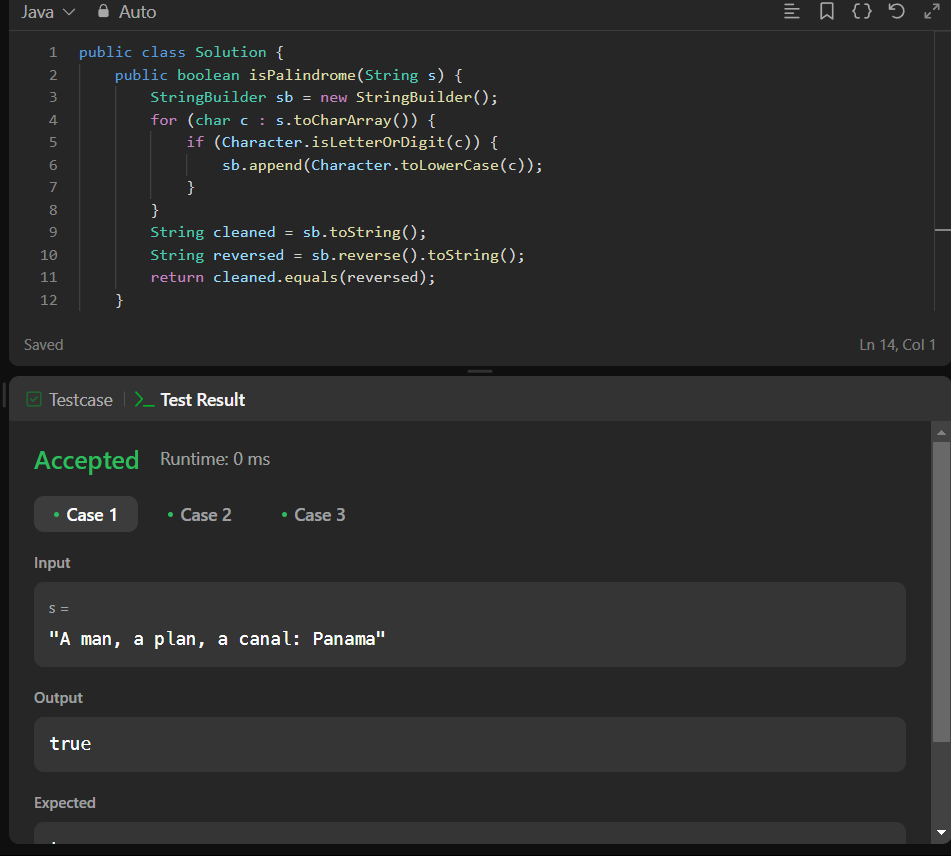
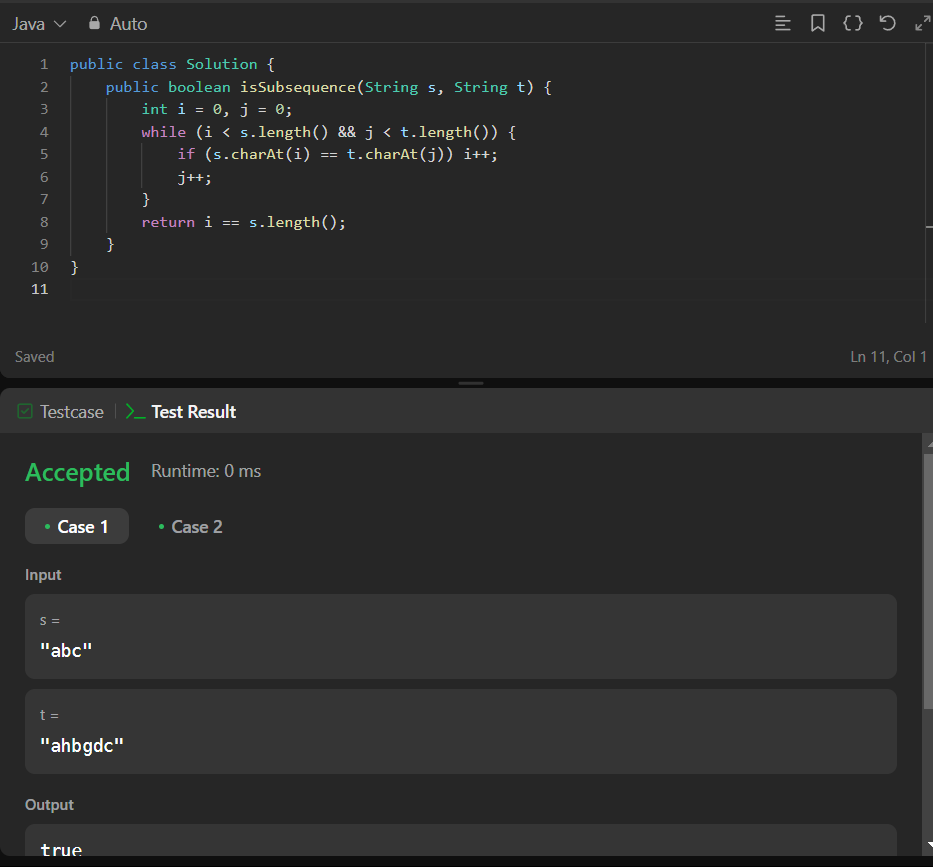
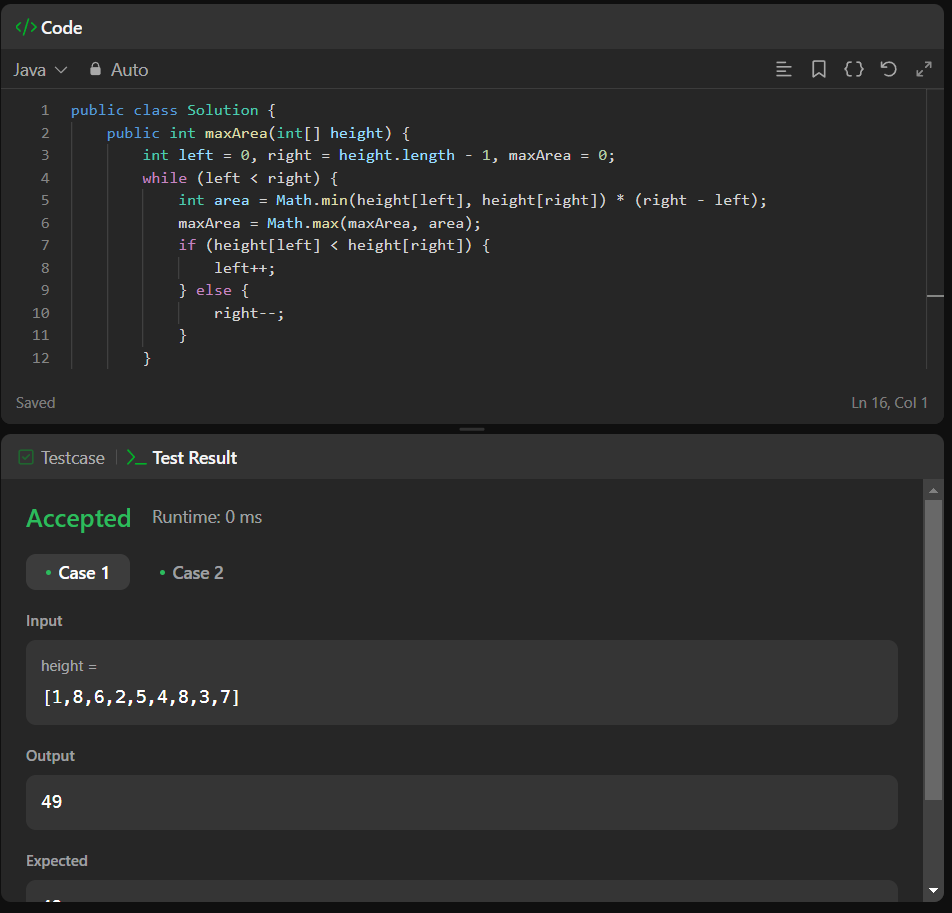
