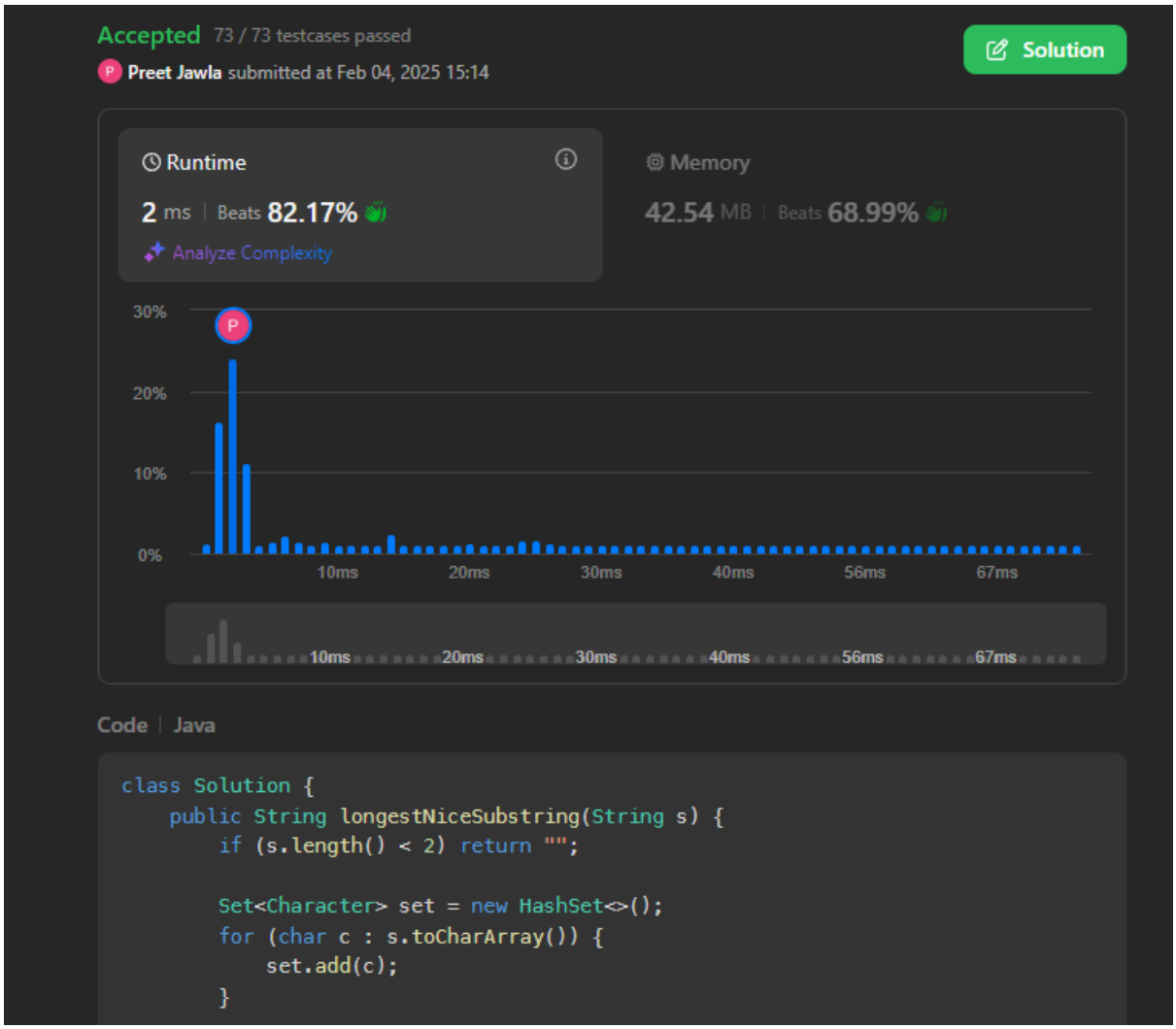
# 1763.Longest Nice Substring

Code:

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) return "";

        Set<Character> set = new HashSet<>();
        for (char c : s.toCharArray()) {
            set.add(c);
        }

        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (set.contains(Character.toLowerCase(c)) && set.contains(Character.toUpperCase(c))) {
                continue;
            }

            String left = longestNiceSubstring(s.substring(0, i));
            String right = longestNiceSubstring(s.substring(i + 1));

            return left.length() >= right.length() ? left : right;
        }

        return s;
    }
}
```

SS:




## 1763.Longest Nice Substring

Code: public class Solution {

public int reverseBits(int n) {

int result = 0;

for (int i = 0; i < 32; i++) {

// Shift result left to make space for the new bit

result <<= 1;

// Get the last bit of n and add it to result

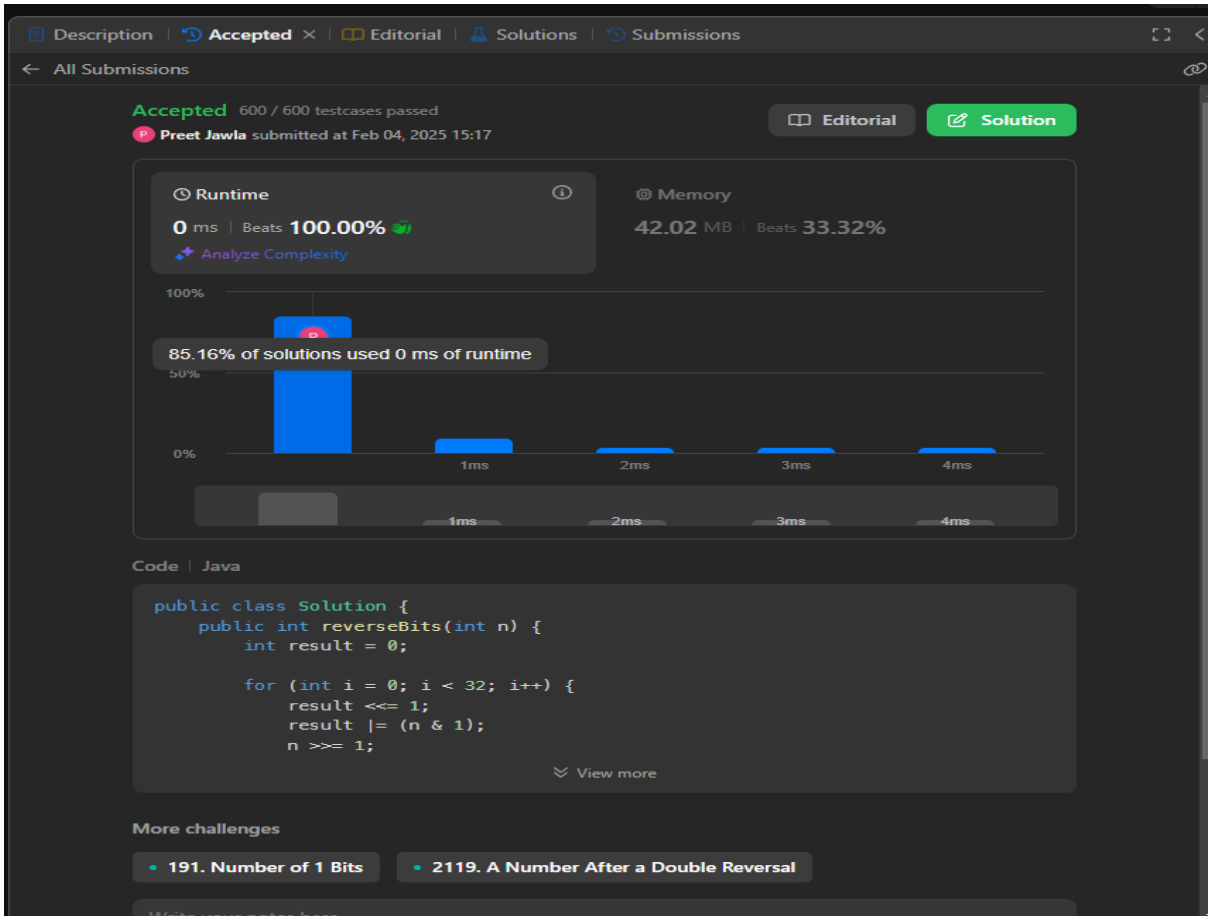
result |= (n & 1);

// Shift n right to process the next bit

n >>= 1;

