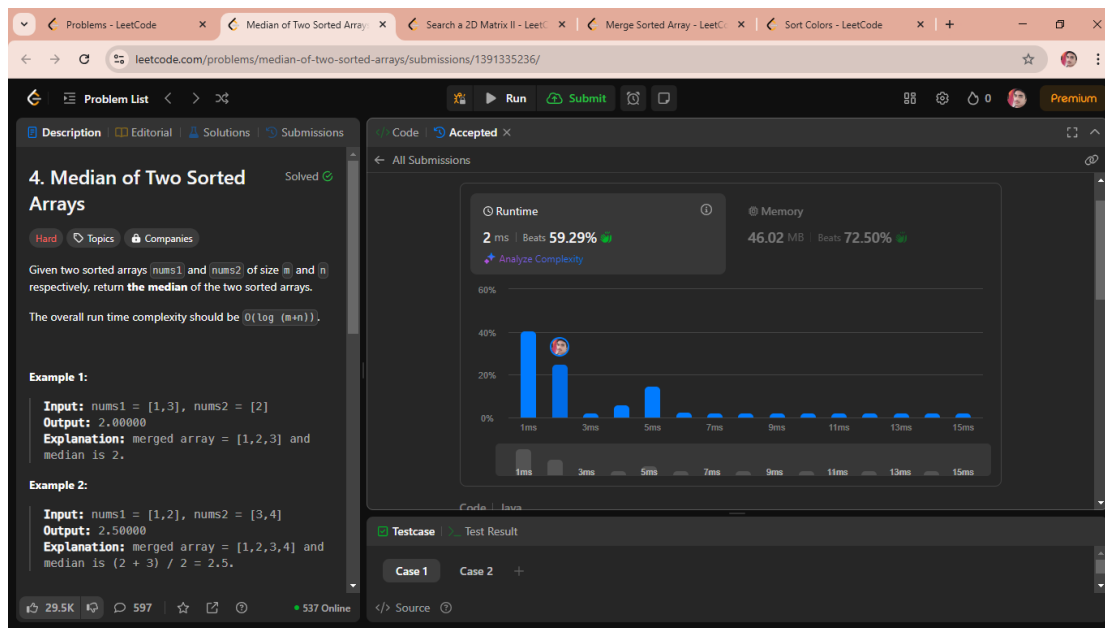
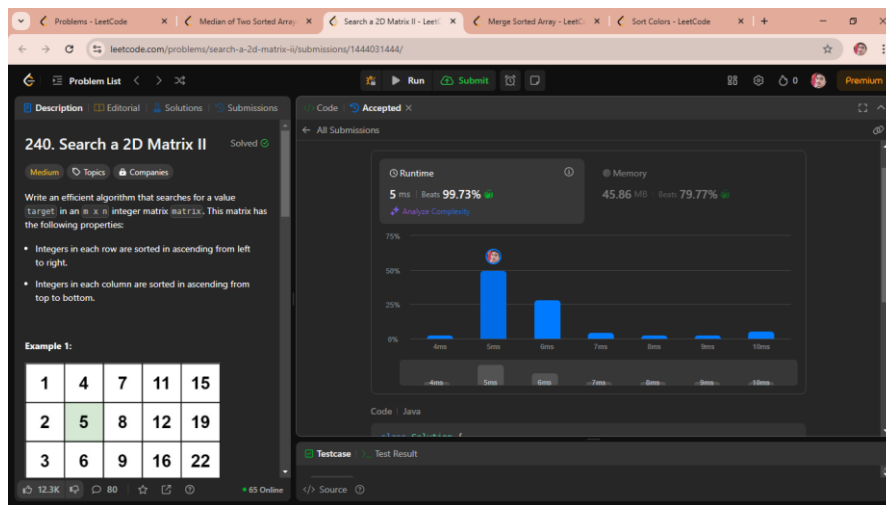


