**DSA PRACTICE – 9 – 21/11/24**

**1.**[**VALID PALINDROME**](https://leetcode.com/problems/valid-palindrome/)

class Solution {

    public boolean isPalindrome(String s) {

        int l = 0;

        int r = s.length() - 1; // right

        while (l < r) {

            while (l < r && !Character.isLetterOrDigit(s.charAt(l))) l++;

            while (l < r && !Character.isLetterOrDigit(s.charAt(r))) r--;

            if (Character.toLowerCase(s.charAt(l)) != Character.toLowerCase(s.charAt(r))) return false;

            l++;

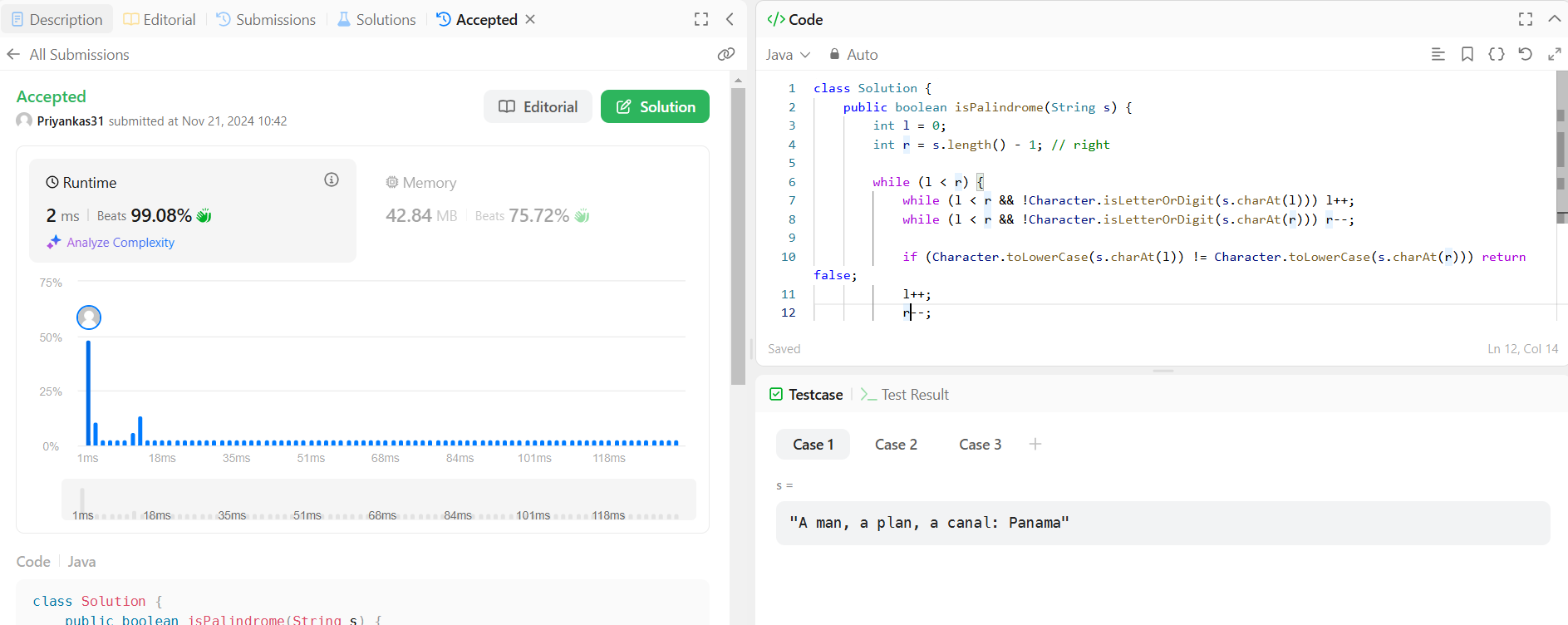
            r--;

        }

        return true;

    }

}



**2.IS SUBSEQUENCE**  
class Solution {

  public boolean isSubsequence(String s, String t) {

    if (s.isEmpty())

      return true;

    int i = 0;

    for (final char c : t.toCharArray())

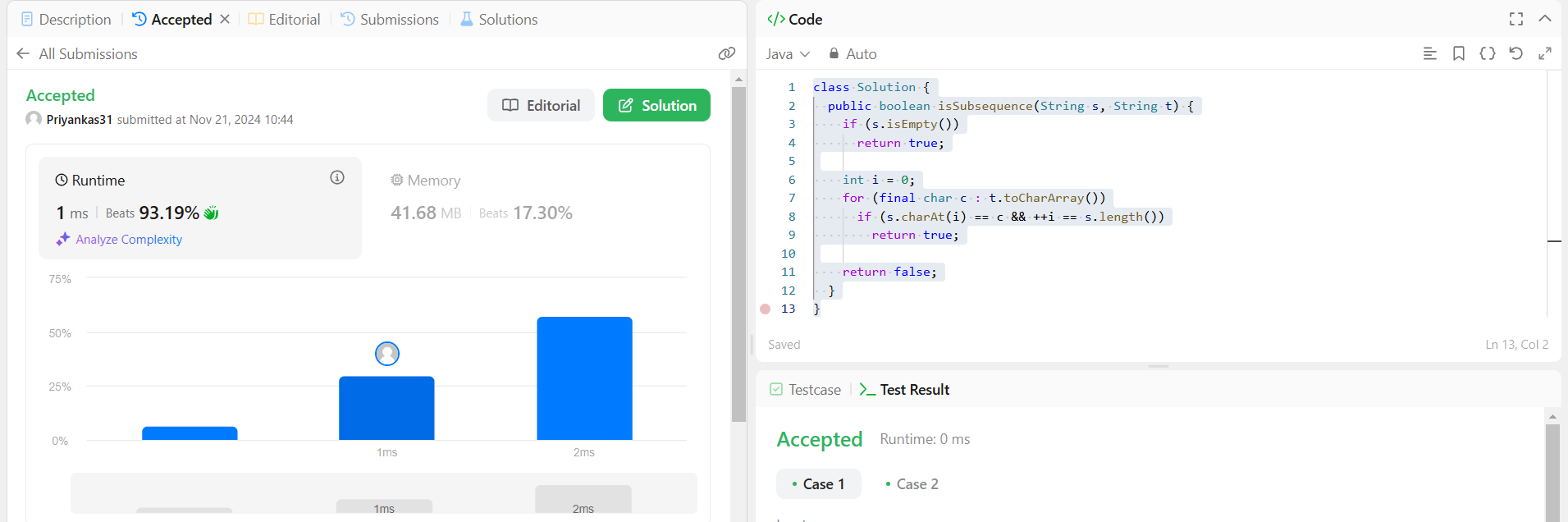
      if (s.charAt(i) == c && ++i == s.length())

