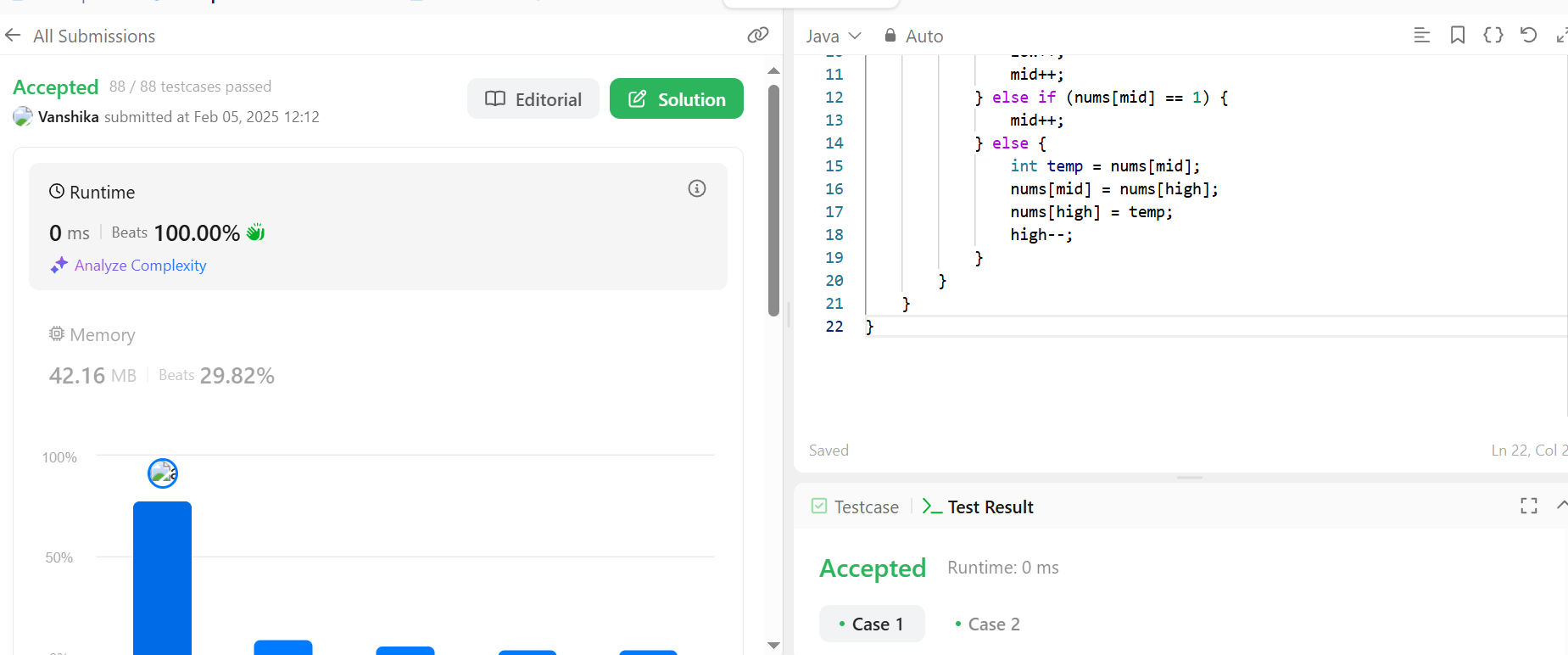
Solution solution = new Solution();

solution.sortColors(nums);

System.out.println(Arrays.toString(nums));

}

}



1. **Top K Frequent Elements**

import java.util.\*;

class Solution {

public int[] topKFrequent(int[] nums, int k) {

Map<Integer, Integer> freqMap = new HashMap<>();

for (int num : nums) {

freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);

}

PriorityQueue<Integer> minHeap = new PriorityQueue<>(Comparator.comparingInt(freqMap::get));

for (int num : freqMap.keySet()) {

minHeap.offer(num);

if (minHeap.size() > k) {

minHeap.poll();