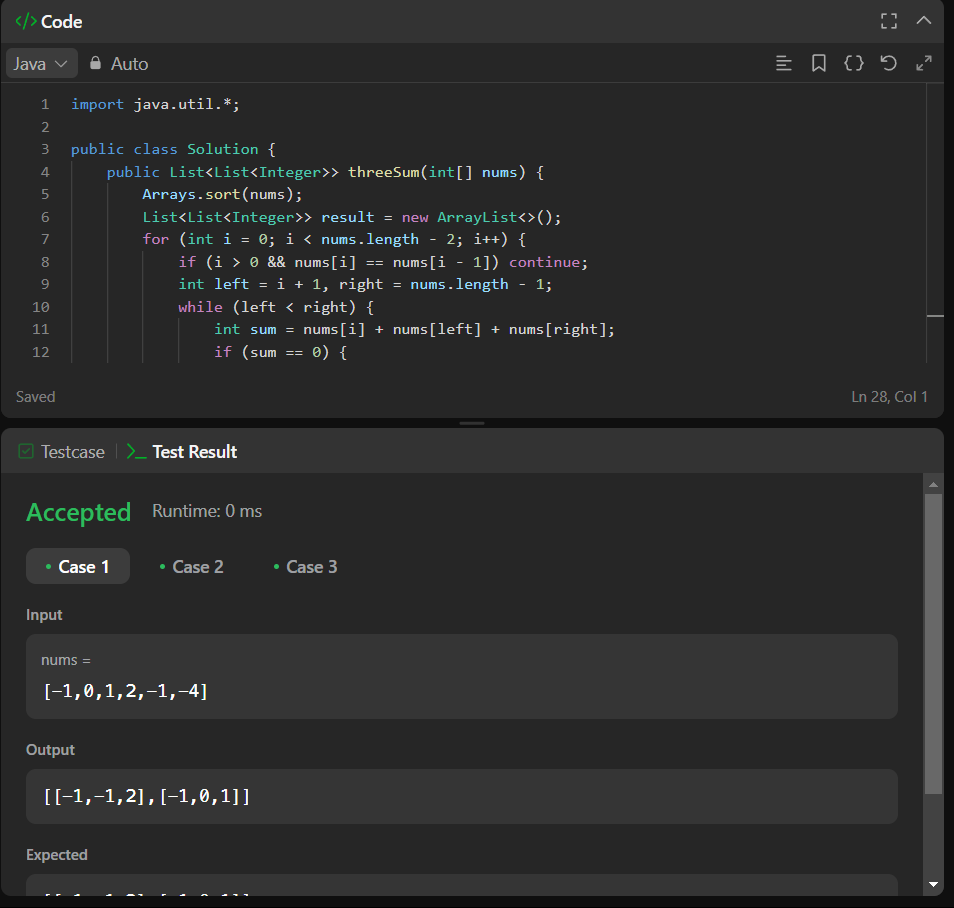
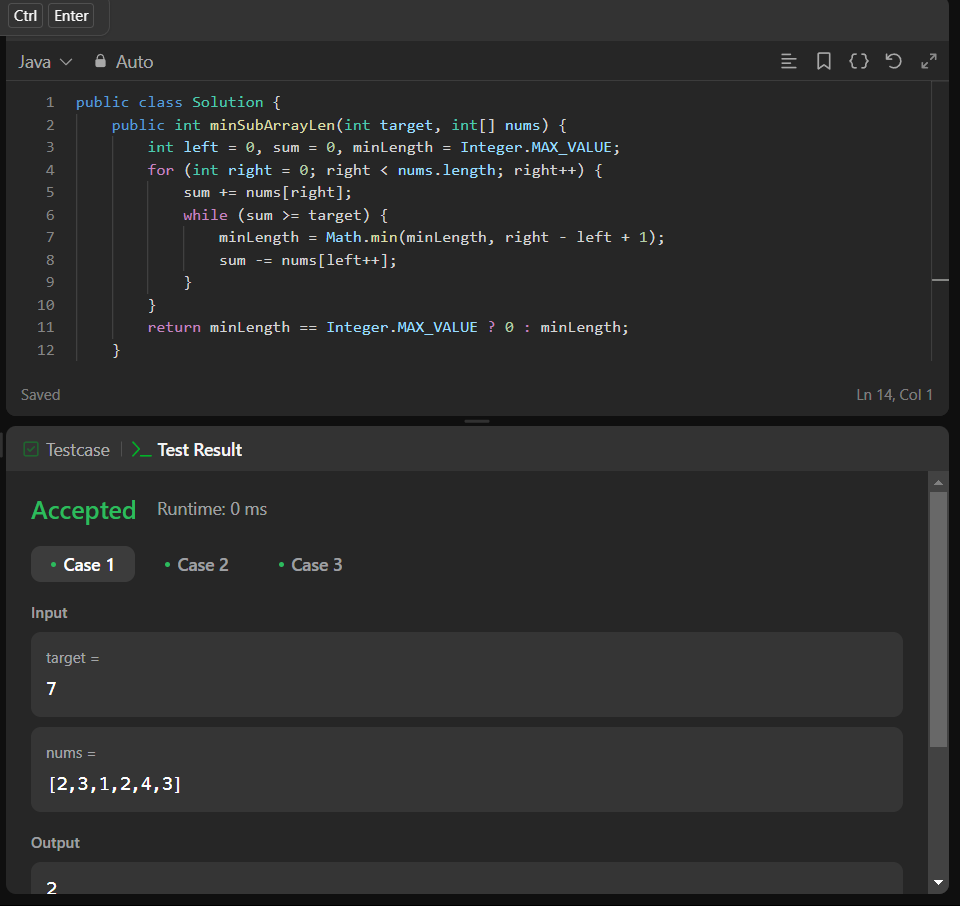
**DSA PROBLEMS**

