Problem 349: Intersection of Two Arrays

Problem Information

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

Problem Description

Given two integer arrays

nums1

and

nums2

, return

an array of their

intersection

. Each element in the result must be

unique

and you may return the result in

any order

.

Example 1:

Input:

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

Output:

[2]

Example 2:

Input:

nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output:

[9,4]

Explanation:

[4,9] is also accepted.

Constraints:

1 <= nums1.length, nums2.length <= 1000

0 <= nums1[i], nums2[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Intersection of Two Arrays
* 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:
vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Intersection of Two Arrays
* 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[] intersection(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Intersection of Two Arrays
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 intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Intersection of Two Arrays
* 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 intersection = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Intersection of Two Arrays
* 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 intersection(nums1: number[], nums2: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Intersection of Two Arrays
* 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[] Intersection(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Intersection of Two Arrays
* 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
*/

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

}
```

Go Solution:

```
// Problem: Intersection of Two Arrays
// 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 intersection(nums1 []int, nums2 []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Intersection of Two Arrays
// 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 intersection(nums1: Vec<i32>, nums2: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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