```
        }

        return result;
    }
}
```

Ss:



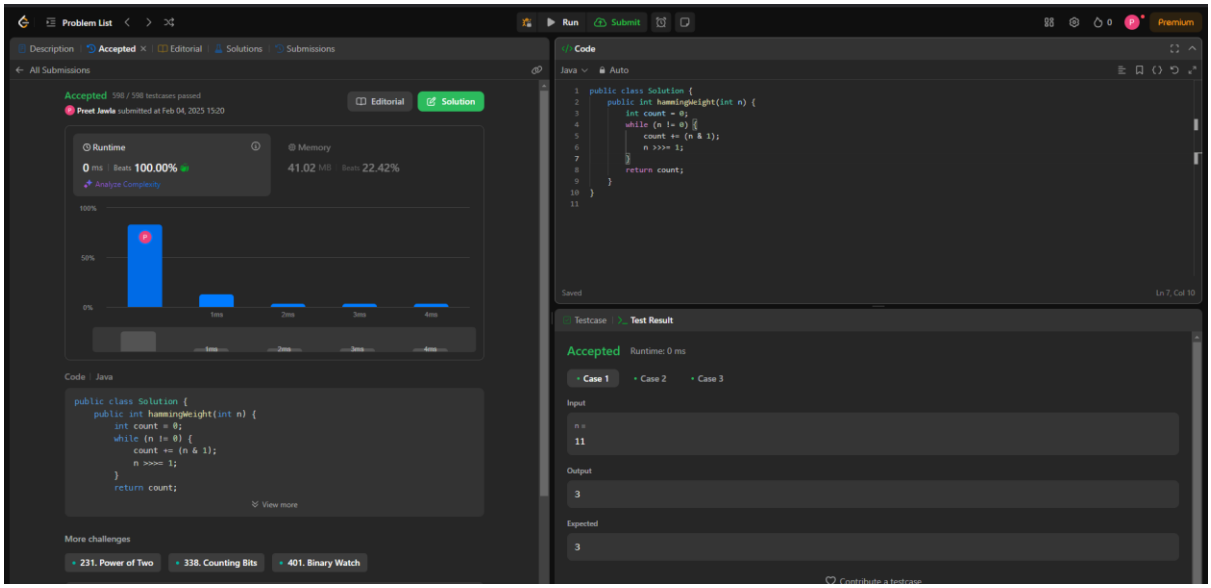

## 190.Reverse Bits

Code:

```
public class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>>= 1;
        }
        return count;
```

```
    }
}
```

SS:

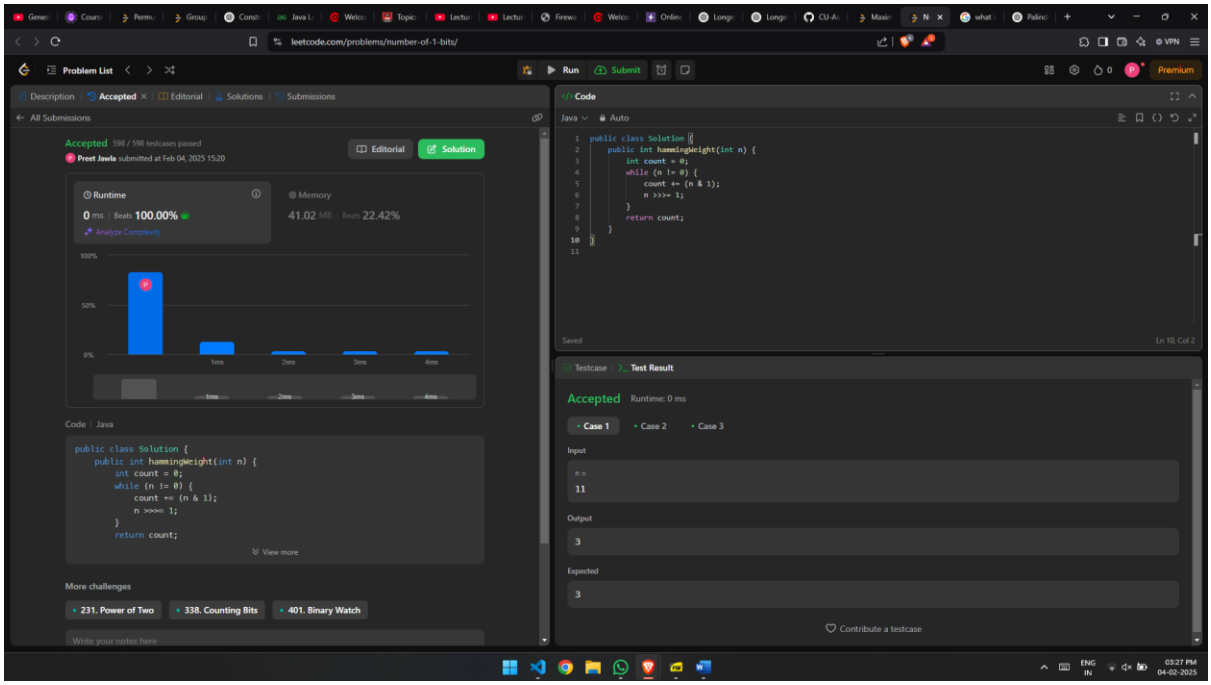



## 191.Number of 1 Bits

Code: public class Solution {

