WORKSHEET 5

Student Name: BHUPESH KUMAR **UID:22BCS11723**

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:

a. Merge Sorted Array

b. First Bad Version

c. Kth Largest Element in an Array

2. Source Code:

a.

```
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
    }
}
```

3. Screenshot of Outputs:

a.

88. Merge Sorted Array

Solved

Easy | Topics | Companies | Hint

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to `0` and should be ignored. `nums2` has a length of `n`.

Example 1:

```
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.
```

Example 2:

```
Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]
Explanation: The arrays we are merging are [1] and [].
The result of the merge is [1].
```

Example 3:

16.5K | 736 | 839 Online

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        // Start from the end of nums1 and nums2
        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

        // Merge the arrays from the back
        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k] = nums1[i];
                i--;
```

Saved | Ln 30, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

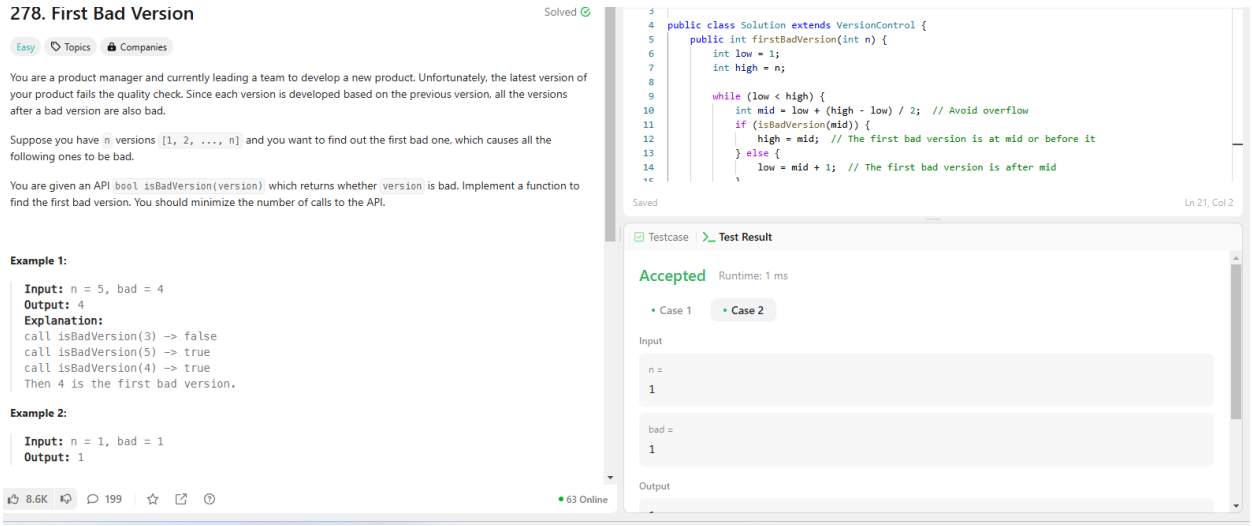
Input

nums1 =
[1,2,3,0,0,0]

m =
3

nums2 =
[2,5,6]

b.



278. First Bad Version

Solved

Easy | Topics | Companies

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, ..., 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.

Example 1:

```
Input: n = 5, bad = 4
Output: 4
Explanation:
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
Then 4 is the first bad version.
```

Example 2:

```
Input: n = 1, bad = 1
Output: 1
```

8.6K | 199 | 63 Online

```java
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
```

Saved | Ln 21, Col 2

Testcase | Test Result

Accepted Runtime: 1 ms

Case 1 | Case 2

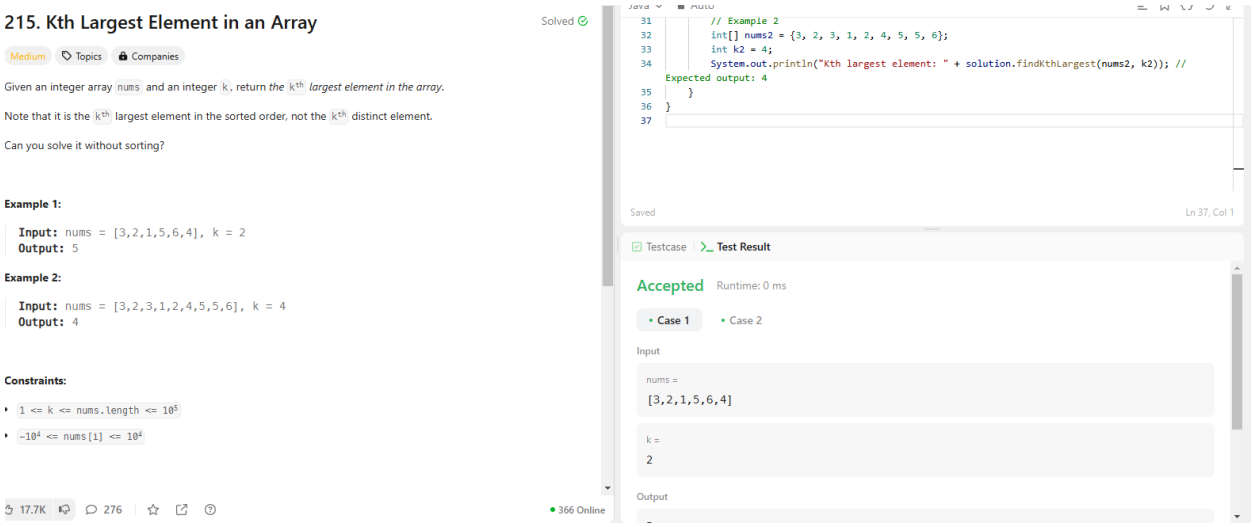
Input

n =
1

bad =
1

Output

c.



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 = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4

Constraints:

- $1 <= k <= nums.length <= 10^5$
- $-10^4 <= nums[i] <= 10^4$

17.7K | 276 | 366 Online

```
31          // Example 2
32          int[] nums2 = {3, 2, 3, 1, 2, 4, 5, 5, 6};
33          int k2 = 4;
34          System.out.println("Kth largest element: " + solution.findKthLargest(nums2, k2)); //
Expected output: 4
35      }
36  }
37
```

Saved | Ln 37, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[3,2,1,5,6,4]

k =
2

Output

4. Learning Outcomes

(i) Learned about various sorting algorithms.

(ii) Learned about various searching algorithms.