**Experiment 2**

**Student Name:** Khushmn Sangha  **UID:** 22BCS14585

**Branch:** BE-CSE  **Section/Group:** 602/A

**Semester:** 6th  **Date of Performance:** 4-02-25

**Subject Name:** Advanced Programming - II **Subject Code:** 22CSP-351

1. **Aim:**

To solve the following problems on Leetcode, with the goal of optimizing solutions in terms of time complexity and space efficiency:

(I) **No. of 1 Bits** –  
Aim: Given a positive integer n, the objective is to return the number of set bits in its binary representation (also known as the Hamming weight). This helps in understanding bit manipulation and the efficient counting of 1-bits.

(II) **Median of Two Sorted Arrays** –  
Aim: Given two sorted arrays nums1 and nums2 of size m and n respectively, the objective is to find and return the median of these two sorted arrays with an optimal time complexity of O(log(m+n)), which involves leveraging binary search techniques.

(III) **Sort Colors** –  
Aim: Given an array nums with n objects colored red, white, or blue, the goal is to sort them in-place such that objects of the same color are adjacent, in the order red, white, and blue. The problem focuses on sorting without using library sort functions and utilizing counting or the Dutch National Flag algorithm.

(IV) **Reverse Bits** –  
Aim: Given a 32-bit unsigned integer, the task is to reverse its bits. This problem emphasizes bitwise operations and understanding the concept of bit reversal.

(V) **Maximum Subarray** –  
Aim: Given an integer array nums, the objective is to find the contiguous subarray that has the largest sum, using the Kadane’s algorithm to achieve an optimal O(n) time complexity.

(VI) **Search a 2D Matrix II** –  
Aim: Given an m x n matrix where each row and column is sorted, the goal is to search for a target value in the matrix. This problem requires efficient traversal techniques to minimize time complexity, using the "top-right" search approach to find the target in O(m + n) time.

(VII) **First Bad Version** –  
Aim: Given a versioning system, the goal is to find the first bad version using a binary search approach, ensuring optimal performance with O(log n) time complexity.

(VII) **First Bad Version** –  
*Aim:* Given a versioning system, the goal is to find the first bad version using a binary search approach, ensuring optimal performance with O(log n) time complexity.

(VIII) **Top K Frequent Elements** –  
*Aim:* Given an integer array nums, the objective is to find the k most frequent elements. This problem aims to utilize a priority queue (max-heap) or bucket sort to efficiently retrieve the top k elements with an optimal time complexity of O(n log k).

(IX) **Merge Sorted Arrays** –  
*Aim:* Given two sorted integer arrays nums1 and nums2, the goal is to merge them into a single sorted array. This problem focuses on merging with an optimal in-place solution by iterating from the end of the arrays, ensuring a time complexity of O(m + n).

1. **Objective:** To efficiently solve a set of algorithmic problems on Leetcode, optimizing for time and space complexity, by implementing advanced techniques such as bit manipulation, binary search, sorting algorithms, and dynamic programming, with the goal of enhancing problem-solving skills and achieving optimal solutions for real-world applications.
2. **Implementation/Code:**

***Problem No. 191 : Number of 1 Bits***  
***Code:***

class Solution {

    public int hammingWeight(int n) {

        int count = 0;

    while (n != 0) {

        count += n & 1;

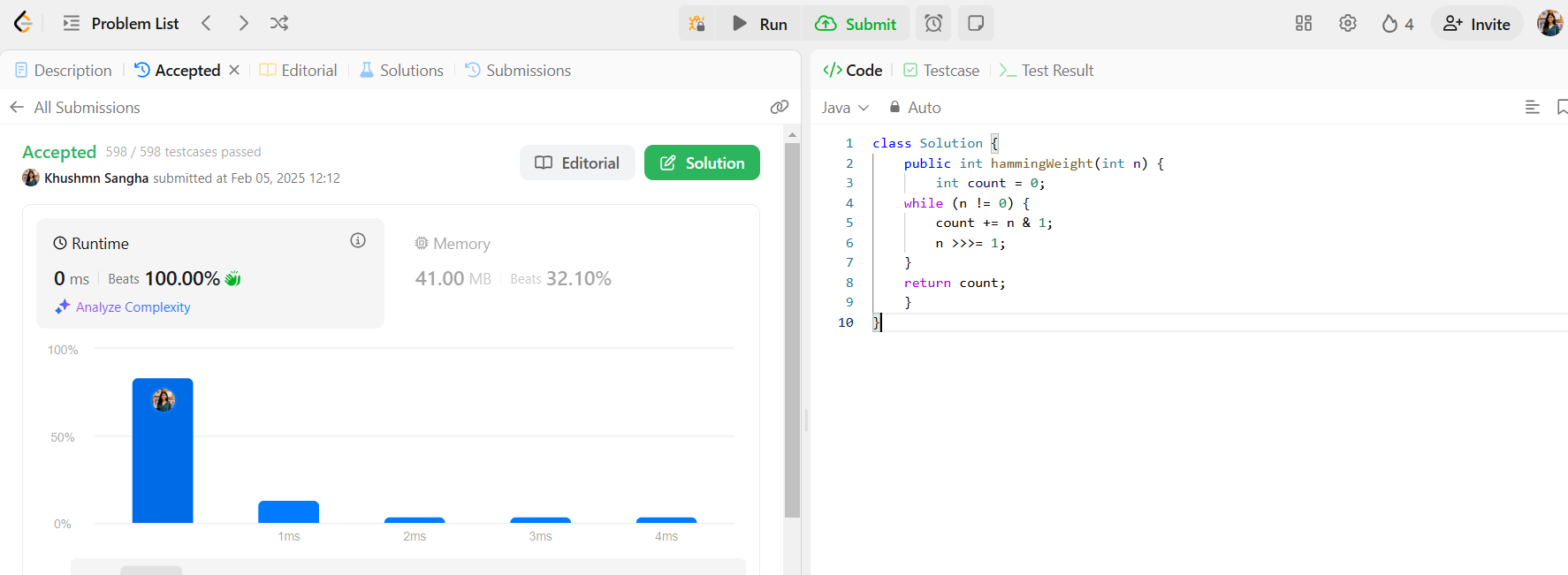
        n >>>= 1;

