**Assignment-6**

**Aviraj  
22BCS17216**

**FL\_IOT-604/A**

**Q .Convert Sorted Array to Binary Search Tree**

Code:

public class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode() {}

TreeNode(int val) { this.val = val; }

TreeNode(int val, TreeNode left, TreeNode right) {

this.val = val;

this.left = left;

this.right = right;

}

}

class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

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

}

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

if (left > right) return null;

int mid = (left + right) / 2;

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

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

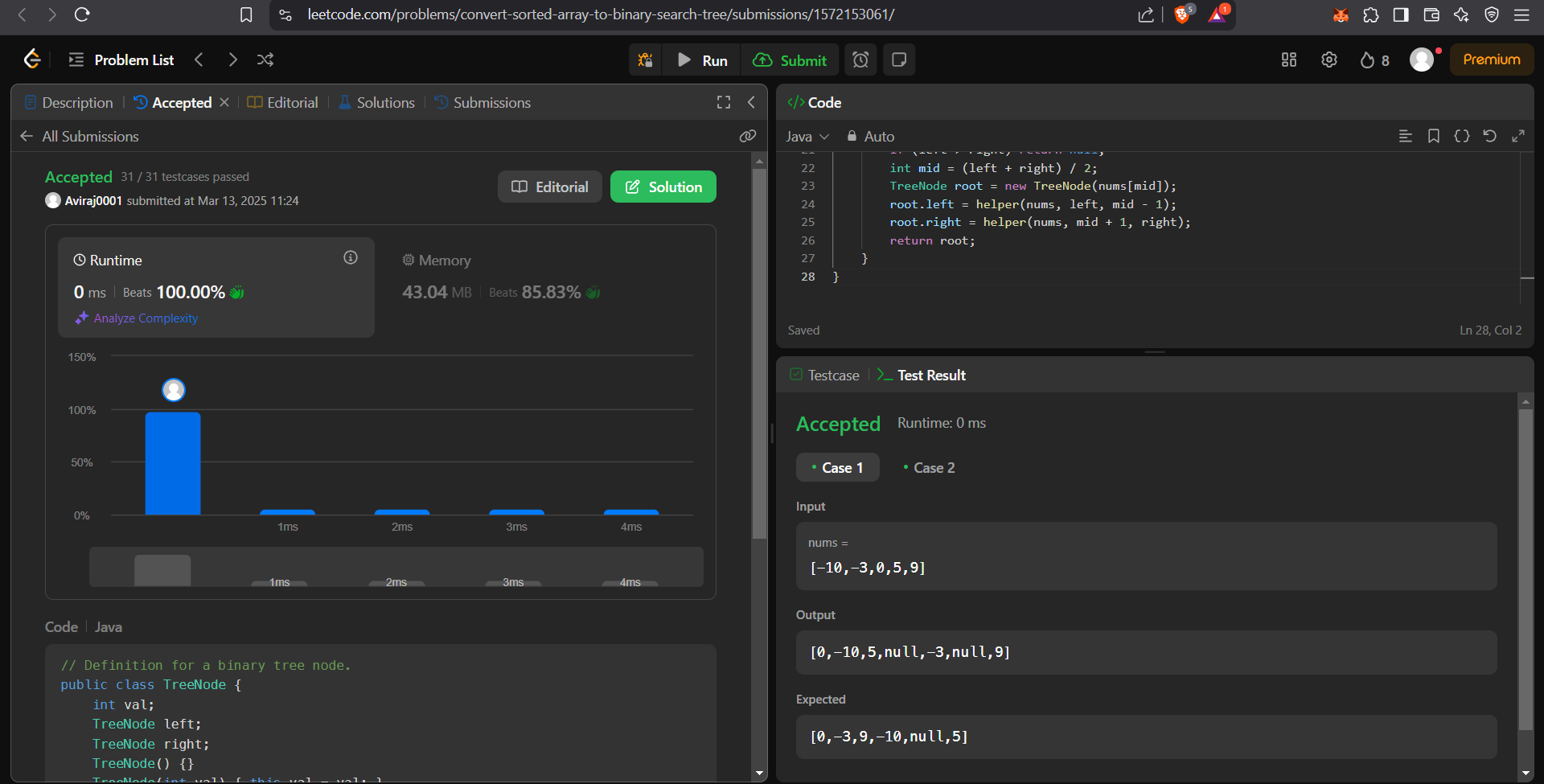
root.right = helper(nums, mid + 1, right);

return root;

