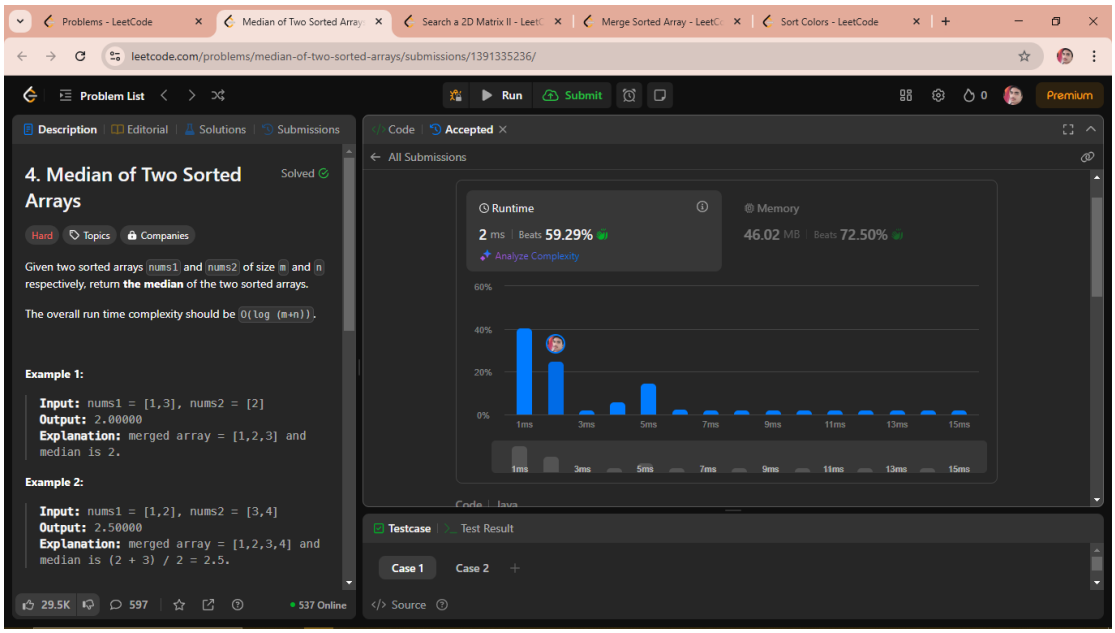
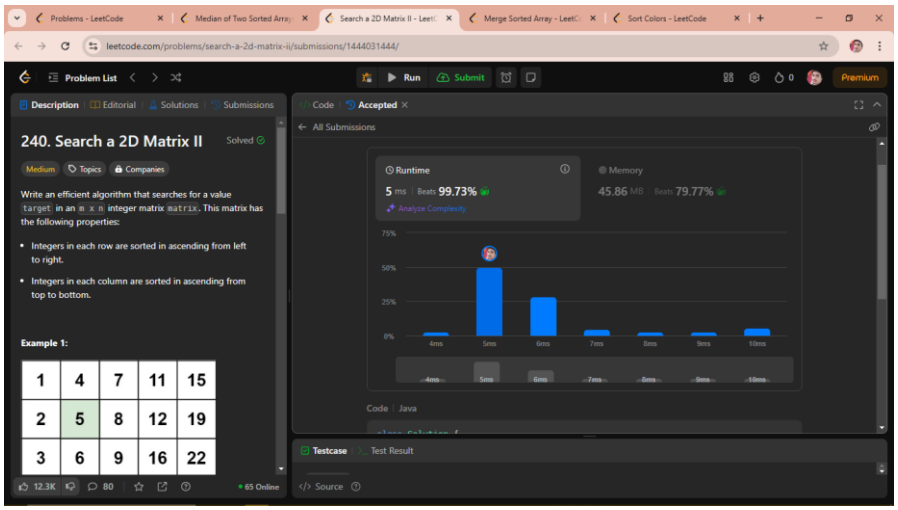
4. Median of Two Sorted Arrays