```
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>>= 1;
        }
        return count;
    }
}
```

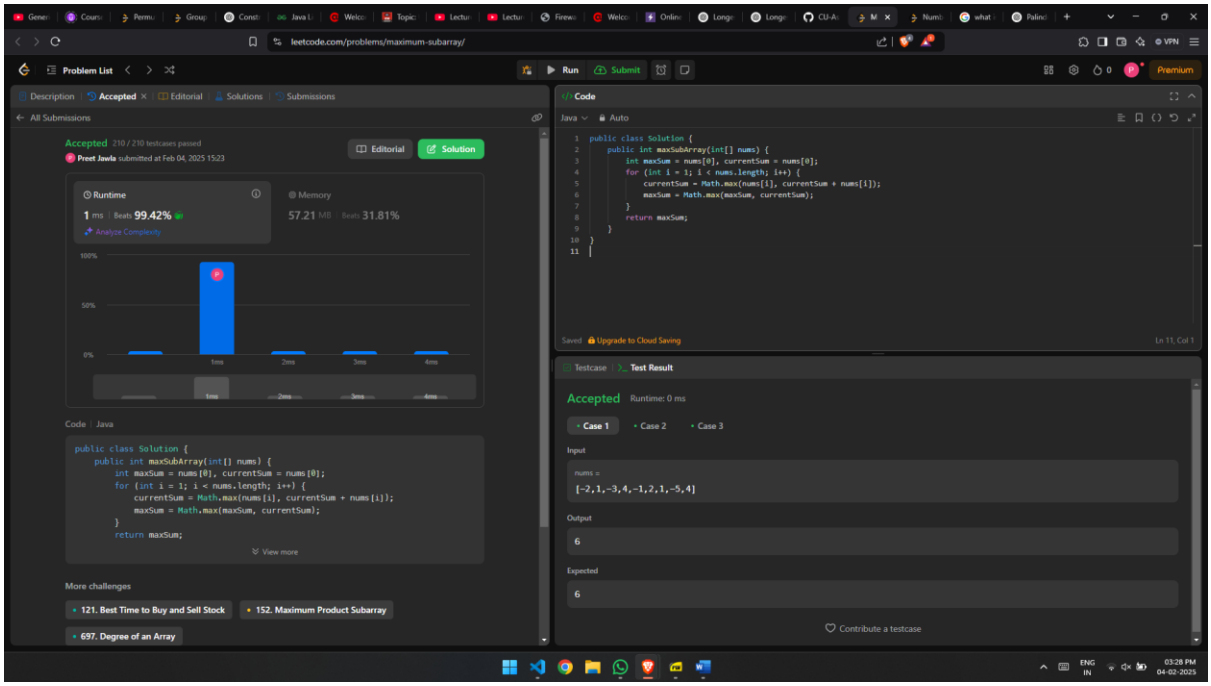
Ss:




## 53.Maximum Subarray

Code: public class Solution {

```
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0], 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;
    }
}
```

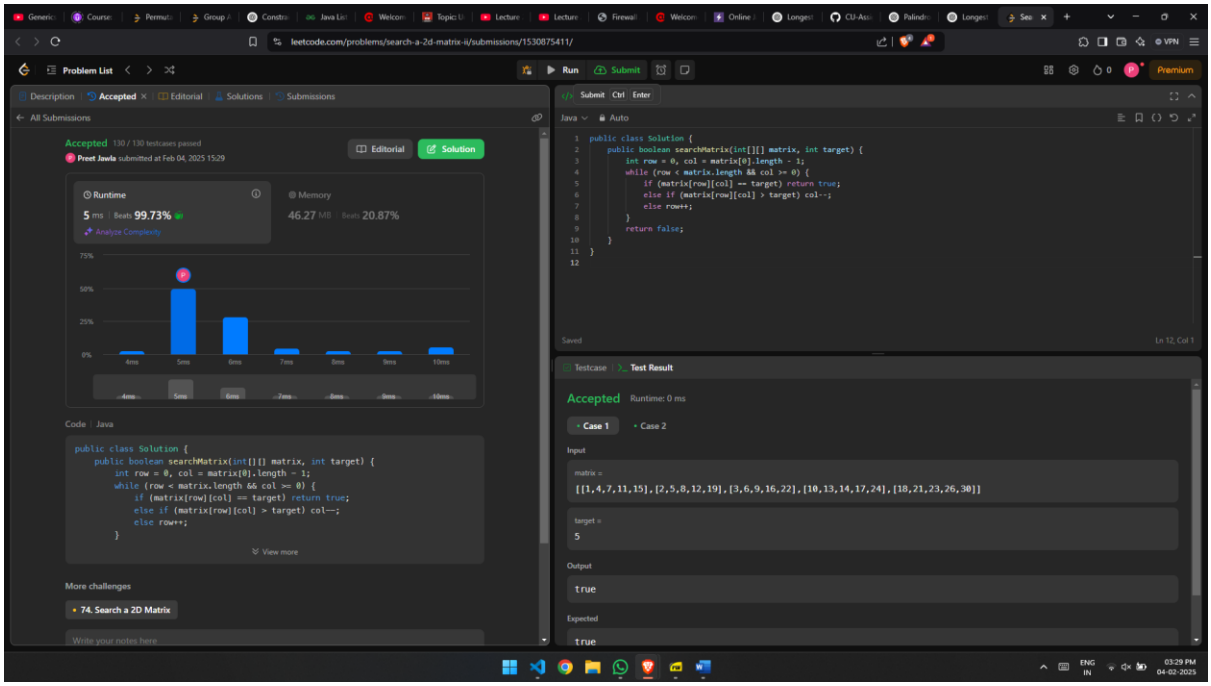



## 240.Search a 2D Matrix II

Code: public class Solution {

```
    public boolean searchMatrix(int[][] matrix, int target) {
        int row = 0, col = matrix[0].length - 1;
        while (row < matrix.length && col >= 0) {
            if (matrix[row][col] == target) return true;
            else if (matrix[row][col] > target) col--;
            else row++;
        }
        return false;
    }
}
```

SS:



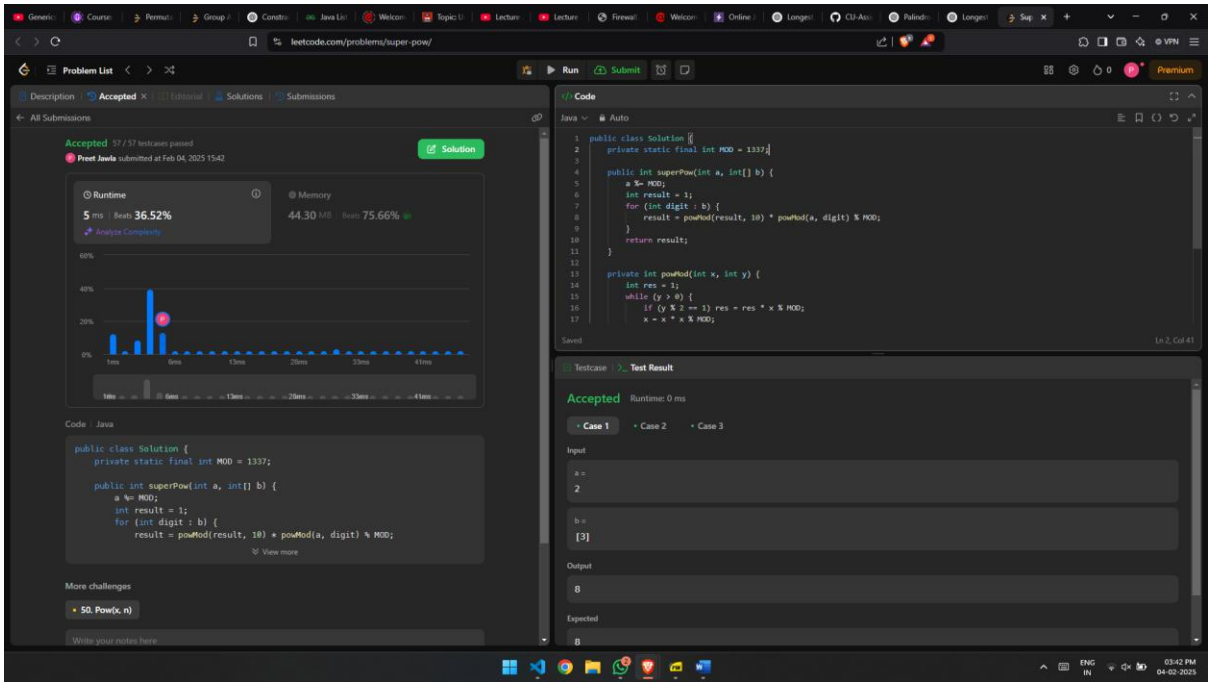

## 372.Super Pow

Code: public class Solution {