4. Median of Two Sorted Arrays



```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int sI1 = 0;
        int sI2 = 0;
        int[] mergeArr = new int[nums1.length + nums2.length];
        int sI = 0;
        while (sI1 < nums1.length && sI2 < nums2.length) {
            if (nums1[sI1] <= nums2[sI2]) {
                mergeArr[sI++] = nums1[sI1++];
            } else {
                mergeArr[sI++] = nums2[sI2++];
            }
        }
        while (sI1 < nums1.length) {
            mergeArr[sI++] = nums1[sI1++];
        }
        while (sI2 < nums2.length) {
            mergeArr[sI++] = nums2[sI2++];
        }
        int mid = mergeArr.length/2;
        if (mergeArr.length%2 == 0) {
            return (double)(mergeArr[mid] + mergeArr[mid-1])/2;
        } else {
            return mergeArr[mid];
        }
    }
}
```

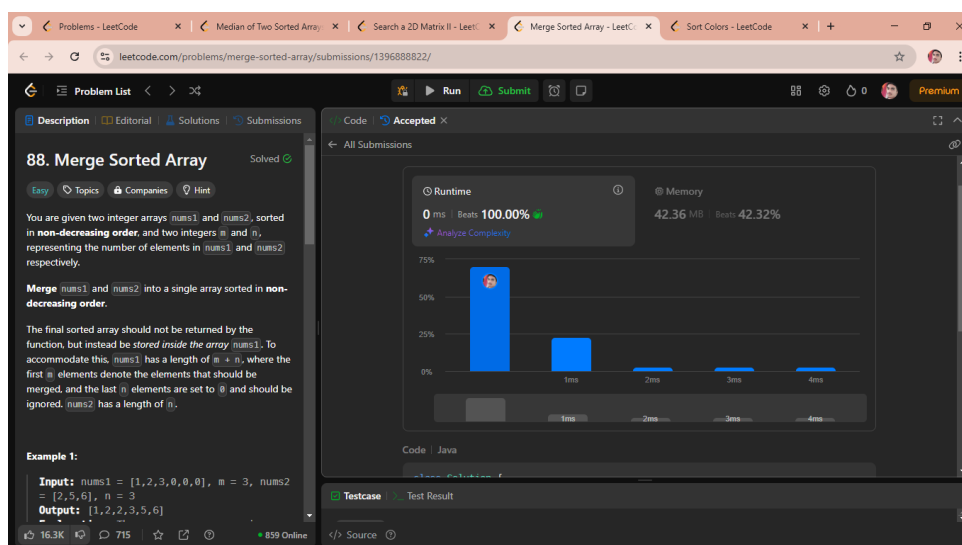
240. Search a 2D Matrix II



```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if(matrix==null || matrix.length<1 || matrix[0].length<1) return false;
        int rP = 0, cP = matrix[0].length-1;
        while (rP < matrix.length && cP >=0) {
            if (matrix[rP][cP] == target) return true;

            if (matrix[rP][cP] > target) {
                cP--;
            } else {
                rP++;
            }
        }
        return false;
    }
}
```

88. Merge Sorted Array



```

class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int[] arr = new int [m+n];
        int k = 0;
        int s1 = 0;
        int s2 = 0;
        while (s1<m && s2<n) {
            if (nums1[s1] <= nums2[s2]) {
                arr[k++] = nums1[s1++];
            } else {
                arr[k++] = nums2[s2++];
            }
        }
        while(s1<m) arr[k++] = nums1[s1++];

        while (s2<n) arr[k++] = nums2[s2++];

        for (int i=0; i<arr.length; i++) {
            nums1[i] = arr[i];
        }
    }
}

```

75. Sort Colors

The screenshot displays the LeetCode problem page for "75. Sort Colors". The problem description on the left states: "Given an array `nums` with `n` objects colored red, white, or blue, sort them *in-place* so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function." It includes two examples: Example 1 with input `[2,0,2,1,1,0]` and output `[0,0,1,1,2,2]`; Example 2 with input `[2,0,1]` and output `[0,1,2]`.

