Problem 1207: Unique Number of Occurrences

Problem Information

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

Problem Description

Given an array of integers

arr

, return

true

if the number of occurrences of each value in the array is

unique

or

false

otherwise

.

Example 1:

Input:

arr = [1,2,2,1,1,3]

Output:

true

Explanation:

The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.

Example 2:

Input:

arr = [1,2]

Output:

false

Example 3:

Input:

arr = [-3,0,1,-3,1,1,1,-3,10,0]

Output:

true

Constraints:

1 <= arr.length <= 1000

-1000 <= arr[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
bool uniqueOccurrences(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public boolean uniqueOccurrences(int[] arr) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function uniqueOccurrences(arr: number[]): boolean {
```

```
};
```

C#:

```
public class Solution {
public bool UniqueOccurrences(int[] arr) {

}
}
```

C:

```
bool uniqueOccurrences(int* arr, int arrSize) {

}
```

Go:

```
func uniqueOccurrences(arr []int) bool {

}
```

Kotlin:

```
class Solution {
fun uniqueOccurrences(arr: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func uniqueOccurrences(_ arr: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn unique_occurrences(arr: Vec<i32>) -> bool {
```

```
}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Boolean}
def unique_occurrences(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Boolean
*/
function uniqueOccurrences($arr) {

}
}
```

Dart:

```
class Solution {
bool uniqueOccurrences(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def uniqueOccurrences(arr: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec unique_occurrences(arr :: [integer]) :: boolean
def unique_occurrences(arr) do

end
end
```

Erlang:

```
-spec unique_occurrences(Arr :: [integer()]) -> boolean().
unique_occurrences(Arr) ->
.
```

Racket:

```
(define/contract (unique-occurrences arr)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unique Number of Occurrences
* 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:
bool uniqueOccurrences(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Unique Number of Occurrences
* 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 boolean uniqueOccurrences(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Unique Number of Occurrences
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 uniqueOccurrences(self, arr: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Unique Number of Occurrences
* 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[]} arr
* @return {boolean}
*/
var uniqueOccurrences = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Unique Number of Occurrences
* 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 uniqueOccurrences(arr: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Unique Number of Occurrences
* 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 bool UniqueOccurrences(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Unique Number of Occurrences
* 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
*/

bool uniqueOccurrences(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Unique Number of Occurrences
// 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 uniqueOccurrences(arr []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun uniqueOccurrences(arr: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func uniqueOccurrences(_ arr: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Unique Number of Occurrences
// 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 unique_occurrences(arr: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Boolean}
def unique_occurrences(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Boolean
*/
function uniqueOccurrences($arr) {

}
}
```

Dart Solution:

```
class Solution {
bool uniqueOccurrences(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def uniqueOccurrences(arr: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec unique_occurrences(arr :: [integer]) :: boolean
def unique_occurrences(arr) do

end
end
```

Erlang Solution:

```
-spec unique_occurrences(Arr :: [integer()]) -> boolean().
unique_occurrences(Arr) ->
.
```

Racket Solution:

```
(define/contract (unique-occurrences arr)
(-> (listof exact-integer?) boolean?)
)
```