```
    private static final int MOD = 1337;


    public int superPow(int a, int[] b) {
        a %= MOD;
        int result = 1;
        for (int digit : b) {
            result = powMod(result, 10) * powMod(a, digit) % MOD;
        }
        return result;
    }


    private int powMod(int x, int y) {
        int res = 1;
        while (y > 0) {
            if (y % 2 == 1) res = res * x % MOD;
```

```
            x = x * x % MOD;
            y /= 2;
        }
        return res;
    }
}
```

SS:

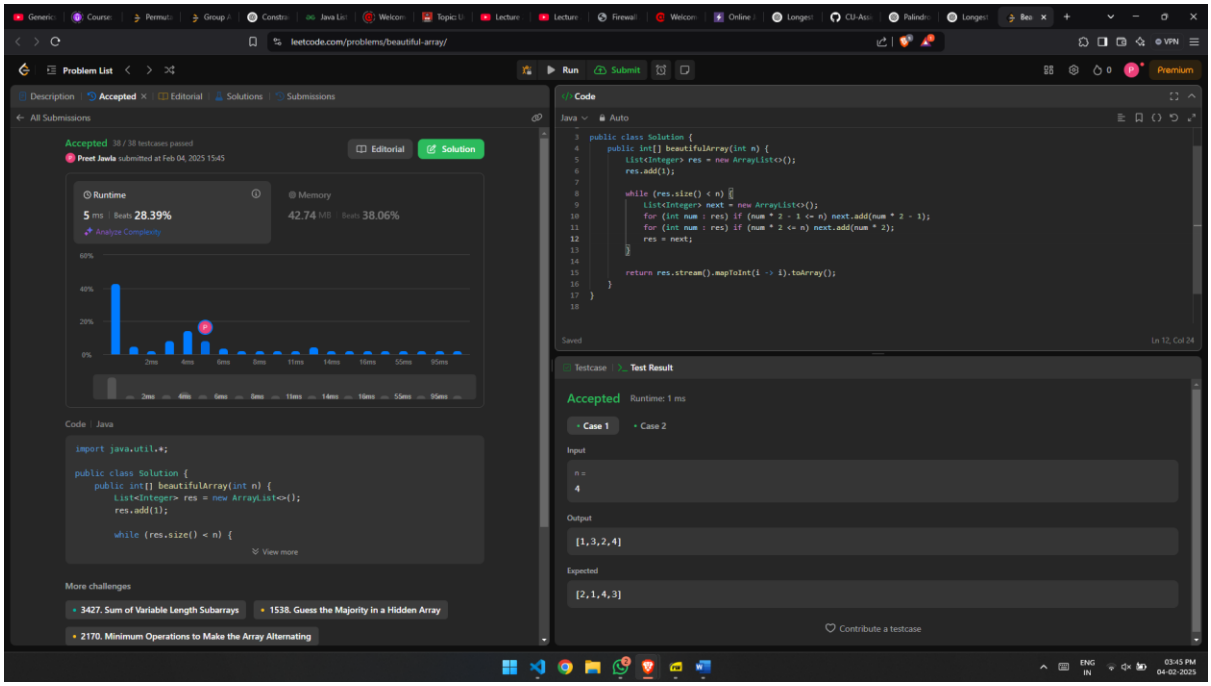



## 932.Beautiful Array

Code: import java.util.*;

```
public class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> res = new ArrayList<>();
        res.add(1);

        while (res.size() < n) {
            List<Integer> next = new ArrayList<>();
            for (int num : res) if (num * 2 - 1 <= n) next.add(num * 2 - 1);
```

```
            for (int num : res) if (num * 2 <= n) next.add(num * 2);
            res = next;
        }

        return res.stream().mapToInt(i -> i).toArray();
    }
}
```



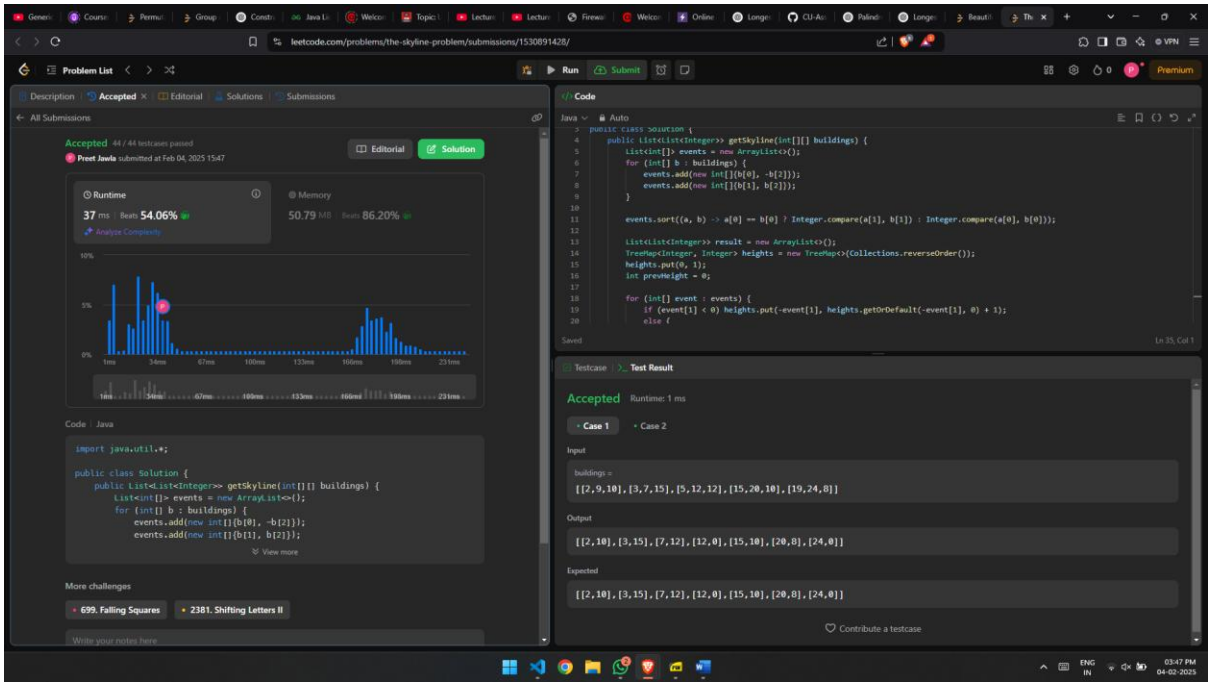

## 218.The Skyline Problem

Code: import java.util.*;

```
public class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> events = new ArrayList<>();
        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2]});
            events.add(new int[]{b[1], b[2]});
        }
```

```
    events.sort((a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));

