**WORKSHEET 5**

**Student Name: Riyan Reyaz UID:22BCS11986**

**Branch: CSE Section/Group: NTPP 603/B**

**Semester: 06 Date of Performance: 27/02/2025**

**Subject Name: AP Lab II Subject Code: 22CSP-351**

1. **Aim**:
2. Merge Sorted Array
3. First Bad Version
4. Kth Largest Element in an Array
5. **Source Code:**

class Solution {

    public void merge(int[] nums1, int m, int[] nums2, int n) {

        int i = m - 1; // pointer for nums1

        int j = n - 1; // pointer for nums2

        int k = m + n - 1; // pointer for the last index of nums1

        while (i >= 0 && j >= 0) {

            if (nums1[i] > nums2[j]) {

                nums1[k] = nums1[i];

                i--;

            } else {

                nums1[k] = nums2[j];

                j--;

            }

            k--;

        }

        while (j >= 0) {

            nums1[k] = nums2[j];

            j--;

            k--;

        }

    }

}

**b.**

/\* 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 low = 1;

int high = n;

while (low < high) {

int mid = low + (high - low) / 2; // Avoid overflow

if (isBadVersion(mid)) {

high = mid; // The first bad version is at mid or before it

} else {

low = mid + 1; // The first bad version is after mid

}

}

// At the end of the loop, low == high, pointing to the first bad version

return low;

}

}

**C.**

import java.util.PriorityQueue;

public class Solution {

    public int findKthLargest(int[] nums, int k) {

        PriorityQueue<Integer> minHeap = new PriorityQueue<>();

        for (int num : nums) {

            minHeap.add(num);

            if (minHeap.size() > k) {

                minHeap.poll();

            }

        }

        return minHeap.peek();

    }

    public static void main(String[] args) {

        Solution solution = new Solution();

        // Example 1

        int[] nums1 = {3, 2, 1, 5, 6, 4};

        int k1 = 2;

        System.out.println("Kth largest element: " + solution.findKthLargest(nums1, k1)); // Expected output: 5

        // Example 2

        int[] nums2 = {3, 2, 3, 1, 2, 4, 5, 5, 6};

        int k2 = 4;

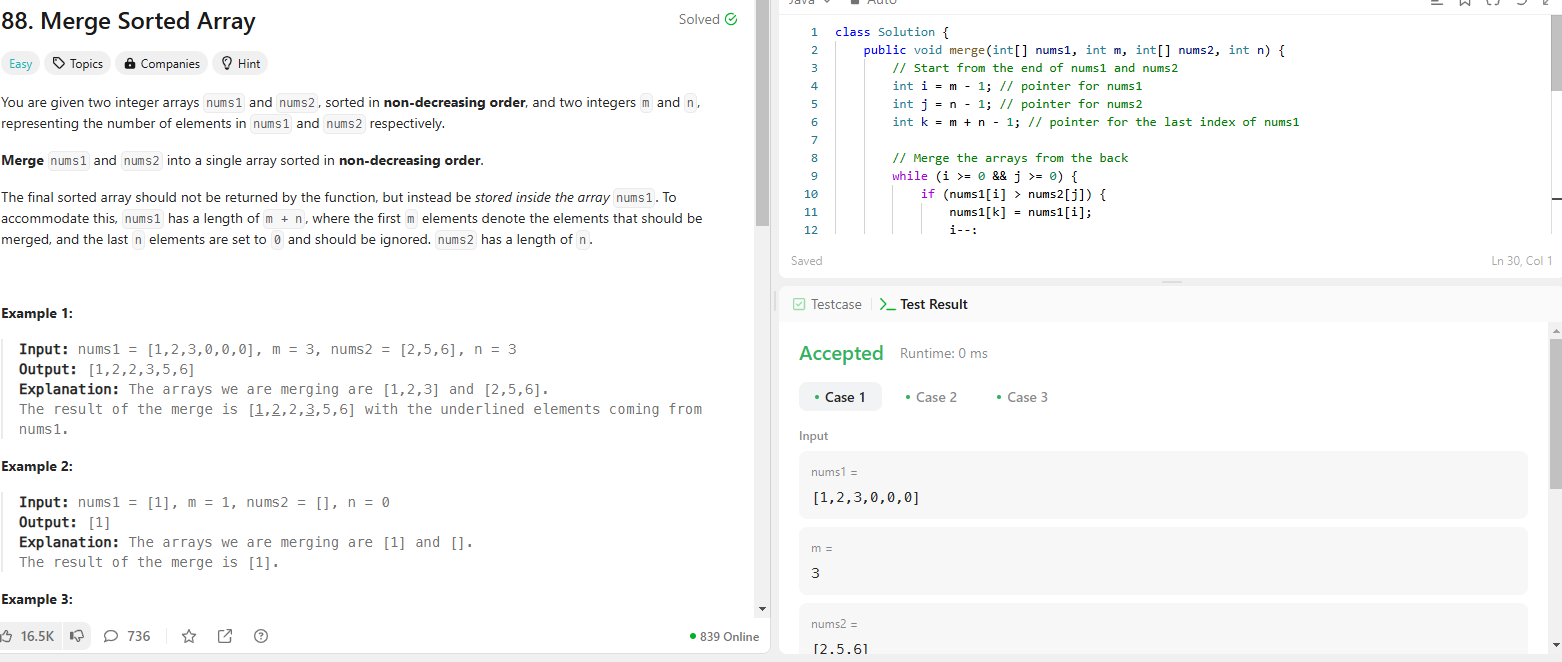
        System.out.println("Kth largest element: " + solution.findKthLargest(nums2, k2)); // Expected output: 4

