

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;
    }
}
```

The screenshot shows the LeetCode platform interface for the "Longest Nice Substring" problem. The top navigation bar includes 'Description', 'Accepted' (with a timestamp of Feb 05, 2028 15:31), 'Editorial', 'Solutions', and 'Submissions'. The main area has tabs for 'All Submissions' and 'Java'. The 'Code' tab displays the provided Java solution and the 'isNice' helper method. The 'Runtime' section shows a histogram with a peak at 32ms (Beats 19.38%) and a summary table with 30ms, 45.46 MB, and 6.25%. The 'Testcase' section shows an accepted submission with a runtime of 0 ms across three test cases: Case 1, Case 2, and Case 3. The input was "YazaAay" and the output was "aAa". The bottom section allows notes to be written.

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;  
    }  
}
```

The screenshot shows the LeetCode submission page for the "Reverse Bits" problem. The code has been accepted with 600/600 testcases passed by Shiv Chauhan at Feb 05, 2025 15:37. The runtime is 0 ms (Beats 100.00%) and memory usage is 41.83 MB (Beats 58.95%). The code editor shows the Java implementation. The test results section shows an accepted case with runtime 0 ms and input/output matching expected values.

```
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;  
    }  
}
```

The screenshot shows a LeetCode problem page for 'Number of 1 Bits'. The code has been submitted and accepted, with 598/598 testcases passed. The submission was made by Shiv Chauhan at Feb 05, 2025 15:39. The runtime is 0 ms (Beats 100.00%) and memory usage is 40.73 MB (Beats 56.55%). The complexity analysis shows O(n) time and O(1) space. The code editor displays the Java solution. The test result section shows Case 1 with input n=11 and output 11.

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;  
    }  
}
```

The screenshot shows the LeetCode platform interface for the "Maximum Subarray" problem. The code editor displays the Java solution provided above. On the left, a performance bar chart indicates the algorithm's efficiency: it beats 99.44% of submissions in runtime (1 ms) and 56.16% in memory (57.02 MB). Below the chart, the code is shown again with line numbers 1 through 14. The right side of the screen shows the "Testcase" tab, which contains a single test case with input [-2,1,-3,4,-1,2,1,-5,4] and no output displayed.

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;  
    }  
}
```

The screenshot shows the LeetCode platform interface for the 'Search a 2D Matrix II' problem. The top navigation bar includes 'Search a 2D Matrix II - LeetCode', 'Run', 'Submit', and other account-related icons.

The left sidebar displays the problem details: 'Accepted' (130 / 130 testcases passed), submitted by 'Shiv Chauhan' at Feb 05, 2025 16:44. It also shows 'Editorial' and 'Solutions' links.

The main area is divided into three sections:

- Runtime & Memory:** Shows performance metrics. Runtime: 5 ms (Beats 99.74%), Memory: 45.29 MB (Beats 99.79%). A chart below shows distribution across 4ms, 5ms, 6ms, 7ms, 8ms, 9ms, and 10ms.
- Code:** Displays the Java solution code.
- Testcase:** Shows the input matrix and target value for testing.

```
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;  
    }  
}
```

The 'Testcase' section shows:

- matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]
- target = 5

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;  
    }  
}
```

The screenshot shows the LeetCode platform interface for the "Super Pow" problem. The top navigation bar includes "Problem List", "Accepted", "Editorial", "Solutions", and "Submissions". The main area displays the Java solution code. On the left, there's a "Runtime" chart showing execution times for different test cases, with a peak at 4ms and a total average of 44.53 MB memory usage. Below the chart, the code editor shows the Java implementation. On the right, the "Testcase" tab is active, showing three test cases: Case 1 (a=2, b=[1]), Case 2 (a=1, b=[2, 1]), and Case 3 (a=1, b=[1, 0]). The status bar indicates "Ln 4, Col 42".

