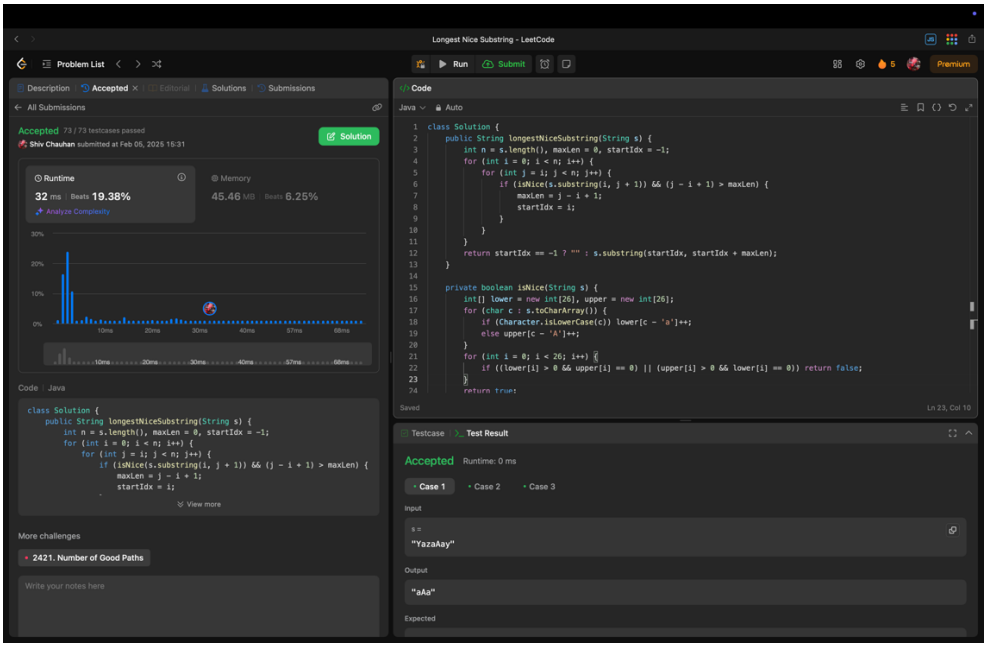
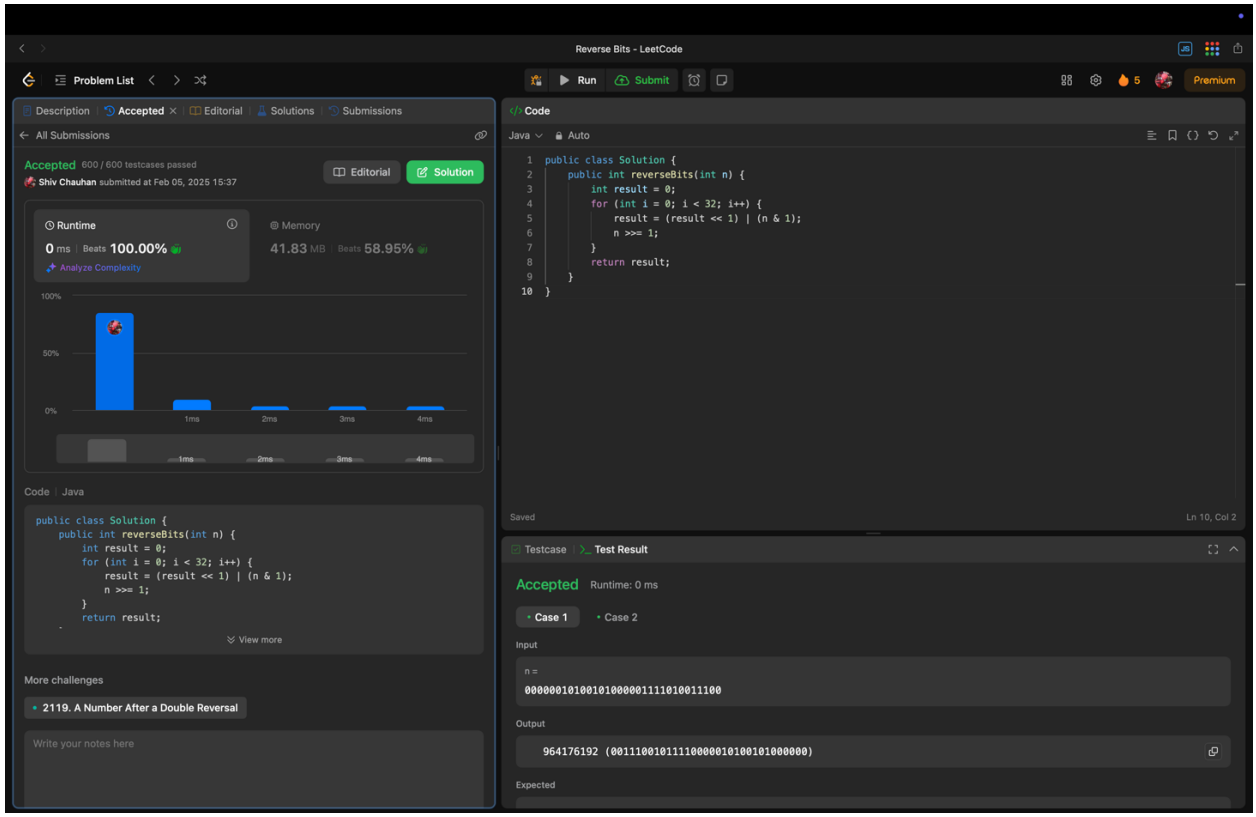
Longest Nice Substring

```
class Solution {
    public String longestNiceSubstring(String s) {
        int n = s.length(), maxLen = 0, startIdx = -1;
        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                if (isNice(s.substring(i, j + 1)) && (j - i + 1) > maxLen) {
                    maxLen = j - i + 1;
                    startIdx = i;
                }
            }
        }
        return startIdx == -1 ? "" : s.substring(startIdx, startIdx + maxLen);
    }

    private boolean isNice(String s) {
        int[] lower = new int[26], upper = new int[26];
        for (char c : s.toCharArray()) {
            if (Character.isLowerCase(c)) lower[c - 'a']++;
            else upper[c - 'A']++;
        }
        for (int i = 0; i < 26; i++) {
            if ((lower[i] > 0 && upper[i] == 0) || (upper[i] > 0 && lower[i] == 0)) return false;
        }
        return true;
    }
}
```



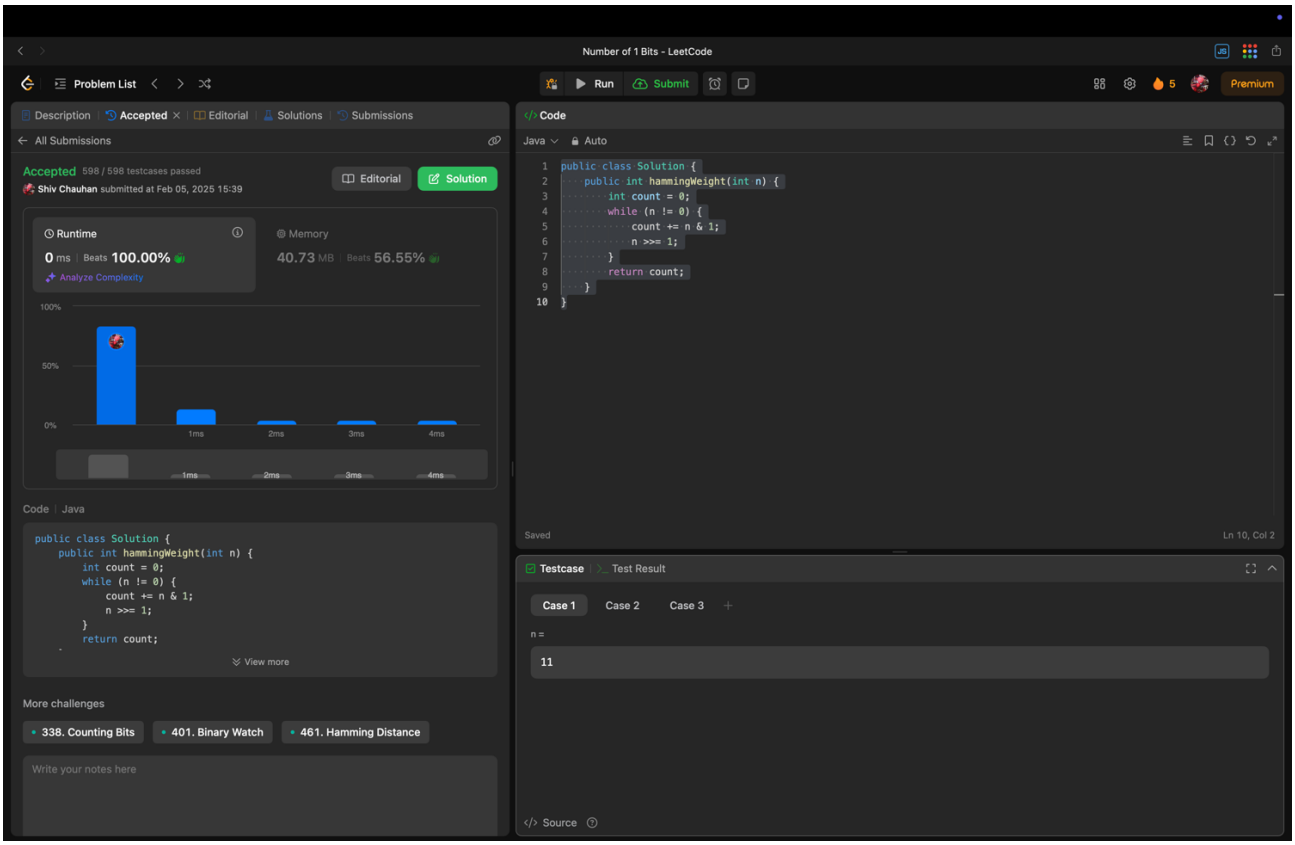
Reverse Bits

```
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;
    }
}
```