}

}

**Output:**



**Q. Maximum Subarray.**

Code:

class Solution {

public int maxSubArray(int[] nums) {

int maxSum = nums[0];

int currentSum = nums[0];

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

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

maxSum = Math.max(maxSum, currentSum);

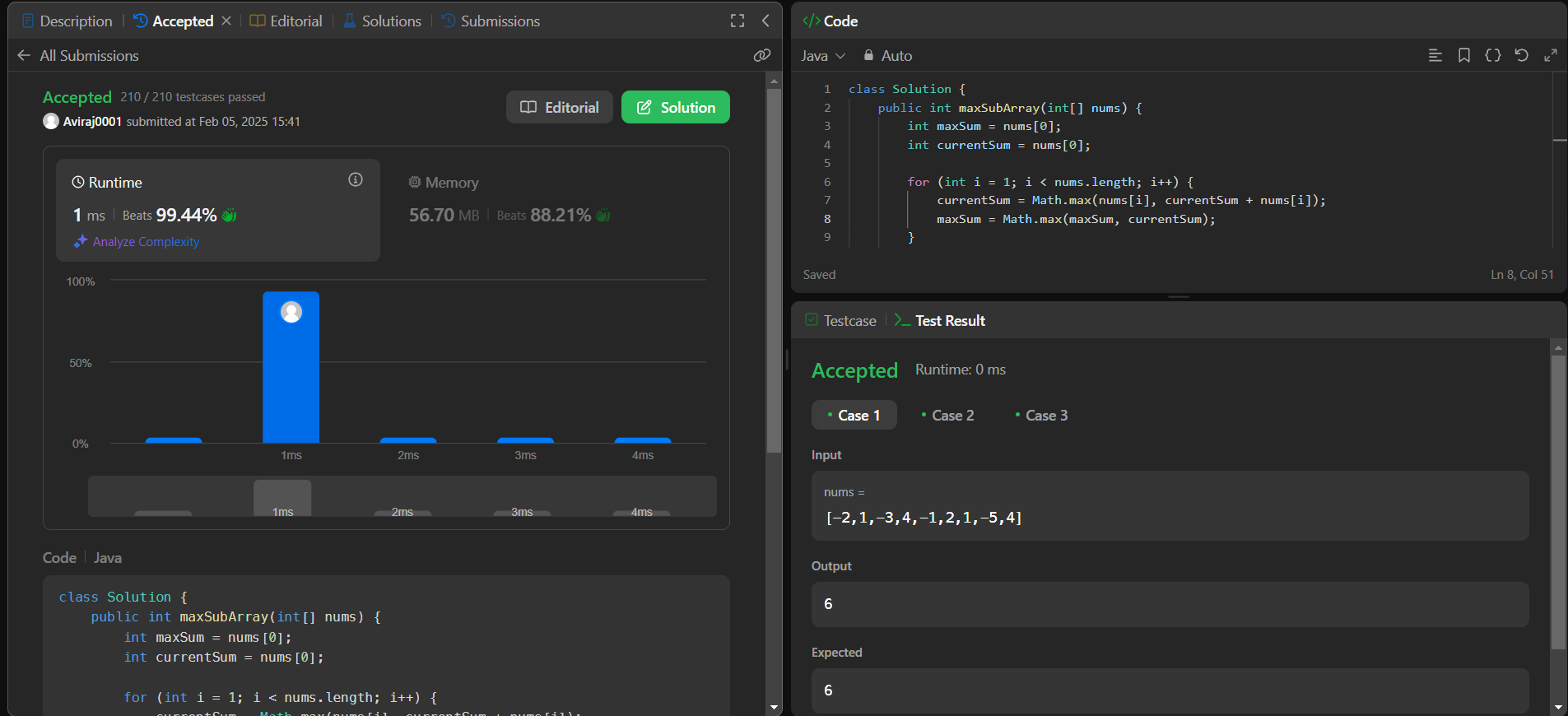
}

return maxSum;