```
1 public class Solution {  
2     private static final int MOD = 1337;  
3  
4     public int superPow(int a, int[] b) {  
5         a %= MOD;  
6         int result = 1;  
7         for (int digit : b) {  
8             result = (powMod(result, 10, MOD) * powMod(a, digit, MOD)) % MOD;  
9         }  
10        return result;  
11    }  
12  
13    private int powMod(int base, int exp, int mod) {  
14        int result = 1;  
15        while (exp > 0) {  
16            if ((exp & 1) == 1) result = (result * base) % mod;  
17            base = (base * base) % mod;  
18            exp >= 1;  
19        }  
20        return result;  
21    }  
22 }
```

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 screenshot shows the LeetCode platform interface for the "Beautiful Array" problem. The top navigation bar includes links for Problem List, Submissions, and Solutions. The main area displays the accepted solution code in Java, its runtime and memory statistics, and a test case editor.

Runtime: 5 ms Beats 28.25%
Memory: 42.48 MB Beats 66.67%

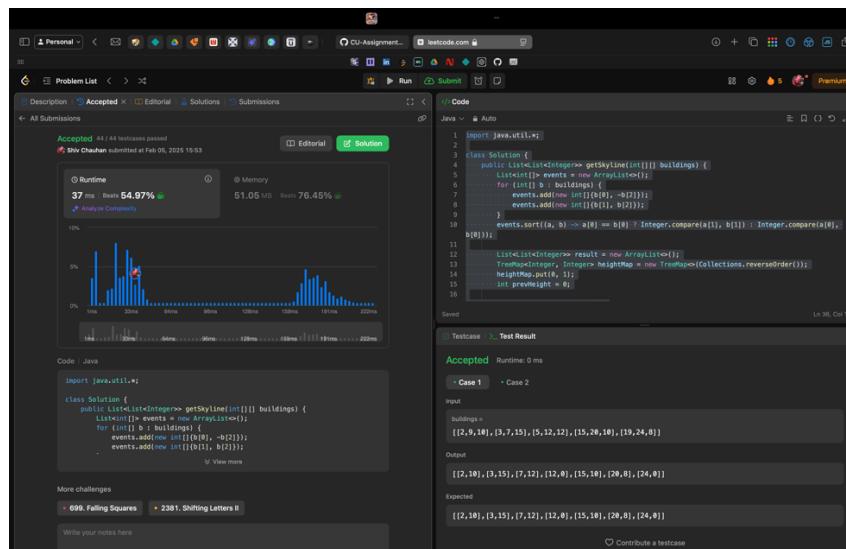
Code Editor:

```
1 class Solution {
2     public int[] beautifulArray(int n) {
3         List<Integer> result = new ArrayList<>();
4         result.add(1);
5         while (result.size() < n) {
6             List<Integer> temp = new ArrayList<>();
7             for (int num : result) {
8                 if (2 * num - 1 <= n) temp.add(2 * num - 1);
9             }
10            for (int num : result) {
11                if (2 * num <= n) temp.add(2 * num);
12            }
13            result = temp;
14        }
15        return result.stream().mapToInt(i -> i).toArray();
16    }
17 }
```

Testcase: Case 1: n = 4

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);
}
}

```

The screenshot shows the LeetCode platform interface for the "Reverse Pairs" problem. The top navigation bar includes links for Problem List, Accepted, Editorial, Solutions, and Submissions. The main area displays the problem description, runtime and memory statistics, and the Java solution code.

Runtime: 43 ms | Beats 74.90% | 55.32 MB | Beats 31.39%

Code Editor:

```

28     int right = mid + 1, cnt = 0;
29     for (int i = low; i <= mid; i++) {
30         while (right <= high && arr[i] > 2L * arr[right]) right++;
31         cnt += right - (mid + 1);
32     }
33     return cnt;
34 }
35
36 private static int mergeSort(int[] arr, int low, int high, int[] temp) {
37     if (low >= high) return 0;
38
39     int mid = (low + high) / 2;
40     int cnt = mergeSort(arr, low, mid, temp) + mergeSort(arr, mid + 1, high, temp);
41     cnt += countPairs(arr, low, mid, high);
42     merge(arr, low, mid, high, temp);
43     return cnt;
44 }
45
46 public int reversePairs(int[] nums) {
47     int[] temp = new int[nums.length];
48     return mergeSort(nums, 0, nums.length - 1, temp);
49 }
50 }

