Problem 697: Degree of an Array

Problem Information

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

Problem Description

Given a non-empty array of non-negative integers

nums

, the

degree

of this array is defined as the maximum frequency of any one of its elements.

Your task is to find the smallest possible length of a (contiguous) subarray of

nums

, that has the same degree as

nums

.

Example 1:

Input:

nums = [1,2,2,3,1]

Output:

2

Explanation:

The input array has a degree of 2 because both elements 1 and 2 appear twice. Of the subarrays that have the same degree: [1, 2, 2, 3, 1], [1, 2, 2, 3], [2, 2, 3, 1], [1, 2, 2], [2, 2, 3], [2, 2] The shortest length is 2. So return 2.

Example 2:

Input:

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

Output:

6

Explanation:

The degree is 3 because the element 2 is repeated 3 times. So [2,2,3,1,4,2] is the shortest subarray, therefore returning 6.

Constraints:

nums.length

will be between 1 and 50,000.

nums[i]

will be an integer between 0 and 49,999.

Code Snippets

C++:

```
class Solution {
public:
int findShortestSubArray(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int findShortestSubArray(int[] nums) {

}
}
```

Python3:

```
class Solution:
def findShortestSubArray(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
def findShortestSubArray(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
var findShortestSubArray = function(nums) {

};
```

TypeScript:

```
function findShortestSubArray(nums: number[]): number {
```

```
};
```

C#:

```
public class Solution {
public int FindShortestSubArray(int[] nums) {

}
}
```

C:

```
int findShortestSubArray(int* nums, int numsSize) {

}
```

Go:

```
func findShortestSubArray(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun findShortestSubArray(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func findShortestSubArray(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_shortest_sub_array(nums: Vec<i32>) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def find_shortest_sub_array(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function findShortestSubArray($nums) {

}
}
```

Dart:

```
class Solution {
int findShortestSubArray(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def findShortestSubArray(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_shortest_sub_array(nums :: [integer]) :: integer
def find_shortest_sub_array(nums) do

end
end
```

Erlang:

```
-spec find_shortest_sub_array(Nums :: [integer()]) -> integer().
find_shortest_sub_array(Nums) ->
.
```

Racket:

```
(define/contract (find-shortest-sub-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Degree of an Array
* 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 findShortestSubArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Degree of an Array
* 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 findShortestSubArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Degree of an Array
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 findShortestSubArray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findShortestSubArray(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Degree of an Array
* 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[]} nums
* @return {number}
*/
var findShortestSubArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Degree of an Array
* 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 findShortestSubArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Degree of an Array
* 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 FindShortestSubArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Degree of an Array
* 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
*/

int findShortestSubArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Degree of an Array
// 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 findShortestSubArray(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findShortestSubArray(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func findShortestSubArray(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Degree of an Array
// 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_shortest_sub_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def find_shortest_sub_array(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function findShortestSubArray($nums) {

}
}
```

Dart Solution:

```
class Solution {
int findShortestSubArray(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def findShortestSubArray(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_shortest_sub_array(nums :: [integer]) :: integer
def find_shortest_sub_array(nums) do

end
end
```

Erlang Solution:

```
-spec find_shortest_sub_array(Nums :: [integer()]) -> integer().
find_shortest_sub_array(Nums) ->
.
```

Racket Solution:

```
(define/contract (find-shortest-sub-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```