    }

    return count;

    }

}



***Problem No. 4 : Median of Two Sorted Arrays***

***Code:***

class Solution {

    public double findMedianSortedArrays(int[] nums1, int[] nums2) {

        if (nums1.length > nums2.length) {

            return findMedianSortedArrays(nums2, nums1);

        }

        int x = nums1.length;

        int y = nums2.length;

        int low = 0, high = x;

        while (low <= high) {

            int partitionX = (low + high) / 2;

            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxX = (partitionX == 0) ? Integer.MIN\_VALUE : nums1[partitionX - 1];

            int minX = (partitionX == x) ? Integer.MAX\_VALUE : nums1[partitionX];

            int maxY = (partitionY == 0) ? Integer.MIN\_VALUE : nums2[partitionY - 1];

            int minY = (partitionY == y) ? Integer.MAX\_VALUE : nums2[partitionY];

            if (maxX <= minY && maxY <= minX) {

                if ((x + y) % 2 == 0) {

                    return (Math.max(maxX, maxY) + Math.min(minX, minY)) / 2.0;

                } else {

                    return Math.max(maxX, maxY);

                }

            } else if (maxX > minY) {

                high = partitionX - 1;

            } else {

                low = partitionX + 1;

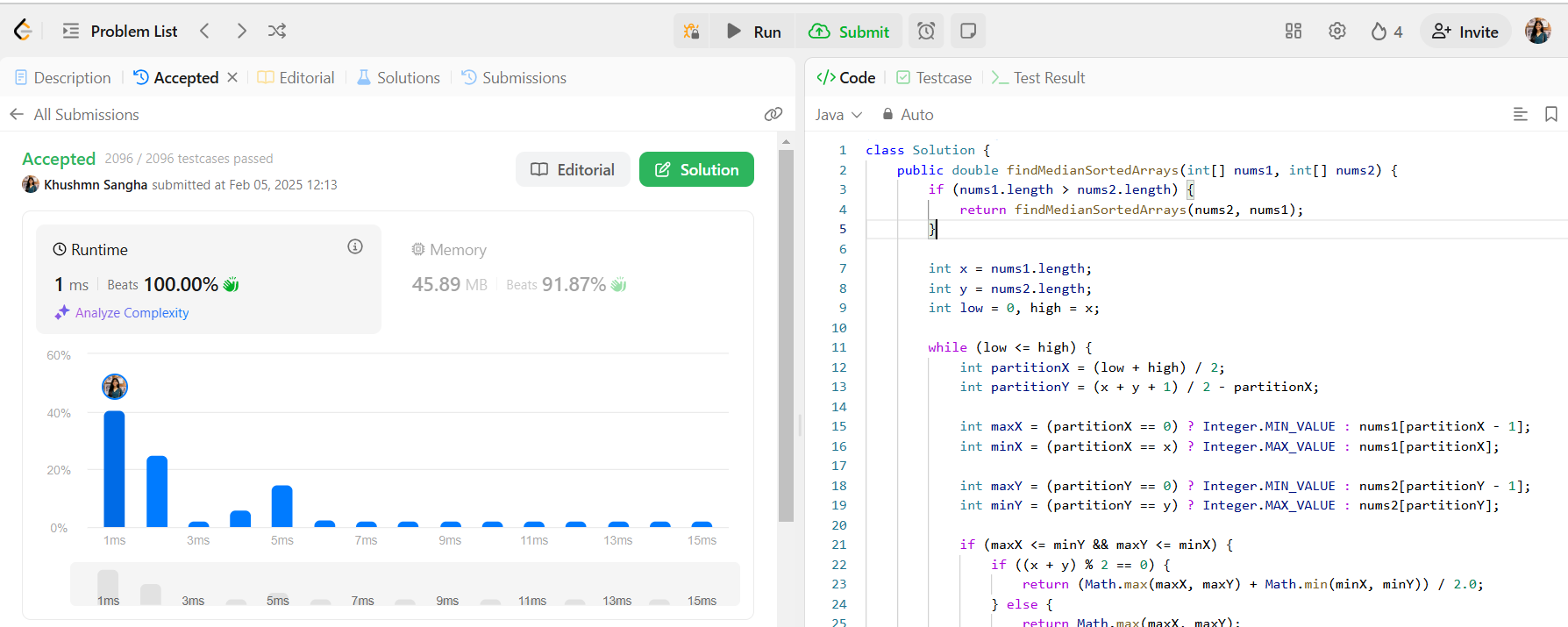
            }

        }

        throw new IllegalArgumentException("Input arrays are not sorted.");

    }

}

******

***Problem No. 75 : Sort Colors***

***Code:***

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;

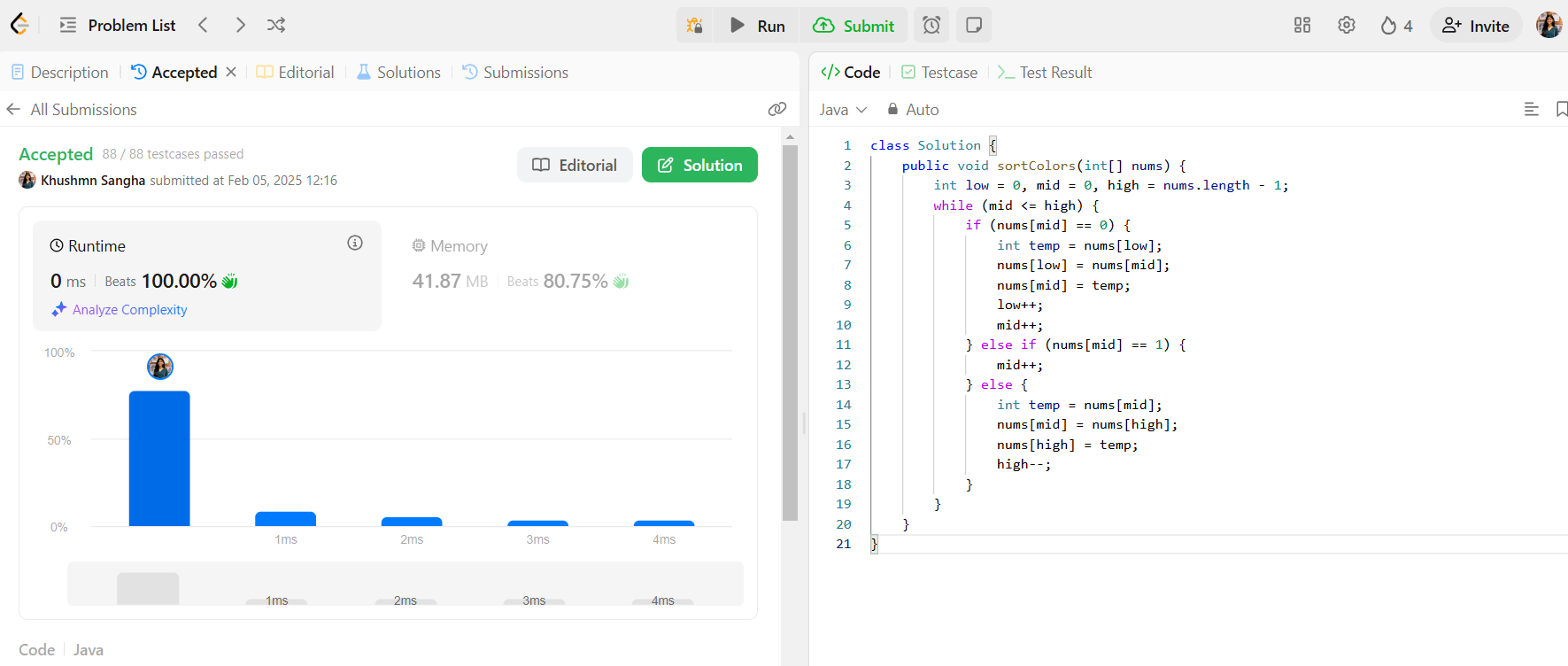
                high--;

            }

        }

    }

}

******

***Problem No. 190 : Reverse Bits***

***Code:***

public class Solution {

    public int reverseBits(int n) {

        int result = 0;

    for (int i = 0; i < 32; i++) {

        result = (result << 1) | (n & 1);