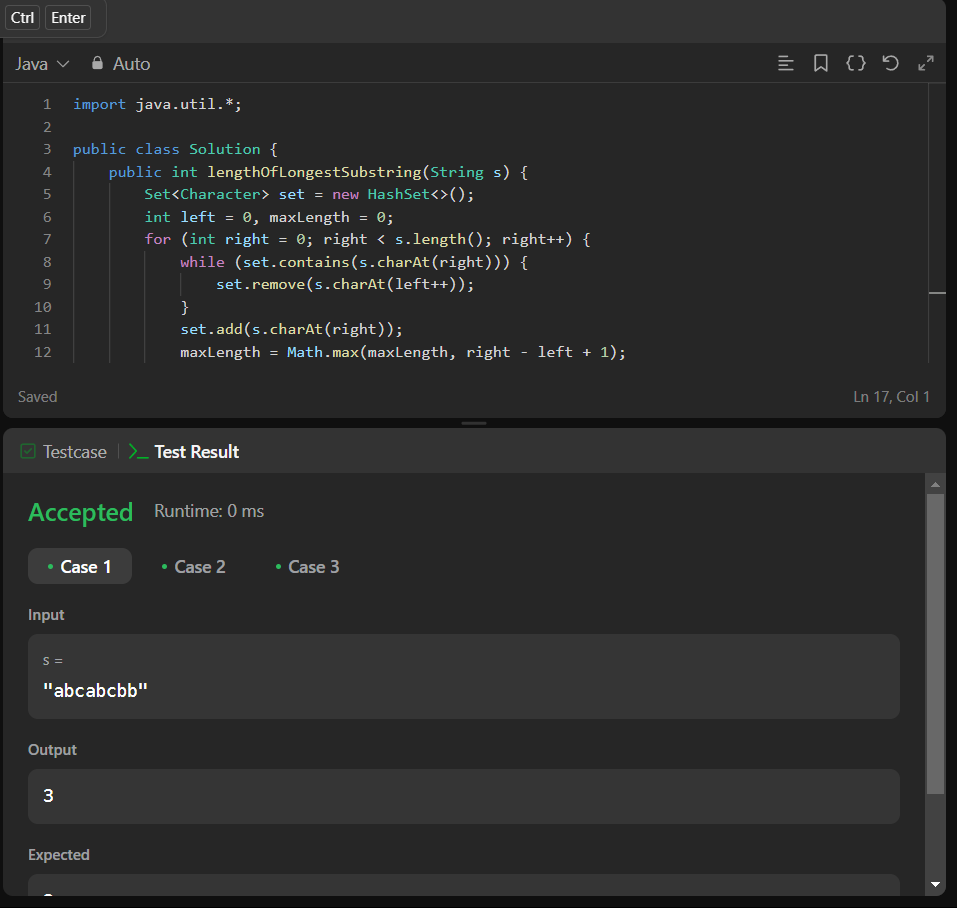
**Valid Palindrome**



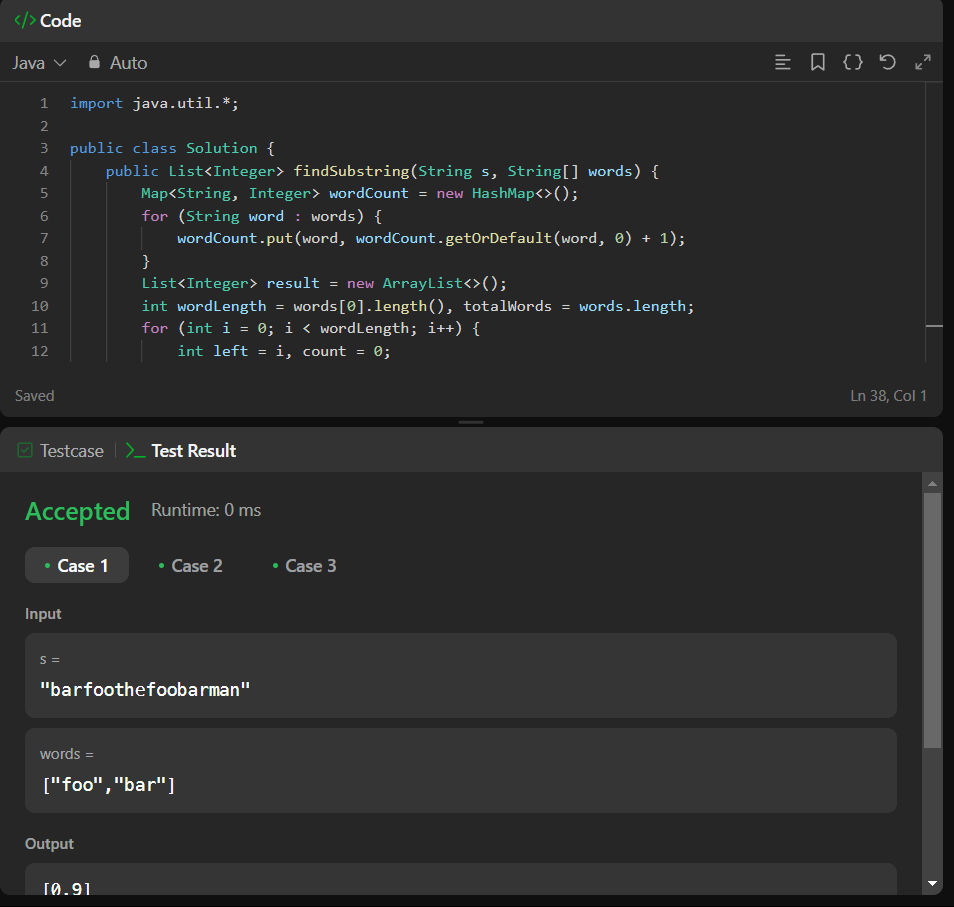
Is Subsequence  
  
  
Two Sum II - Input Array is Sorted  
  
  
Container with Most Water  
  
  
3Sum  
  