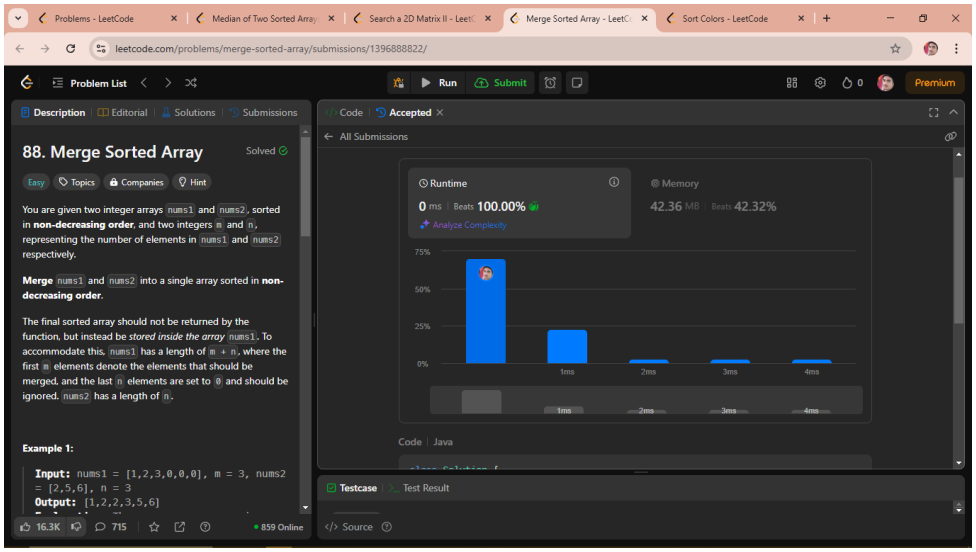

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int sI1 = 0;
        int sI2 = 0;
        int[] mergeArr = new int[nums1.length + nums2.length];
        int sI = 0;
        while (sI1 < nums1.length && sI2 < nums2.length) {
            if (nums1[sI1] <= 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