The right panel shows the submission results for user "Shaurya270204" submitted on Feb 04, 2025 at 10:43. The status is "Accepted" with "88 / 88 testcases passed". Performance metrics are: Runtime: 0 ms (Beats 100.00%), Memory: 41.92 MB (Beats 64.71%). A bar chart shows the runtime distribution, with a single bar at 0 ms. Below the chart, there are tabs for "Testcase" and "Test Result", with "Case 1" and "Case 2" visible.

```

class Solution {
    public void sortColors(int[] nums) {
        int count2 = 0;
        int sum = 0;
        for (int num : nums) {
            sum += num;
        }
    }
}

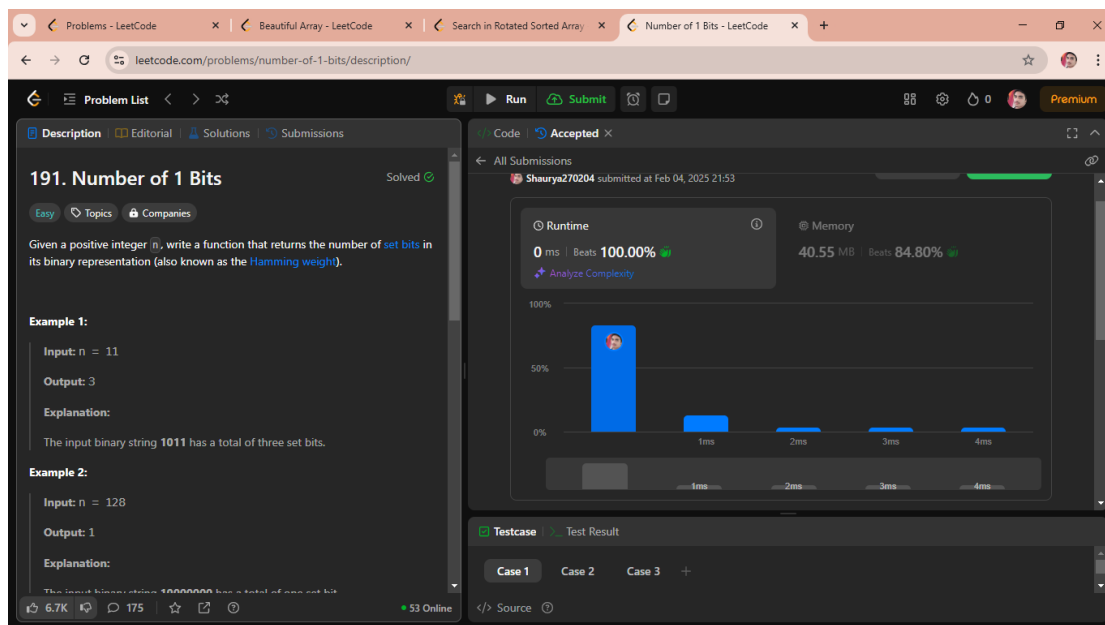
```

```

        if (num == 2) {
            count2++;
        }
    }
    for (int i=nums.length-1; i>=0; i--) {
        if (count2 == 0 && sum == 0) {
            nums[i] = 0;
        } else if (count2 == 0 && sum != 0) {
            nums[i] = 1;
            sum--;
        } else {
            nums[i] = 2;
            count2--;
            sum -=2;
        }
    }
}
}
}

```

191. Number of 1 Bits



```

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

```

932. Beautiful Array

932. Beautiful Array Solved

Medium Topics Companies

An array `nums` of length `n` is **beautiful** if:

- `nums` is a permutation of the integers in the range `[1, n]`.
- For every $0 \leq i < j < n$, there is no index `k` with $i < k < j$ where $2 * \text{nums}[k] == \text{nums}[i] + \text{nums}[j]$.

Given the integer `n`, return *any beautiful array* `nums` of length `n`. There will be at least one valid answer for the given `n`.