    List<List<Integer>> result = new ArrayList<>();
    TreeMap<Integer, Integer> heights = new TreeMap<>(Collections.reverseOrder());
    heights.put(0, 1);
    int prevHeight = 0;

    for (int[] event : events) {
        if (event[1] < 0) heights.put(-event[1], heights.getOrDefault(-event[1], 0) + 1);
        else {
            if (heights.get(event[1]) == 1) heights.remove(event[1]);
            else heights.put(event[1], heights.get(event[1]) - 1);
        }

        int currentHeight = heights.firstKey();
        if (currentHeight != prevHeight) {
            result.add(Arrays.asList(event[0], currentHeight));
            prevHeight = currentHeight;
        }
    }

    return result;
}
```

}




SS:

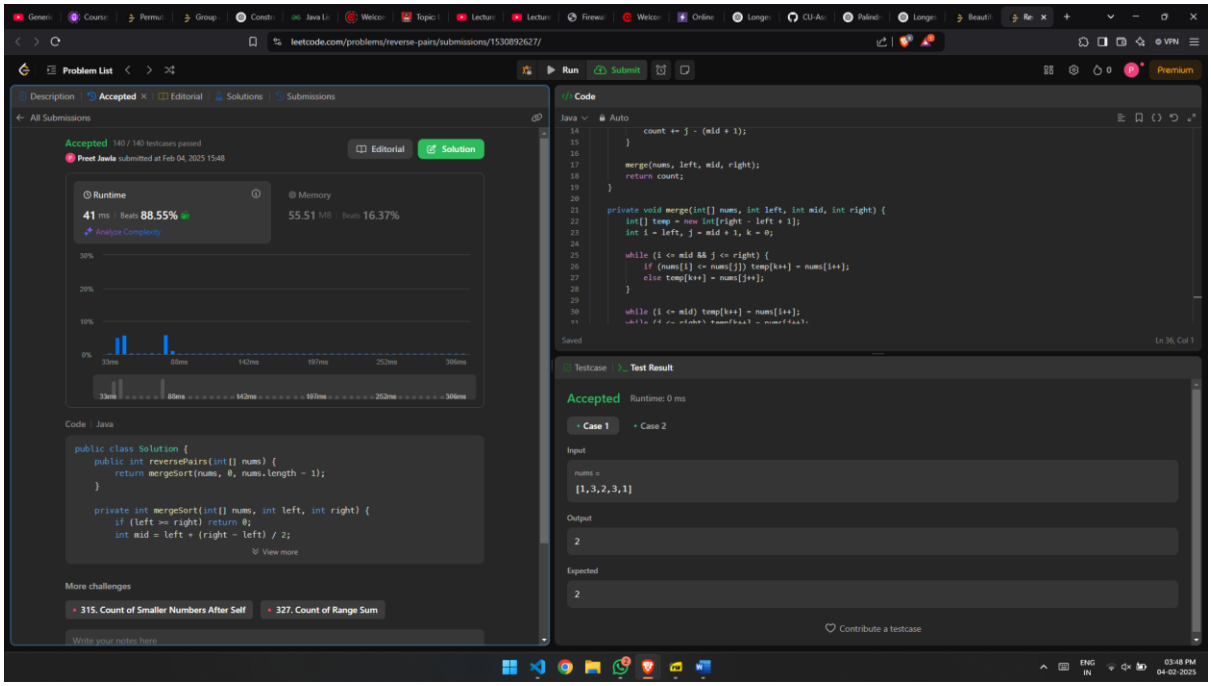
## 493.Reverse Pairs

Code: public 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) {
        if (nums[i] <= nums[j]) temp[k++] = nums[i++];
        else temp[k++] = nums[j++];
    }

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

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

}



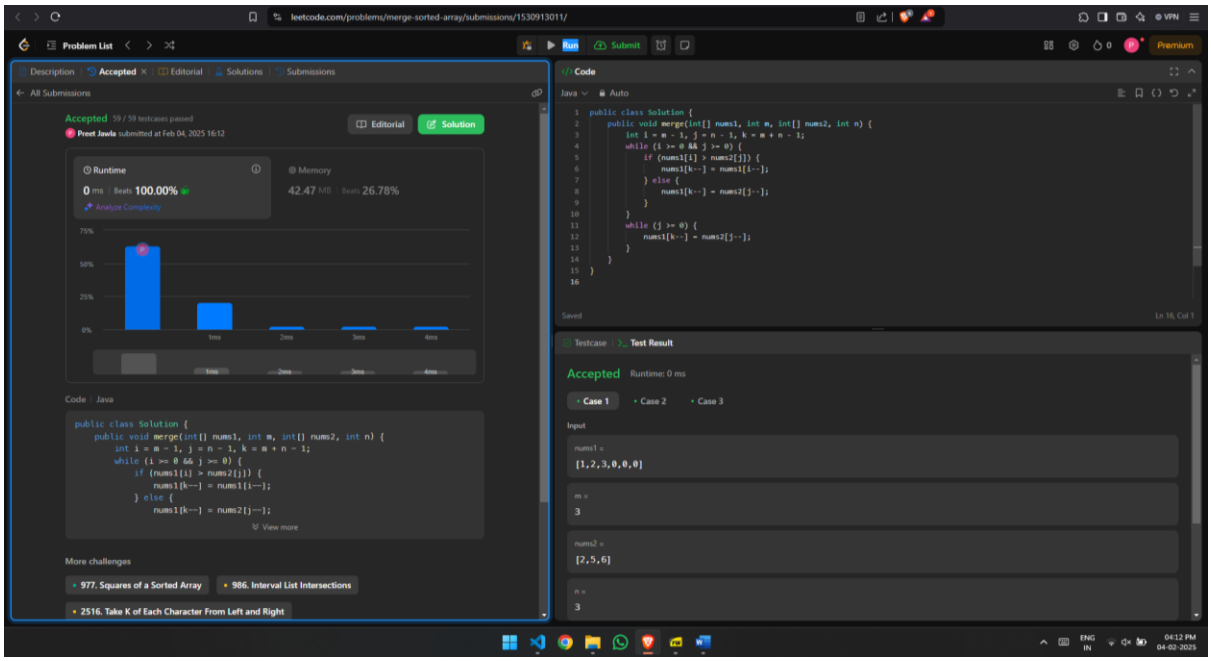

SS:

## 88.Merge Sorted Array

Code: 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--];
        }
    }
```