}

}

int[] result = new int[k];

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

result[i] = minHeap.poll();

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] nums = new int[n];

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

nums[i] = scanner.nextInt();

}

int k = scanner.nextInt();

scanner.close();

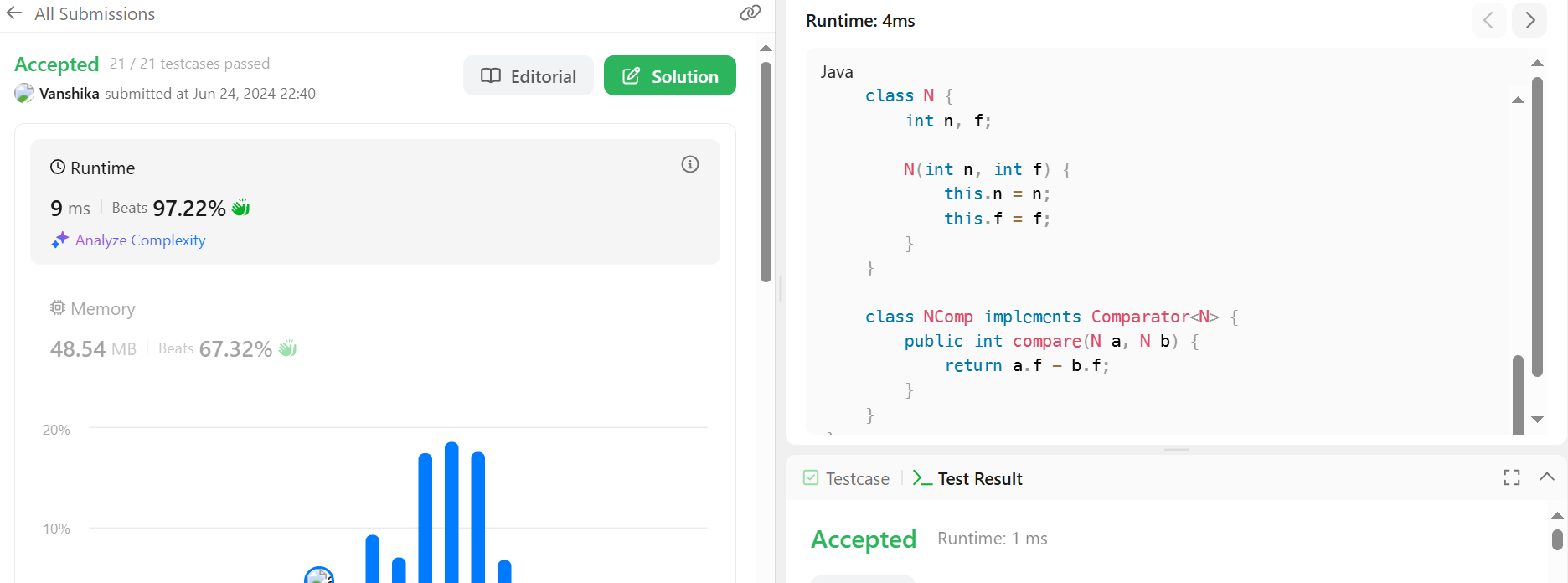
Solution solution = new Solution();

int[] result = solution.topKFrequent(nums, k);

System.out.println(Arrays.toString(result));

}

}



1. **Kth Largest Element in an Array**

import java.util.\*;

class Solution {

public int findKthLargest(int[] nums, int k) {

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

for (int num : nums) {

minHeap.offer(num);

if (minHeap.size() > k) {

minHeap.poll();

}

}

return minHeap.peek();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] nums = new int[n];

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

nums[i] = scanner.nextInt();

}

int k = scanner.nextInt();

scanner.close();

Solution solution = new Solution();

System.out.println(solution.findKthLargest(nums, k));

}

}