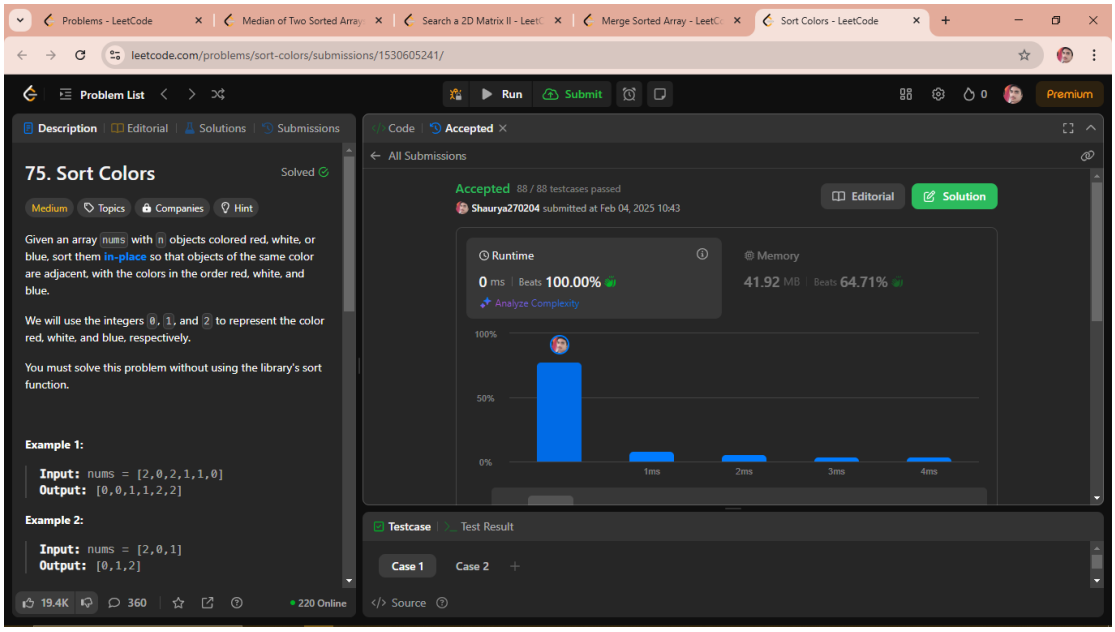

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if(matrix==null || matrix.length<1 || matrix[0].length<1) return false;
        int rP = 0, cP = matrix[0].length-1;
        while (rP < matrix.length && cP >=0) {
            if (matrix[rP][cP] == target) return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

