**ASSIGNMENT -6 (ADVANCED PROGRAMMING)**

**PISINI JOEL – 22BCS12519**

1. **Problem 1:** Convert Sorted Array to Binary Search Tree
2. **Implementation/Code:**

class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

return constructBST(nums, 0, nums.length - 1); }

private TreeNode constructBST(int[] nums, int left, int right) {

if (left > right) {

return null; }

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

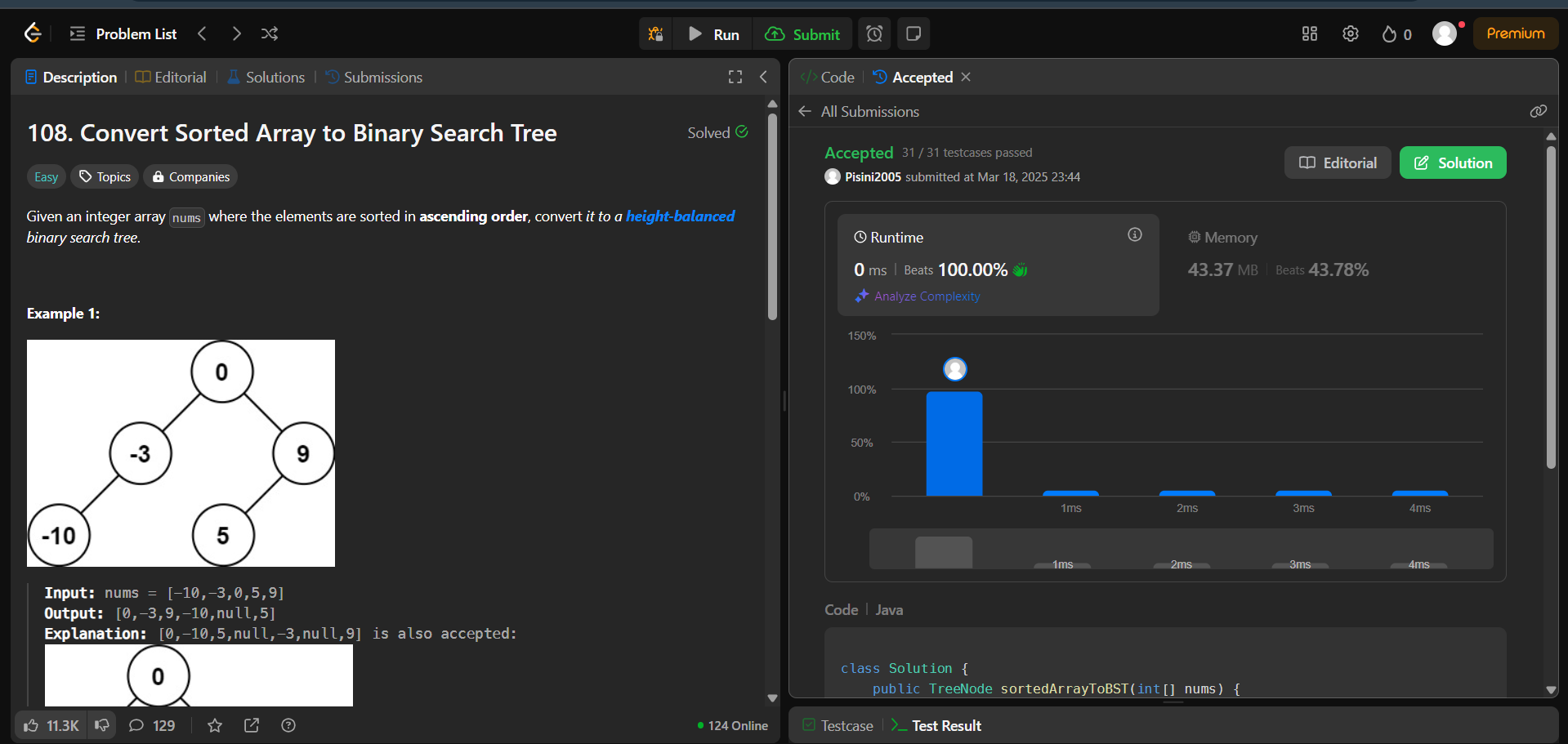
TreeNode root = new TreeNode(nums[mid]);

root.left = constructBST(nums, left, mid - 1);

root.right = constructBST(nums, mid + 1, right);

return root; }}

1. **Output:**

****

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

public class Solution {

public int hammingWeight(int n) {

int count = 0;

while (n != 0) {

count += (n & 1);

n >>>= 1;