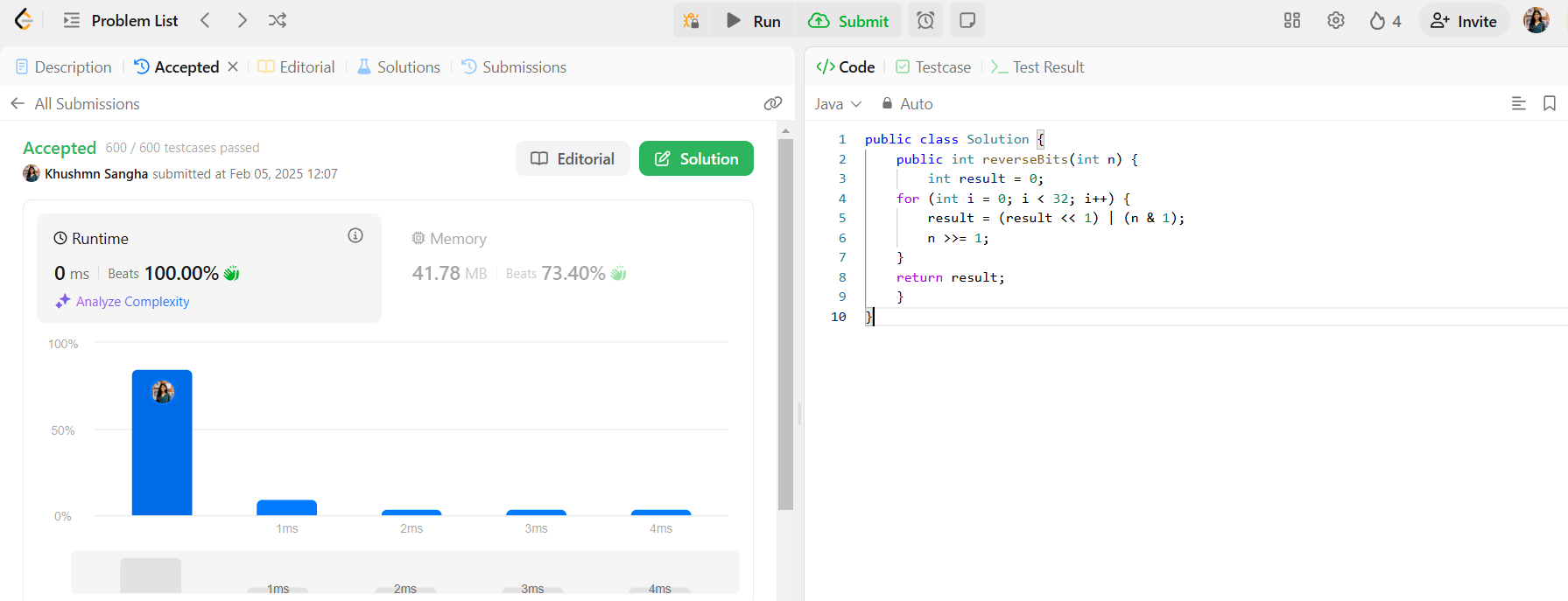
        n >>= 1;

    }

    return result;

    }

}



***Problem No. 53 : Maximum Subarray -***

***Code:***

class Solution {

    public int maxSubArray(int[] nums) {

        int currsum=nums[0];

        int globalsum=nums[0];

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

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

          if(currsum>globalsum){

            globalsum=currsum;