        return true;

    return false;

  }

}



**3.TWO SUM II**

class Solution {

    public int[] twoSum(int[] numbers, int target) {

        int left = 0;

        int right = numbers.length - 1;

        while (left < right) {

            int total = numbers[left] + numbers[right];

            if (total == target) {

                return new int[]{left + 1, right + 1};

            } else if (total > target) {

                right--;

            } else {

                left++;

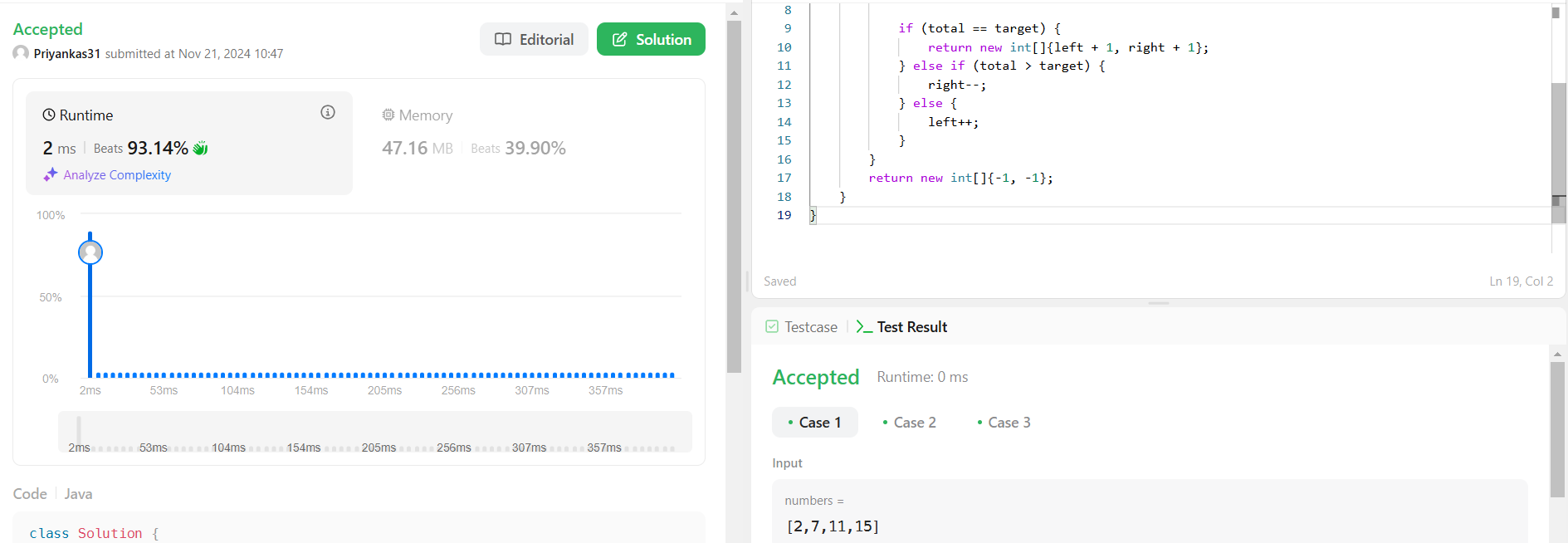
            }

        }

        return new int[]{-1, -1};

    }

}



**4. CONTAINER WITH MOST WATER**

class Solution {

    public int maxArea(int[] height) {

        int maxArea = 0;

        int left = 0;

        int right = height.length - 1;

        while (left < right) {

            maxArea = Math.max(maxArea, (right - left) \* Math.min(height[left], height[right]));

            if (height[left] < height[right]) {

                left++;

            } else {

                right--;