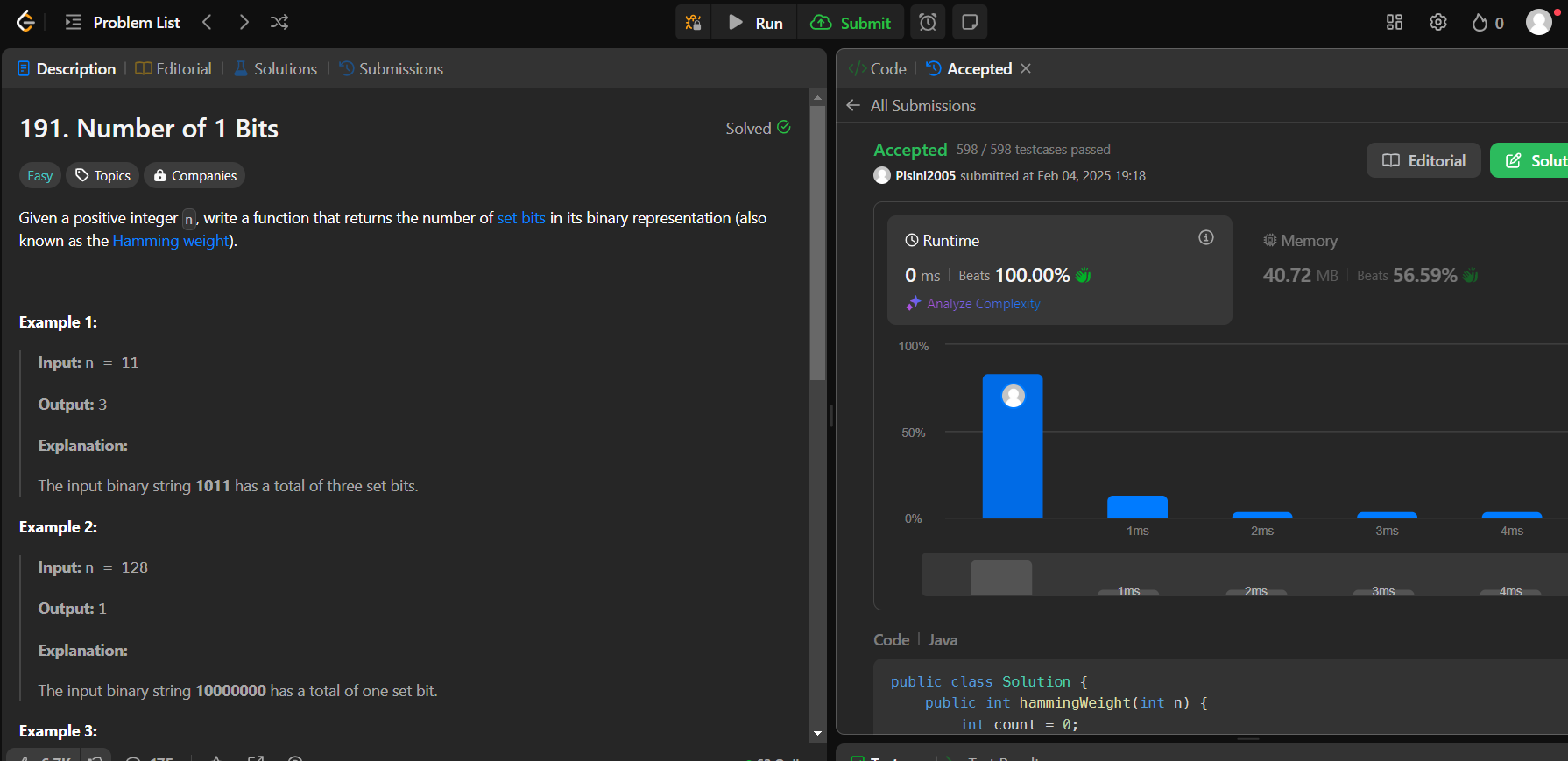
}

return count;