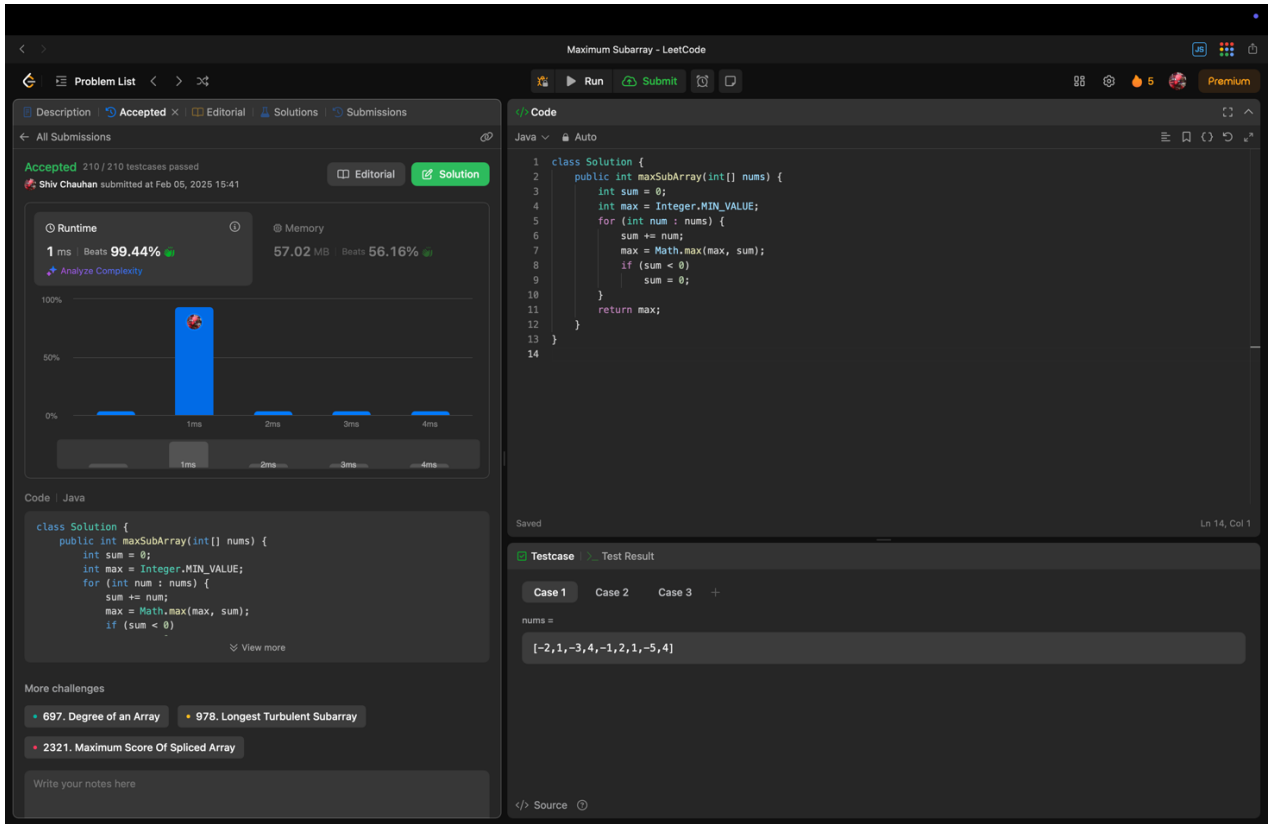
Number of 1 Bits

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



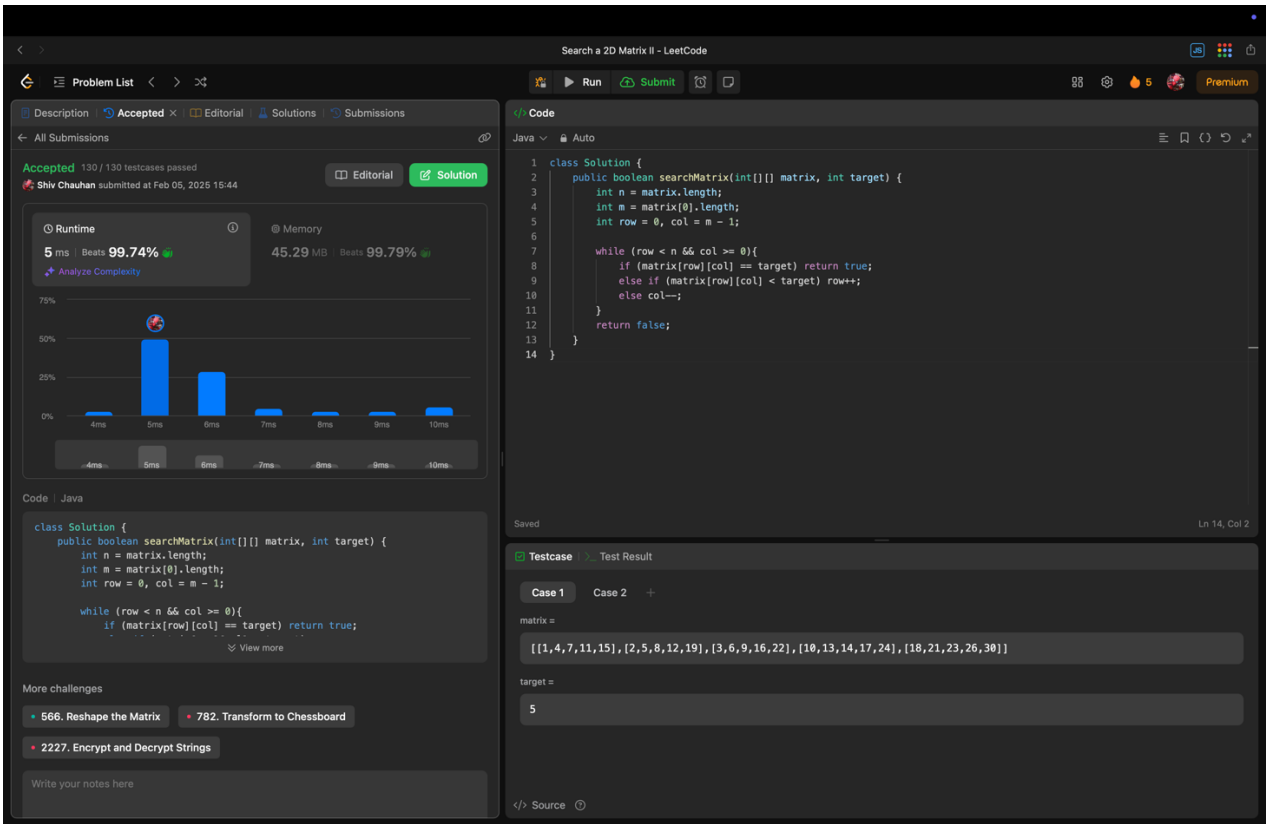
Maximum Subarray

```
class Solution {
    public int maxSubArray(int[] nums) {
        int sum = 0;
        int max = Integer.MIN_VALUE;
        for (int num : nums) {
            sum += num;
            max = Math.max(max, sum);
            if (sum < 0)
                sum = 0;
        }
        return max;
    }
}
```