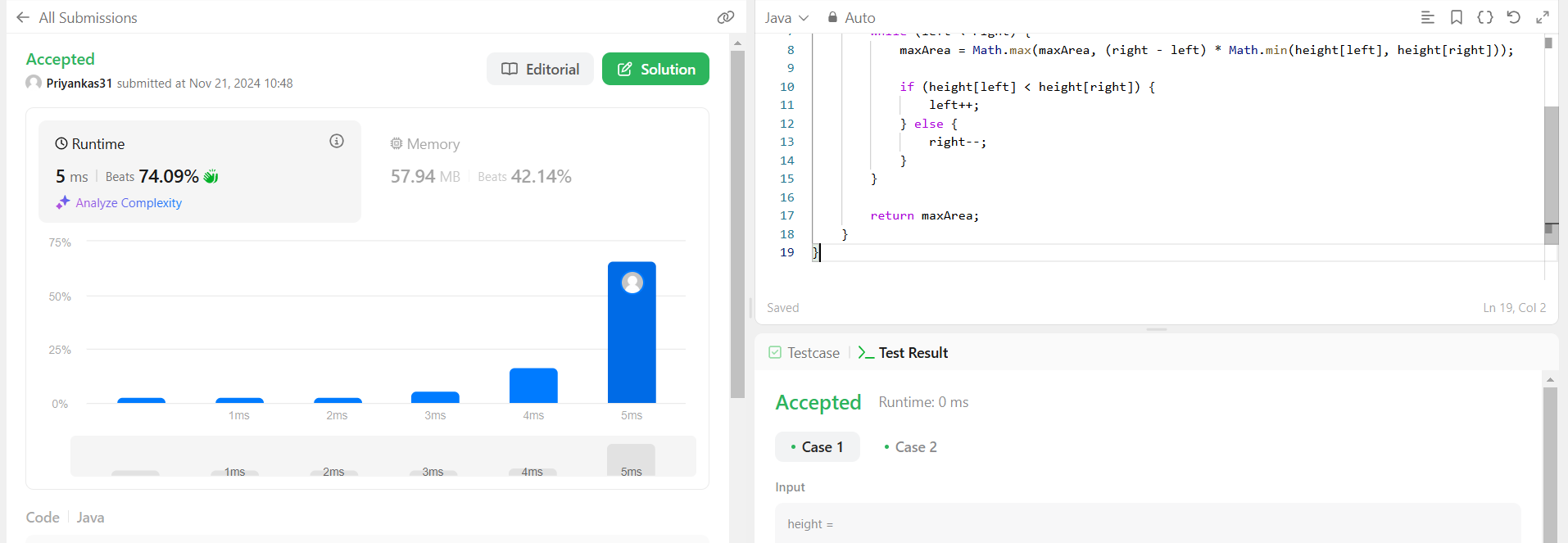
            }

        }

        return maxArea;

    }

}



**5. 3SUM**

class Solution {

    public List<List<Integer>> threeSum(int[] nums) {

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

        Arrays.sort(nums);

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

            if (i > 0 && nums[i] == nums[i-1]) {

                continue;

            }

            int j = i + 1;

            int k = nums.length - 1;

            while (j < k) {

                int total = nums[i] + nums[j] + nums[k];

                if (total > 0) {

                    k--;

                } else if (total < 0) {

                    j++;

                } else {

                    res.add(Arrays.asList(nums[i], nums[j], nums[k]));

                    j++;

                    while (nums[j] == nums[j-1] && j < k) {

                        j++;

                    }

                }

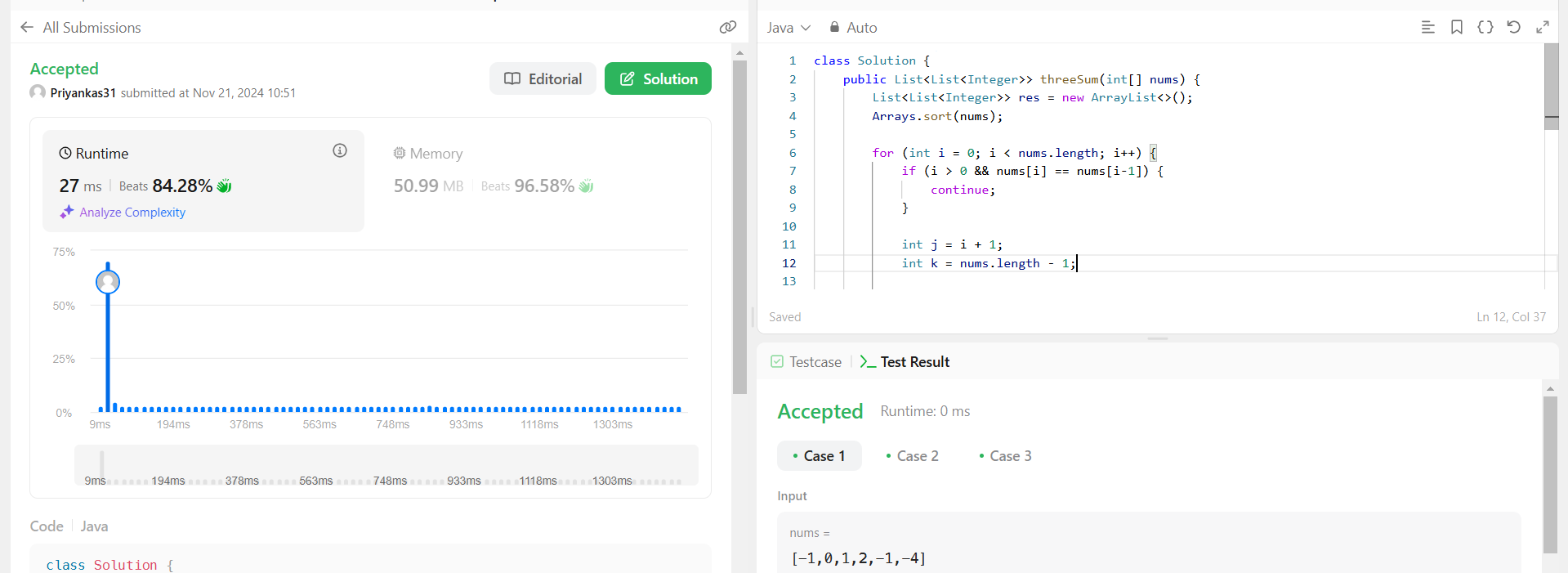
            }

        }

        return res;