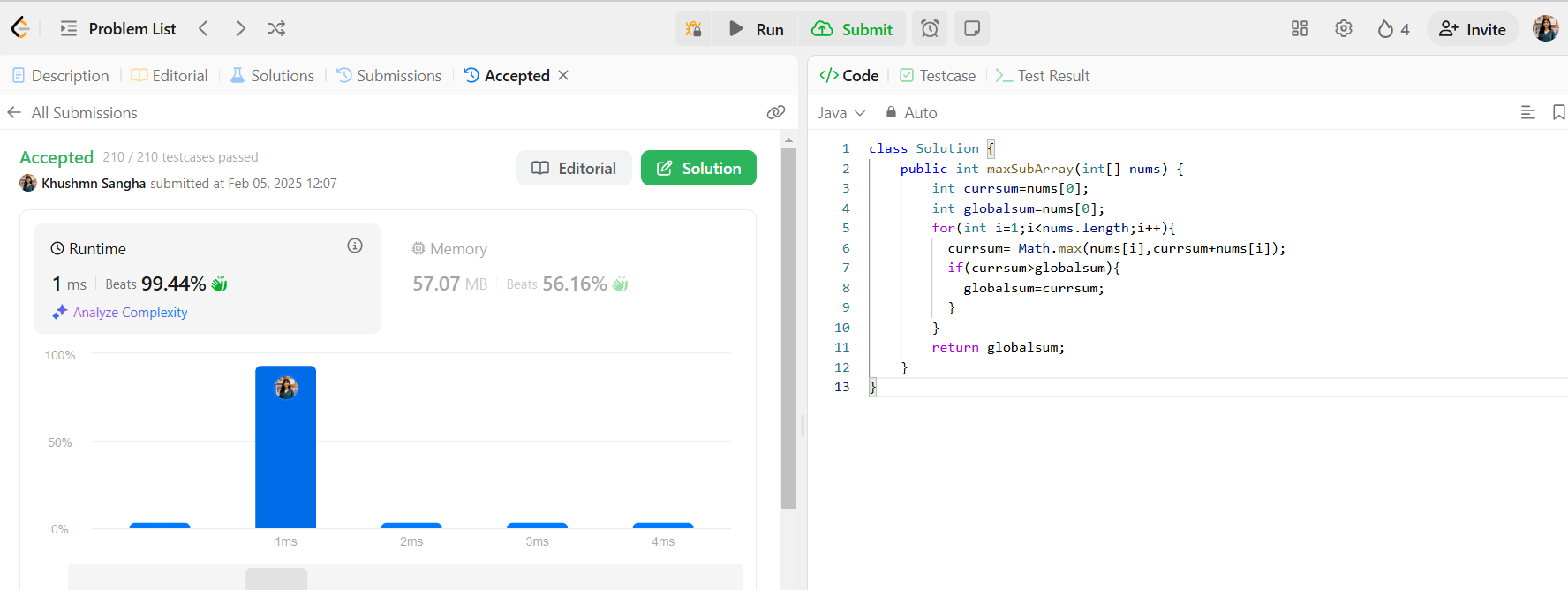
          }

        }

        return globalsum;

    }

}

****

***Problem No. 240 : Search a 2D Matrix –***

***Code:***

class Solution {

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

      int m=arr.length;

      int n=arr[0].length;

      int i=0;

      int j=n-1;

      while(i>=0 && i<m && j>=0 &&j<n){

        if(arr[i][j]==target){

            return true;

        }else if(target>arr[i][j]){

            i++;

        }else if(target<arr[i][j]){

            j--;

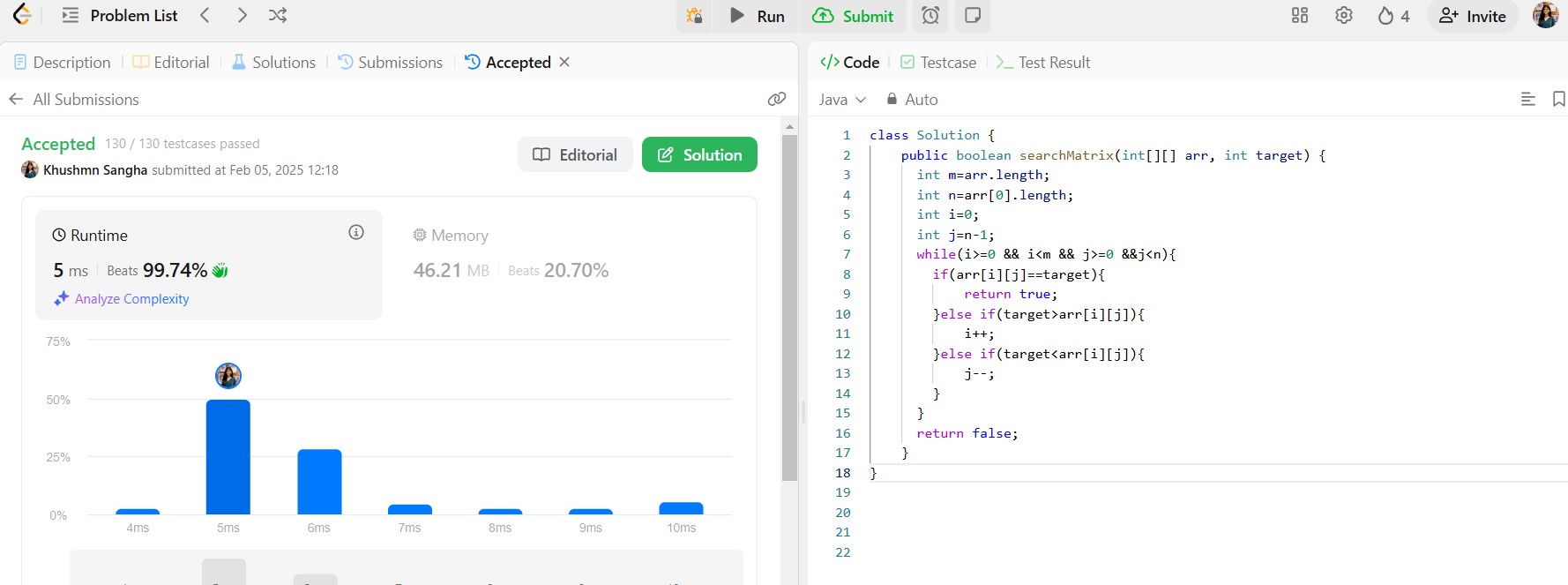
        }

      }

      return false;