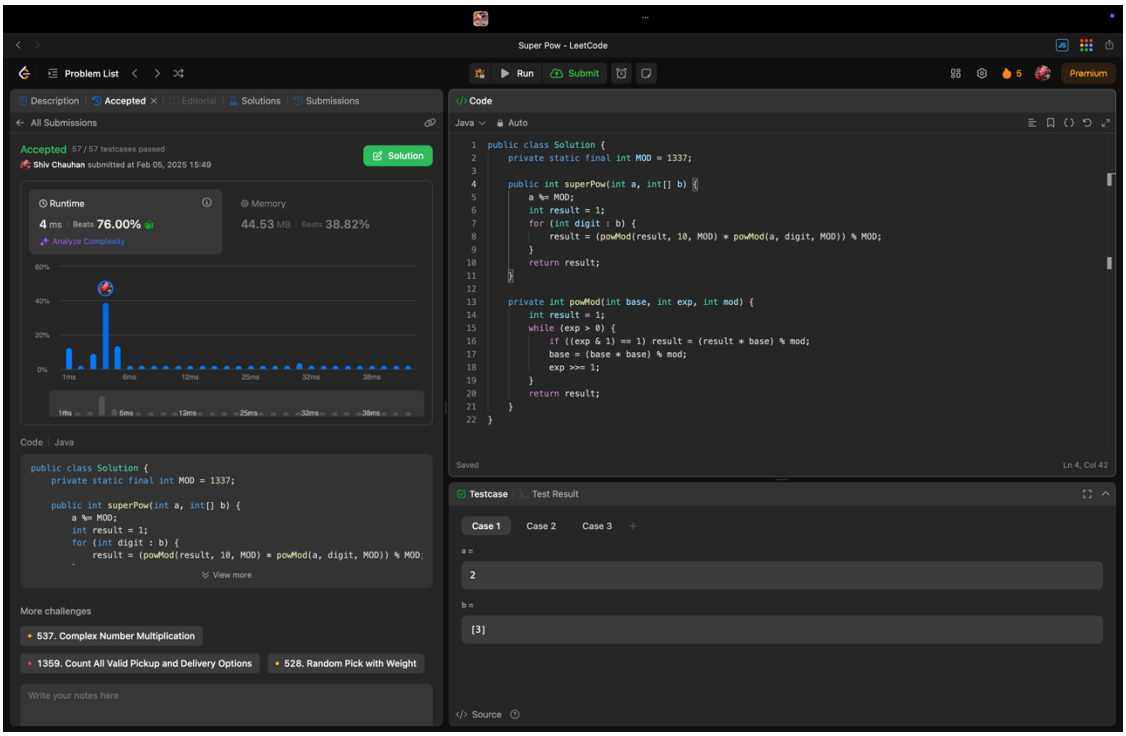
Search a 2D Matrix II

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int n = matrix.length;
        int m = matrix[0].length;
        int row = 0, col = m - 1;

        while (row < n && col >= 0){
            if (matrix[row][col] == target) return true;
            else if (matrix[row][col] < target) row++;
            else col--;
        }
        return false;
    }
}
```



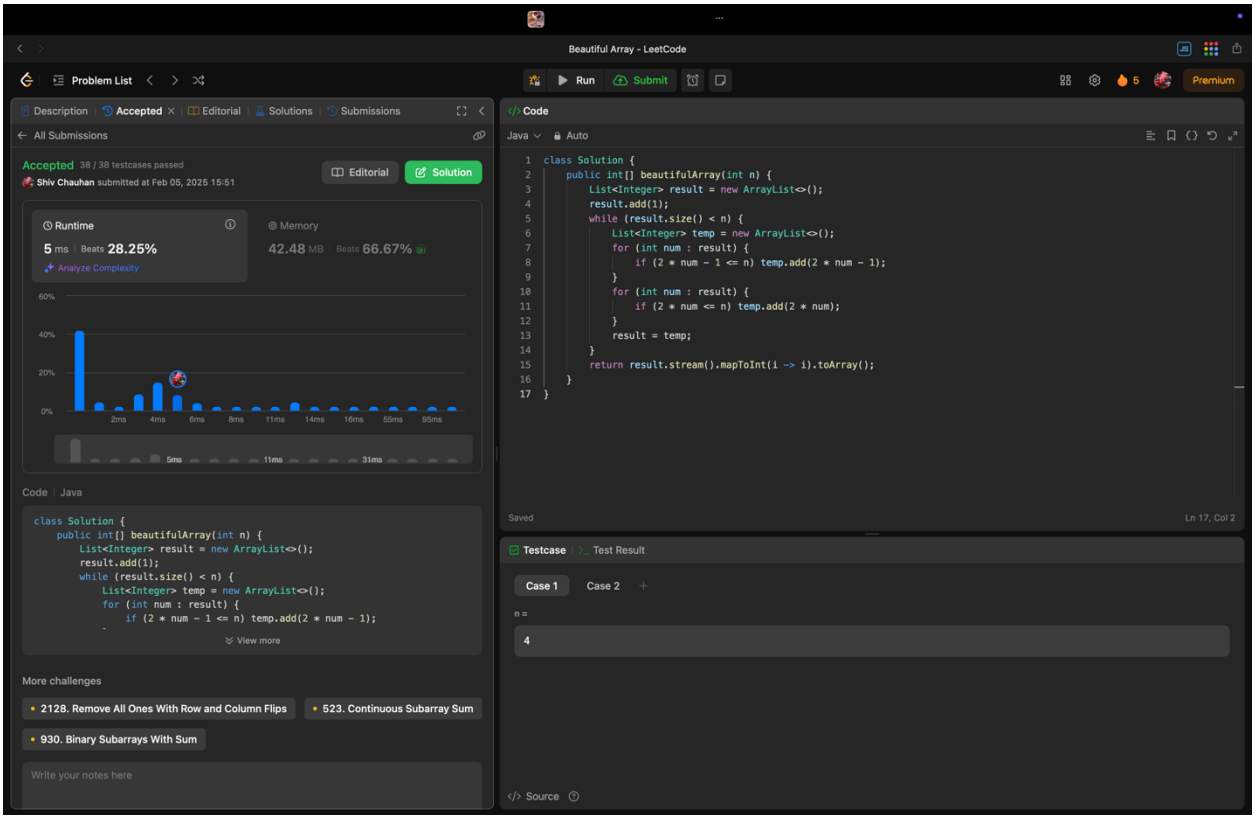
Super Pow

```
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, MOD) * powMod(a, digit, MOD)) % MOD;
        }
        return result;
    }

    private int powMod(int base, int exp, int mod) {
        int result = 1;
        while (exp > 0) {
            if ((exp & 1) == 1) result = (result * base) % mod;
            base = (base * base) % mod;
            exp >>= 1;
        }
        return result;
    }
}
```



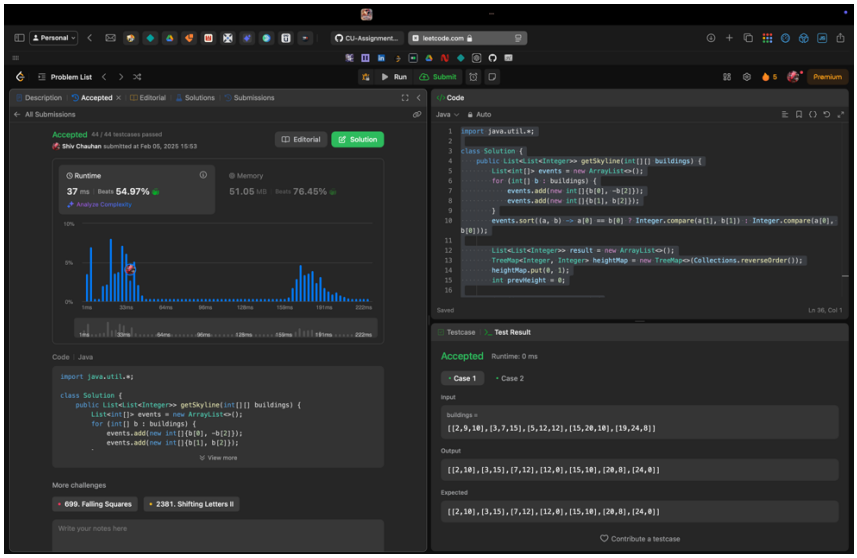
Beautiful Array

```
class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = new ArrayList<>();
        result.add(1);
        while (result.size() < n) {
            List<Integer> temp = new ArrayList<>();
            for (int num : result) {
                if (2 * num - 1 <= n) temp.add(2 * num - 1);
            }
            for (int num : result) {
                if (2 * num <= n) temp.add(2 * num);
            }
            result = temp;
        }
        return result.stream().mapToInt(i -> i).toArray();
    }
}
```



The Skyline Problem

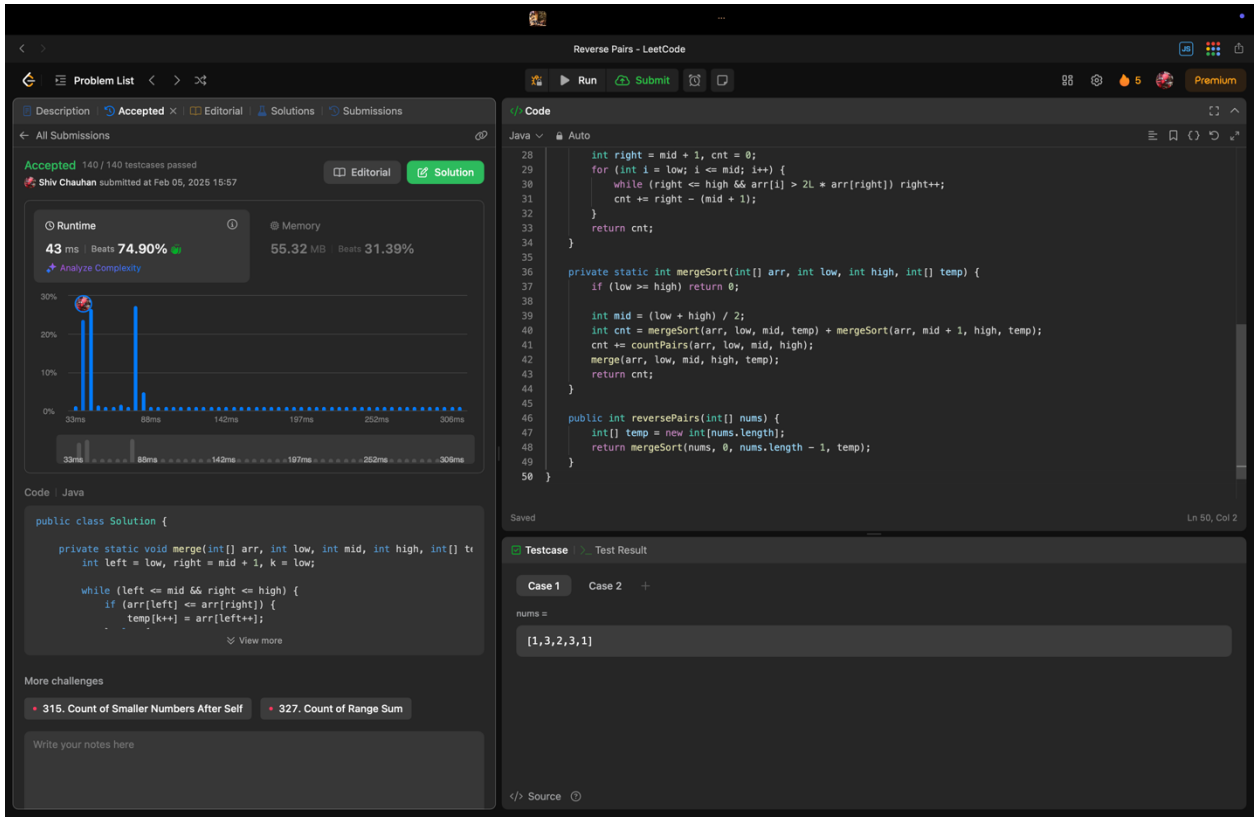
```
import java.util.*;