```

Testcase: Case 1: nums = [1,3,2,3,1]

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]);  
}
```

The screenshot shows a LeetCode submission page for the problem "Median of Two Sorted Arrays".

Accepted 84 / 84 testcases passed
Shiv Chauhan submitted at Feb 05, 2025 16:20

Runtime: 97 ms Beats 62.07%
Memory: 57.85 MB Beats 53.45%

Code (Java)

```
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);
}
```

Testcase | Test Result

Accepted Runtime: 0 ms

Input: nums = [4,2,1,4,3,4,5,8,15]
k = 3

Output: 5
Expected: 5

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);
        }
    }
}
```

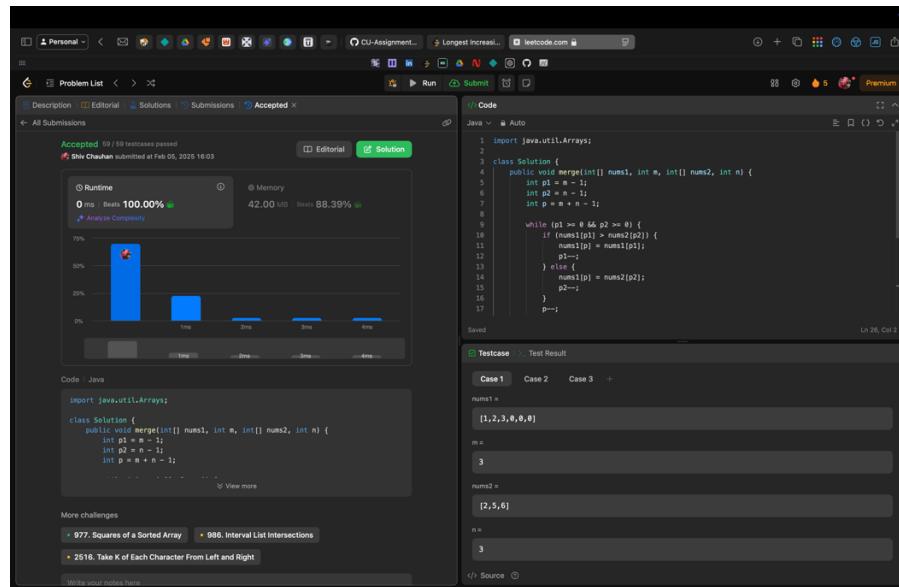
More challenges

- 300. Longest Increasing Subsequence
- 673. Number of Longest Increasing Subsequence
- 674. Longest Continuous Increasing Subsequence

Write your notes here

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--;  
        }  
    }  
}
```



The screenshot shows a LeetCode submission page for the 'Merge Sorted Array' problem. The code has been accepted and passed all test cases. The runtime is 0 ms (100.00%) and the memory usage is 42.00 MB (88.39%). The code editor displays the Java solution provided above. Below the editor, there are sections for 'Testcase' and 'Test Result' where specific input and output pairs are shown.

```
import java.util.Arrays;  
  