    }

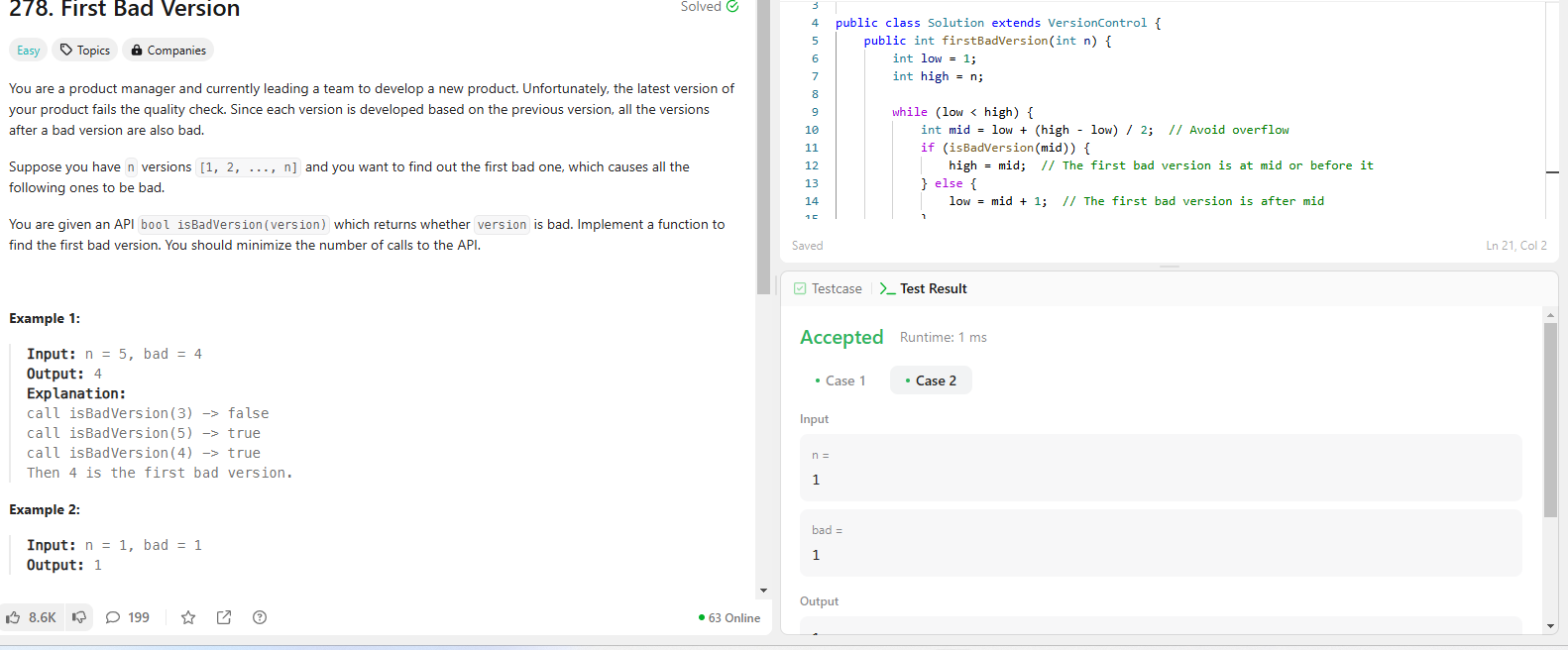
}

1. **Screenshot of Outputs:**

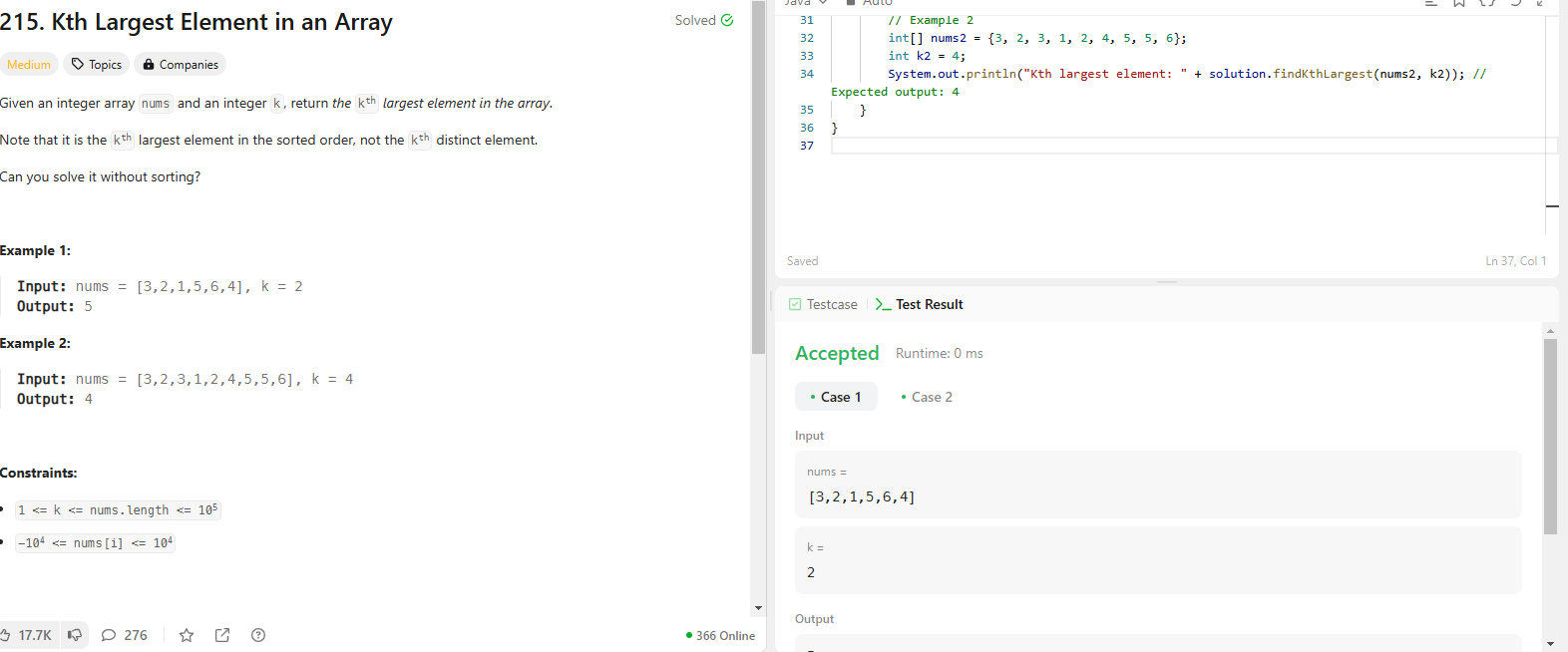
**a.**



**b.**



**c.**



1. **Learning Outcomes**
2. **Learned about various sorting algorithms.**
3. **Learned about various searching algorithms.**