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> heightMap = new TreeMap<>(Collections.reverseOrder());
        heightMap.put(0, 1);
        int prevHeight = 0;

        for (int[] event : events) {
            if (event[1] < 0) {
                heightMap.put(-event[1], heightMap.getOrDefault(-event[1], 0) + 1);
            } else {
                if (heightMap.get(event[1]) == 1) {
                    heightMap.remove(event[1]);
                } else {
                    heightMap.put(event[1], heightMap.get(event[1]) - 1);
                }
            }
            int currHeight = heightMap.firstKey();
            if (currHeight != prevHeight) {
                result.add(Arrays.asList(event[0], currHeight));
                prevHeight = currHeight;
            }
        }
        return result;
    }
}
```



Reverse Pairs

```
public class Solution {

    private static void merge(int[] arr, int low, int mid, int high, int[] temp) {
        int left = low, right = mid + 1, k = low;

        while (left <= mid && right <= high) {
            if (arr[left] <= arr[right]) {
                temp[k++] = arr[left++];
            } else {
                temp[k++] = arr[right++];
            }
        }

        while (left <= mid) {
            temp[k++] = arr[left++];
        }

        while (right <= high) {
            temp[k++] = arr[right++];
        }

        for (int i = low; i <= high; i++) {
            arr[i] = temp[i];
        }
    }

    private static int countPairs(int[] arr, int low, int mid, int high) {
        int right = mid + 1, cnt = 0;
        for (int i = low; i <= mid; i++) {
            while (right <= high && arr[i] > 2L * arr[right]) right++;
            cnt += right - (mid + 1);
        }
        return cnt;
    }

    private static int mergeSort(int[] arr, int low, int high, int[] temp) {
```

```
        if (low >= high) return 0;

        int mid = (low + high) / 2;
        int cnt = mergeSort(arr, low, mid, temp) + mergeSort(arr, mid + 1, high, temp);
        cnt += countPairs(arr, low, mid, high);
        merge(arr, low, mid, high, temp);
        return cnt;
    }

    public int reversePairs(int[] nums) {
        int[] temp = new int[nums.length];
        return mergeSort(nums, 0, nums.length - 1, temp);
    }
}
```

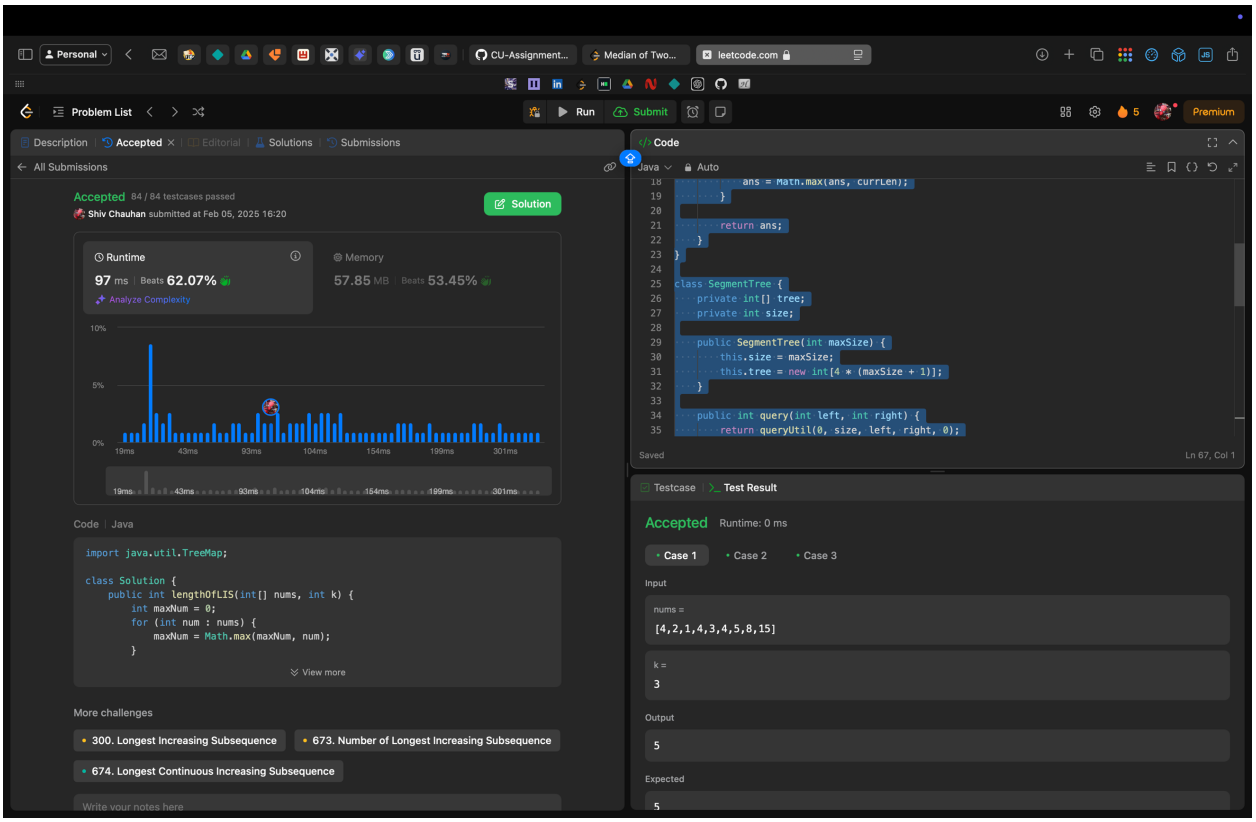


Longest Increasing Subsequence II

```
import java.util.TreeMap;


class Solution {
    public int lengthOfLIS(int[] nums, int k) {
        int maxNum = 0;
        for (int num : nums) {
            maxNum = Math.max(maxNum, num);
        }

        SegmentTree segTree = new SegmentTree(maxNum);
        int ans = 1;

        for (int num : nums) {
            int left = Math.max(0, num - k);
            int best = segTree.query(left, num - 1);
            int currLen = best + 1;
            segTree.update(num, currLen);
            ans = Math.max(ans, currLen);
        }

        return ans;
    }
}
```

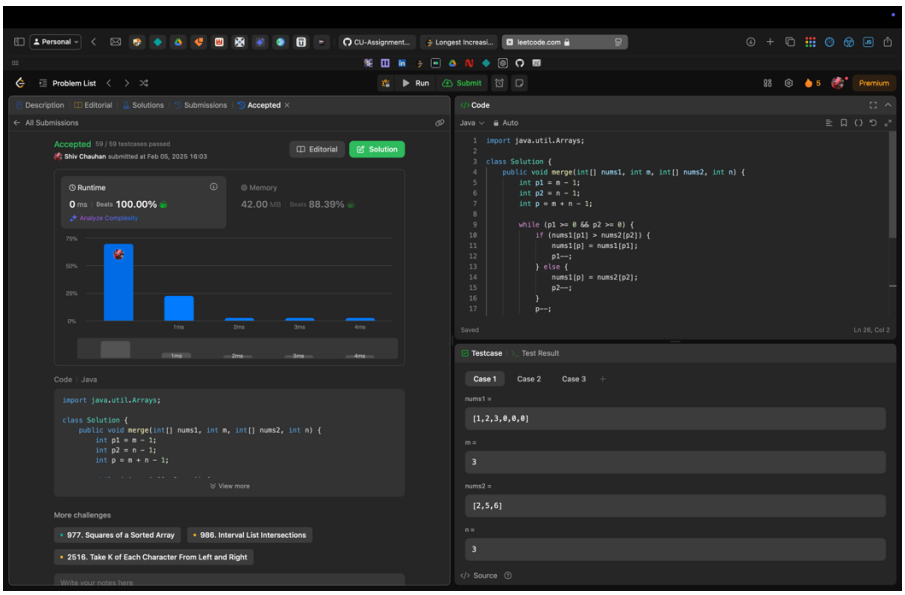
```
class SegmentTree {
    private int[] tree;
    private int size;

    public SegmentTree(int maxSize) {
        this.size = maxSize;
        this.tree = new int[4 * (maxSize + 1)];
    }

    public int query(int left, int right) {
        return queryUtil(0, size, left, right, 0);
    }

    private int queryUtil(int start, int end, int left, int right, int index) {
        if (left > end || right < start) return 0;
        if (left <= start && end <= right) return tree[index];

        int mid = (start + end) / 2;
        return Math.max(queryUtil(start, mid, left, right, 2 * index + 1),
                        queryUtil(mid + 1, end, left, right, 2 * index + 2));
    }

    public void update(int pos, int value) {
        updateUtil(0, size, pos, value, 0);
    }
```

