Problem 2956: Find Common Elements Between Two Arrays

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

of sizes

n

and

m

, respectively. Calculate the following values:

answer1

: the number of indices

i

such that

nums1[i]

exists in

nums2

.

answer2

: the number of indices

i

such that

nums2[i]

exists in

nums1

.

Return

[answer1,answer2]

.

Example 1:

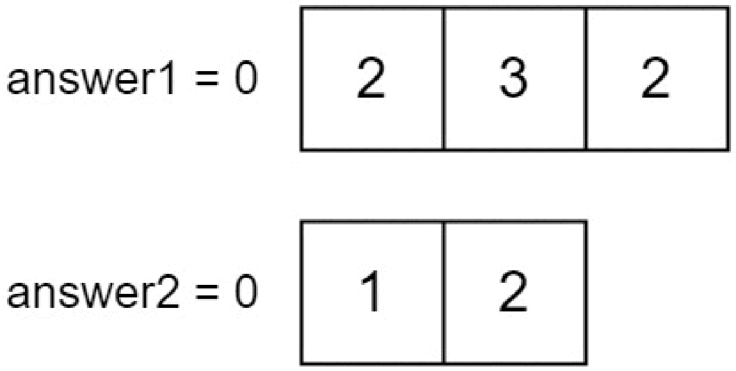
Input:

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

Output:

[2,1]

Explanation:



Example 2:

Input:

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

Output:

[3,4]

Explanation:

The elements at indices 1, 2, and 3 in

nums1

exist in

nums2

as well. So

answer1

is 3.

The elements at indices 0, 1, 3, and 4 in

nums2

exist in

nums1

. So

answer2

is 4.

Example 3:

Input:

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

Output:

[0,0]

Explanation:

No numbers are common between

nums1

and

nums2

, so answer is [0,0].

Constraints:

n == nums1.length

m == nums2.length

1 <= n, m <= 100

1 <= nums1[i], nums2[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findIntersectionValues(int* nums1, int nums1Size, int* nums2, int
nums2Size, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
find_intersection_values(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (find-intersection-values nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Common Elements Between Two Arrays
* Difficulty: Easy
* Tags: array, hash
*
* 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:
vector<int> findIntersectionValues(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Find Common Elements Between Two Arrays
* Difficulty: Easy
* Tags: array, hash
*
* 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[] findIntersectionValues(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Common Elements Between Two Arrays
Difficulty: Easy
Tags: array, hash

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Common Elements Between Two Arrays
* Difficulty: Easy
* Tags: array, hash
*
* 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 findIntersectionValues = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Common Elements Between Two Arrays
* Difficulty: Easy
* Tags: array, hash
*
* 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 findIntersectionValues(nums1: number[], nums2: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Common Elements Between Two Arrays
* Difficulty: Easy
* Tags: array, hash
*
* 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[] FindIntersectionValues(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Find Common Elements Between Two Arrays
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findIntersectionValues(int* nums1, int nums1Size, int* nums2, int
nums2Size, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find Common Elements Between Two Arrays
// Difficulty: Easy
// Tags: array, hash
//
// 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 findIntersectionValues(nums1 []int, nums2 []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Common Elements Between Two Arrays
// Difficulty: Easy
// Tags: array, hash
//
// 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 find_intersection_values(nums1: Vec<i32>, nums2: Vec<i32>) -> Vec<i32>
{

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-intersection-values nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```