Minimum Size Subarray Sum

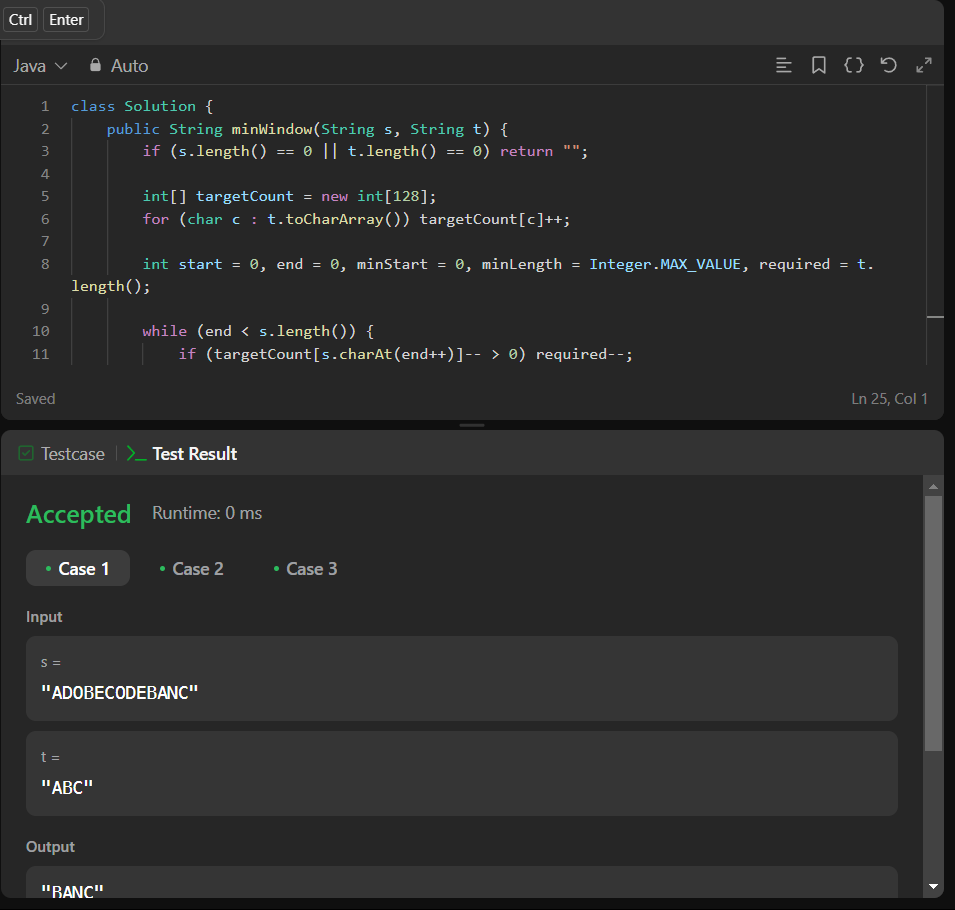


Longest Substring without Repeating Characters  


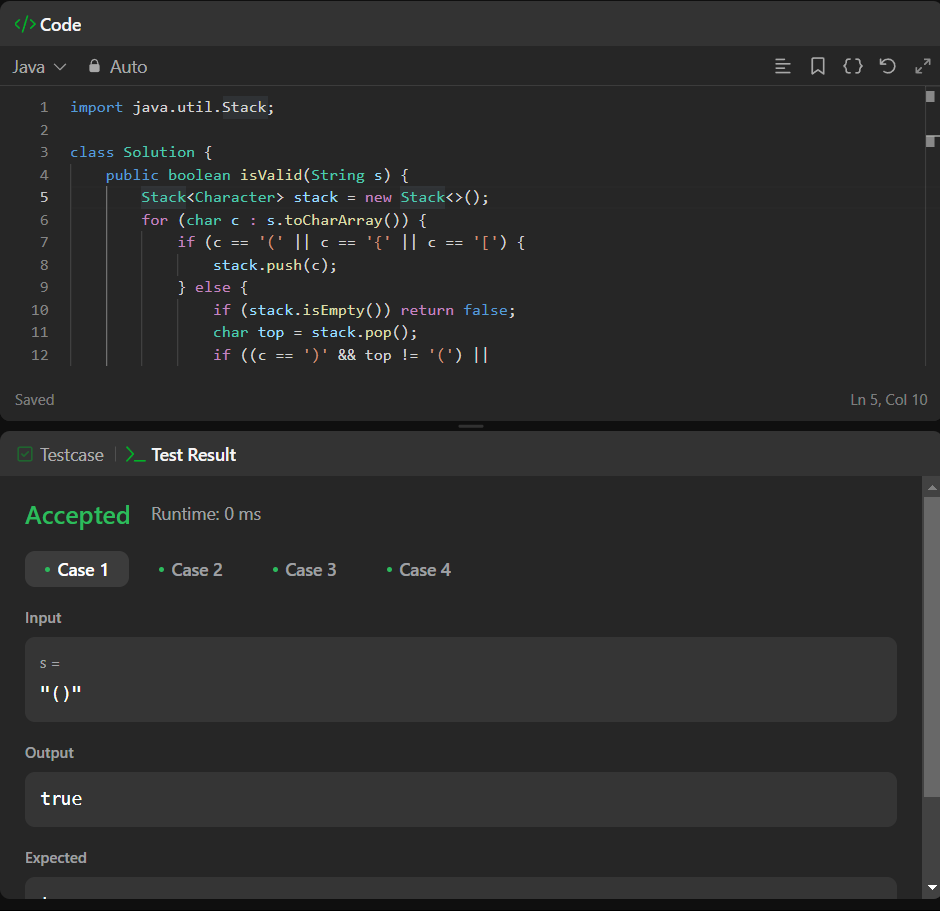
Substring with Concatenation of all Words