}

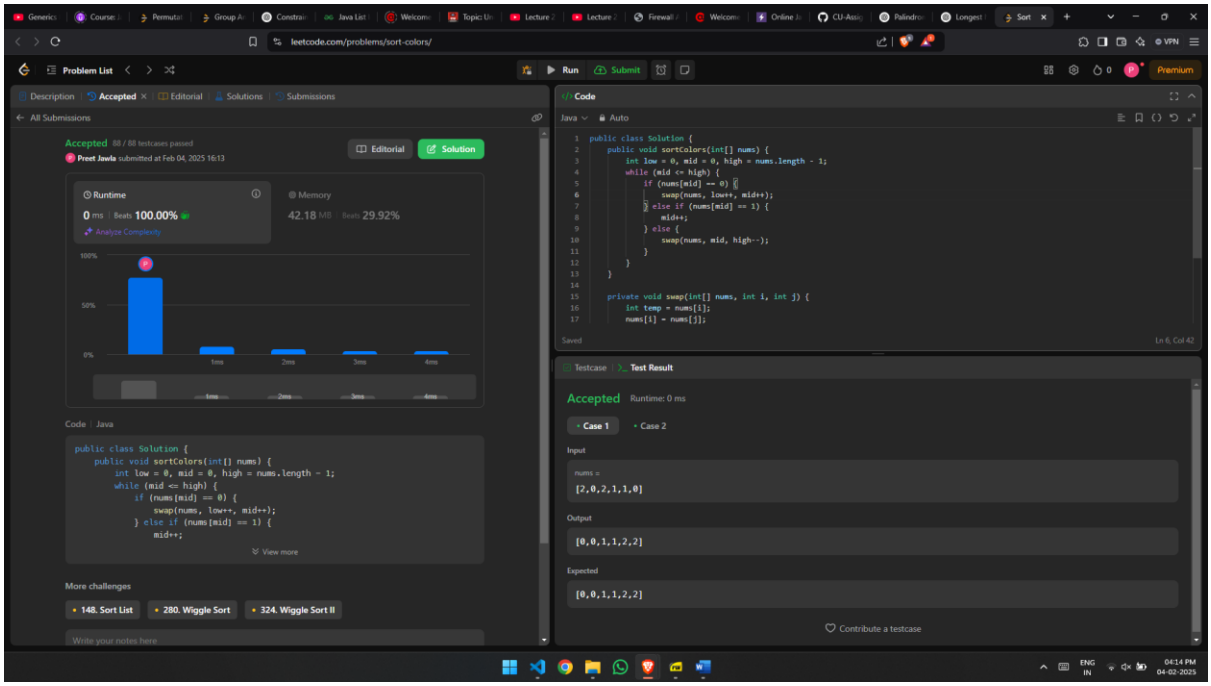



## 75.Sort Colors

Code: 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;
    }
}
```



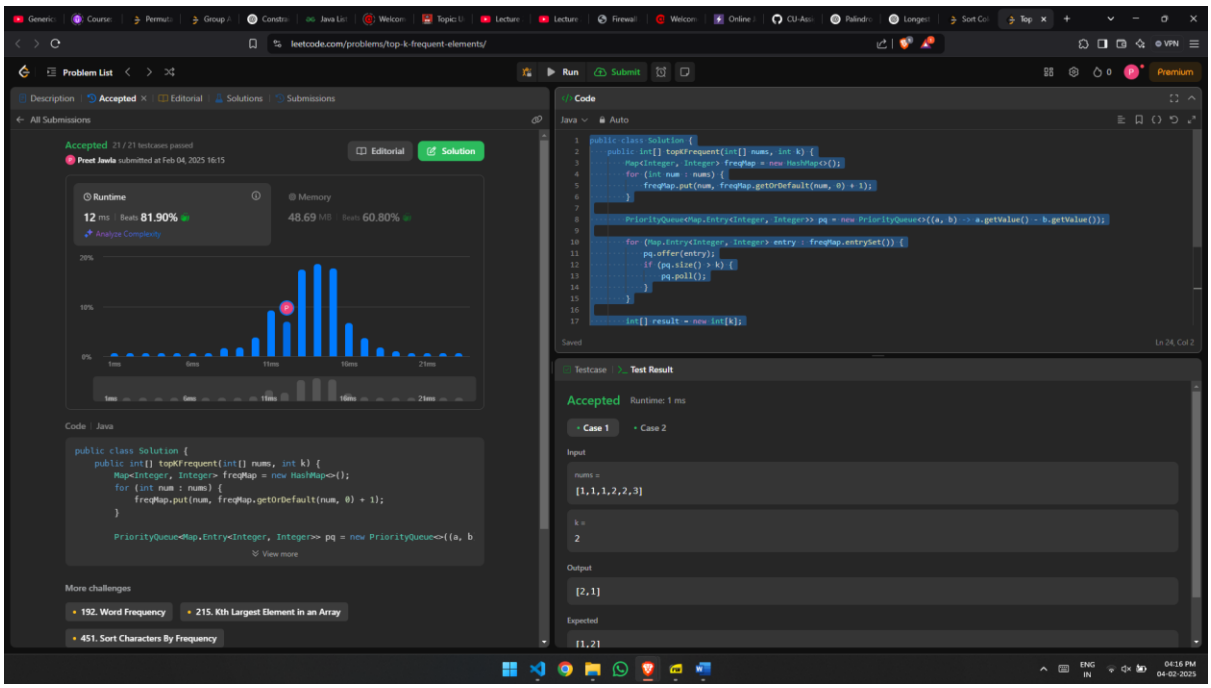

SS:

## 347.Top K Frequent Elements

Code: public 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<Map.Entry<Integer, Integer>> pq = new PriorityQueue<>((a, b) -> a.getValue() - b.getValue());

        for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {
```

```
            pq.offer(entry);
            if (pq.size() > k) {
                pq.poll();
            }
        }

        int[] result = new int[k];
        for (int i = 0; i < k; i++) {
            result[i] = pq.poll().getKey();
        }

        return result;
    }
}
```