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;  
    }  
}
```

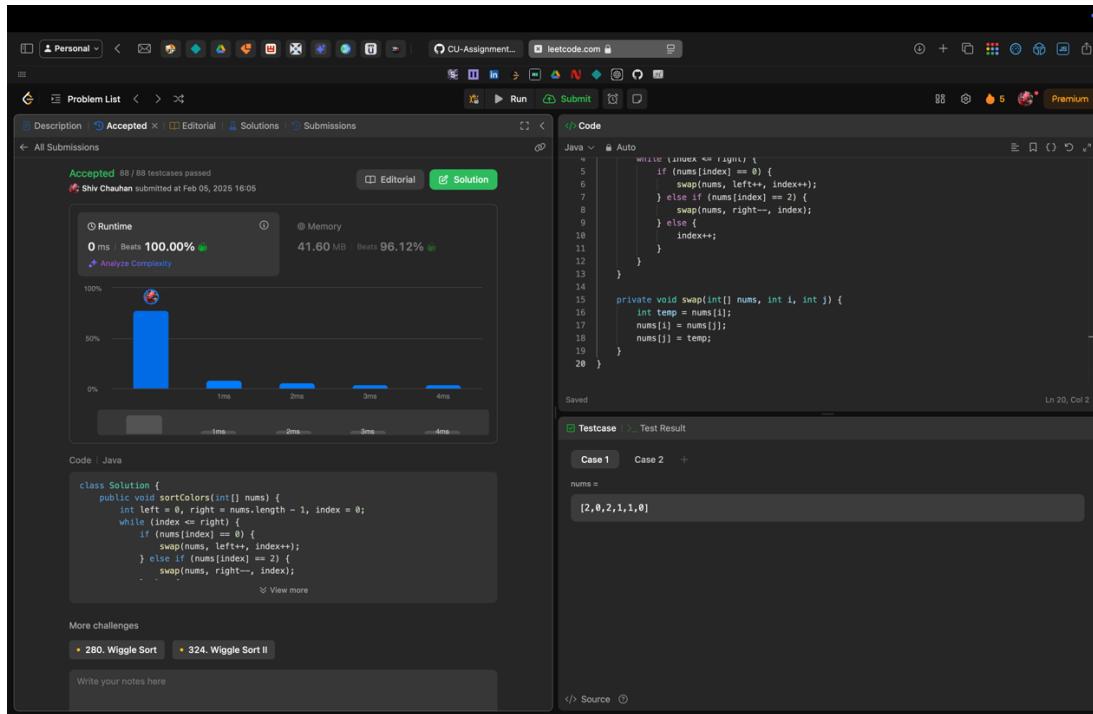
The screenshot shows a LeetCode submission page for the 'First Bad Version' problem. The code has been accepted with 24/24 testcases passed by Shiv Chauhan at Feb 05, 2025 16:04. The runtime is 24 ms (Beats 98.12%) and memory usage is 40.37 MB (Beats 64.00%). The complexity analysis shows a time complexity of O(log n). The code editor displays the Java implementation. The test case section shows two cases: Case 1 with input 5 and output 4, and Case 2 with input 10 and output 5.

```
/* The isBadVersion API is defined in the parent class VersionControl.  
 * boolean isBadVersion(int 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++;  
            }  
        }  
    }  
}
```

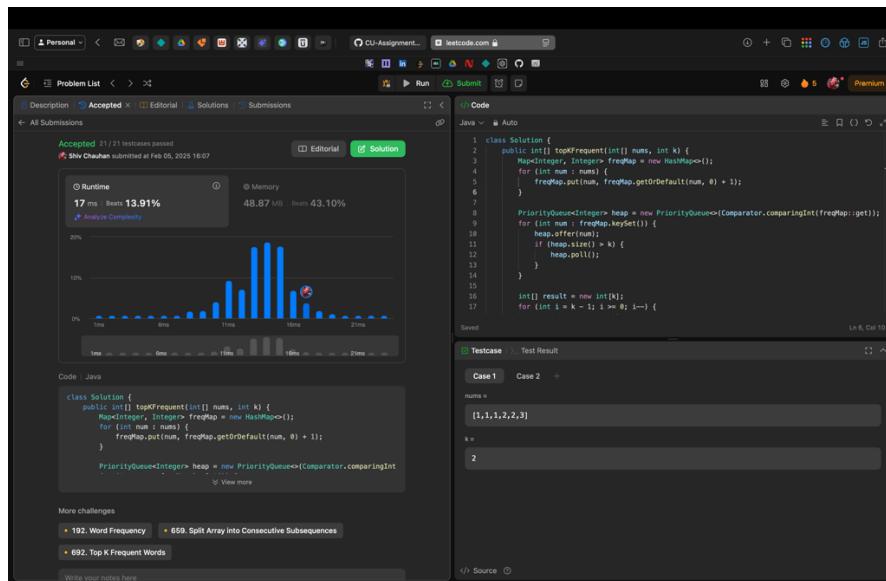
```
private void swap(int[] nums, int i, int j) {  
    int temp = nums[i];  
    nums[i] = nums[j];  
    nums[j] = temp;  
}  
}
```



The screenshot shows a LeetCode submission page for the "Sort Colors" problem. The code has been accepted with 88/88 testcases passed. The runtime is 0 ms (Beats 100.00%) and the memory usage is 41.60 MB (Beats 96.12%). The complexity analysis shows a time complexity of O(n). The code editor displays the Java implementation. Below the editor, the test case results show Case 1: [2,0,2,1,1,0] and Case 2: [2,0,2,1,1,0]. The bottom of the page includes links to other challenges like "280. Wiggle Sort" and "324. Wiggle Sort II".

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();
    }
}
```