    }

}



**6. MINIMUM SIZE SUBARRAY SUM**

class Solution {

    public int minSubArrayLen(int target, int[] nums) {

        int left=0,right=0,sum =0;

        int ans = Integer.MAX\_VALUE;

        for(right=0;right<nums.length;right++){

            sum +=nums[right];

            while(sum>=target){

                ans=Math.min(ans,right-left+1);

                sum -=nums[left++];

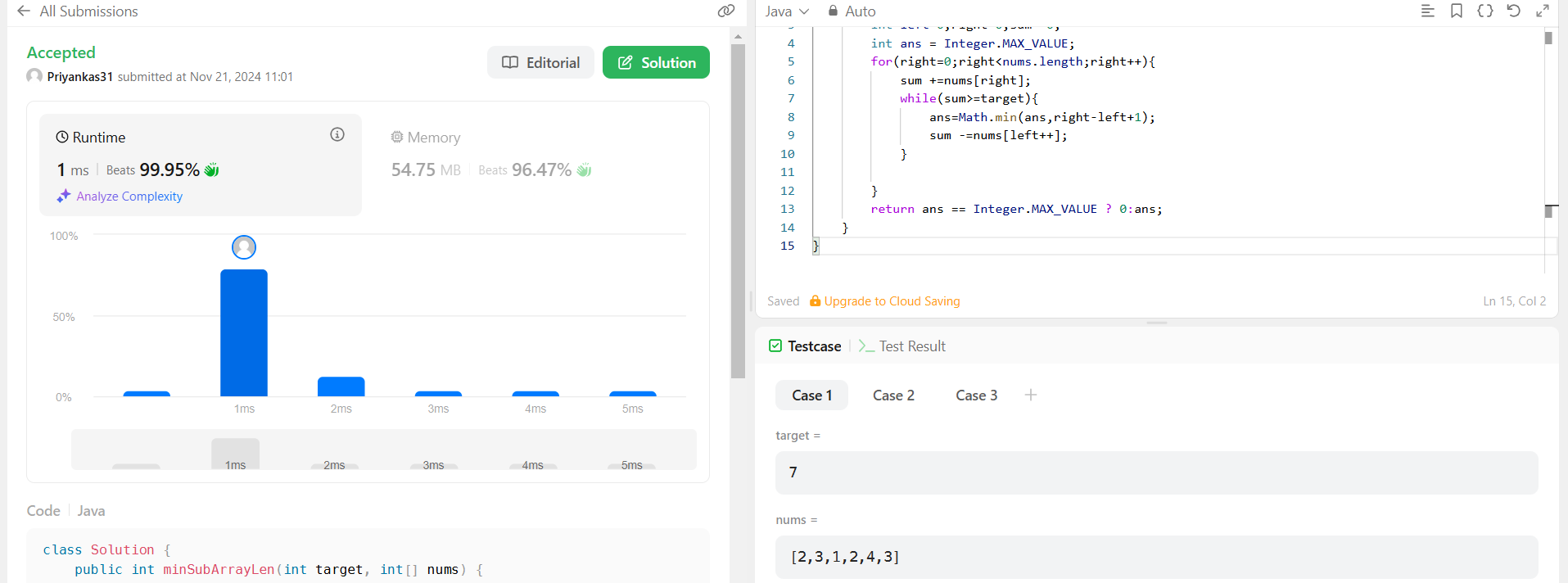
            }

        }

        return ans == Integer.MAX\_VALUE ? 0:ans;

    }

}



**7. LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS**class Solution {

    public int lengthOfLongestSubstring(String s) {

        int left = 0;

        int maxLength = 0;

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

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

            while (charSet.contains(s.charAt(right))) {

                charSet.remove(s.charAt(left));

                left++;

            }

            charSet.add(s.charAt(right));

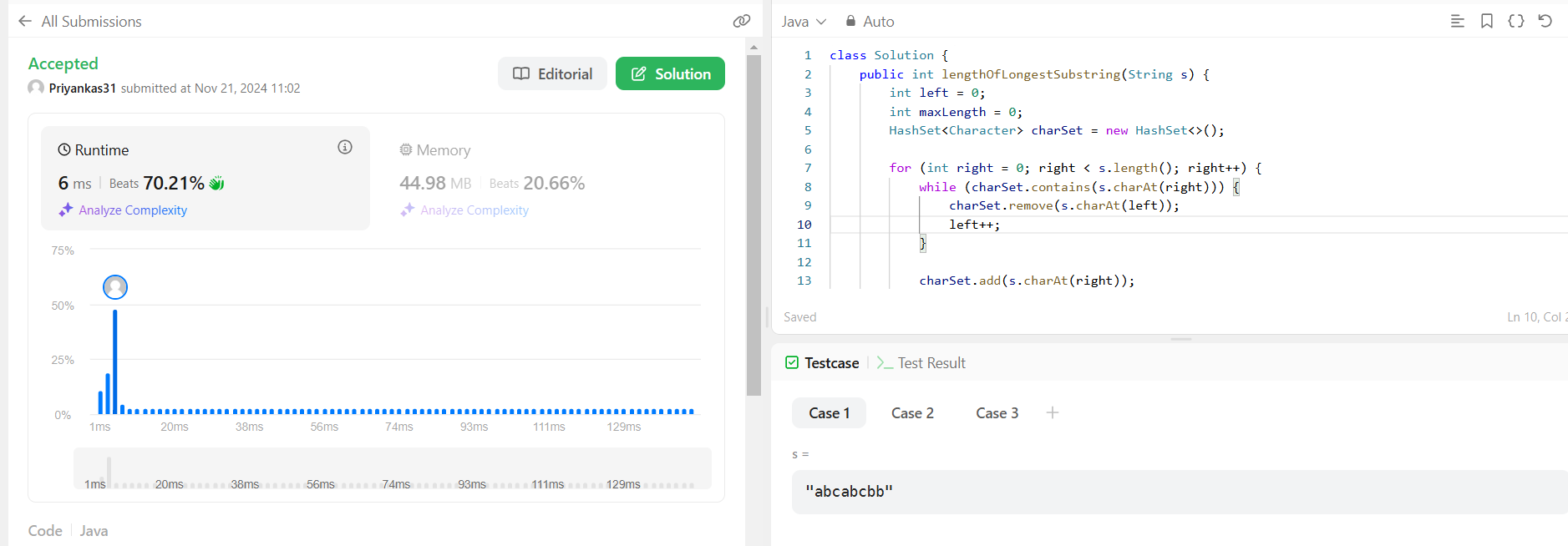
            maxLength = Math.max(maxLength, right - left + 1);