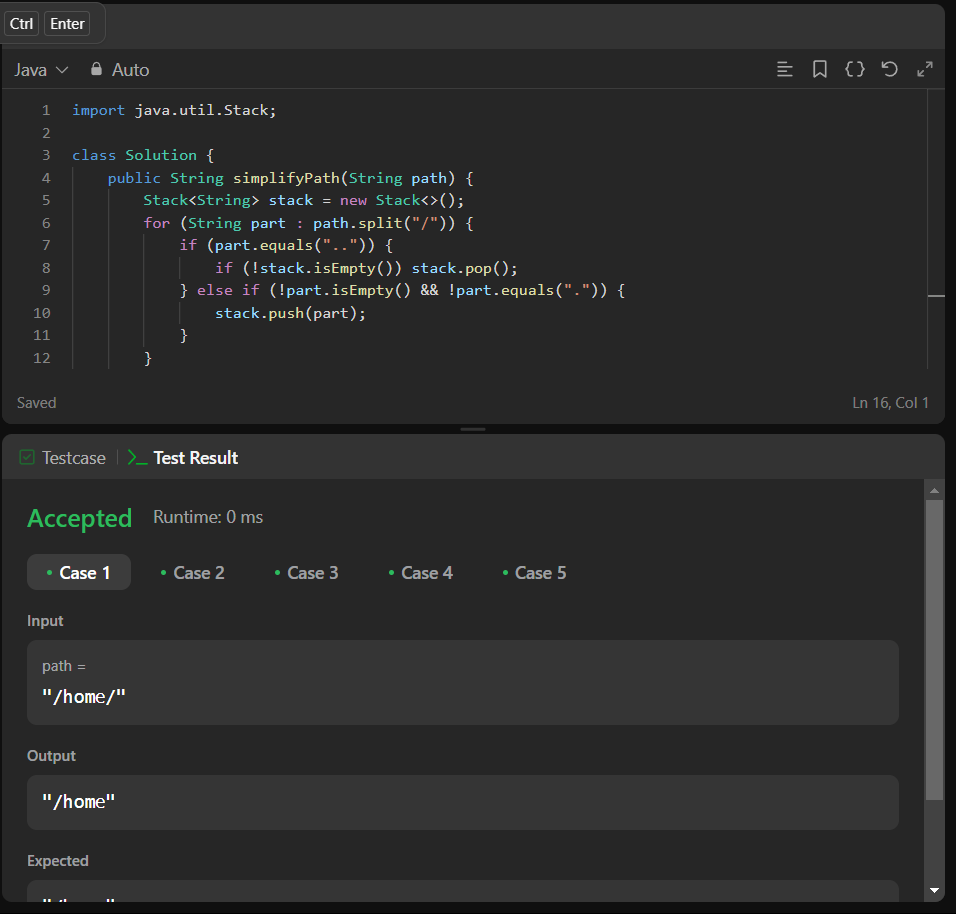
Minimise Window Substring



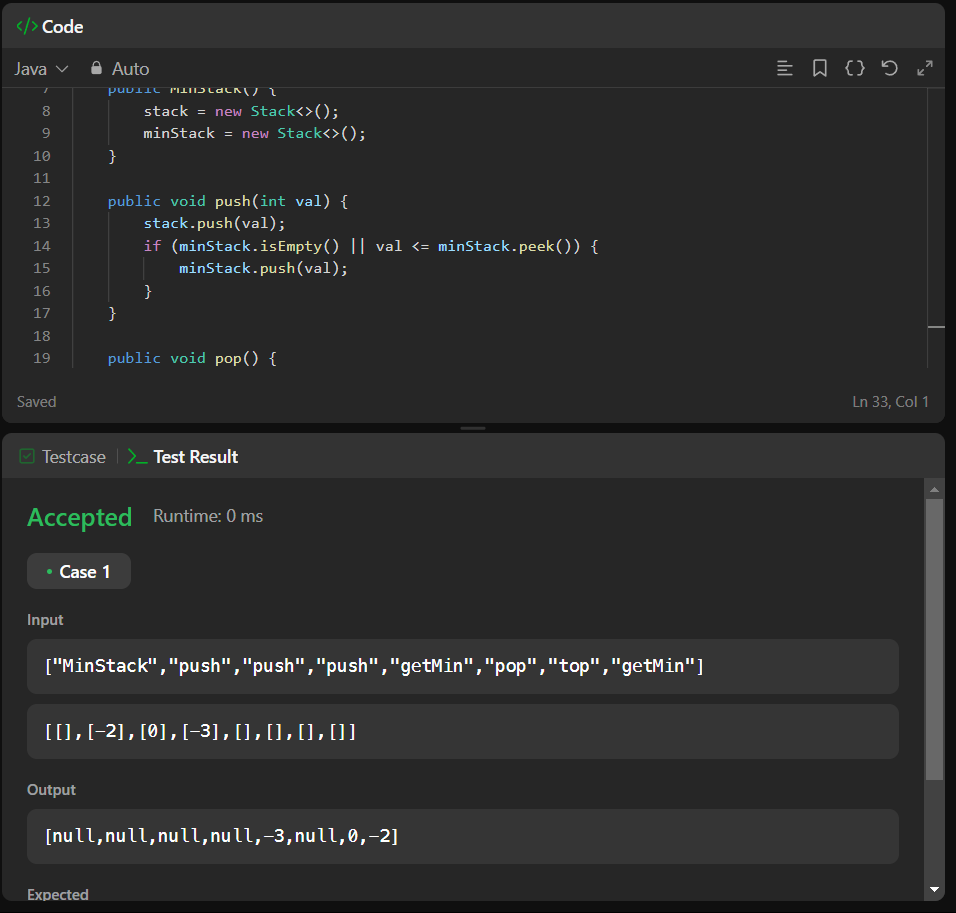
Valid Parantheses



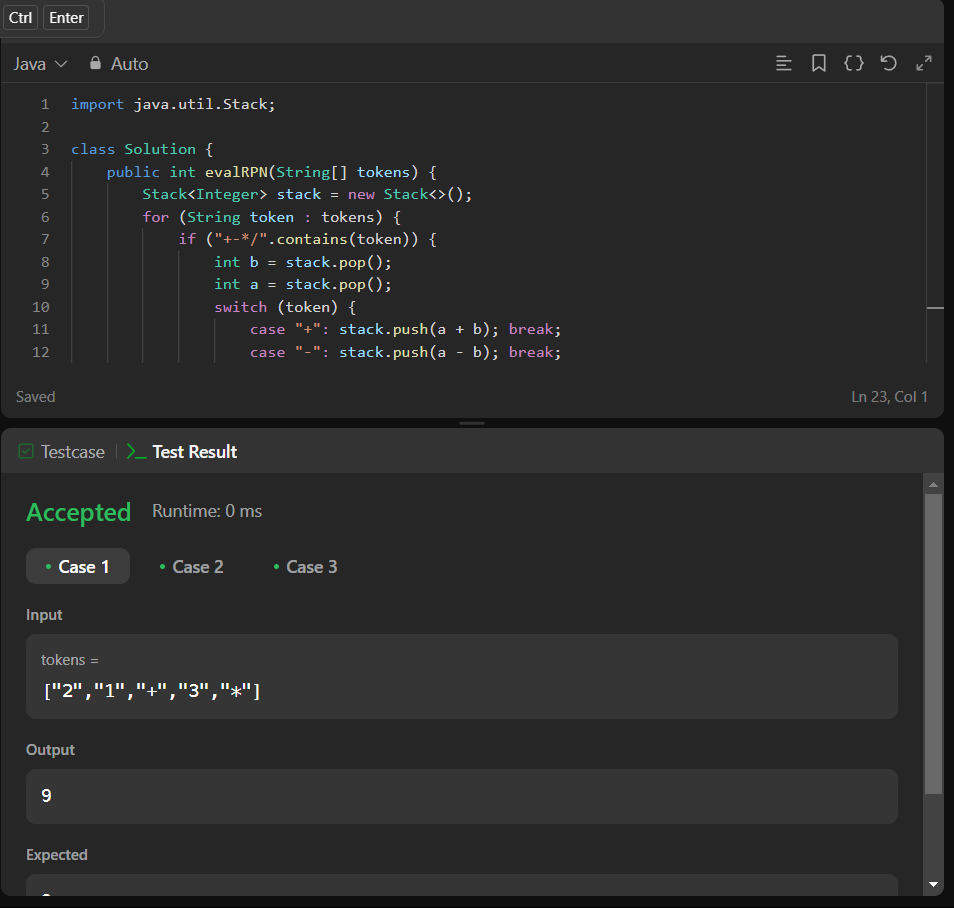
Simplify Path



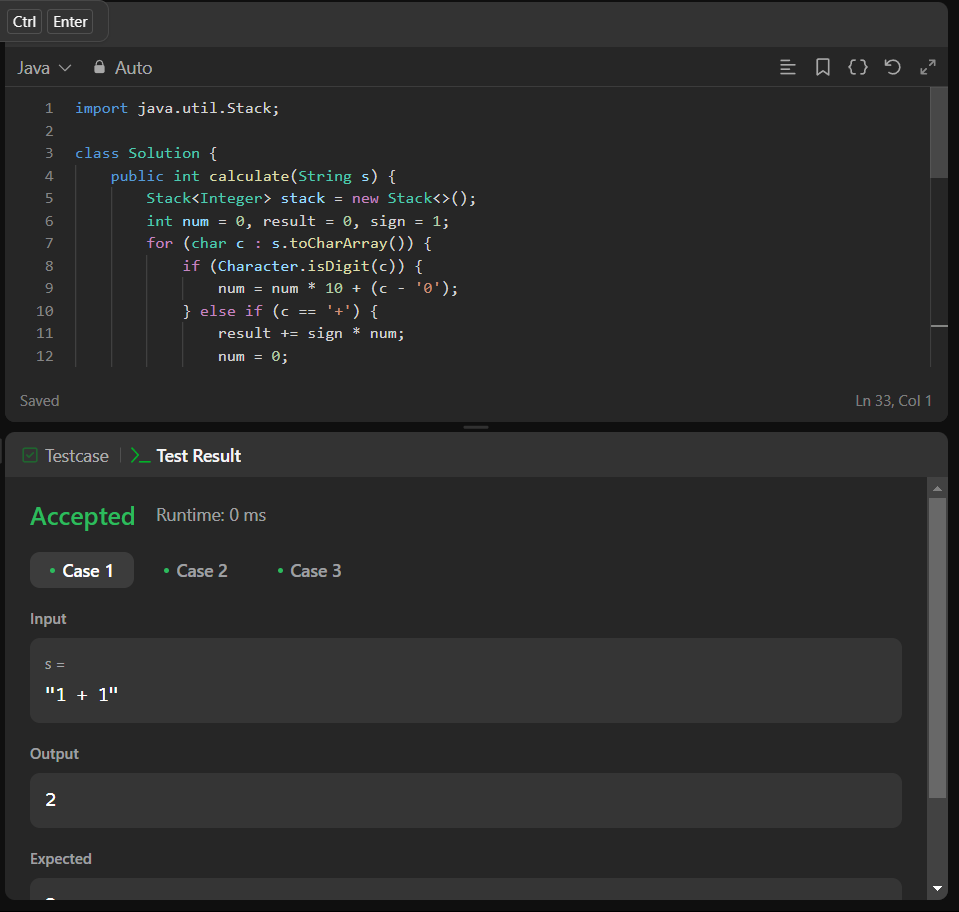
Min Stack



Evaluate Reverse Polish Notation



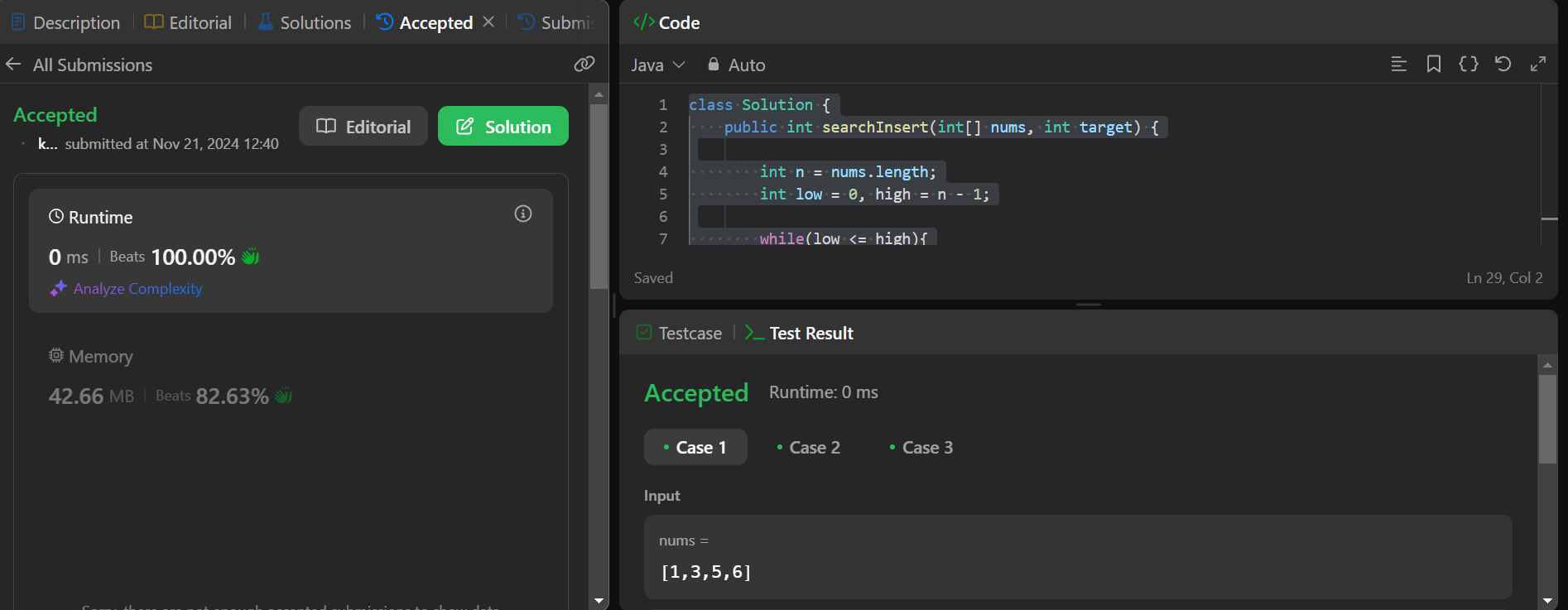
Basic Calculator



**Binary search**

1. **Search in insert position**
2. class Solution {
3. public int searchInsert(int[] nums, int target) {
4. int n = nums.length;
5. int low = 0, high = n - 1;
6. while(low <= high){
7. int mid = (high + low) / 2;
8. if(nums[mid] == target){
9. return mid;
10. }
11. else if(target > nums[mid]){
12. low = mid + 1;
13. }
14. else{
15. high = mid - 1;
16. }
18. }
19. return low;

22. }
23. }

****

**2. Search in 2 D matrix**

class Solution {

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

        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){

            return false;

        }

        int m = matrix.length;

        int n = matrix[0].length;

        int left = 0, right = m \* n - 1;

        while(left <= right){

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

            int midvalue = matrix [mid / n][mid % n];

            if(midvalue == target){

                return true;

            }

            else if(target > midvalue){

                left = mid + 1;

            }

            else{

                right = mid - 1;