The screenshot shows a LeetCode submission page for the problem "Kth Largest Element in the Array".

- Runtime:** 63 ms | Beats 30.87%
- Memory:** 61.46 MB | Beats 31.30%
- Code Editor:** Java code for the solution.
- Testcase:** Case 1: nums = [3,2,3,1,2,4,5,5,6], k = 4. Result: Accepted.

Find Peak Element

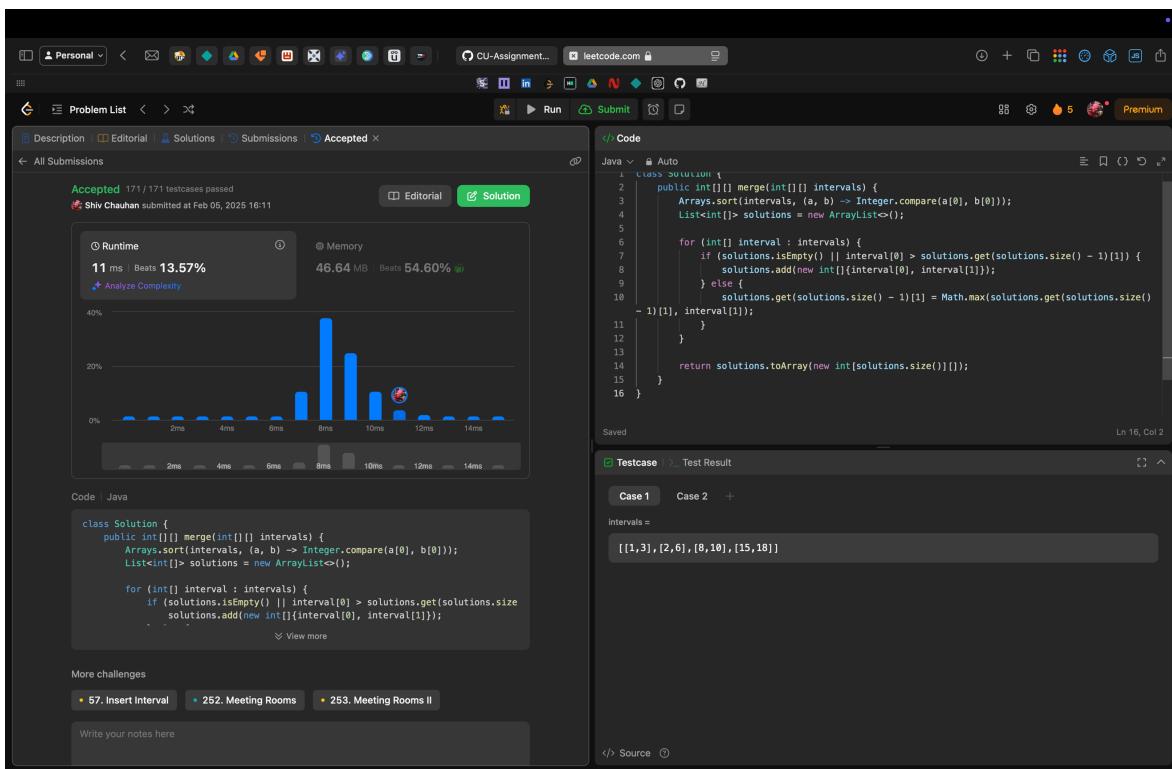
```
class Solution {  
    public int findPeakElement(int[] nums) {  
        int low = 1;  
        int high = nums.length - 2;  
        int n = nums.length;  
        if (n == 1) return 0;  
        if (nums[0] > nums[1]) return 0;  
        if (nums[n - 1] > nums[n - 2]) return n - 1;  
        while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

The screenshot shows a LeetCode submission page for the 'Find Peak Element' problem. The code has been accepted with 68/68 testcases passed. The runtime is 0 ms (beats 100.00%) and the memory usage is 42.94 MB (beats 7.38%). The code editor shows the Java implementation. The test case input is [1,2,3,1].