        }

        return maxLength;

    }

}



**8. VALID PARENTHESES**import java.util.Stack;

class Solution {

    public boolean isValid(String s) {

        Stack<Character> stack = new Stack<>();

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

            if (c == '(' || c == '{' || c == '[') {

                stack.push(c);

            } else {

                if (stack.isEmpty()) return false;

                char top = stack.pop();

                if ((c == ')' && top != '(') ||

                    (c == '}' && top != '{') ||

                    (c == ']' && top != '[')) {

                    return false;

                }

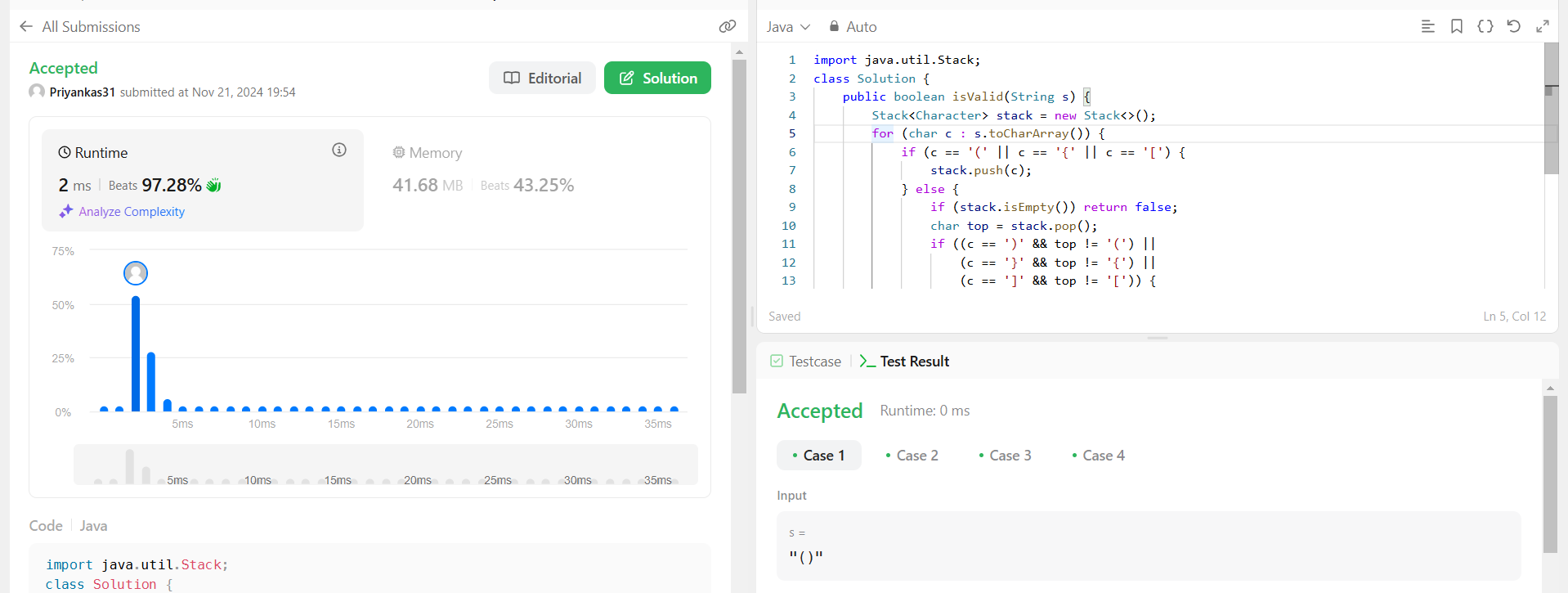
            }

        }

        return stack.isEmpty();

    }

}



9.Simplify Path  
class Solution {

    public String simplifyPath(String path) {

        Stack<String> stack = new Stack<>();

        String[] directories = path.split("/");

        for (String dir : directories) {

            if (dir.equals(".") || dir.isEmpty()) {

                continue;

            } else if (dir.equals("..")) {

                if (!stack.isEmpty()) {

                    stack.pop();

                }

            } else {

                stack.push(dir);

