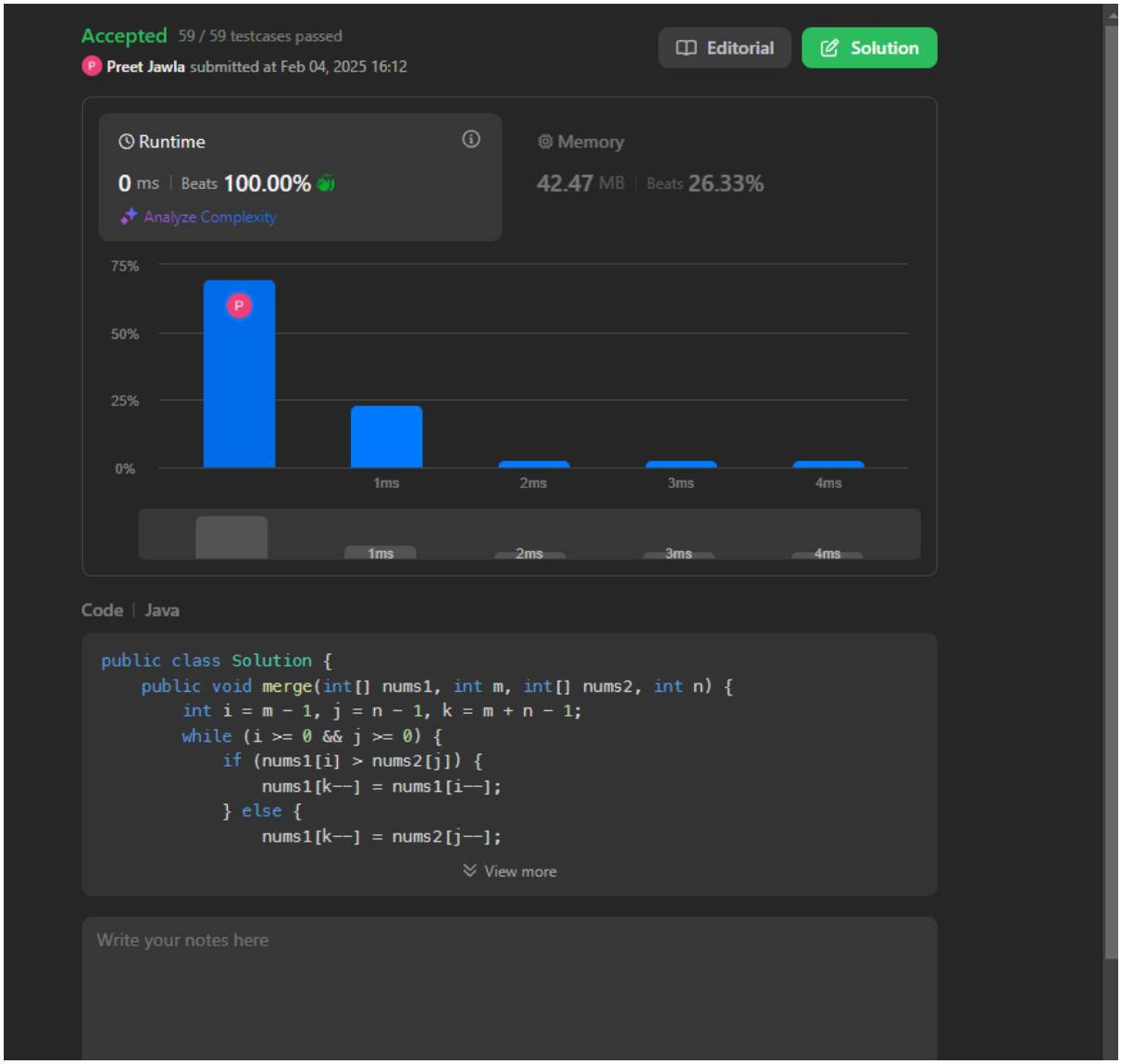
1. Merge Sorted Array

```
public class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m - 1, j = n - 1, k = m + n - 1;
        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k--] = nums1[i--];
            } else {
                nums1[k--] = nums2[j--];
            }
        }
        while (j >= 0) {
            nums1[k--] = nums2[j--];
        }
    }
}
```