```
    private void updateUtil(int start, int end, int pos, int value, int index) {
        if (start == end) {
            tree[index] = Math.max(tree[index], value);
            return;
        }

        int mid = (start + end) / 2;
        if (pos <= mid) {
            updateUtil(start, mid, pos, value, 2 * index + 1);
        } else {
            updateUtil(mid + 1, end, pos, value, 2 * index + 2);
        }

        tree[index] = Math.max(tree[2 * index + 1], tree[2 * index + 2]);
    }
}
```


Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 84 / 84 testcases passed
Shiv Chauhan submitted at Feb 05, 2025 16:20
Solution
Runtime
97 ms | Beats 62.07%
Analyze Complexity
Memory
57.85 MB | Beats 53.45%
Code | Java
import java.util.TreeMap;
class Solution {
public int lengthOfLIS(int[] nums, int k) {
int maxNum = 0;
for (int num : nums) {
maxNum = Math.max(maxNum, num);
}
View more
More challenges
300. Longest Increasing Subsequence
673. Number of Longest Increasing Subsequence
674. Longest Continuous Increasing Subsequence
Write your notes here
Code
Java
Auto
ans = Math.max(ans, currLen);
}
return ans;
}
}
class SegmentTree {
private int[] tree;
private int size;
public SegmentTree(int maxSize) {
this.size = maxSize;
this.tree = new int[4 * (maxSize + 1)];
}
public int query(int left, int right) {
return queryUtil(0, size, left, right, 0);
Saved
Ln 67, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[4,2,1,4,3,4,5,8,15]
k =
3
Output
5
Expected
5

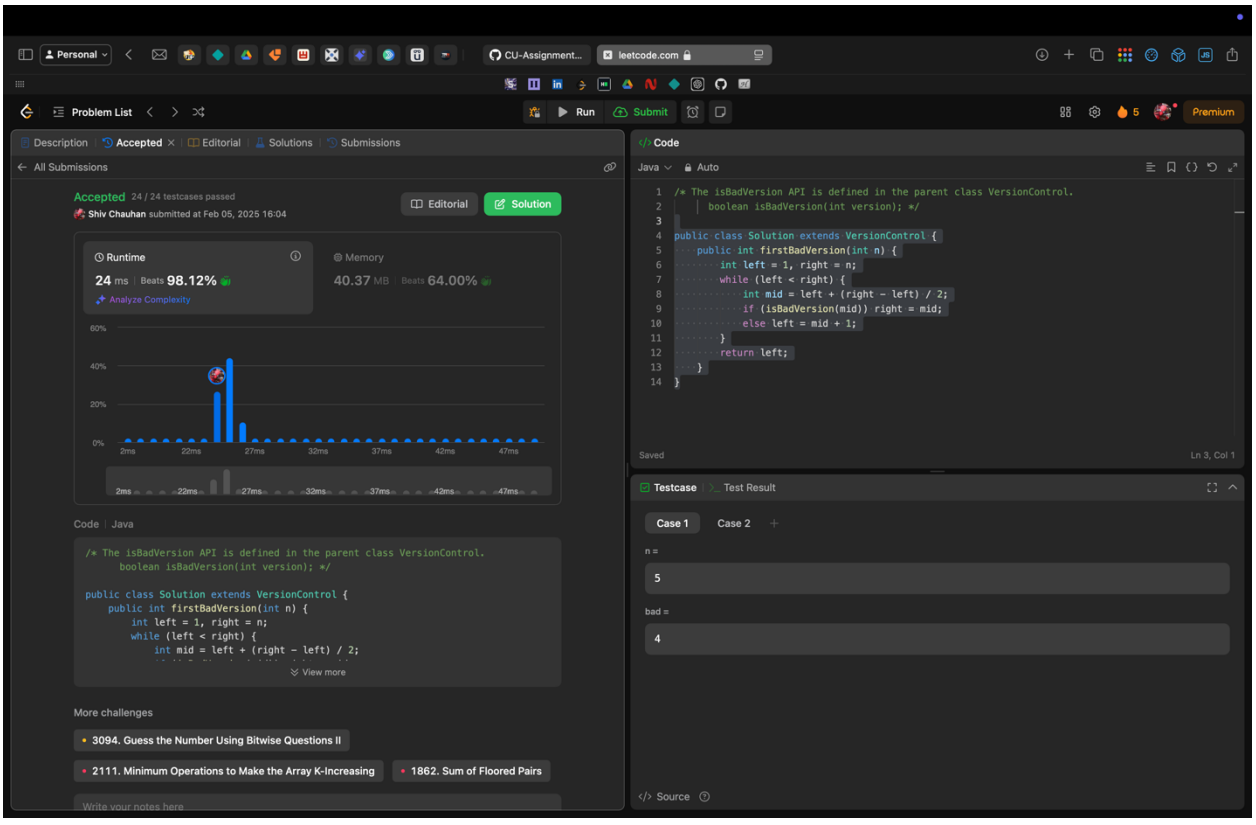
Merge Sorted Array

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int p1 = m - 1;
        int p2 = n - 1;
        int p = m + n - 1;
        while (p1 >= 0 && p2 >= 0) {
            if (nums1[p1] > nums2[p2]) {
                nums1[p] = nums1[p1];
                p1--;
            } else {
                nums1[p] = nums2[p2];
                p2--;
            }
            p--;
        }
        while (p2 >= 0) {
            nums1[p] = nums2[p2];
            p2--;
            p--;
        }
    }
}
```



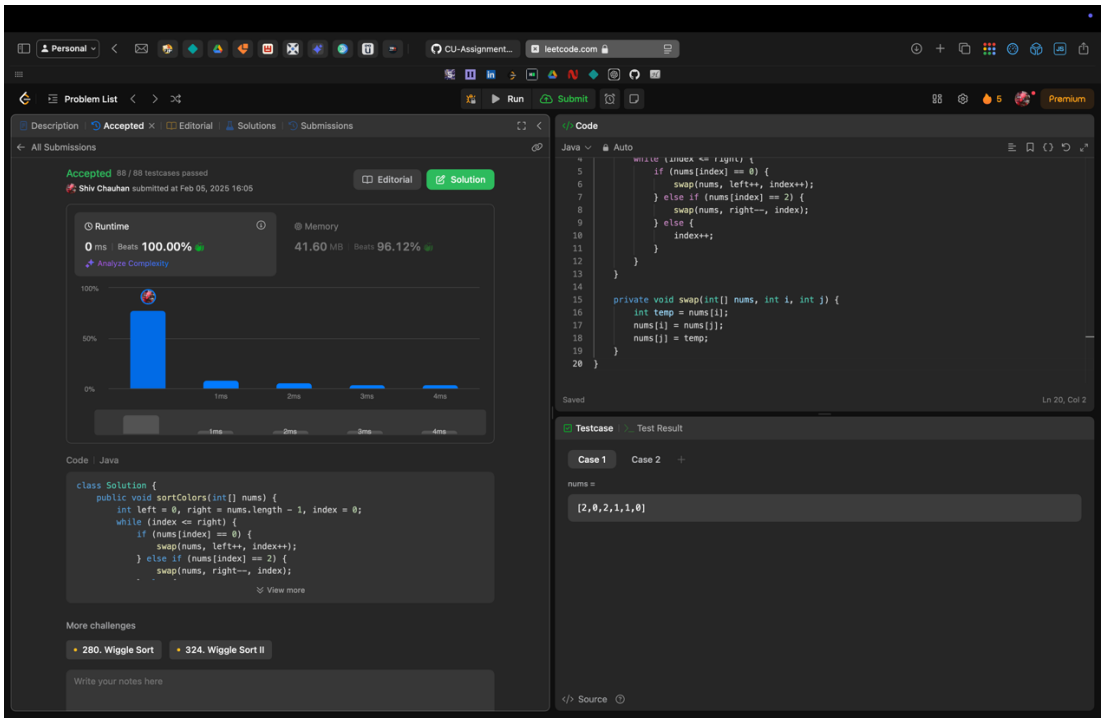
First Bad Version

```
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;
    }
}
```