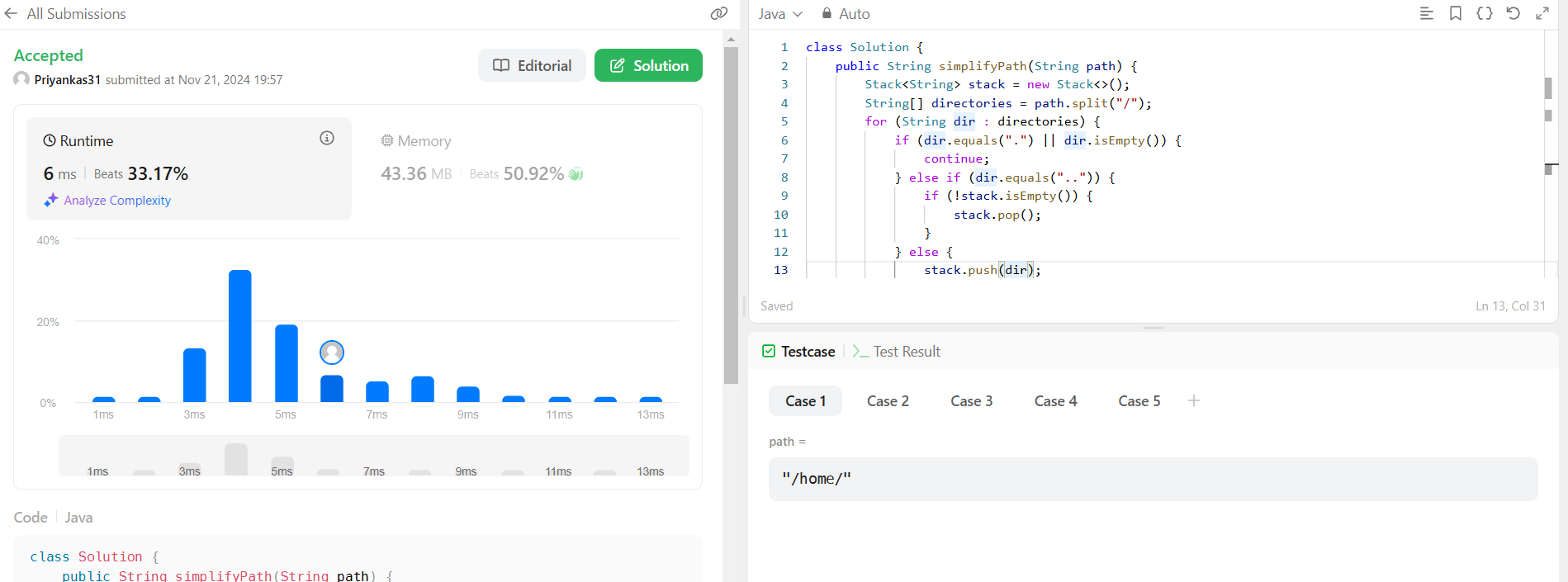
            }

        }

        return "/" + String.join("/", stack);

    }

}



10. Evaluate Reverse Polish Notation  
class Solution {

    public int evalRPN(String[] tokens) {

        Stack<Integer> stack = new Stack<>();

        for (String c : tokens) {

            if (c.equals("+")) {

                stack.push(stack.pop() + stack.pop());

            } else if (c.equals("-")) {

                int second = stack.pop();

                int first = stack.pop();

                stack.push(first - second);

            } else if (c.equals("\*")) {

                stack.push(stack.pop() \* stack.pop());

            } else if (c.equals("/")) {

                int second = stack.pop();

                int first = stack.pop();

                stack.push(first / second);

            } else {

                stack.push(Integer.parseInt(c));

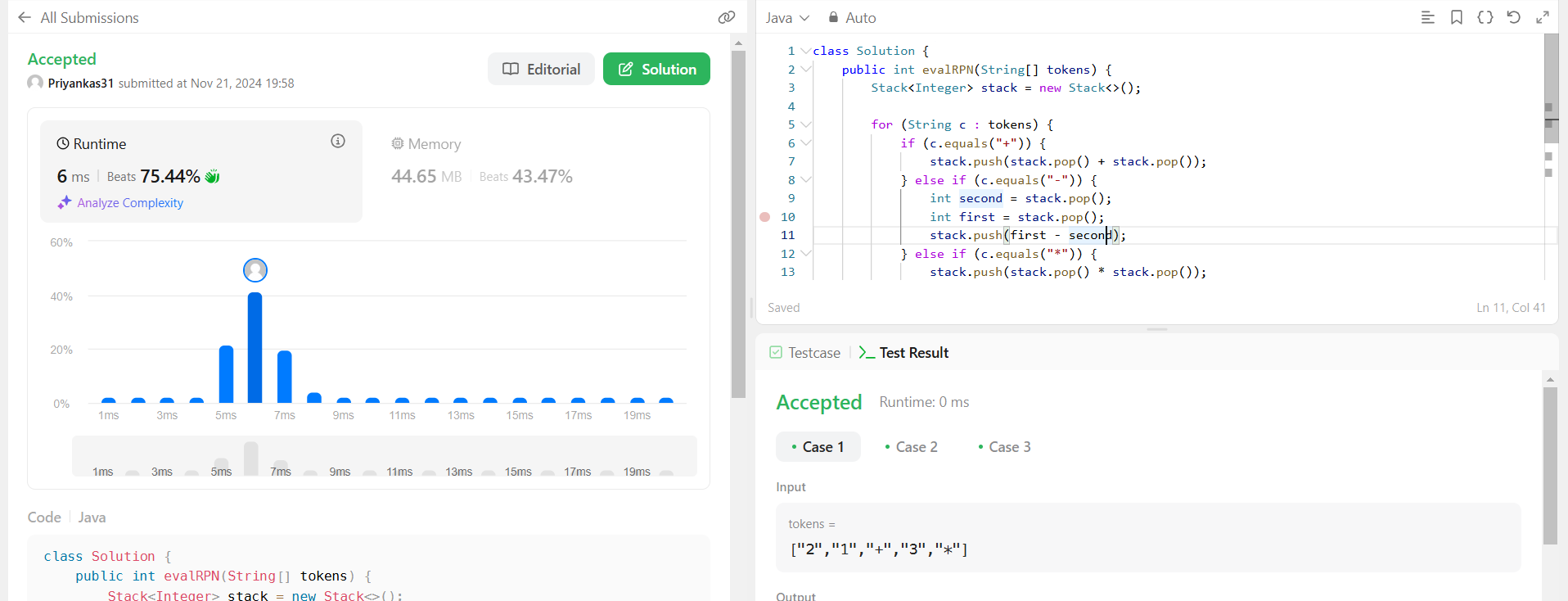
            }

        }

        return stack.peek();