    }

}

****

***Problem No. 278 First Bad Version***

***Code:***

public class Solution extends VersionControl {

    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 {

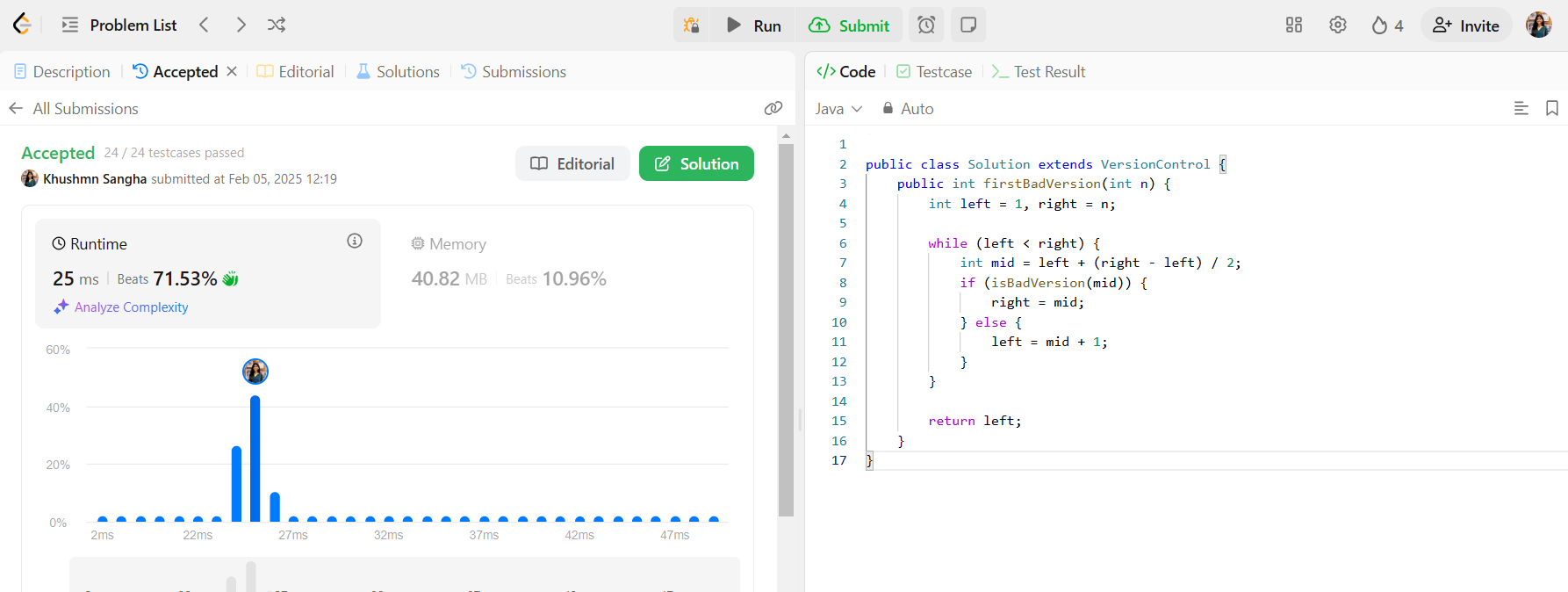
                left = mid + 1;

            }

        }

        return left;

    }

******

***Prolem No. 347: Top K Frequent Elements –***

***Code:***

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<>((a, b) -> freqMap.get(a) - freqMap.get(b));

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

            minHeap.offer(key);

            if (minHeap.size() > k) {

                minHeap.poll();

            }

        }

        int[] result = new int[k];

        for (int i = k - 1; i >= 0; i--) {

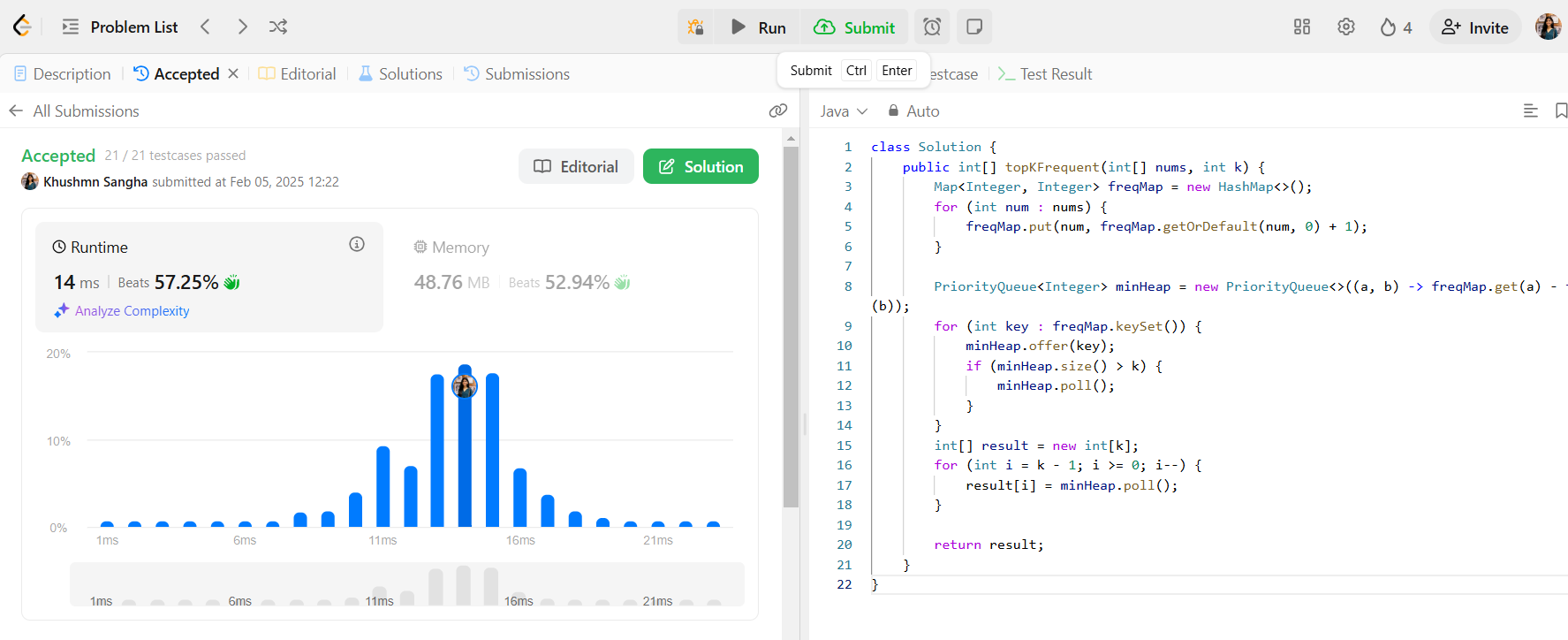
            result[i] = minHeap.poll();