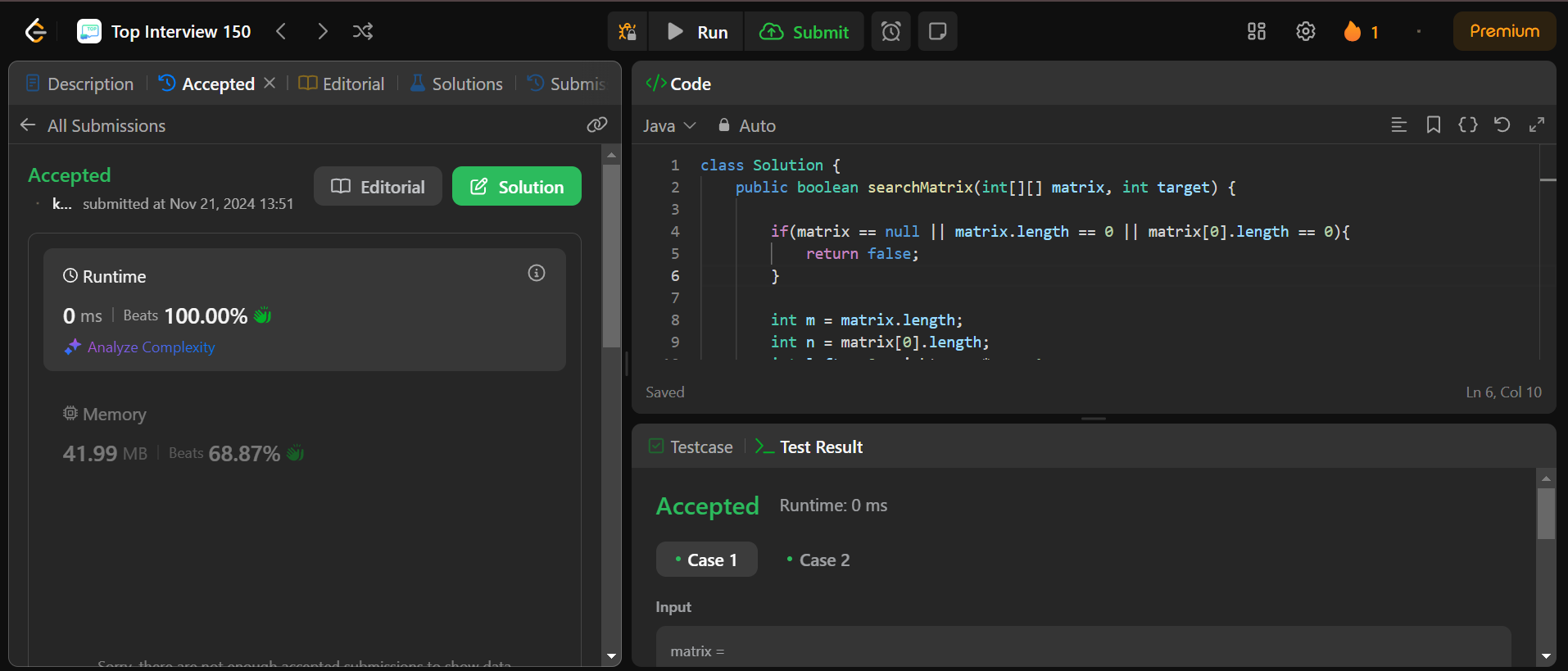
            }

        }

        return false;

    }

}



**3. Peak element**

class Solution {

    public int findPeakElement(int[] nums) {

        int n = nums.length;

        int left = 0, right = n - 1;

        while(left<right){

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

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

                left = mid + 1;

            }

            else{

                right = mid;

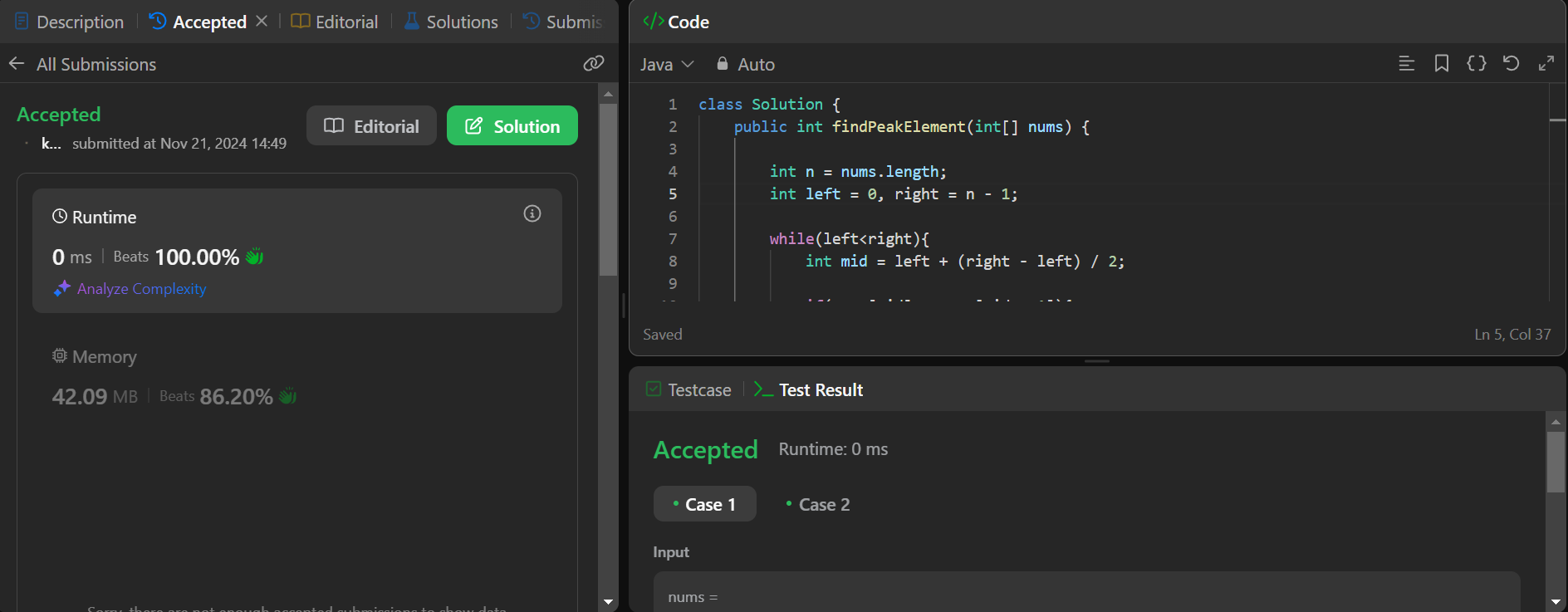
            }

        }

        return left;

    }

}



4. **Search in rotated array**

class Solution {

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

        int n = nums.length;

        int left = 0, right = n - 1;

        while(left <= right){

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

            if(nums[mid] == target){

                return mid;

            }

            else if(nums[mid] >= nums[left]){

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

                    right = mid - 1;

                }

                else{

                    left = mid + 1;

                }

            }

            else{

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

                    left = mid + 1;

                }

                else{

                    right = mid - 1;