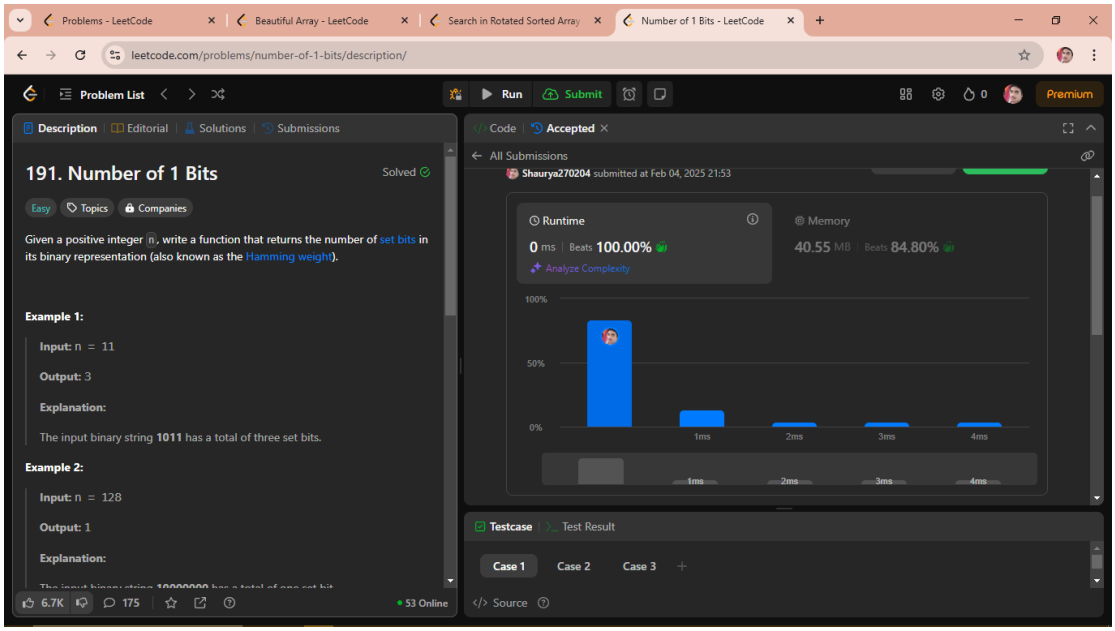
75. Sort Colors




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

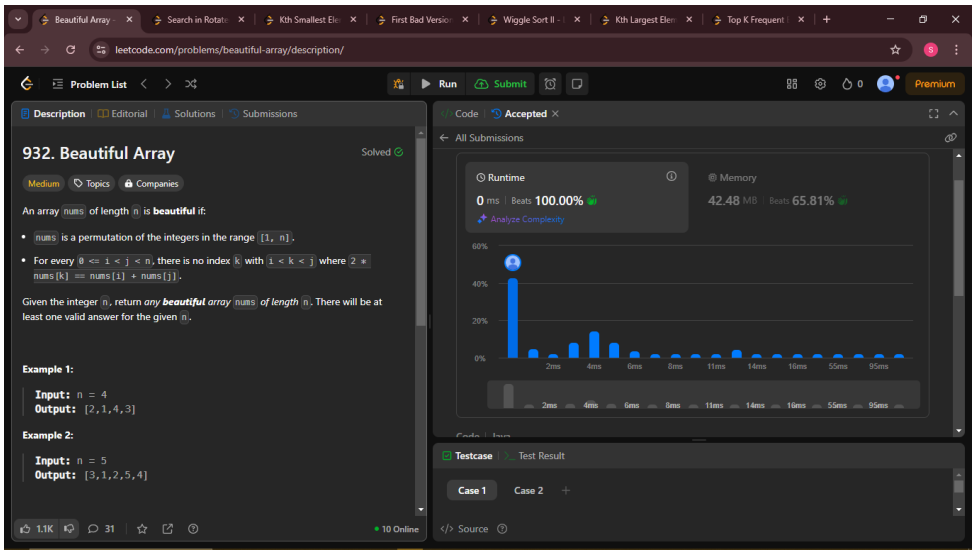
```
            if (num == 2) {
                count2++;
            }
        }
        for (int i=nums.length-1; i>=0; i--) {
            if (count2 == 0 && sum == 0) {
                nums[i] = 0;
            } else if (count2 == 0 && sum != 0) {
                nums[i] = 1;
                sum--;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

191. Number of 1 Bits



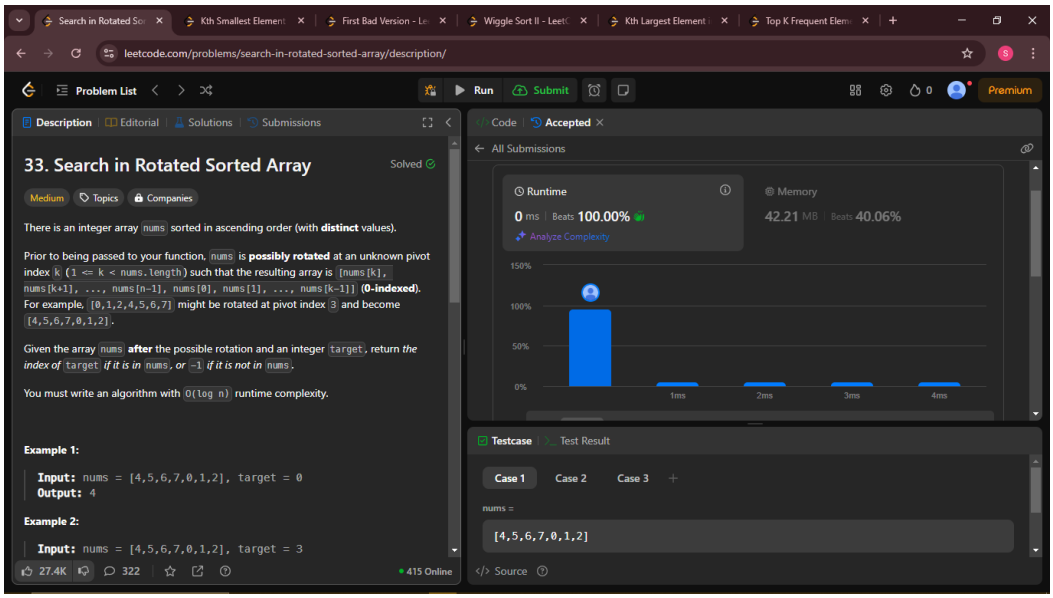