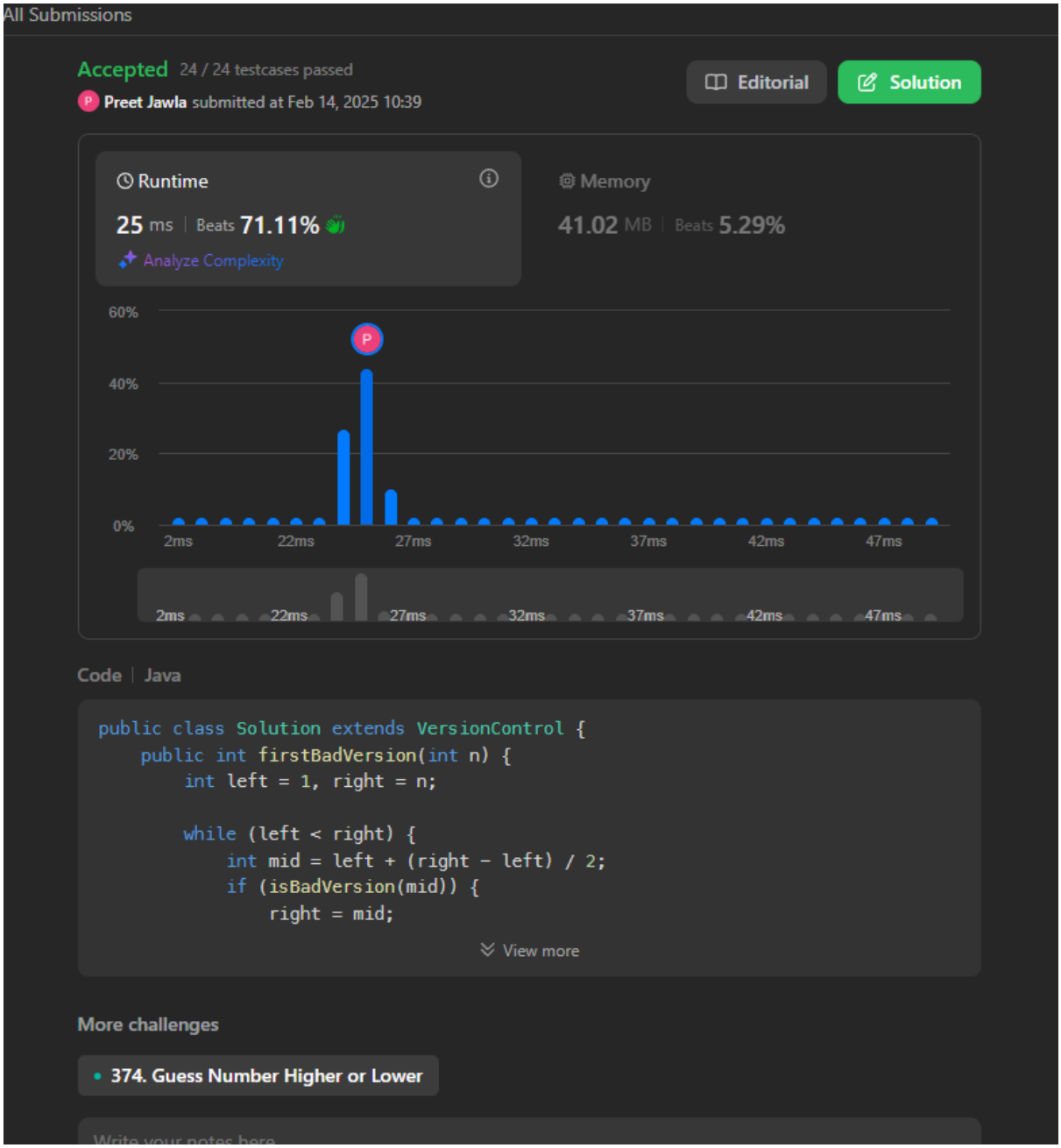


2. First-Bad Version

```
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;
    }
}
```



3. Sort Color

```
public class Solution {
    public void sortColors(int[] nums) {
        int low = 0, mid = 0, high = nums.length - 1;
        while (mid <= high) {
            if (nums[mid] == 0) {
                swap(nums, low++, mid++);
            } else if (nums[mid] == 1) {
                mid++;
            } else {
                swap(nums, mid, high--);
```

```
            }
        }
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

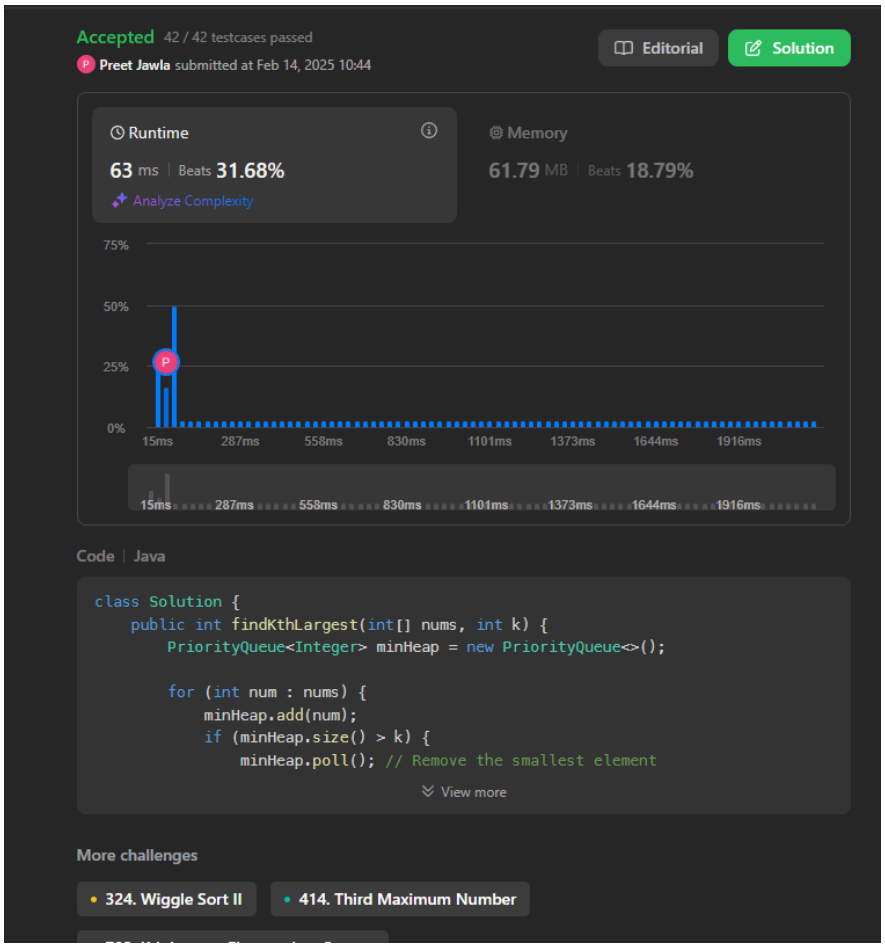
4. Kth largest element in an array

```
class Solution {
    public int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();

        for (int num : nums) {
            minHeap.add(num);
            if (minHeap.size() > k) {
```

```
            minHeap.poll();
        }
    }

    return minHeap.peek();
  }
}
```



5. Merge interval

```
public class Solution {
  public int[][] merge(int[][] intervals) {
    if (intervals.length == 0) return new int[0][0];

    Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

    List<int[]> result = new ArrayList<>();
    int[] currentInterval = intervals[0];
    result.add(currentInterval);

    for (int[] interval : intervals) {
      if (interval[0] <= currentInterval[1]) {
```

```
            currentInterval[1] = Math.max(currentInterval[1], interval[1]);
        } else {
            currentInterval = interval;
            result.add(currentInterval);
        }
    }

    return result.toArray(new int[result.size()][]);
  }
}
```