Example 1:
Input: `n = 4`
Output: `[2, 1, 4, 3]`

Example 2:
Input: `n = 5`
Output: `[3, 1, 2, 5, 4]`

Runtime: 0 ms | Beats: 100.00% | Memory: 42.48 MB | Beats: 65.81%

Testcase: Case 1 Case 2

```
class Solution {  
    public int[] beautifulArray(int n) {  
        if (n == 1) return new int[] {1};  
        int[] left = beautifulArray((n + 1) >> 1);  
        int[] right = beautifulArray(n >> 1);  
        int[] ans = new int[n];  
        int i = 0;  
        for (int x : left) ans[i++] = x * 2 - 1;  
  
        for (int x : right) ans[i++] = x * 2;  
        return ans;  
    }  
}
```

33. Search in Rotated Sorted Array

33. Search in Rotated Sorted Array Solved

Medium Topics Companies

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` ($1 \leq k < \text{nums.length}$) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**).
For example, `[0, 1, 2, 4, 5, 6, 7]` might be rotated at pivot index `3` and become `[4, 5, 6, 7, 0, 1, 2]`.

Given the array `nums` after the possible rotation and an integer `target`, return the *index of target* if it is in `nums`, or `-1` if it is not in `nums`.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:
Input: `nums = [4, 5, 6, 7, 0, 1, 2]`, `target = 0`
Output: `4`

Example 2:
Input: `nums = [4, 5, 6, 7, 0, 1, 2]`, `target = 3`

Runtime: 0 ms | Beats: 100.00% | Memory: 42.21 MB | Beats: 40.06%

Testcase: Case 1 Case 2 Case 3

nums = [4, 5, 6, 7, 0, 1, 2]

```

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;
    }
}

```

378. Kth Smallest Element in a Sorted Matrix

The screenshot shows the LeetCode problem page for '378. Kth Smallest Element in a Sorted Matrix'. The problem description states: 'Given an $n \times n$ matrix where each of the rows and columns is sorted in ascending order, return the k^{th} smallest element in the matrix. Note that it is the k^{th} smallest element in the sorted order, not the k^{th} distinct element. You must find a solution with a memory complexity better than $O(n^2)$.' Example 1: Input: matrix = [[1,5,9],[10,11,13],[12,13,15]], k = 8; Output: 13; Explanation: The elements in the matrix are [1,5,9,10,11,12,13,13,15], and the 8th smallest number is 13. Example 2: Input: matrix = [[-5]], k = 1; Output: -5. The submission result shows a runtime of 1 ms, beating 74.56% of submissions, and a memory usage of 49.50 MB, beating 43.40% of submissions. The test result is 'Accepted'.

```

class Solution {
    public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length, left = matrix[0][0], right = matrix[n - 1][n - 1];
        while (left < right) {
            int mid = left + (right - left) / 2, count = 0, j = n - 1;
            for (int i = 0; i < n; i++) {
                while (j >= 0 && matrix[i][j] > mid) j--;
                count += (j + 1);
            }
            if (count < k) left = mid + 1;
            else right = mid;
        }
        return left;
    }
}

```

278. First Bad Version

The screenshot shows the LeetCode interface for problem 278. The problem description is on the left, and the submission results are on the right. The problem description states: "You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad. Suppose you have n versions $[1, 2, \dots, n]$ and you want to find out the first bad one, which causes all the following ones to be bad. You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API." The submission results on the right show a runtime of 25 ms, beating 71.56% of submissions, and a memory usage of 40.89 MB, beating 10.94% of submissions. A bar chart shows the distribution of runtimes, with a peak at 25 ms. The test case section shows Case 1 with input `n = 5` and output `4`.

/* 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;  
    }  
}
```

324. Wiggle Sort II

The screenshot shows the LeetCode interface for problem 324. The problem description is on the left, and the submission results are on the right. The problem description states: "Given an integer array `nums`, reorder it such that `nums[0] < nums[1] > nums[2] < nums[3] > ...`. You may assume the input array always has a valid answer." The submission results on the right show a runtime of 5 ms, beating 95.23% of submissions, and a memory usage of 47.73 MB, beating 65.37% of submissions. A bar chart shows the distribution of runtimes, with a peak at 5 ms. The test case section shows Case 1 with input `nums = [1,5,1,1,6,4]` and output `[1,6,1,5,1,4]`.