    }

}



11. Search Insert Point  
class Solution {

    public int searchInsert(int[] nums, int target) {

        int left = 0;

        int right = nums.length - 1;

        while (left <= right) {

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

            if (nums[mid] == target) {

                return mid;

            } else if (nums[mid] > target) {

                right = mid - 1;

            } else {

                left = mid + 1;

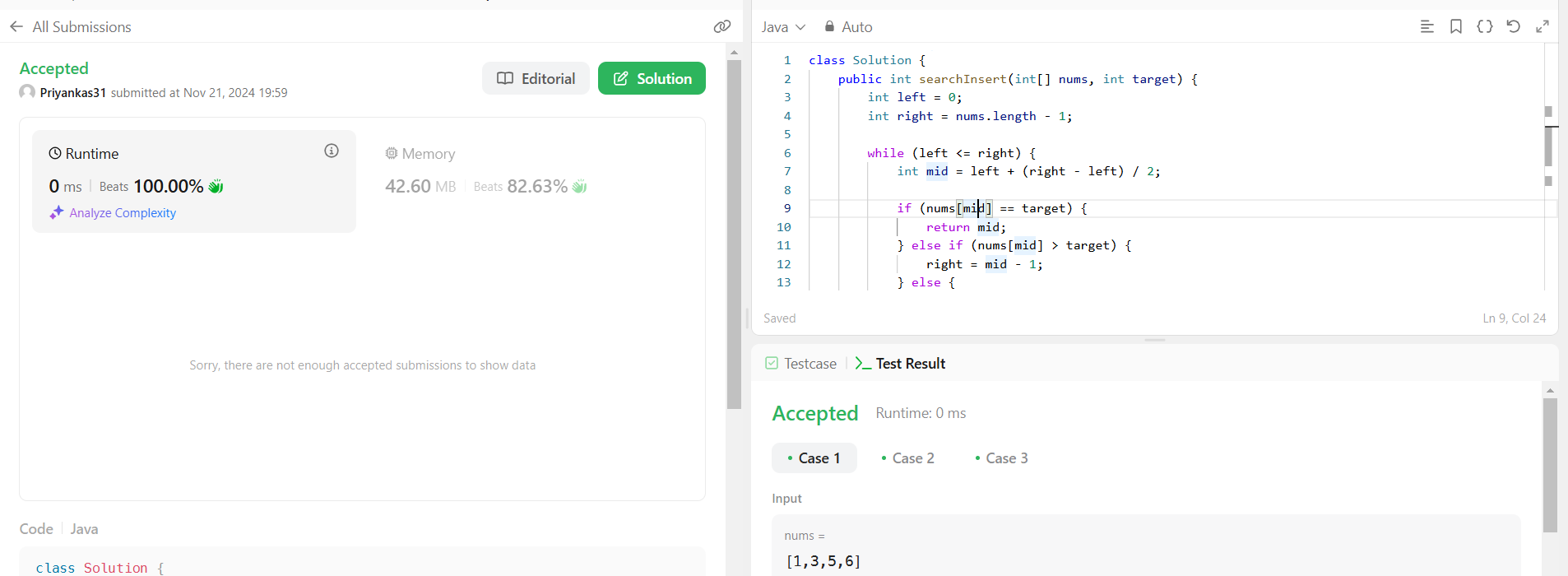
            }

        }

        return left;

    }

}



12. Search in 2D Matrix

class Solution {

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

        int m = matrix.length;

        int n = matrix[0].length;

        int i=0;

        int j=n-1;

        while(i<m && j>=0){

            if(matrix[i][j]==target) return true;

            if(matrix[i][j]>target){

                j--;