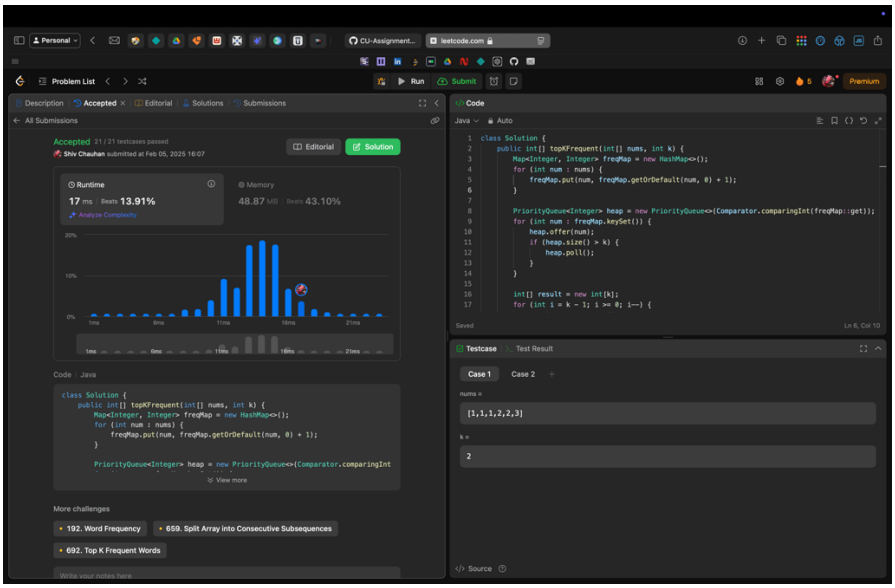
Sort Colors

```
class Solution {
    public void sortColors(int[] nums) {
        int left = 0, right = nums.length - 1, index = 0;
        while (index <= right) {
            if (nums[index] == 0) {
                swap(nums, left++, index++);
            } else if (nums[index] == 2) {
                swap(nums, right--, index);
            } else {
                index++;
            }
        }
    }

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



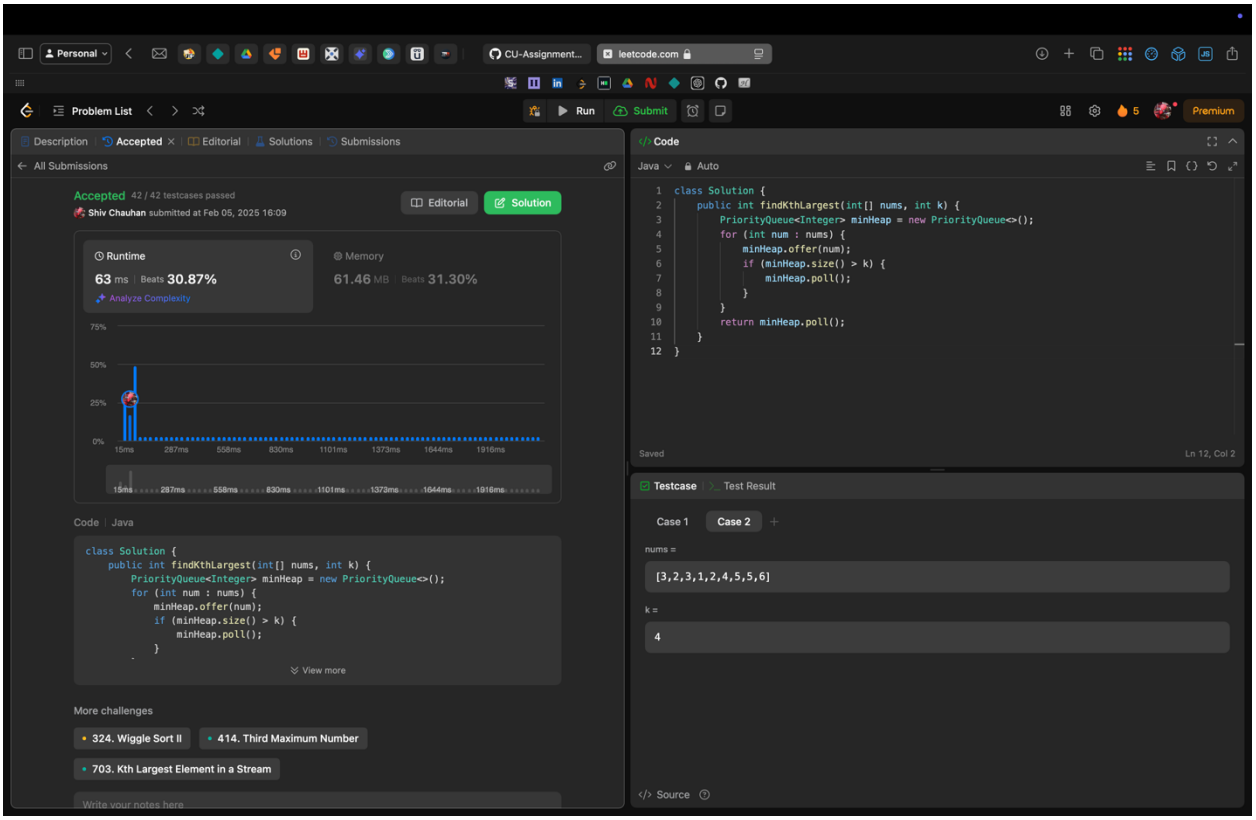
Top K Frequent Elements

```
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> heap = new
PriorityQueue<>(Comparator.comparingInt(freqMap::get));
        for (int num : freqMap.keySet()) {
            heap.offer(num);
            if (heap.size() > k) {
                heap.poll();
            }
        }

        int[] result = new int[k];
        for (int i = k - 1; i >= 0; i--) {
            result[i] = heap.poll();
        }
        return result;
    }
}
```



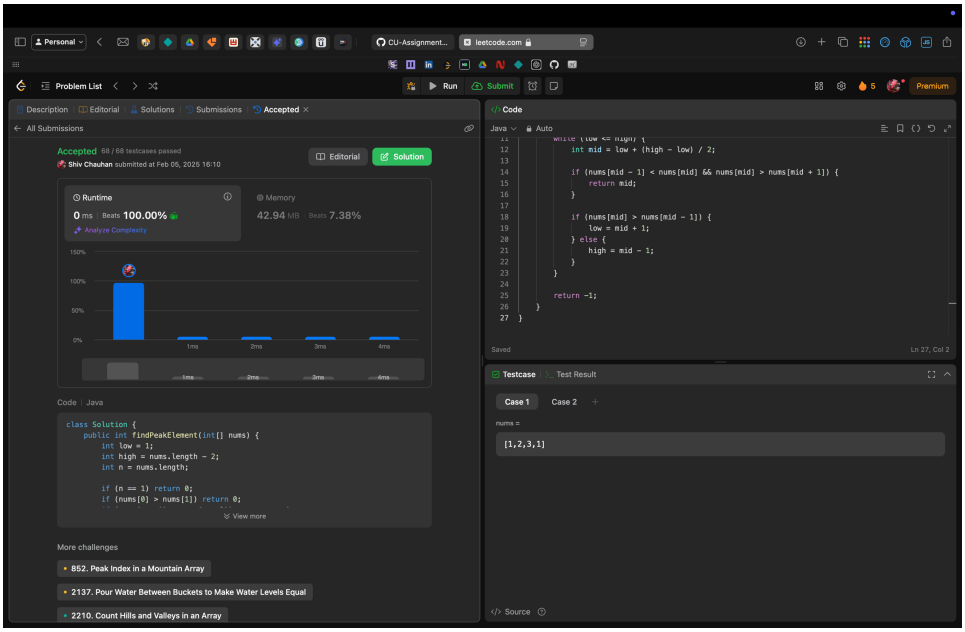
Kth Largest Element in the Array