}

}

**Output:**

****

**Q. Number of 1 bits.**

**Code:**

class Solution {

public int hammingWeight(int n) {

int count = 0;

while (n != 0) {

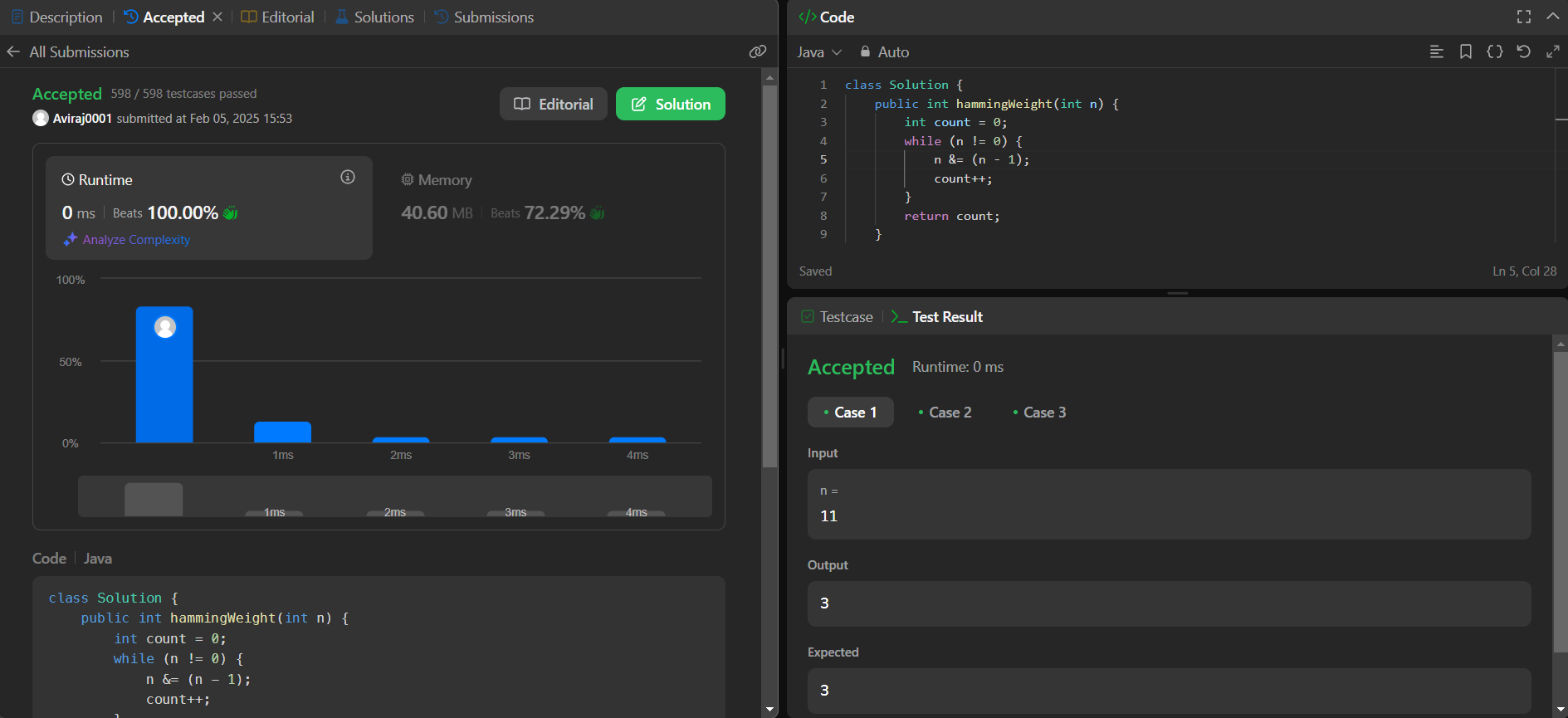
n &= (n - 1);

count++; }

return count;

}}

**Output:**

****

**Q Sort an Array.**

**Code:**

class Solution {

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

var map = new int[100001];

for (var n : nums)

map[n + 50000]++;

var k = 0;

for (var i = 0; i < 100001; i++)

for (var j = 0; j < map[i]; j++)