}

}

1. **Output:**



1. **Problem 3:** Sort an Array
2. **Implementation/Code:**

class Solution {

public int[] sortArray(int[] nums) {

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

return nums; }

public static void mergeFun(int[] arr, int l, int m, int r) {

int n1 = m + 1 - l; int n2 = r - m; int[] left = new int[n1];

for (int i = 0; i < n1; i++) {left[i] = arr[l + i]; }

int[] right = new int[n2];

for (int i = 0; i < n2; i++) { right[i] = arr[m + 1 + i]; }

int i = 0, j = 0, k = l;

while (i < n1 || j < n2) {

if (j == n2 || i < n1 && left[i] < right[j]) arr[k++] = left[i++];

else arr[k++] = right[j++]; } }

public static void mergeSort(int[] arr, int low, int high) {

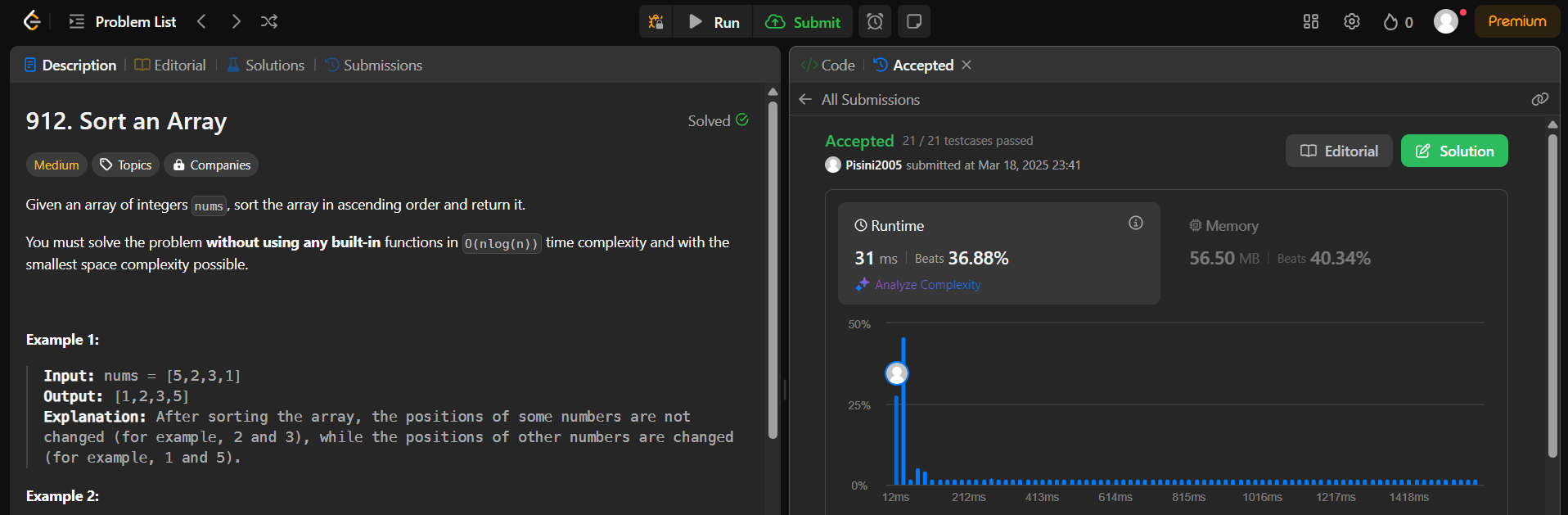
if (low < high) { int middle = (high - low) / 2 + low;

mergeSort(arr, low, middle);

mergeSort(arr, middle + 1, high);

mergeFun(arr, low, middle, high); } }}

1. **Output:**

****

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

public class Solution {

public int maxSubArray(int[] nums) {

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

for (int num : nums) {

currentSum = Math.max(num, currentSum + num);

maxSum = Math.max(maxSum, currentSum);

