

# 2540. Minimum Common Value

Easy   Array   Hash Table   Two Pointers   Binary Search

Leetcode Link

## Problem Description

The problem involves finding the smallest integer that is common to two sorted arrays of integers `nums1` and `nums2`. Both arrays are sorted in non-decreasing order. To solve this problem, we are tasked with comparing the integers from both arrays to identify a common integer. If such an integer is found, it should be returned as the result. If there is no integer common to both arrays, the function should return `-1`.

## Intuition

The solution strategy is based on the fact that both arrays are already sorted in non-decreasing order. This allows us to use a two-pointer approach to efficiently compare elements of the two arrays without the need to look at every possible pair of elements.

Here's how the two-pointer approach works:

- Start with two pointers, `i` and `j`, both initialized to 0, which will traverse through `nums1` and `nums2` respectively.
- While both pointers have not reached the end of their respective arrays, compare the elements pointed to by `i` and `j`.
- If the elements are equal, that means we've found a common integer, and we return that integer immediately since we are looking for the minimum and the array is sorted.
- If they are not equal, we need to move the pointer that points to the smaller element to the right to find a potential match, as the larger element will never match with any previous elements in the other array due to sorting.
- If we reach the end of either array without finding a match, we conclude there is no common integer, and thus return `-1`.

Using this method, we efficiently move through both arrays, comparing only the necessary elements, and we are guaranteed to find the smallest common integer if one exists.

## Solution Approach

The implementation of the solution is a direct application of the two-pointer approach described in the intuition section. The approach utilizes the given sorting of `nums1` and `nums2` to compare elements and find the least common integer.

Here is how the algorithm is applied through the given Python code:

- Initialize two pointers `i` and `j` to 0. These pointers are indices for iterating through `nums1` and `nums2` respectively.
- Determine the lengths of the two arrays `nums1` and `nums2` and store them in variables `m` and `n`. This is done to avoid repeated calculation of lengths within the loop.
- Use a while loop that continues as long as `i < m` and `j < n`. This condition ensures that we do not go out of bounds in either array.
- Inside the loop, compare the elements at the current indices of the two arrays, `nums1[i]` and `nums2[j]`. If they are equal, `nums1[i]` (or `nums2[j]`, since they are the same) is immediately returned as the solution since it's the first common integer found when traversing the arrays from left to right.
- If the elements are not equal, increment the pointer `i` if `nums1[i]` is less than `nums2[j]`, because we are looking for the possibility of `nums1[i]` being in the subsequent elements of `nums2`.
- Otherwise, increment the pointer `j` since `nums2[j]` is less, and we want to find if `nums2[j]` matches any subsequent elements in `nums1`.
- If the while loop ends without returning, this implies that there was no common element between the two arrays. Therefore, we return `-1` at the end of the function to indicate that no common integer was found.

In summary, the algorithm leverages the sorted nature of the inputs to use a methodical, step-by-step comparison that conserves unnecessary checks. This is a common technique in problems involving sorted arrays, known for its efficiency and simplicity.

## Example Walkthrough

Let's consider two sorted arrays `nums1` and `nums2` for the walkthrough:

- `nums1 = [1, 3, 5, 7]`
- `nums2 = [2, 3, 6, 8]`

We need to find the smallest integer common to both `nums1` and `nums2`. According to the solution approach, we will use the two-pointer technique.

- Initialize two pointers, `i` and `j`, at 0. So `i` points to `nums1[i]` which is 1, and `j` points to `nums2[j]` which is 2.
- Compare the elements at `nums1[i]` and `nums2[j]`. Since `1 < 2`, it doesn't match, and we increment `i` since `nums1[i]` is smaller.
- Now `i = 1` and `j = 0`. The new values at the pointers are `nums1[1]` which is 3, and `nums2[0]` which is 2.
- Again, compare `nums1[i]` with `nums2[j]`. Since `3 > 2`, we increment `j` this time as `nums2[j]` is smaller.
- `i` is still at 1 and `j` is incremented to 1. We now have `nums1[1]` as 3, and `nums2[1]` also as 3.
- Since `nums1[i]` is equal to `nums2[j]`, we have found the smallest common integer, which is 3.
- Return 3 as the result.

According to the example, 3 is the smallest integer that is found in both arrays `nums1` and `nums2`. Thus demonstrating the efficiency of the two-pointer approach in solving such problems. If there were no common elements, the pointers would reach the end of their respective arrays, and the function would return `-1`.