                }

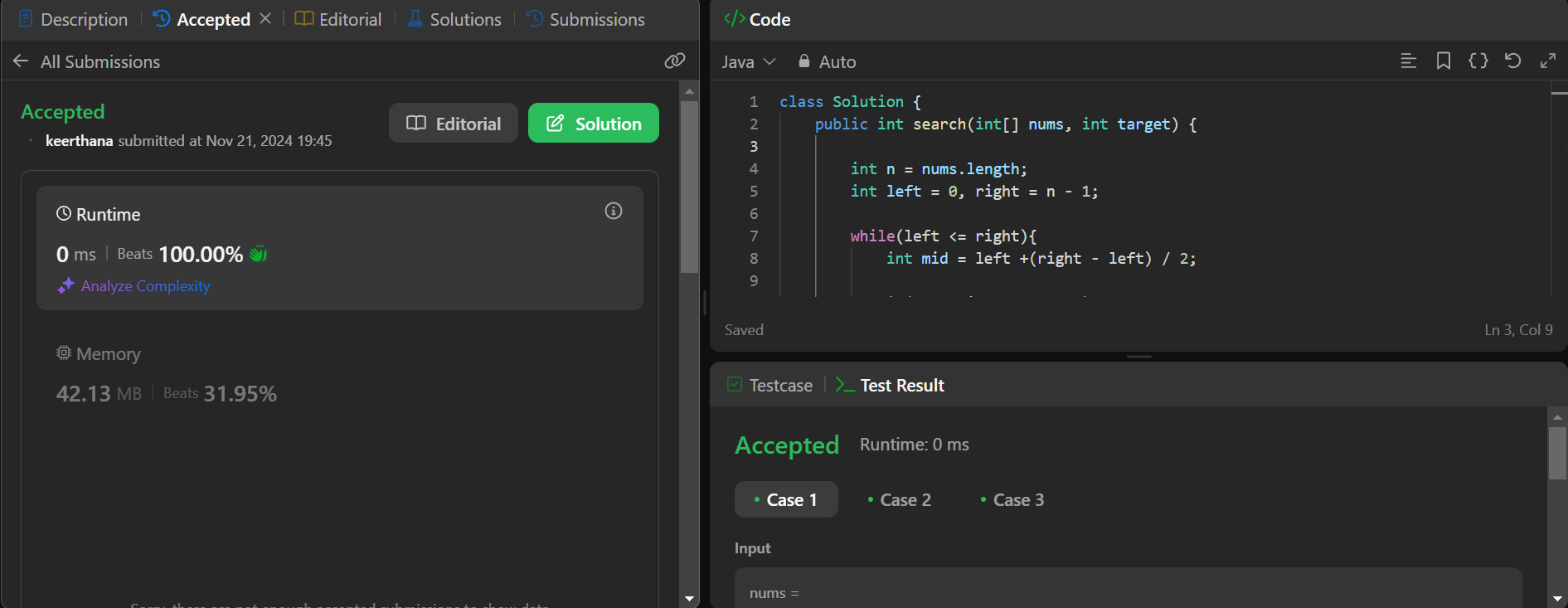
            }

        }

        return -1;

    }

}



**5. Find first and last element in sorted array**

class Solution {

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

        int[] result = new int[2];

        result[0] = findleftmost(nums, target);

        result[1] = findrightmost(nums, target);

        return result;

    }

        private int findleftmost(int[] nums, int target){

            int n = nums.length;

            int left = 0, right = n - 1;

            int index = -1;

            while(left <= right){

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

                if(nums[mid] == target){

                    index = mid;

                    right = mid - 1;

                }

                else if(nums[mid] > target){

                    right = mid - 1;

                }

                else{

                    left = mid + 1;

                }

            }

            return index;

        }

        private int findrightmost(int[] nums, int target){

            int n = nums.length;

            int left = 0, right = n - 1;

            int index = -1;

            while(left <= right){

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

                if(nums[mid] == target){

                    index = mid;

                    left = mid + 1;

                }

                else if(nums[mid] > target){

                    right = mid - 1;

                }

                else{

                    left = mid + 1;

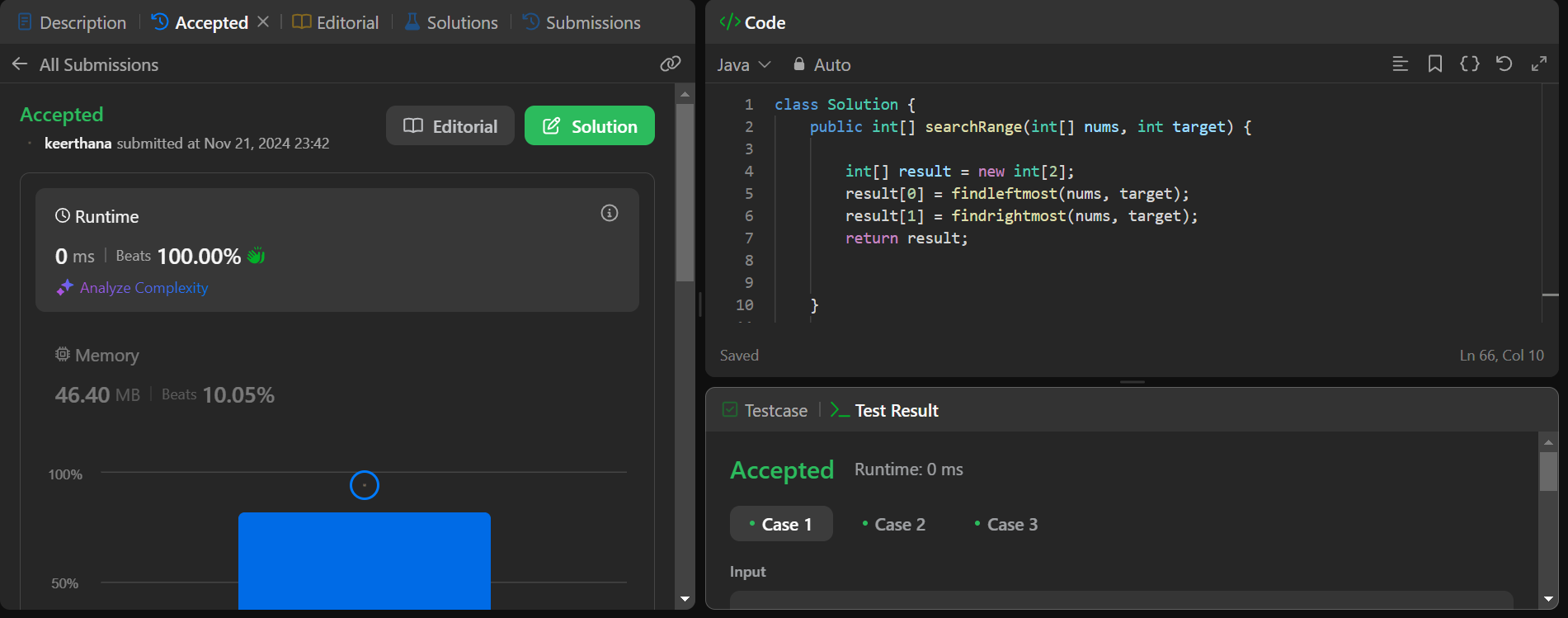
                }

            }

            return index;

        }

        }



**6. Find minimum in a rotated sorted array**

class Solution {

    public int findMin(int[] nums) {

        int n = nums.length;

        int left = 0, right = n - 1;

        while(left < right){

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

            if(nums[mid] > nums[right]){

                left = mid + 1;

            }

            else{

                right = mid;

            }

        }

        return nums[left];

    }

}