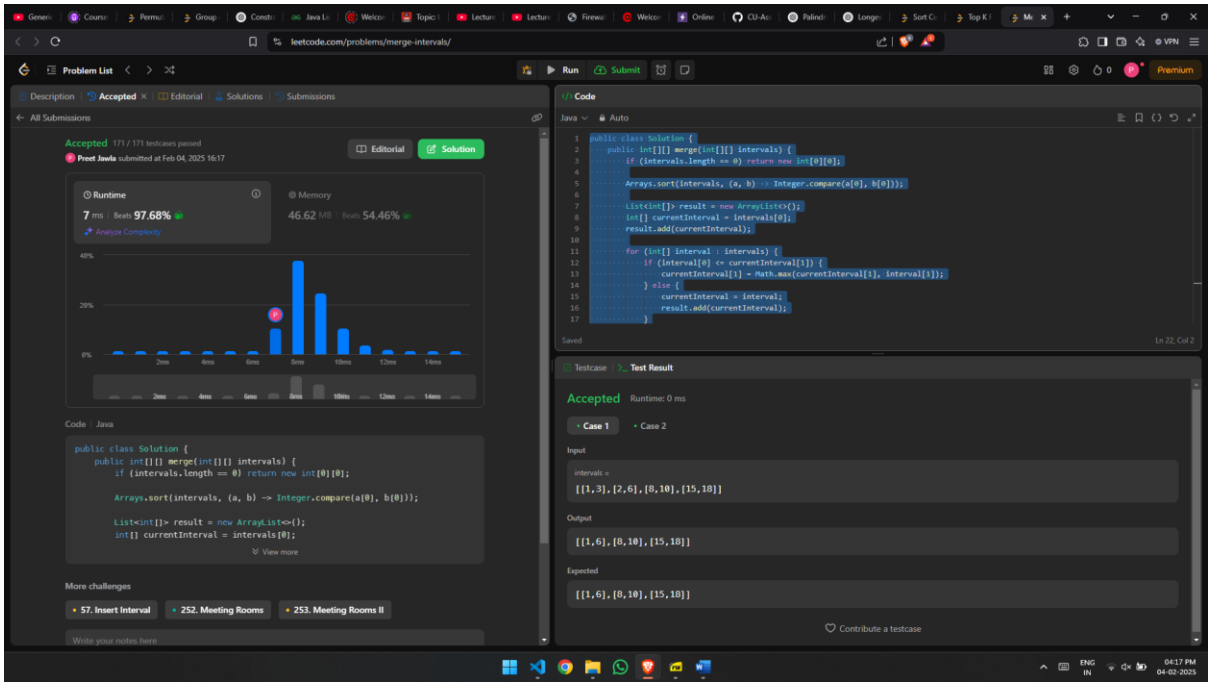



## 56.Merge Intervals

Code: 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()][]);
    }
}
```



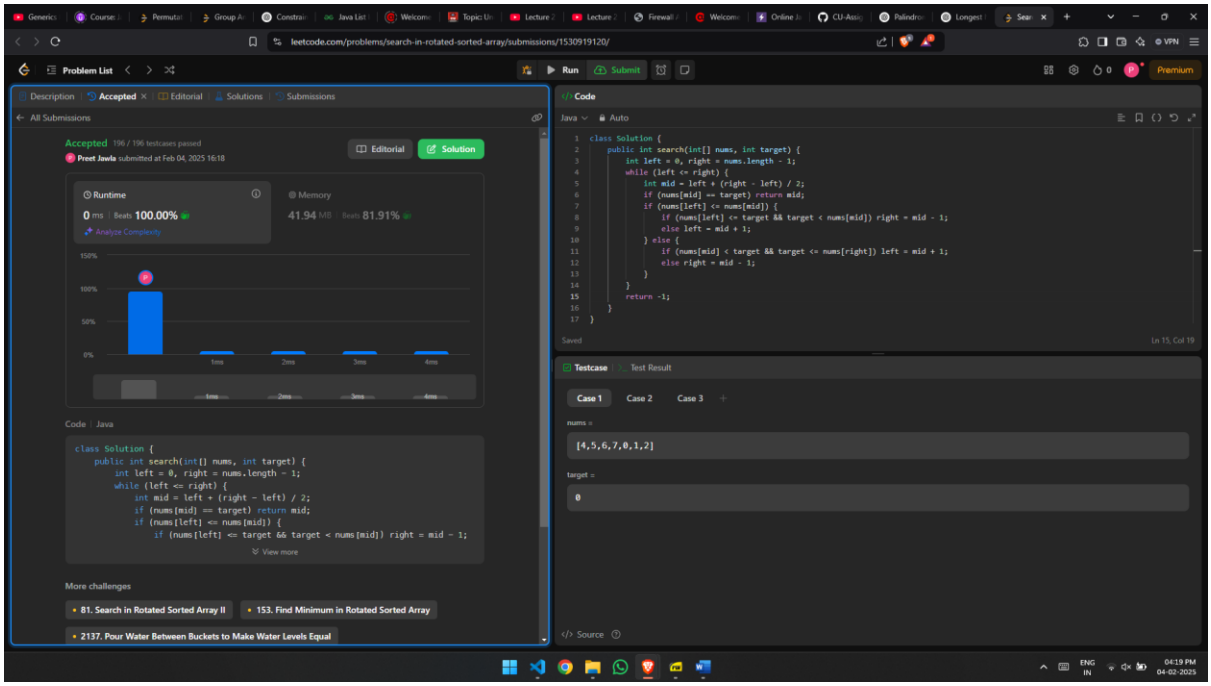

SS:

## 33.Search in Rotated Sorted Array

Code: class Solution {

```
    public int search(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) return mid;
            if (nums[left] <= nums[mid]) {
                if (nums[left] <= target && target < nums[mid]) right = mid - 1;
                else left = mid + 1;
            } else {
                if (nums[mid] < target && target <= nums[right]) left = mid + 1;
                else right = mid - 1;
            }
        }
        return -1;
```

```
    }
}
```

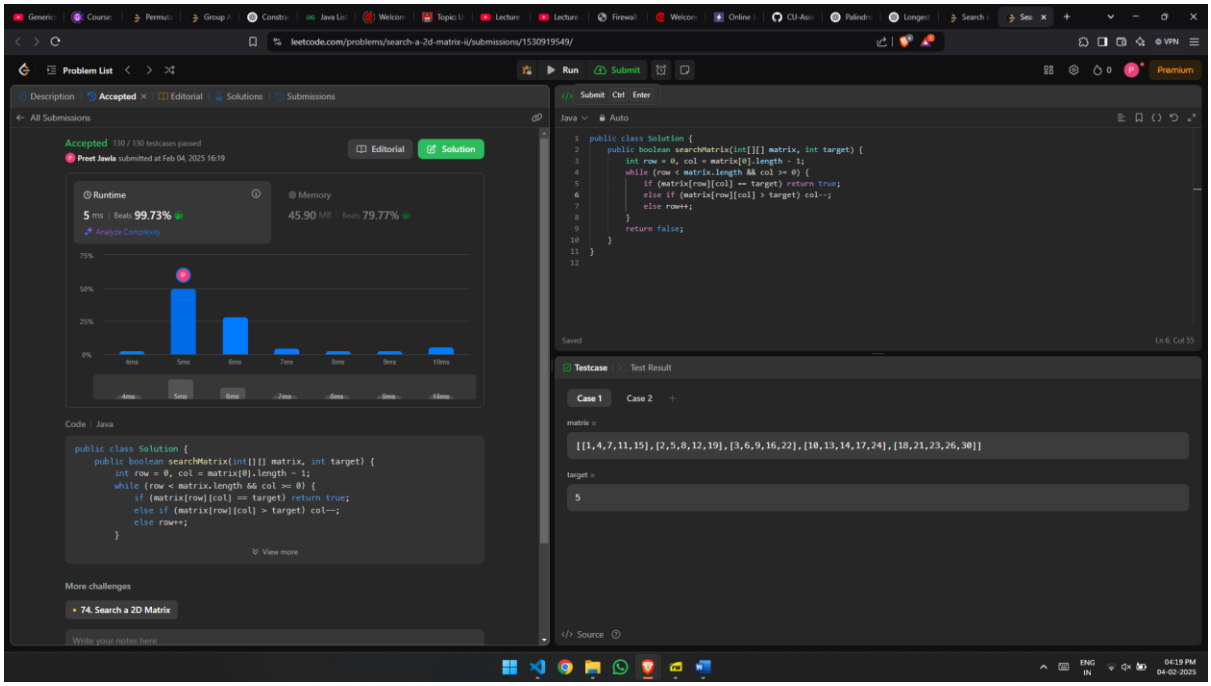



SS:

## 240.Search a 2D Matrix II

Code: public class Solution {

```
    public boolean searchMatrix(int[][] matrix, int target) {
        int row = 0, col = matrix[0].length - 1;
        while (row < matrix.length && col >= 0) {
            if (matrix[row][col] == target) return true;
            else if (matrix[row][col] > target) col--;
            else row++;
        }
        return false;
    }
}
```

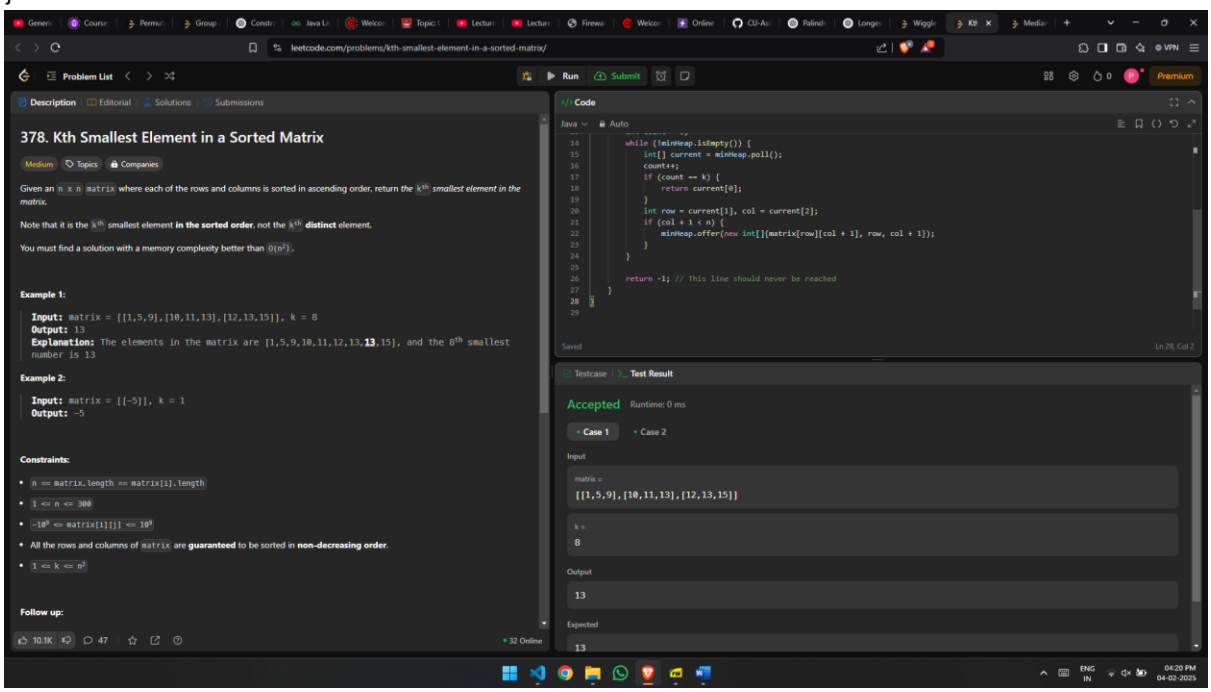



## 378.Kth Smallest Element in a Sorted Matrix

Code: import java.util.*;

