Problem 3591: Check if Any Element Has Prime Frequency

Problem Information

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

Problem Description

You are given an integer array

nums

.

Return

true

if the frequency of any element of the array is

prime

, otherwise, return

false

.

The

frequency

of an element

x

is the number of times it occurs in the array.

A prime number is a natural number greater than 1 with only two factors, 1 and itself.

Example 1:

Input:

nums = [1,2,3,4,5,4]

Output:

true

Explanation:

4 has a frequency of two, which is a prime number.

Example 2:

Input:

nums = [1,2,3,4,5]

Output:

false

Explanation:

All elements have a frequency of one.

Example 3:

Input:

nums = [2,2,2,4,4]

Output:

true

Explanation:

Both 2 and 4 have a prime frequency.

Constraints:

1 <= nums.length <= 100

0 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean checkPrimeFrequency(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function checkPrimeFrequency(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CheckPrimeFrequency(int[] nums) {

}
}
```

C:

```
bool checkPrimeFrequency(int* nums, int numsSize) {

}
```

Go:

```
func checkPrimeFrequency(nums []int) bool {
}
```

Kotlin:

```
class Solution {
fun checkPrimeFrequency(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool checkPrimeFrequency(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-prime-frequency nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if Any Element Has Prime Frequency
* Difficulty: Easy
* Tags: array, math, 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:
bool checkPrimeFrequency(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Check if Any Element Has Prime Frequency
* Difficulty: Easy
* Tags: array, math, 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 boolean checkPrimeFrequency(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if Any Element Has Prime Frequency
Difficulty: Easy
Tags: array, math, 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 checkPrimeFrequency(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if Any Element Has Prime Frequency
* Difficulty: Easy
* Tags: array, math, 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 {boolean}
*/
var checkPrimeFrequency = function(nums) {
};
```

TypeScript Solution:

```
/**
* Problem: Check if Any Element Has Prime Frequency
* Difficulty: Easy
* Tags: array, math, 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 checkPrimeFrequency(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if Any Element Has Prime Frequency
* Difficulty: Easy
* Tags: array, math, 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 bool CheckPrimeFrequency(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Check if Any Element Has Prime Frequency
* Difficulty: Easy
* Tags: array, math, 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
*/

bool checkPrimeFrequency(int* nums, int numsSize) {

}
```

Go Solution:

```go
// Problem: Check if Any Element Has Prime Frequency
// Difficulty: Easy
// Tags: array, math, 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 checkPrimeFrequency(nums []int) bool {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun checkPrimeFrequency(nums: IntArray): Boolean {

}
}
```

Swift Solution:

```swift
class Solution {
func checkPrimeFrequency(_ nums: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check if Any Element Has Prime Frequency
// Difficulty: Easy
// Tags: array, math, 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 check_prime_frequency(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool checkPrimeFrequency(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-prime-frequency nums)
(-> (listof exact-integer?) boolean?)
)
```