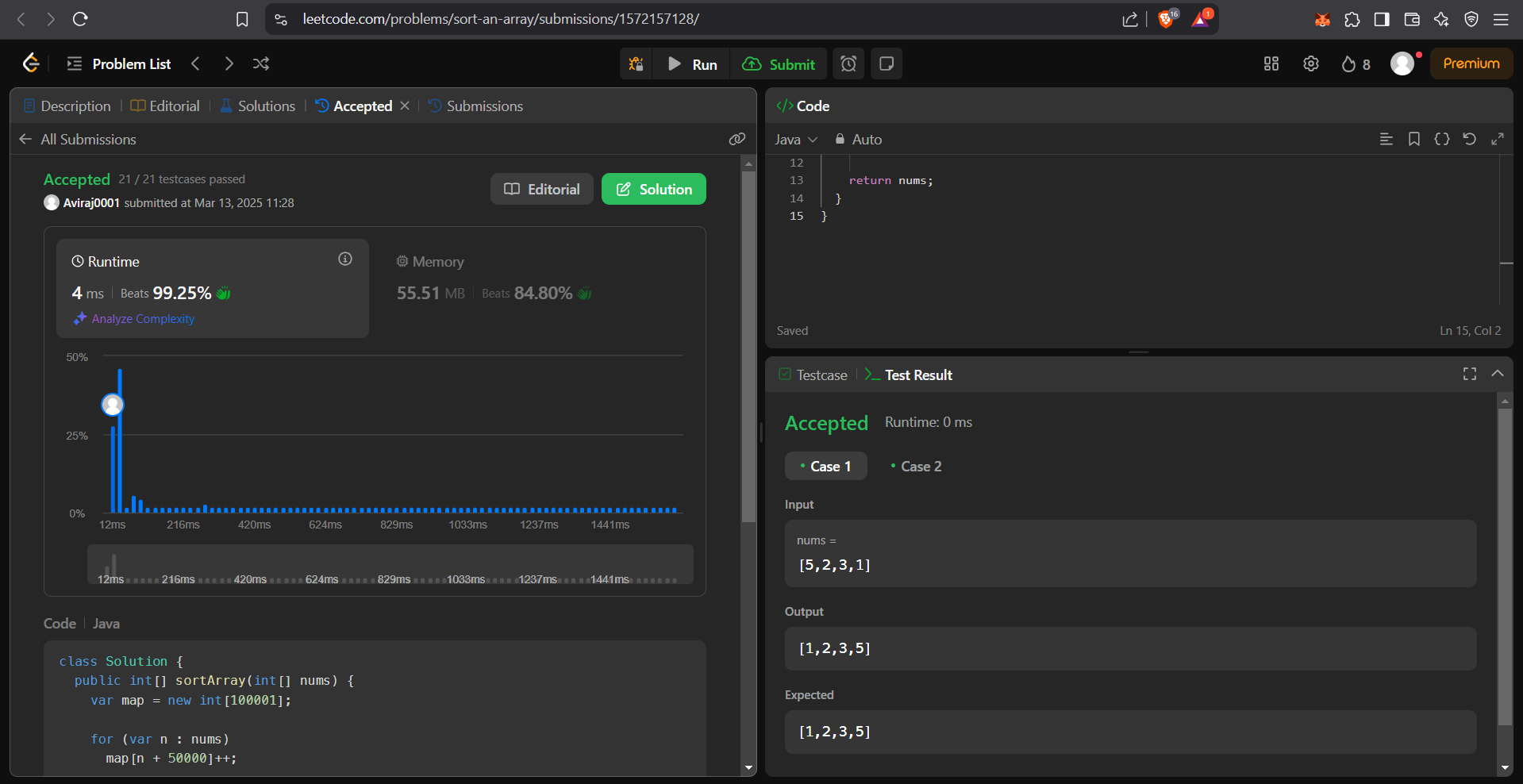
nums[k++] = i - 50000;

return nums;

}

}

**Output:**



**Q. Super Pow.**

**Code:**

class Solution {

private static final int MOD = 1337;

private int pow(int a, int b) {

int result = 1;

a %= MOD; // Taking mod to prevent overflow

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

result = (result \* a) % MOD;

}

return result;

}

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

int result = 1;