        }

        return result;

    }

}



***Problem No. 88 : Merge Sorted Arrays***

***Code:***

class Solution {

    public void merge(int[] nums1, int m, int[] nums2, int n) {

    int i = m - 1;

    int j = n - 1;

    int k = m + n - 1;

    while (j >= 0)

      if (i >= 0 && nums1[i] > nums2[j])

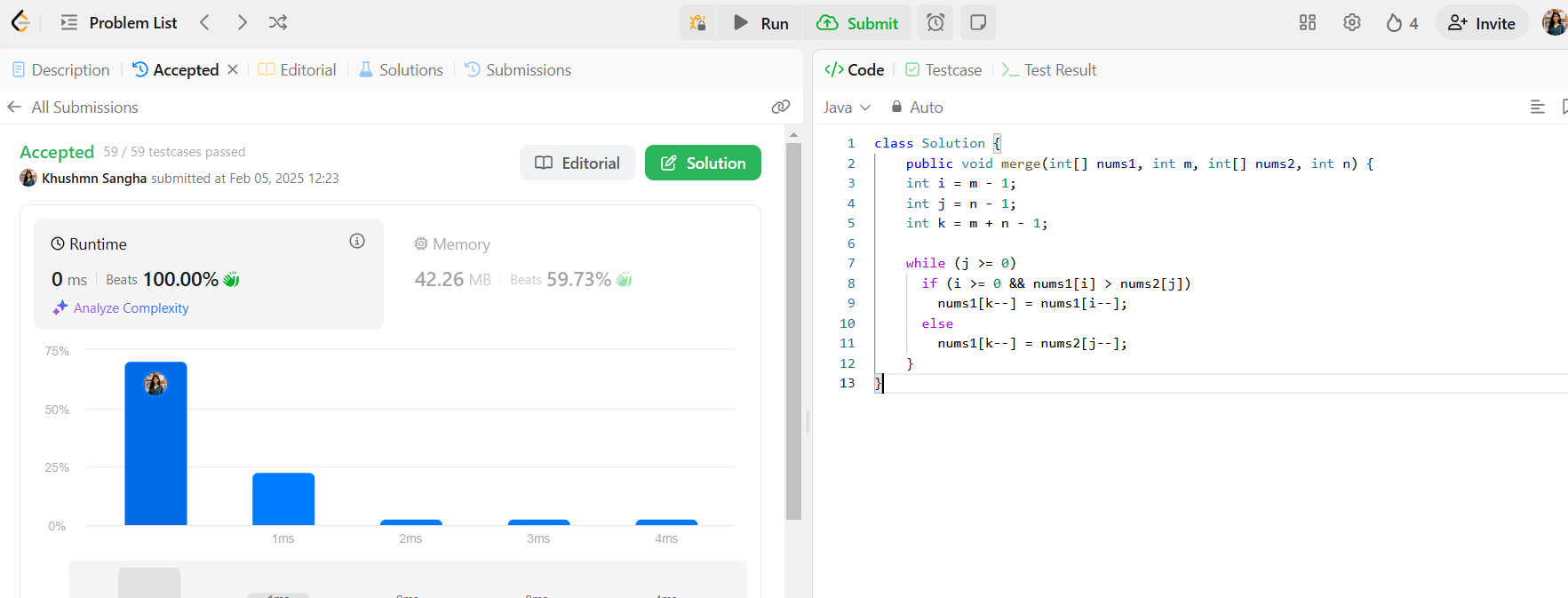
        nums1[k--] = nums1[i--];

      else

        nums1[k--] = nums2[j--];