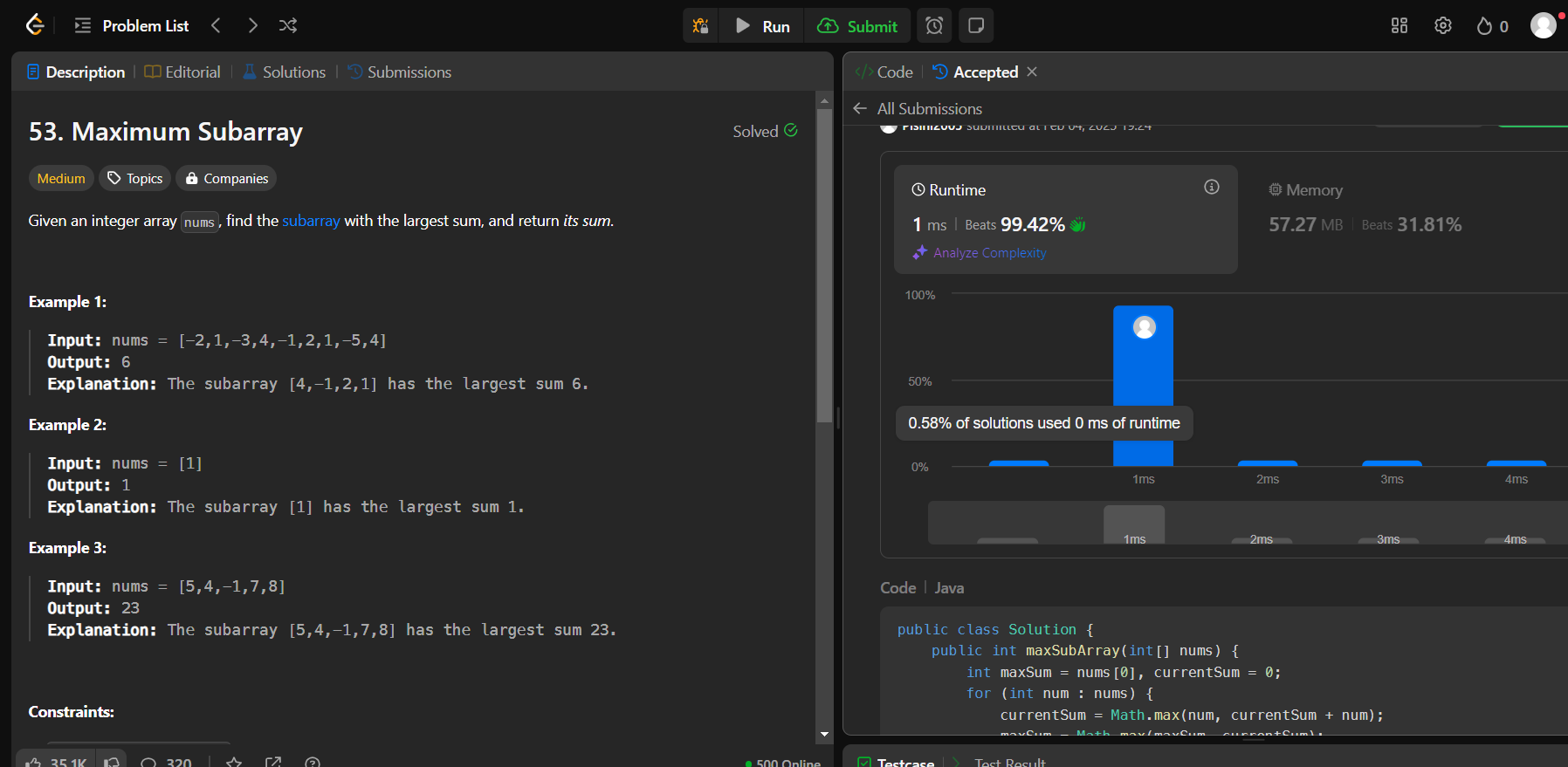
}

return maxSum;