```
class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n>0) {
            if ((n&1) == 1) count++;
            n >>= 1;
        }
        return count;
    }
}
```

932. Beautiful Array



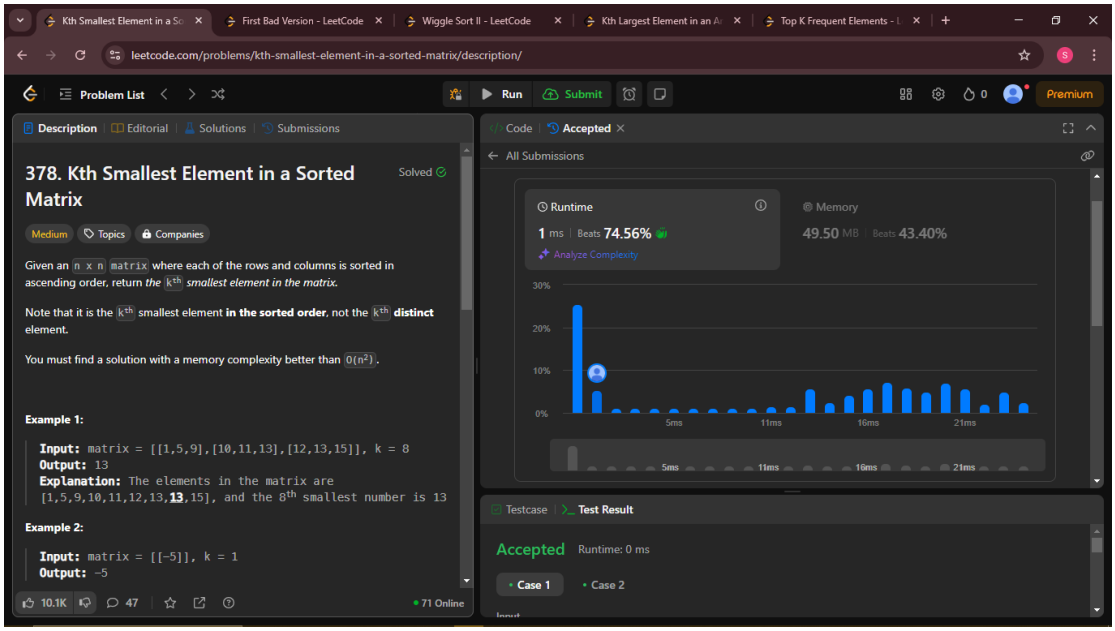

```
class Solution {
    public int[] beautifulArray(int n) {
        if (n == 1)  return new int[] {1};
        int[] left = beautifulArray((n + 1) >> 1);
        int[] right = beautifulArray(n >> 1);
        int[] ans = new int[n];
        int i = 0;
        for (int x : left) ans[i++] = x * 2 - 1;

        for (int x : right) ans[i++] = x * 2;
        return ans;
    }
}
```

33. Search in Rotated Sorted Array



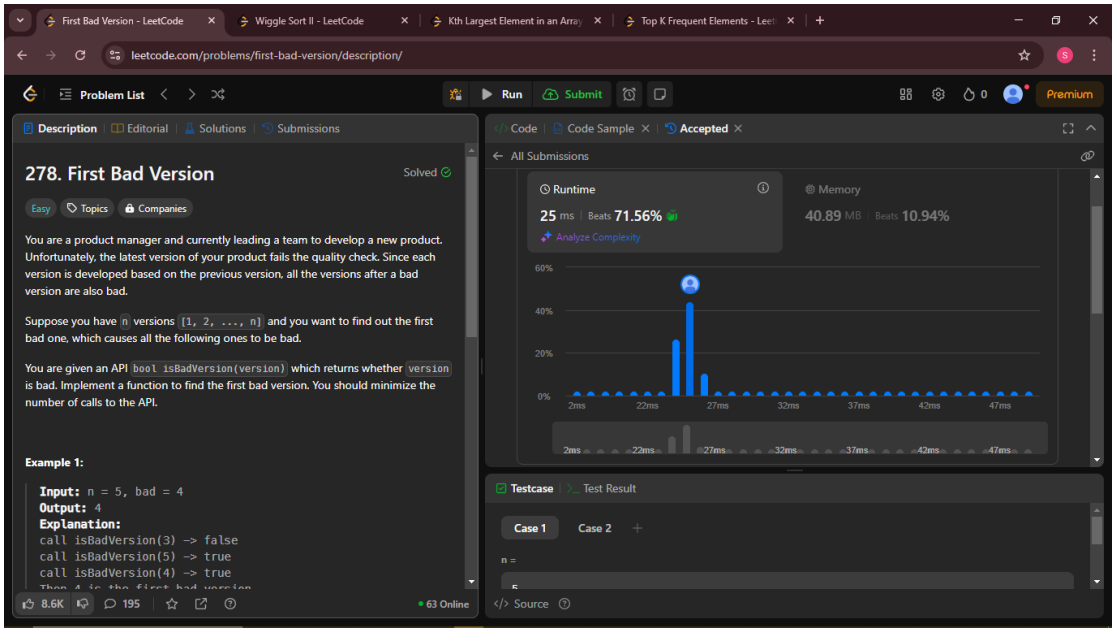
```
class Solution {
    public int search(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) return mid;
            if (nums[left] <= nums[mid]) {
           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