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

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

a = pow(a, 10); // Power up for the next iteration

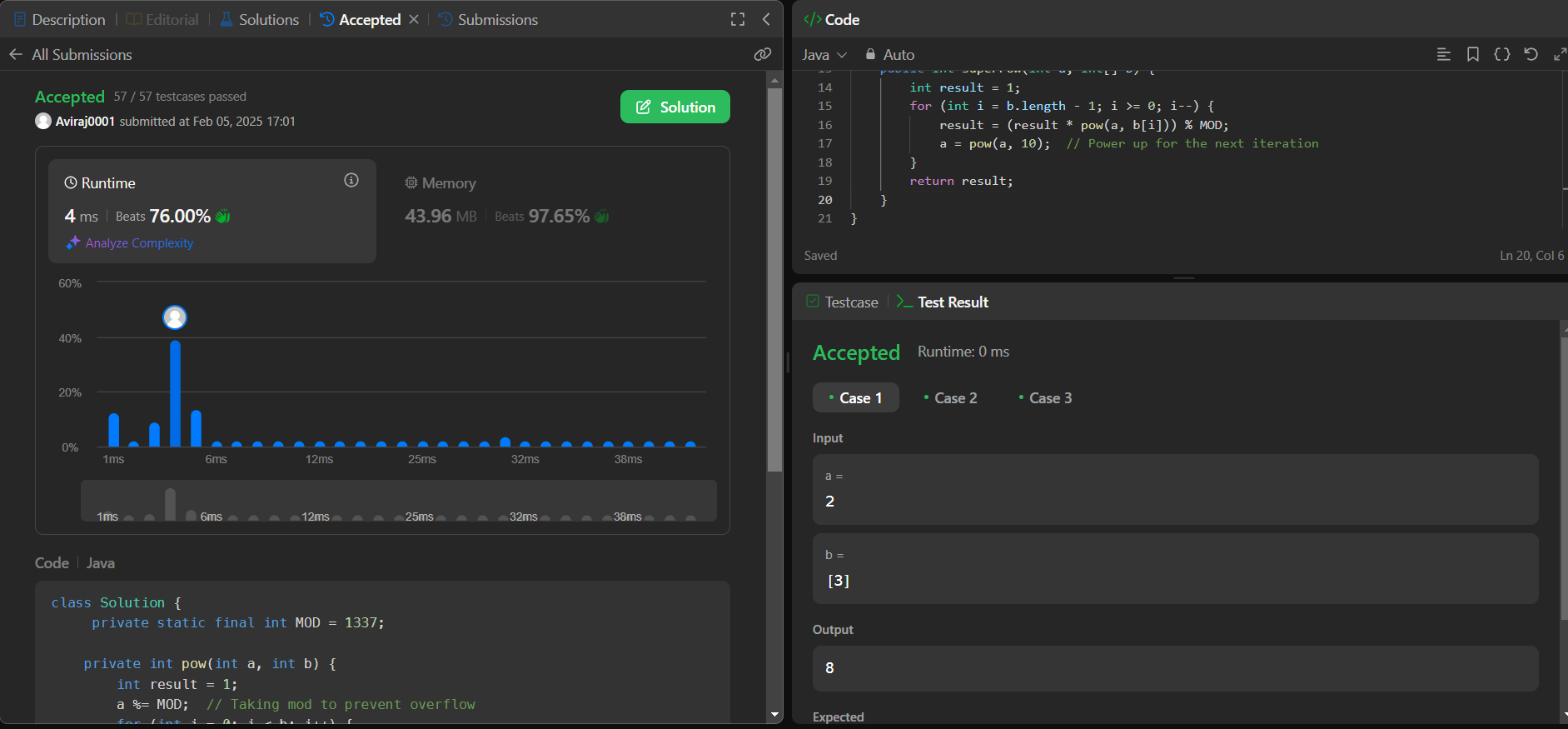
}

return result;

}

}

**Output:**

****

**Q. Beautiful Array.**

**Code:**

class Solution {

public int[] beautifulArray(int N) {

int[] res = new int[N];

if (N == 1)

{

return new int[] {1};

}

else if (N == 2)

{

return new int[] {1, 2};

}

else

{

int[] odds = beautifulArray((N + 1) / 2);

int[] even = beautifulArray(N / 2);

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

{

res[i] = odds[i] \* 2 - 1;

}

for (int j = 0; j < even.length; j ++)

{

res[odds.length + j] = even[j] \* 2;