    }

}



***Problem No. 493:*** [***Reverse Pairs***](https://leetcode.com/problems/reverse-pairs/)

***Code:***

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) {

        temp[k++] = nums[i] <= nums[j] ? nums[i++] : nums[j++];

    }

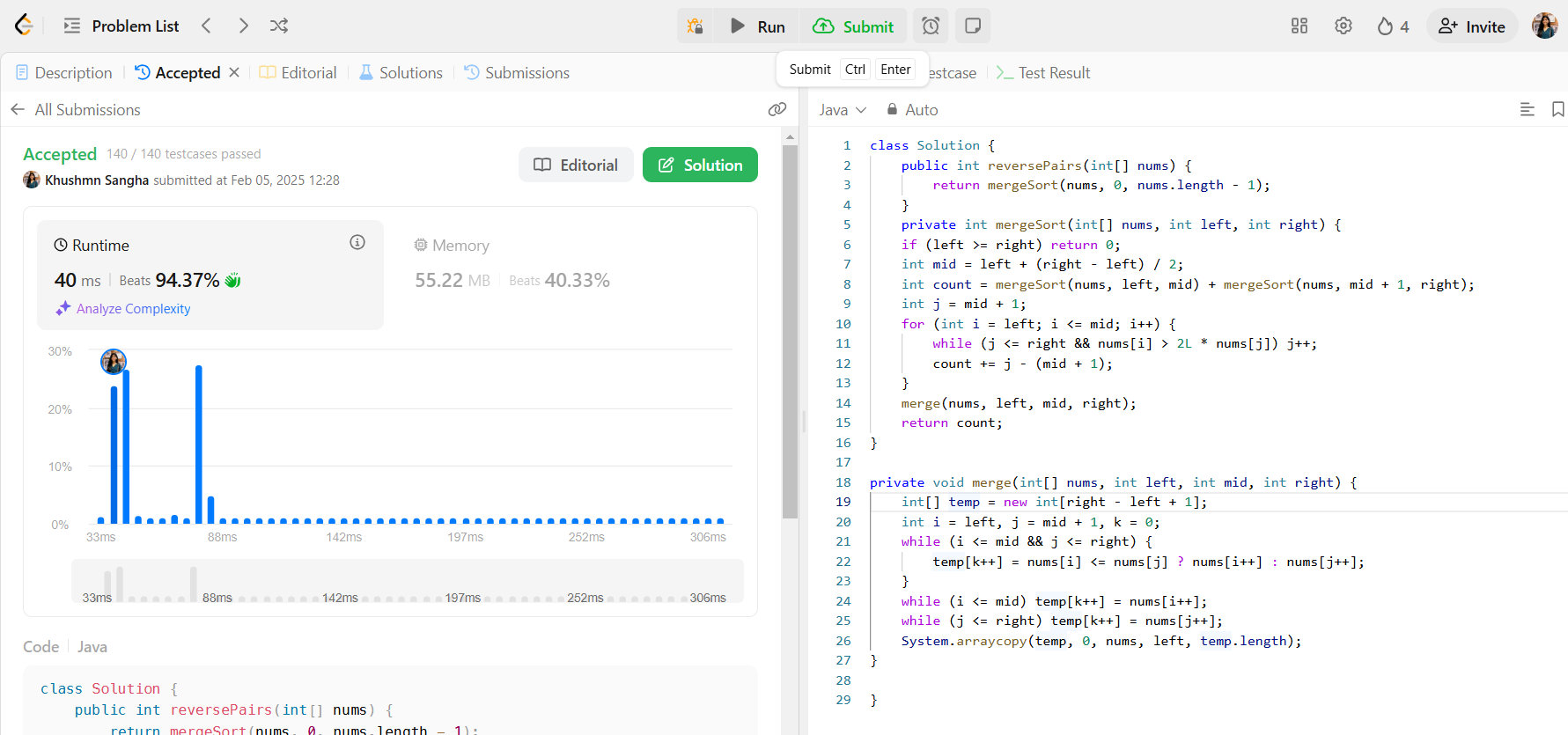
    while (i <= mid) temp[k++] = nums[i++];

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

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

}

}



***Problem No. 218:*** [***The Skyline Problem***](https://leetcode.com/problems/the-skyline-problem/)

***Code:***

class Solution {

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

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

    for (int[] b : buildings) {

        height.add(new int[]{b[0], -b[2]});

        height.add(new int[]{b[1], b[2]});

    }

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

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

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

    pq.add(0);

    int prev = 0;

    for (int[] h : height) {

        if (h[1] < 0) {

            pq.add(-h[1]);

        } else {

            pq.remove(h[1]);

        }

        int curr = pq.peek();

        if (curr != prev) {

            result.add(Arrays.asList(h[0], curr));

            prev = curr;

        }

    }

    return result;

    }

}