}

}

1. **Output:**



1. **Problem 5:** Beautiful Array
2. **Implementation/Code:**

import java.util.\*;

public class Solution {

public int[] beautifulArray(int N) {

List<Integer> result = new ArrayList<>();

result.add(1);

while (result.size() < N) {

List<Integer> temp = new ArrayList<>();

for (int num : result) {

if (num \* 2 - 1 <= N) temp.add(num \* 2 - 1); }

for (int num : result) {

if (num \* 2 <= N) temp.add(num \* 2); }

result = temp; }

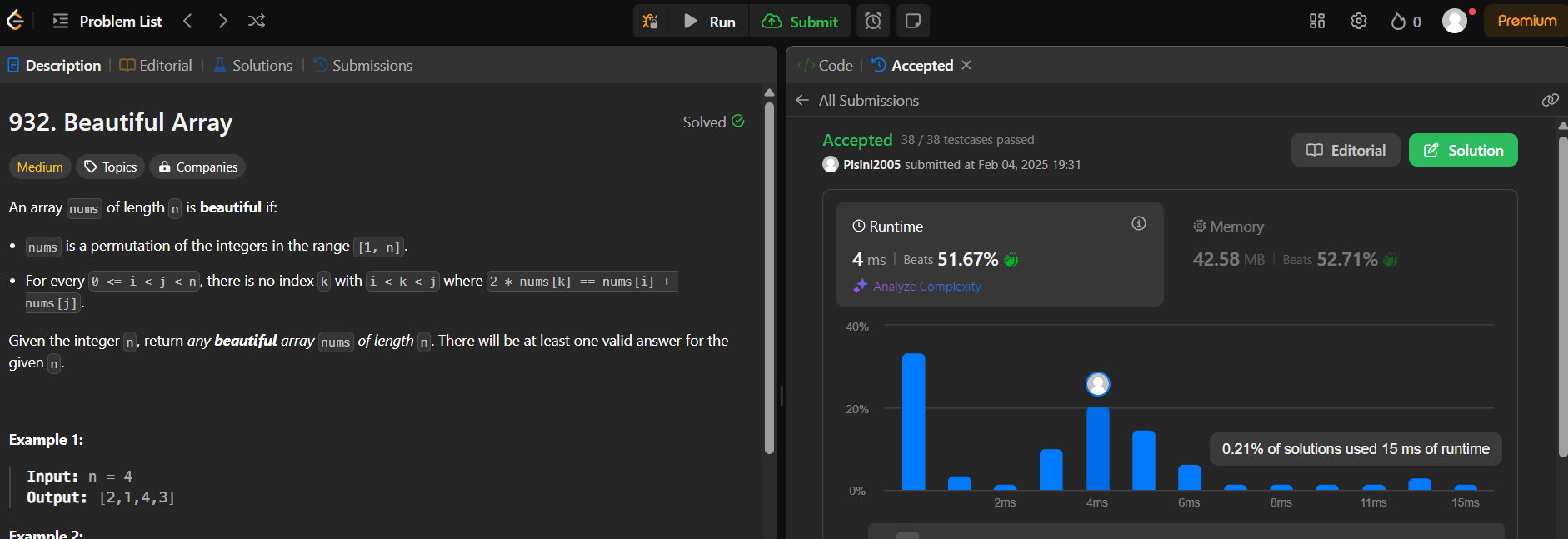
int[] arr = new int[result.size()];

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

arr[i] = result.get(i); }

return arr; }}

1. **Output:**

****

1. **Problem 6:** Super Pow
2. **Implementation/Code:**

public class Solution {

private static final int MOD = 1337;

private int pow(int a, int b) {

int res = 1;

a %= MOD;

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

res = (res \* a) % MOD; }

return res; }

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

int res = 1;