}

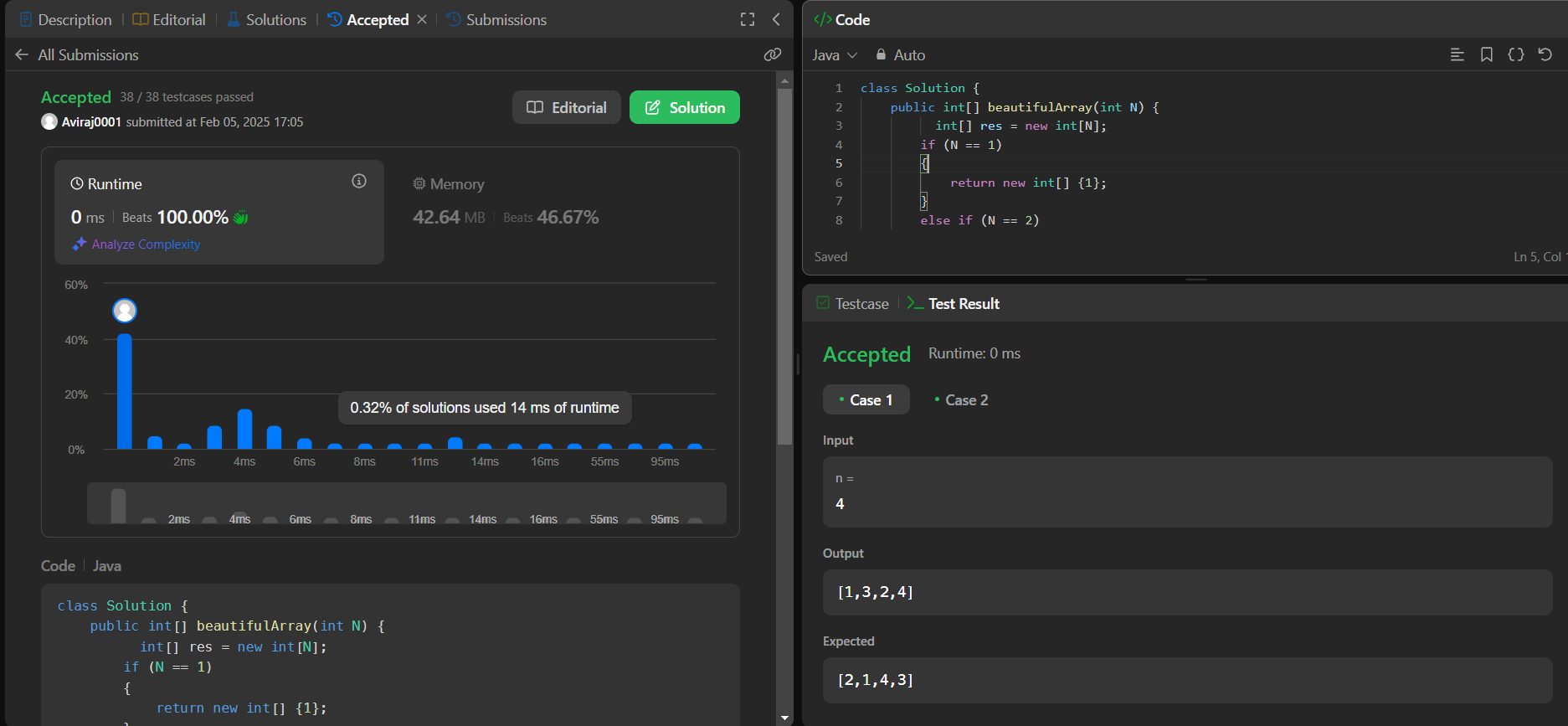
}

return res;

}

}

**Output:**

****

**Q. The Skyline Problem.**

**Code:**

class Solution {

class KeyPoint {

public int key;

public int height;

public KeyPoint next = null;

public KeyPoint(int key, int height) {

this.key = key;

this.height = height;

}

}

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

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

KeyPoint dummy = new KeyPoint(-1, 0); // dummy head

KeyPoint pre = dummy;

for (int[] bd : buildings) {

int L = bd[0];

int R = bd[1];

int H = bd[2];

while (pre.next != null && pre.next.key <= L)

pre = pre.next;

int preH = pre.height;

if (pre.key == L)

pre.height = Math.max(pre.height, H);

else if (pre.height < H) {

KeyPoint next = pre.next;

pre.next = new KeyPoint(L, H);

pre = pre.next;

pre.next = next;

}

KeyPoint preIter = pre;

KeyPoint curIter = pre.next;

while (curIter != null && curIter.key < R) {

preH = curIter.height;

curIter.height = Math.max(curIter.height, H);

if (curIter.height == preIter.height)

preIter.next = curIter.next;

else

preIter = curIter;

curIter = curIter.next;

}

if (preIter.height != preH && preIter.key != R && (curIter == null || curIter.key != R)) {

KeyPoint next = preIter.next;

preIter.next = new KeyPoint(R, preH);

preIter.next.next = next;

}

}

KeyPoint first = dummy;

KeyPoint second = dummy.next;

while (second != null) {

if (second.height != first.height)

res.add(Arrays.asList(second.key, second.height));

first = first.next;

second = second.next;

}

return res;

}

}

**Output:**