```
public class Solution {
    public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        PriorityQueue<int[]> minHeap = new PriorityQueue<>((a, b) -> a[0] - b[0]);

        // Push the first element of each row into the heap
        for (int i = 0; i < n; i++) {
            minHeap.offer(new int[]{matrix[i][0], i, 0});
        }

        int count = 0;
        while (!minHeap.isEmpty()) {
            int[] current = minHeap.poll();
            count++;
```

```
            if (count == k) {
                return current[0];
            }
            int row = current[1], col = current[2];
            if (col + 1 < n) {
                minHeap.offer(new int[]{matrix[row][col + 1], row, col + 1});
            }
        }

        return -1; // This line should never be reached
    }
}
```




SS:

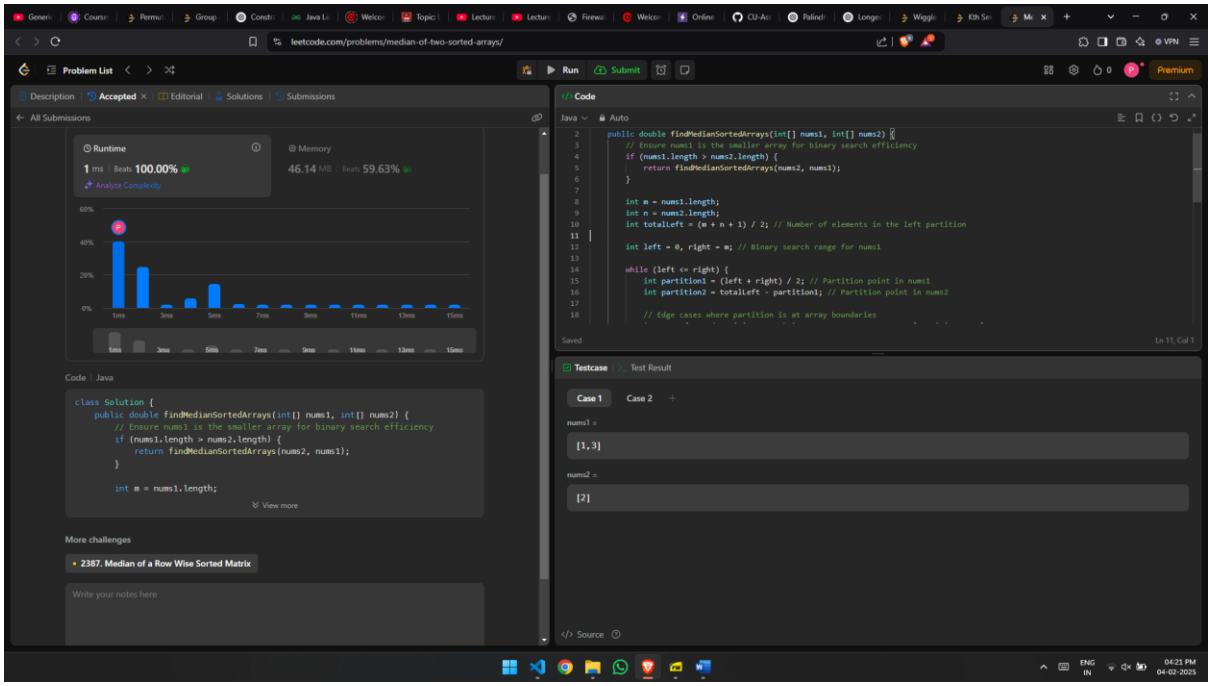
## 4.Median of Two Sorted Arrays

Code: class Solution {

```
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        // Ensure nums1 is the smaller array for binary search efficiency
```

```
    if (nums1.length > nums2.length) {
        return findMedianSortedArrays(nums2, nums1);
    }

    int m = nums1.length;
    int n = nums2.length;
    int totalLeft = (m + n + 1) / 2; // Number of elements in the left partition

    int left = 0, right = m; // Binary search range for nums1

    while (left <= right) {
        int partition1 = (left + right) / 2; // Partition point in nums1
        int partition2 = totalLeft - partition1; // Partition point in nums2

        // Edge cases where partition is at array boundaries
        int maxLeft1 = (partition1 == 0) ? Integer.MIN_VALUE : nums1[partition1 - 1];
        int minRight1 = (partition1 == m) ? Integer.MAX_VALUE : nums1[partition1];
        int maxLeft2 = (partition2 == 0) ? Integer.MIN_VALUE : nums2[partition2 - 1];
        int minRight2 = (partition2 == n) ? Integer.MAX_VALUE : nums2[partition2];
```

```
            // Check if the partition is correct
            if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {
                // If total length is odd, return max of left partition
                if ((m + n) % 2 == 1) {
                    return Math.max(maxLeft1, maxLeft2);
                }
                // If total length is even, return average of middle two elements
                return (Math.max(maxLeft1, maxLeft2) + Math.min(minRight1, minRight2)) / 2.0;
            } else if (maxLeft1 > minRight2) {
                // Move partition1 to the left
                right = partition1 - 1;
            } else {
                // Move partition1 to the right
                left = partition1 + 1;
            }
        }

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


leetcode.com/problems/median-of-two-sorted-arrays/
Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Runtime
1 ms | Beats 100.00%
Analyze Complexity
Memory
46.14 MB | Beats 59.63%
Code | Java
class Solution {
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
// Ensure nums1 is the smaller array for binary search efficiency
if (nums1.length > nums2.length) {
return findMedianSortedArrays(nums2, nums1);
}
int m = nums1.length;
View more
More challenges
2387. Median of a Row Wise Sorted Matrix
Write your notes here
Code
Java
Auto
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
// Ensure nums1 is the smaller array for binary search efficiency
if (nums1.length > nums2.length) {
return findMedianSortedArrays(nums2, nums1);
}
int m = nums1.length;
int n = nums2.length;
int totalLeft = (m + n + 1) / 2; // Number of elements in the left partition
int left = 0, right = m; // Binary search range for nums1
while (left <= right) {
int partition1 = (left + right) / 2; // Partition point in nums1
int partition2 = totalLeft - partition1; // Partition point in nums2
// Edge cases where partition is at array boundaries
Saved
Ln 11, Col 1
Testcase
Test Result
Case 1
Case 2
nums1 =
[1,3]
nums2 =
[2]
Source
ENG
IN
04:21 PM
04-02-2025