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

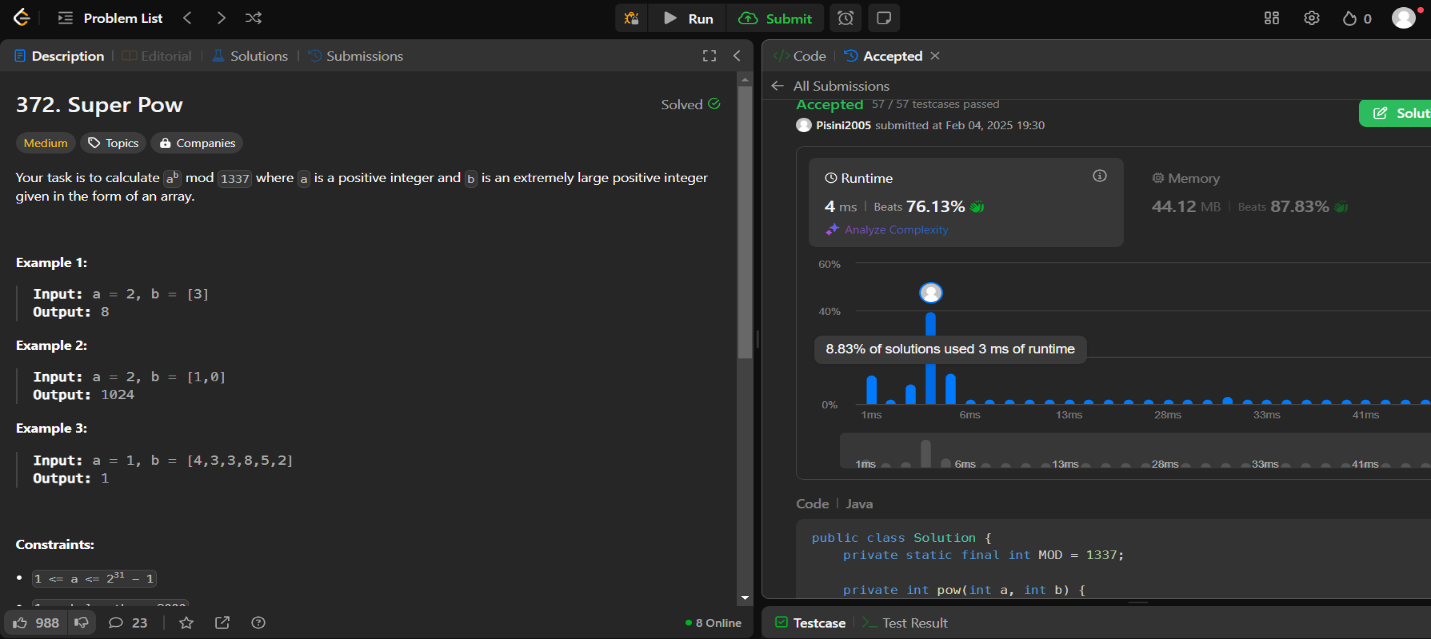
res = (res \* pow(a, b[i])) % MOD;

a = pow(a, 10);

}

return res; }}

1. **Output:**



1. **Problem 7:** The Skyline Problem
2. **Implementation/Code:**

import java.util.\*;

class Solution {

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

return divideAndConquer(buildings, 0, buildings.length - 1);

}

private List<List<Integer>> divideAndConquer(int[][] buildings, int left, int right) {

if (left > right) return new ArrayList<>();

if (left == right) {

List<List<Integer>> result = new ArrayList<>();

result.add(Arrays.asList(buildings[left][0], buildings[left][2]));

result.add(Arrays.asList(buildings[left][1], 0));

return result;

}

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

List<List<Integer>> leftSkyline = divideAndConquer(buildings, left, mid);

List<List<Integer>> rightSkyline = divideAndConquer(buildings, mid + 1, right);

return mergeSkylines(leftSkyline, rightSkyline);

}

private List<List<Integer>> mergeSkylines(List<List<Integer>> left, List<List<Integer>> right) {

List<List<Integer>> result = new ArrayList<>();

int h1 = 0, h2 = 0, i = 0, j = 0;

while (i < left.size() && j < right.size()) {

List<Integer> point1 = left.get(i);

List<Integer> point2 = right.get(j);

int x;

if (point1.get(0) < point2.get(0)) {

x = point1.get(0);

h1 = point1.get(1);

i++;