```
Accepted 68 / 68 testcases passed  
Shiv Chauhan submitted at Feb 05, 2025 16:10  
Editorial Solution  
Runtime 0 ms Beats 100.00% Memory 42.94 MB Beats 7.38%  
Analyze Complexity  
Code Java  
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;  
        ...  
    }  
}  
More challenges  
• 862. Peak Index in a Mountain Array  
• 2137. Pour Water Between Buckets to Make Water Levels Equal  
• 2210. Count Hills and Valleys in an Array
```

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()][]);  
    }  
}
```



The screenshot shows a LeetCode submission page for the "Merge Intervals" problem. The code has been accepted with 171 / 171 testcases passed. The author is Shiv Chauhan, submitted at Feb 05, 2025 16:11. The runtime is 11 ms (Beats 13.57%) and the memory usage is 46.64 MB (Beats 54.60%). The complexity analysis shows a bar chart with the following approximate data:

Time Range (ms)	Percentage
0-2	~1%
2-4	~1%
4-6	~1%
6-8	~10%
8-10	~35%
10-12	~15%
12-14	~10%
14-16	~1%

The code editor contains the Java solution provided above. The test case section shows the input intervals [[1,3],[2,6],[8,10],[15,18]] and the expected output [[1,6],[8,10],[15,18]]. The bottom navigation bar includes links to other challenges like 57. Insert Interval, 252. Meeting Rooms, and 253. Meeting Rooms II.

Search in a Sorted Rotated Array

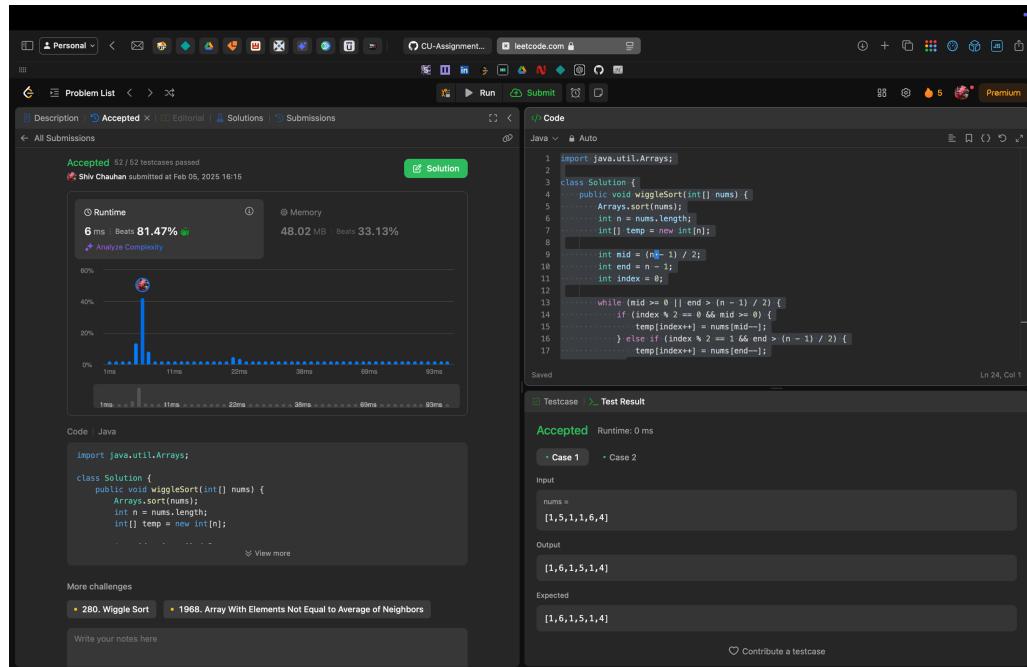
```
class Solution {  
    public int search(int[] nums, int target) {  
        int low = 0, high = nums.length - 1;  
        while (low <= high) {  
            int mid = low + (high - low) / 2;  
            if (nums[mid] == target) return mid;  
            if (nums[low] <= nums[mid]) {  
     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

The screenshot shows the LeetCode platform interface for problem 33. The code has been submitted and accepted, with 196/196 test cases passed. The runtime is 0 ms (Beats 100.00%) and memory usage is 42.02 MB (Beats 67.28%). The code editor displays the Java solution provided above. Below the editor, there is a 'Testcase' table with three rows: Case 1, Case 2, and Case 3. The first two rows have their 'Code' and 'target' fields filled with the values from the screenshot. The third row has its 'Code' field filled with '0'. At the bottom of the editor, there is a note: 'More challenges' followed by a link to '2137. Pour Water Between Buckets to Make Water Levels Equal'.

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);
    }
}
```