```

class Solution {
    public void wiggleSort(int[] nums) {
        int n = nums.length;
        int[] temp = nums.clone();
        Arrays.sort(temp);
        int i = n - 1;
        int left = 0;
        int right = i / 2 + 1;

        while (i >= 0) {
            if (i % 2 == 1) {
                nums[i] = temp[right];
                right++;
            } else {
                nums[i] = temp[left];
                left++;
            }
            i--;
        }
    }
}

```

215. Kth Largest Element in an Array

215. Kth Largest Element in an Array Solved

Medium Topics Companies

Given an integer array `nums` and an integer `k`, return the k^{th} largest element in the array.

Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.

Can you solve it without sorting?

Example 1:
Input: `nums = [3,2,1,5,6,4], k = 2`
Output: `5`

Example 2:
Input: `nums = [3,2,3,1,2,4,5,6], k = 4`
Output: `4`

Constraints:

- `1 <= k <= nums.length <= 10^4`
- `-10^4 <= nums[i] <= 10^4`

Runtime: 2 ms | Beats 100.00% | Memory: 55.01 MB | Beats 99.04%

Accepted

Case 1 Case 2

```

class Solution {
    public int findKthLargest(int[] nums, int k) {
        int[] count = new int[20001];

        for (int num : nums) count[num + 10000]++;

        for (int i = count.length - 1; i >= 0; i--)
            if (count[i] > 0) {
                k -= count[i];
            }
    }
}

```

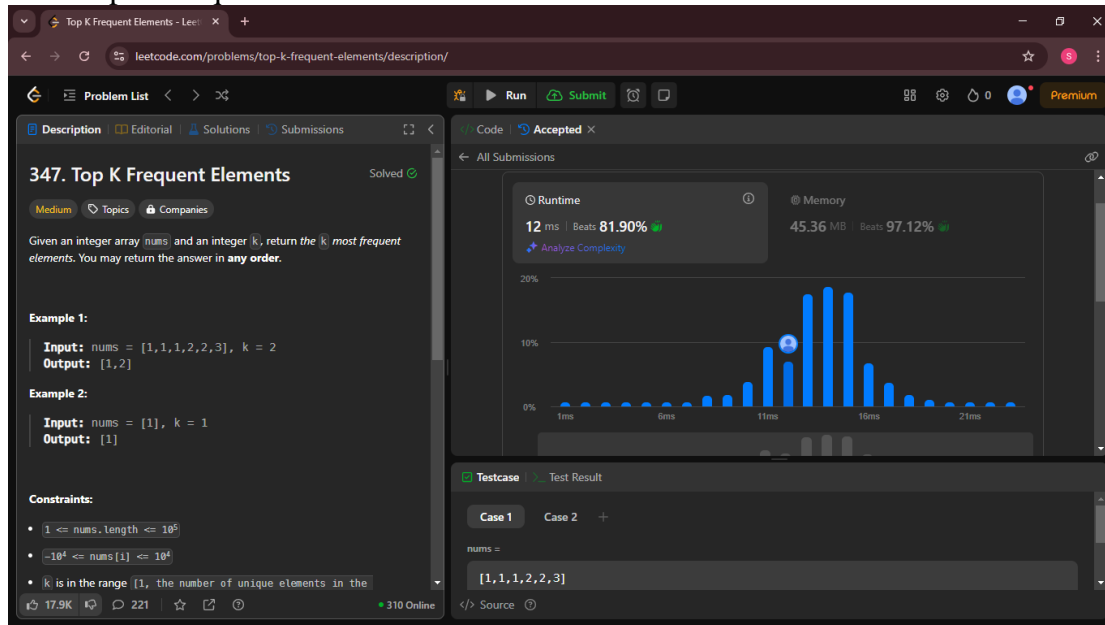


```

        if (k <= 0) return i - 10000;
    }
    return -1;
}
}

```

347. 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);

        List<Integer>[] bucket = new List[nums.length + 1];
        for (int key : freqMap.keySet()) {
            int freq = freqMap.get(key);
            if (bucket[freq] == null) bucket[freq] = new ArrayList<>();
            bucket[freq].add(key);
        }

        int[] result = new int[k];
        int index = 0;
        for (int i = bucket.length - 1; i >= 0 && index < k; i--) {
            if (bucket[i] != null) {
                for (int num : bucket[i]) {
                    result[index++] = num;
                    if (index == k) return result;
                }
            }
        }
        return result;
    }
}

```