} else if (point1.get(0) > point2.get(0)) {

x = point2.get(0);

h2 = point2.get(1);

j++;

} else {

x = point1.get(0);

h1 = point1.get(1);

h2 = point2.get(1);

i++;

j++; }

int maxHeight = Math.max(h1, h2);

if (result.isEmpty() || result.get(result.size() - 1).get(1) != maxHeight) {

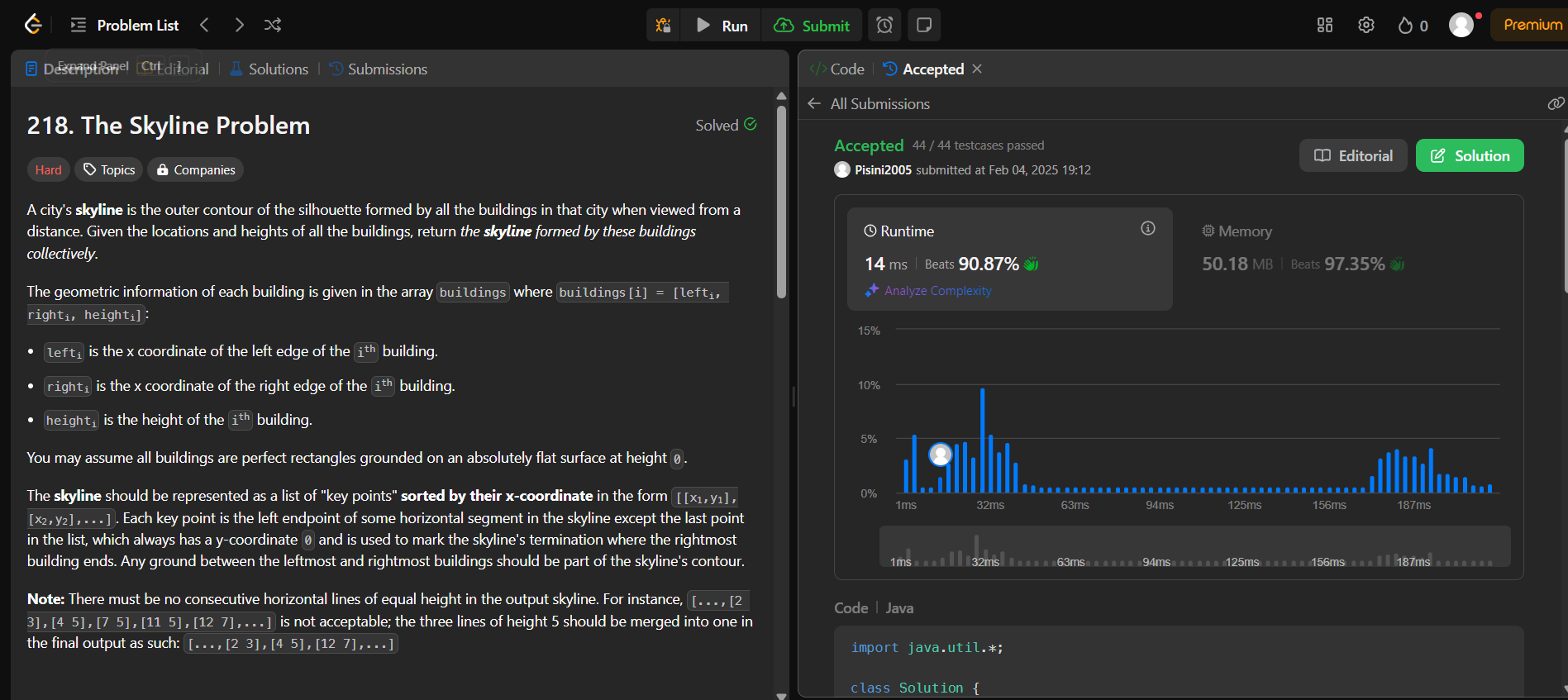
result.add(Arrays.asList(x, maxHeight)); } }

while (i < left.size()) result.add(left.get(i++));

while (j < right.size()) result.add(right.get(j++));

return result; }}

1. **Output:**

****