The screenshot shows a LeetCode submission page for the "Wiggle Sort II" problem. The code has been accepted, having passed 52/52 testcases. It was submitted by Shiv Chauhan on Feb 05, 2025 at 16:15. The runtime is 6 ms (Beats 81.47%), and the memory usage is 48.02 MB (Beats 33.13%). A histogram indicates the distribution of execution times. The code itself is identical to the one provided above.

```
1 import java.util.Arrays;
2
3 class Solution {
4     public void wiggleSort(int[] nums) {
5         Arrays.sort(nums);
6         int n = nums.length;
7         int[] temp = new int[n];
8
9         int mid = (n-1) / 2;
10        int end = n - 1;
11        int index = 0;
12
13        while (mid >= 0 || end > (n - 1) / 2) {
14            if (index % 2 == 0 && mid >= 0) {
15                temp[index++] = nums[mid--];
16            } else if (index % 2 == 1 && end > (n - 1) / 2) {
17                temp[index++] = nums[end--];
18            }
19        }
20
21        System.arraycopy(temp, 0, nums, 0, n);
22    }
23}
```

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;  
    }  
}
```

The screenshot shows a LeetCode problem page for challenge 373. The code has been accepted with 67/67 testcases passed. The runtime is 0 ms (Beats 100.00%) and memory usage is 49.52 MB (Beats 42.13%). The code implements a binary search algorithm to find the k-th smallest element in a matrix. The Java code is as follows:

```
1 class Solution {
2     public int kthSmallest(int[][] matrix, int k) {
3         int n = matrix.length;
4         int low = matrix[0][0], high = matrix[n - 1][n - 1];
5
6         while (low < high) {
7             int mid = low + (high - low) / 2;
8             int count = countLessEqual(matrix, mid, n);
9
10            if (count >= k) {
11                high = mid;
12            } else {
13                low = mid + 1;
14            }
15        }
16        return low;
17    }
}
```

The Testcase section shows the following input:

```
matrix = [[1,5,9],[10,11,13],[12,13,15]]  
k = 8
```

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];  
        }  
    }  
}
```

Accepted 2096 / 2096 testcases passed

Shiv Chauhan submitted at Feb 05, 2025 16:17

Runtime: 2 ms | Beats 59.23% | Analyze Complexity

Memory: 46.44 MB | Beats 12.00%

Code

```
Java | Auto
14 while (i < nums1.length) {
15     merged[idx++] = nums1[i++];
16 }
17
18 while (j < nums2.length) {
19     merged[idx++] = nums2[j++];
20 }
21
22 int n = merged.length;
23 if (n % 2 == 0) {
24     return (merged[n / 2 - 1] + merged[n / 2]) / 2.0;
25 } else {
26     return merged[n / 2];
27 }
28 }
```

Testcase | Test Result

Case 1 Case 2 +

nums1 = [1,3]

nums2 = [2]

Source