Problem 2540: Minimum Common Value

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given two integer arrays

nums1

and

nums2

, sorted in non-decreasing order, return

the

minimum integer common

to both arrays

. If there is no common integer amongst

nums1

and

nums2

, return

-1

.

Note that an integer is said to be

common

to

nums1

and

nums2

if both arrays have

at least one

occurrence of that integer.

Example 1:

Input:

nums1 = [1,2,3], nums2 = [2,4]

Output:

2

Explanation:

The smallest element common to both arrays is 2, so we return 2.

Example 2:

Input:

nums1 = [1,2,3,6], nums2 = [2,3,4,5]

Output:

2

Explanation:

There are two common elements in the array 2 and 3 out of which 2 is the smallest, so 2 is returned.

Constraints:

1 <= nums1.length, nums2.length <= 10

5

1 <= nums1[i], nums2[j] <= 10

9

Both

nums1

and

nums2

are sorted in

non-decreasing

order.

Code Snippets

C++:

```
class Solution {
public:
int getCommon(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int getCommon(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def getCommon(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def getCommon(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var getCommon = function(nums1, nums2) {

};
```

TypeScript:

```
function getCommon(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public int GetCommon(int[] nums1, int[] nums2) {

}
}
```

C:

```
int getCommon(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func getCommon(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun getCommon(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func getCommon(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn get_common(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def get_common(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function getCommon($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
int getCommon(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def getCommon(nums1: Array[Int], nums2: Array[Int]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec get_common(nums1 :: [integer], nums2 :: [integer]) :: integer
def get_common(nums1, nums2) do

end
end
```

Erlang:

```
-spec get_common(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
get_common(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (get-common nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Common Value
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int getCommon(vector<int>& nums1, vector<int>& nums2) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Minimum Common Value
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int getCommon(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Common Value
Difficulty: Easy
Tags: array, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def getCommon(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getCommon(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Common Value
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var getCommon = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Common Value
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function getCommon(nums1: number[], nums2: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Common Value
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int GetCommon(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Common Value
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int getCommon(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Minimum Common Value
// Difficulty: Easy
```

```
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func getCommon(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun getCommon(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func getCommon(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Common Value
// Difficulty: Easy
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn get_common(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def get_common(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function getCommon($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int getCommon(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def getCommon(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_common(nums1 :: [integer], nums2 :: [integer]) :: integer
def get_common(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec get_common(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
get_common(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (get-common nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```