```
class Solution {
    public int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        for (int num : nums) {
            minHeap.offer(num);
            if (minHeap.size() > k) {
                minHeap.poll();
            }
        }
        return minHeap.poll();
    }
}
```



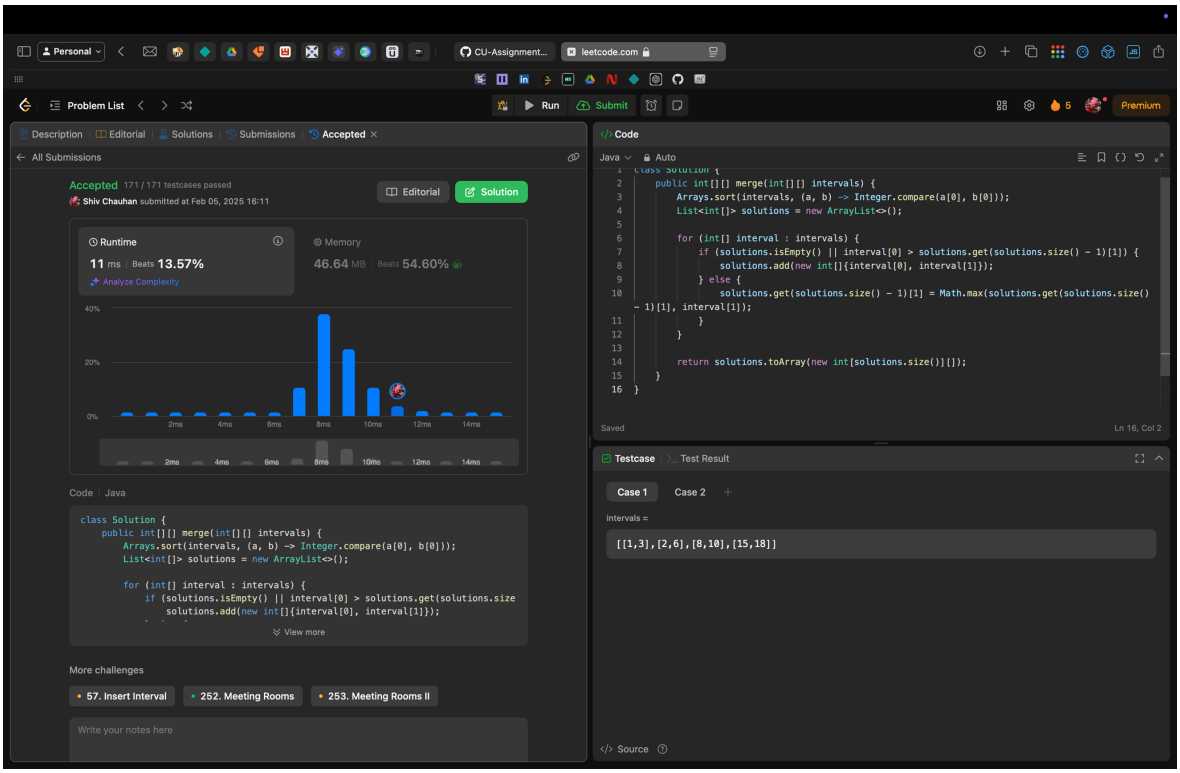
Find Peak Element

```
class Solution {
    public int findPeakElement(int[] nums) {
        int low = 1;
        int high = nums.length - 2;
        int n = nums.length;
        if (n == 1) return 0;
        if (nums[0] > nums[1]) return 0;
        if (nums[n - 1] > nums[n - 2]) return n - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (nums[mid - 1] < nums[mid] && nums[mid] > nums[mid + 1]) {
                return mid;
            }
            if (nums[mid] > nums[mid - 1]) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return -1;
    }
}
```



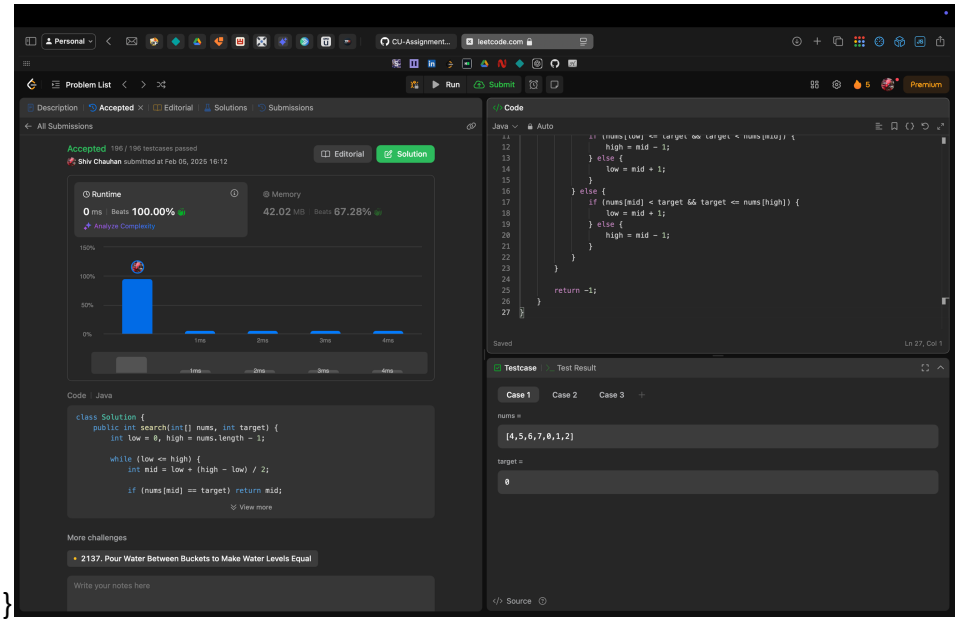
Merge Intervals

```
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> solutions = new ArrayList<>();

        for (int[] interval : intervals) {
            if (solutions.isEmpty() || interval[0] > solutions.get(solutions.size() - 1)[1]) {
                solutions.add(new int[]{interval[0], interval[1]});
            } else {
                solutions.get(solutions.size() - 1)[1] = Math.max(solutions.get(solutions.size() - 1)[1],
interval[1]);
            }
        }