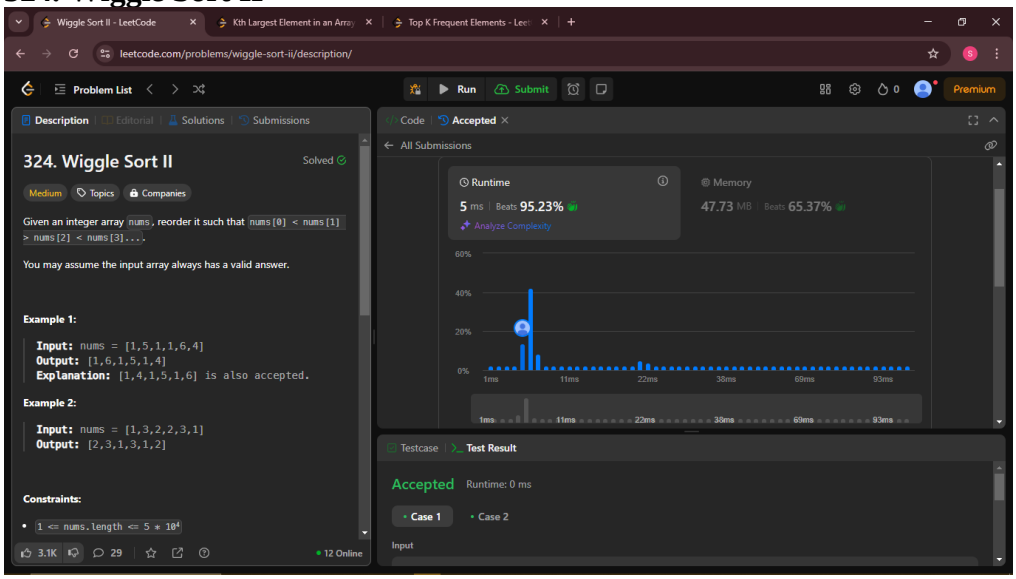

```
class Solution {
    public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length, left = matrix[0][0], right = matrix[n - 1][n - 1];
        while (left < right) {
            int mid = left + (right - left) / 2, count = 0, j = n - 1;
            for (int i = 0; i < n; i++) {
 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



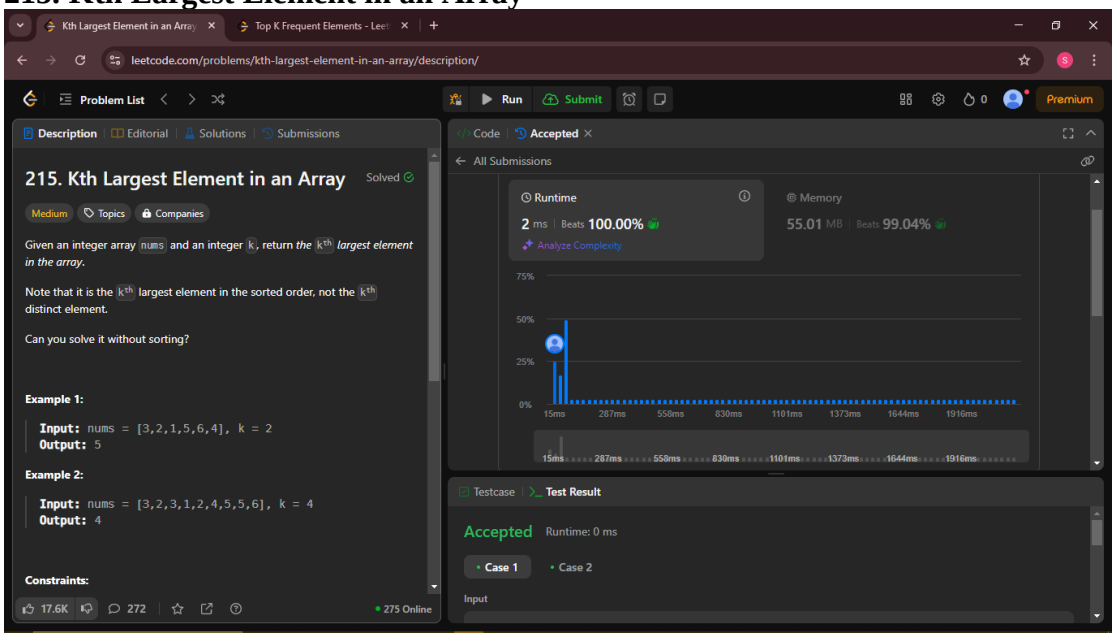