            }

            else{

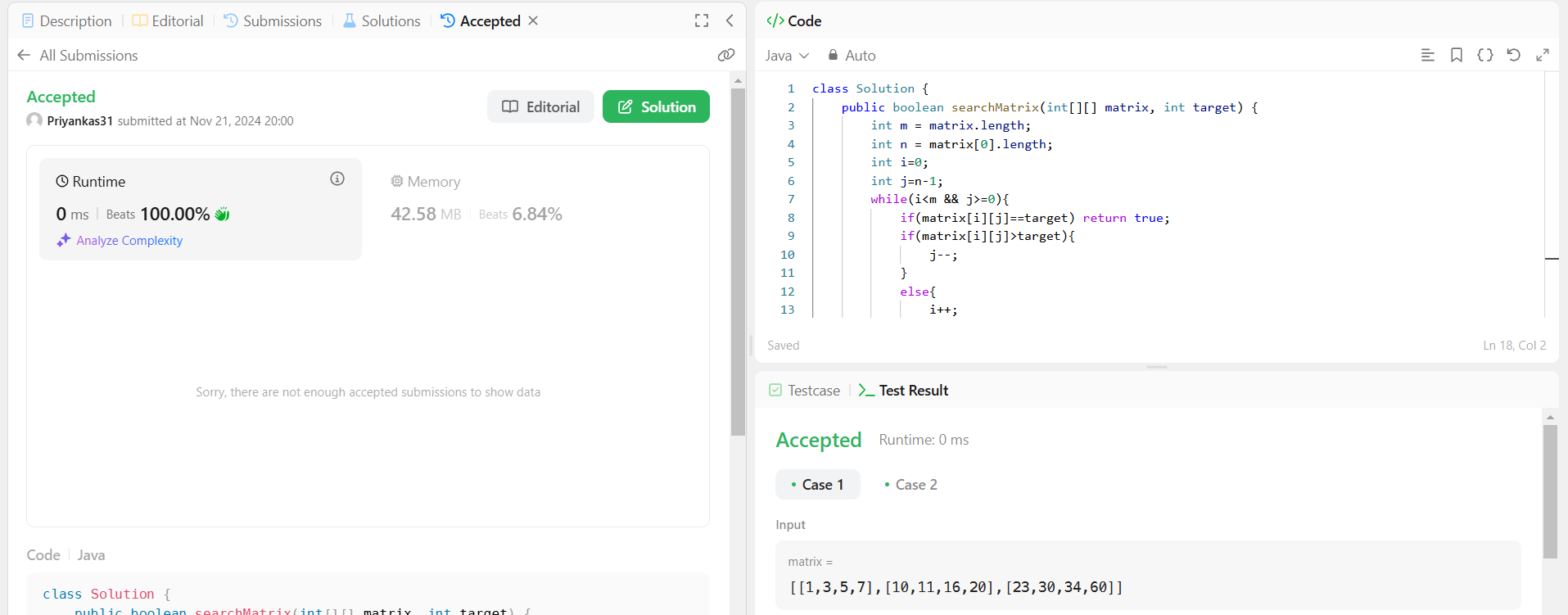
                i++;

            }

        }

        return false;

    }

}  


13. Find Peak Element  
class Solution {

    public int findPeakElement(int[] nums) {

        int left = 0;

        int right = nums.length - 1;

        while (left < right) {

            int mid = (left + right) / 2;

            if (nums[mid] > nums[mid + 1]) {

                right = mid;

            } else {

                left = mid + 1;

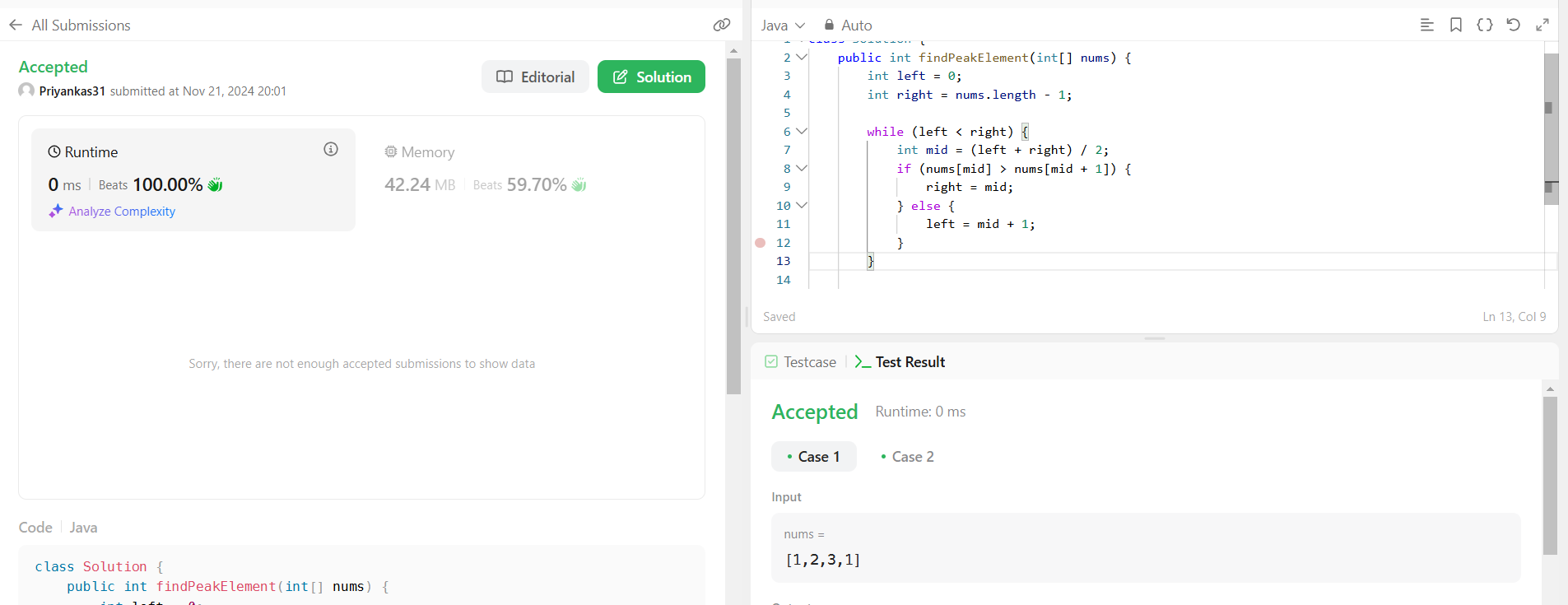
            }

        }

        return left;

    }

}



14. Find Minimum in Rotated Sorted Array  
class Solution {

    public int findMin(int[] nums) {

        int left = 0;

        int right = nums.length - 1;

        while (left < right) {

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

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

                right = mid;

            } else {

                left = mid + 1;

            }

        }

        return nums[left];

    }

}