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



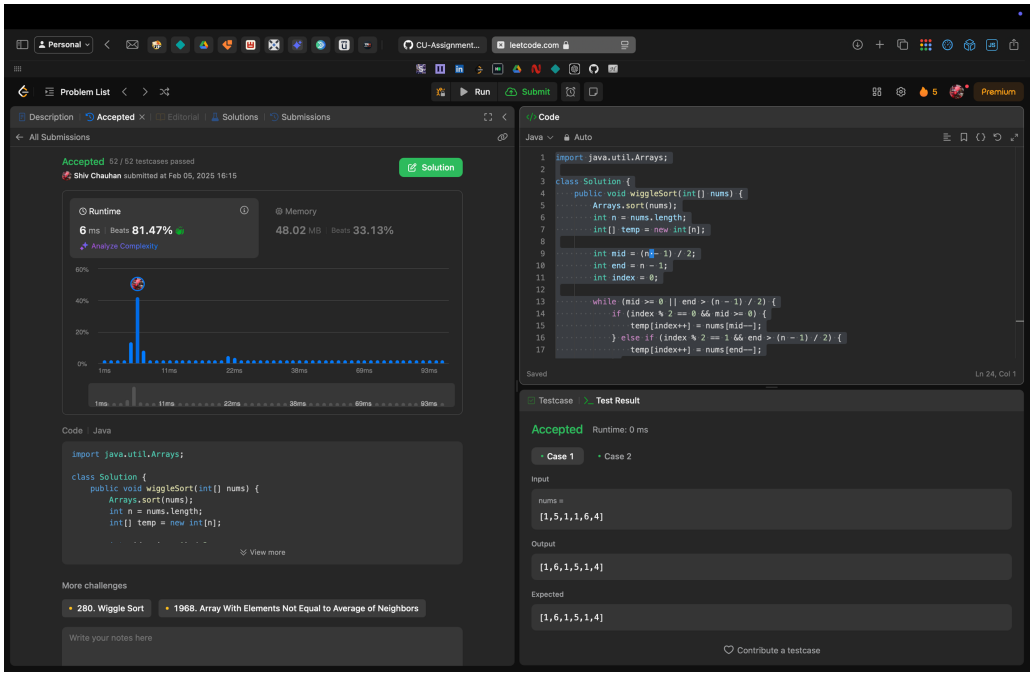
Search in a Sorted Rotated Array

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



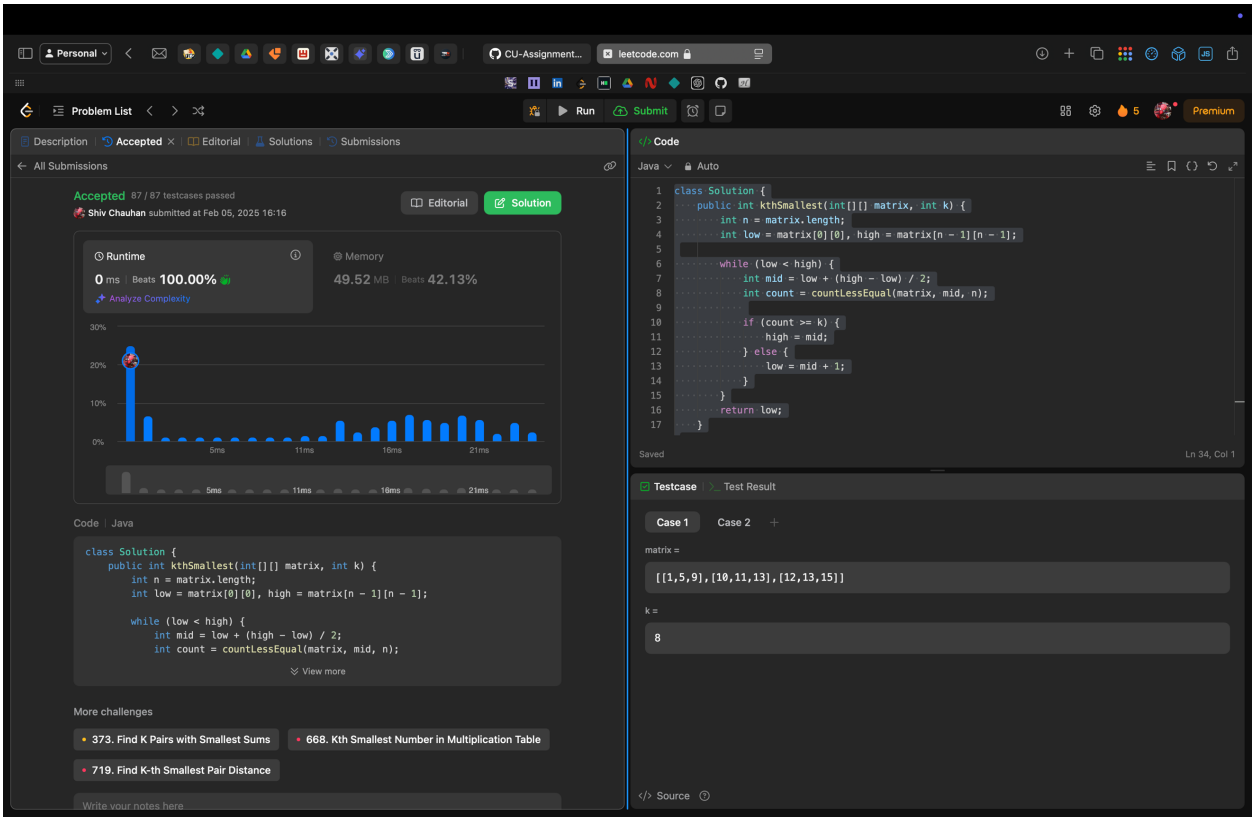
Wiggle Sort II

```
import java.util.Arrays;

class Solution {
    public void wiggleSort(int[] nums) {
        Arrays.sort(nums);
        int n = nums.length;
        int[] temp = new int[n];
        int mid = (n - 1) / 2;
        int end = n - 1;
        int index = 0;
        while (mid >= 0 || end > (n - 1) / 2) {
            if (index % 2 == 0 && mid >= 0) {
                temp[index++] = nums[mid--];
            } else if (index % 2 == 1 && end > (n - 1) / 2) {
                temp[index++] = nums[end--];
            }
        }
        System.arraycopy(temp, 0, nums, 0, n);
    }
}
```



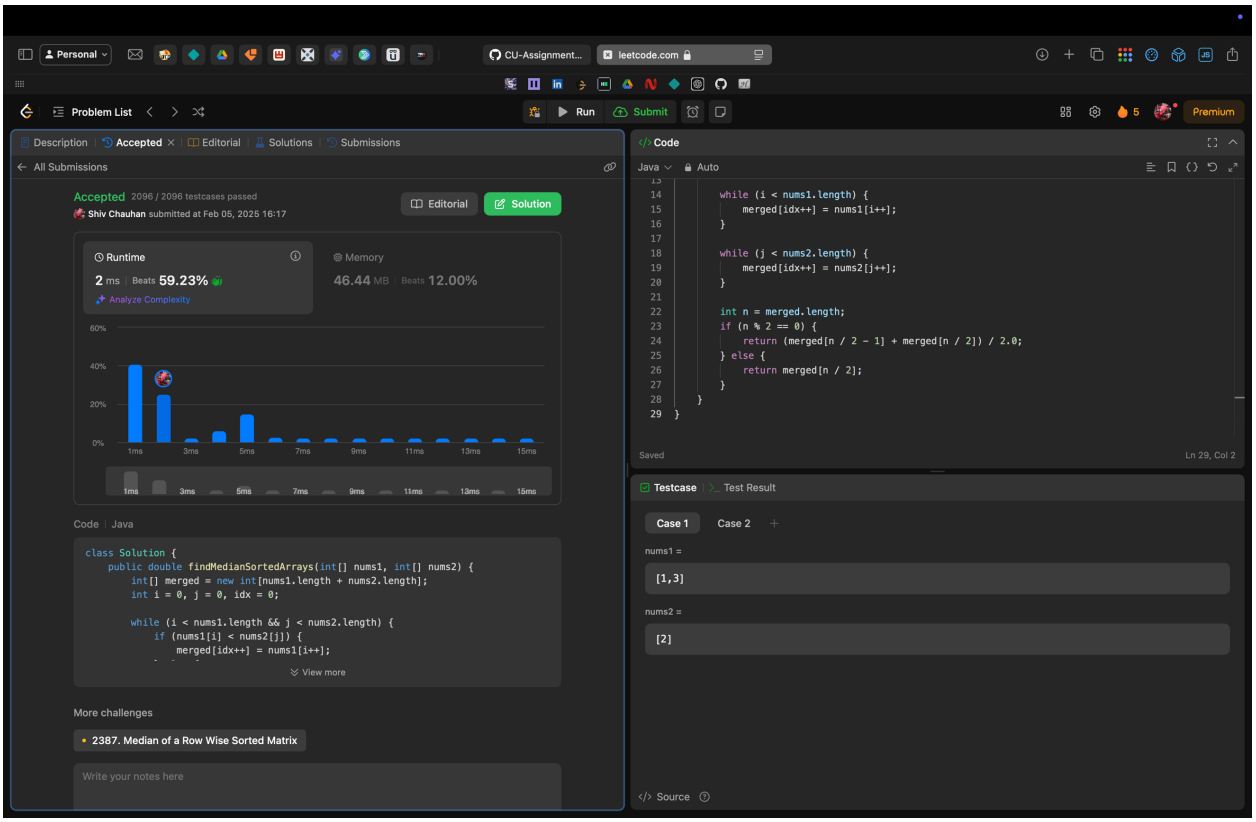
Kth Smallest Element in Sorted Matrix

```
class Solution {
    public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        int low = matrix[0][0], high = matrix[n - 1][n - 1];

        while (low < high) {
            int mid = low + (high - low) / 2;
            int count = countLessEqual(matrix, mid, n);

            if (count >= k) {
                high = mid;
            } else {
                low = mid + 1;
            }
        }
        return low;
    }

    private int countLessEqual(int[][] matrix, int mid, int n) {
        int count = 0;
        int row = n - 1, col = 0;

        while (row >= 0 && col < n) {
            if (matrix[row][col] <= mid) {
                count += row + 1;
                col++;
            } else {
                row--;
            }
        }
        return count;
    }
}
```


Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 87 / 87 testcases passed
Shiv Chauhan submitted at Feb 05, 2025 16:16
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
49.52 MB | Beats 42.13%
30%
20%
10%
0%
5ms
11ms
16ms
21ms
Code | Java
class Solution {
public int kthSmallest(int[][] matrix, int k) {
int n = matrix.length;
int low = matrix[0][0], high = matrix[n - 1][n - 1];
while (low < high) {
int mid = low + (high - low) / 2;
int count = countLessEqual(matrix, mid, n);
View more
More challenges
373. Find K Pairs with Smallest Sums
668. Kth Smallest Number in Multiplication Table
719. Find K-th Smallest Pair Distance
Write your notes here
Code
Java
Auto
class Solution {
public int kthSmallest(int[][] matrix, int k) {
int n = matrix.length;
int low = matrix[0][0], high = matrix[n - 1][n - 1];
while (low < high) {
int mid = low + (high - low) / 2;
int count = countLessEqual(matrix, mid, n);
if (count >= k) {
high = mid;
} else {
low = mid + 1;
}
}
return low;
}
Saved
Ln 34, Col 1
Testcase
Test Result
Case 1
Case 2
matrix =
[[1,5,9],[10,11,13],[12,13,15]]
k =
8
Source

Median of Two Sorted Arrays

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int[] merged = new int[nums1.length + nums2.length];
        int i = 0, j = 0, idx = 0;

        while (i < nums1.length && j < nums2.length) {
            if (nums1[i] < nums2[j]) {
                merged[idx++] = nums1[i++];
            } else {
                merged[idx++] = nums2[j++];
            }
        }

        while (i < nums1.length) {
            merged[idx++] = nums1[i++];
        }

        while (j < nums2.length) {
            merged[idx++] = nums2[j++];
        }

        int n = merged.length;
        if (n % 2 == 0) {
            return (merged[n / 2 - 1] + merged[n / 2]) / 2.0;
        } else {
            return merged[n / 2];
        }
    }
}
```


Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 2096 / 2096 testcases passed
Shiv Chauhan submitted at Feb 05, 2025 16:17
Editorial
Solution
Runtime
2 ms | Beats 59.23%
Analyze Complexity
Memory
46.44 MB | Beats 12.00%
Code | Java
class Solution {
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
int[] merged = new int[nums1.length + nums2.length];
int i = 0, j = 0, idx = 0;
while (i < nums1.length && j < nums2.length) {
if (nums1[i] < nums2[j]) {
merged[idx++] = nums1[i++];
View more
More challenges
2387. Median of a Row Wise Sorted Matrix
Write your notes here
Code
Java
Auto
while (i < nums1.length) {
merged[idx++] = nums1[i++];
}
while (j < nums2.length) {
merged[idx++] = nums2[j++];
}
int n = merged.length;
if (n % 2 == 0) {
return (merged[n / 2 - 1] + merged[n / 2]) / 2.0;
} else {
return merged[n / 2];
}
}
}
Saved
Ln 29, Col 2
Testcase
Test Result
Case 1
Case 2
nums1 =
[1,3]
nums2 =
[2]
Source