```
/* The isBadVersion API is defined in the parent class VersionControl.
      boolean isBadVersion(int version); */

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



```
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--;
        }
    }
}
```

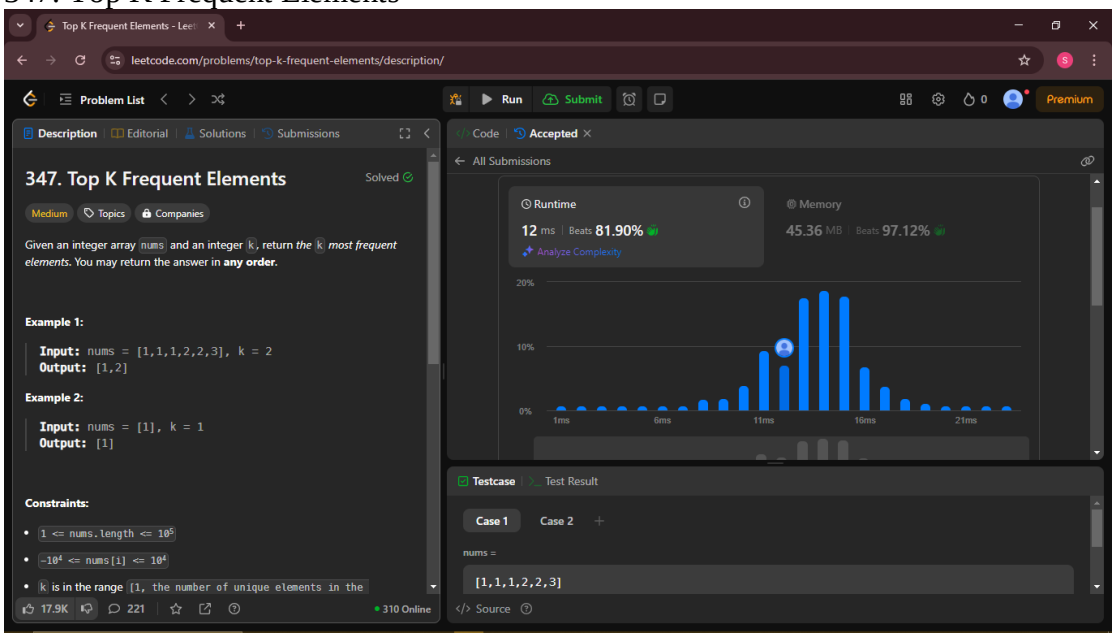
215. Kth Largest Element in an Array




```
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;
    }
}
```