## Python Solution

```
1 class Solution:
2     def getCommon(self, nums1: List[int], nums2: List[int]) -> int:
3         # Initialize pointers for both lists
4         index1 = index2 = 0
5
6         # Get the lengths of both lists
7         length1, length2 = len(nums1), len(nums2)
8
9         # Iterate over both lists as long as there are elements in each
10        while index1 < length1 and index2 < length2:
11            # If the current elements are the same, return the common element
12            if nums1[index1] == nums2[index2]:
13                return nums1[index1]
14
15            # If the current element in nums1 is smaller, move to the next element in nums1
16            if nums1[index1] < nums2[index2]:
17                index1 += 1
18            else:
19                # If the current element in nums2 is smaller, move to the next element in nums2
20                index2 += 1
21
22        # If no common elements are found, return -1
23        return -1
24
```

## Java Solution

```
1 class Solution {
2
3     /**
4      * Finds the first common element in two sorted arrays.
5      *
6      * If a common element is found, this method returns that element.
7      * If there are no common elements, the method returns -1.
8      *
9      * @param nums1 The first sorted array.
10     * @param nums2 The second sorted array.
11     * @return The first common element or -1 if none found.
12     */
13     public int getCommon(int[] nums1, int[] nums2) {
14         int nums1Length = nums1.length;
15         int nums2Length = nums2.length;
16
17         // Initialize indices for iterating through the arrays
18         int index1 = 0;
19         int index2 = 0;
20
21         // Loop through both arrays until one array is fully traversed
22         while (index1 < nums1Length && index2 < nums2Length) {
23             // Check the current elements in each array for a match
24             if (nums1[index1] == nums2[index2]) {
25                 // If a match is found, return the common element
26                 return nums1[index1];
27             }
28
29             // Increment the index of the smaller element
30             if (nums1[index1] < nums2[index2]) {
31                 ++index1;
32             } else {
33                 ++index2;
34             }
35         }
36
37         // Return -1 if no common element is found
38         return -1;
39     }
40 }
41
```

## C++ Solution

```
1 class Solution {
2 public:
3     // Function to find the first common element in two sorted arrays
4     int getCommon(vector<int>& nums1, vector<int>& nums2) {
5         int sizeNums1 = nums1.size(); // Size of the first array
6         int sizeNums2 = nums2.size(); // Size of the second array
7
8         // Initialize pointers for both arrays
9         int indexNums1 = 0;
10        int indexNums2 = 0;
11
12        // Loop through both arrays until one array is fully traversed
13        while (indexNums1 < sizeNums1 && indexNums2 < sizeNums2) {
14            // If a common element is found, return it
15            if (nums1[indexNums1] == nums2[indexNums2]) {
16                return nums1[indexNums1];
17            }
18
19            // Move the pointer in the smaller-value array to find a match
20            if (nums1[indexNums1] < nums2[indexNums2]) {
21                ++indexNums1;
22            } else {
23                ++indexNums2;
24            }
25        }
26
27        // If no common element is found, return -1
28        return -1;
29    }
30 };
31
```

## Typescript Solution

```
1 // Function to find the first common element between two sorted arrays.
2 // If no common element is found, returns -1.
3 function getCommon(nums1: number[], nums2: number[]): number {
4     // Length of the first array.
5     const length1 = nums1.length;
6     // Length of the second array.
7     const length2 = nums2.length;
8     // Initialize pointers for both arrays.
9     let index1 = 0;
10    let index2 = 0;
11
12    // Continue looping until the end of one array is reached.
13    while (index1 < length1 && index2 < length2) {
14        // Check if the current elements are the same.
15        if (nums1[index1] === nums2[index2]) {
16            // If they are the same, return the common element.
17            return nums1[index1];
18        }
19        // If the current element of nums1 is smaller, move to the next element in nums1.
20        if (nums1[index1] < nums2[index2]) {
21            index1++;
22        } else {
23            // Otherwise, move to the next element in nums2.
24            index2++;
25        }
26    }
27    // If no common element is found, return -1.
28    return -1;
29 }
30
```

## Time and Space Complexity

The time complexity of this code is  $O(m + n)$ , where `m` is the length of `nums1` and `n` is the length of `nums2`. This is because each while loop iteration increments either `i` or `j`, but never both at the same time, thus at most `m + n` iterations occur before the loop terminates.

The space complexity of the code is  $O(1)$  because there are no additional data structures that grow with the input size. Only a fixed number of integer variables `i`, `j`, `m`, and `n` are used regardless of the input size.