Problem 1224: Maximum Equal Frequency

Problem Information

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

Problem Description

Given an array

nums

of positive integers, return the longest possible length of an array prefix of

nums

, such that it is possible to remove

exactly one

element from this prefix so that every number that has appeared in it will have the same number of occurrences.

If after removing one element there are no remaining elements, it's still considered that every appeared number has the same number of ocurrences (0).

Example 1:

Input:

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

Output:

7

Explanation:

For the subarray [2,2,1,1,5,3,3] of length 7, if we remove nums[4] = 5, we will get [2,2,1,1,3,3], so that each number will appear exactly twice.

Example 2:

Input:

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

Output:

13

Constraints:

2 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-equal-freq nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Equal Frequency
* Difficulty: Hard
* 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 maxEqualFreq(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Equal Frequency
* Difficulty: Hard
* 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 maxEqualFreq(int[] nums) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Equal Frequency
Difficulty: Hard
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 maxEqualFreq(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Equal Frequency
* Difficulty: Hard
* 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 maxEqualFreq = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Equal Frequency
* Difficulty: Hard
* 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 maxEqualFreq(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Equal Frequency
* Difficulty: Hard
* 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 MaxEqualFreq(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Maximum Equal Frequency
* Difficulty: Hard
* 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 maxEqualFreq(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Equal Frequency
// Difficulty: Hard
// 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 maxEqualFreq(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Equal Frequency
// Difficulty: Hard
// 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 max_equal_freq(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-equal-freq nums)
(-> (listof exact-integer?) exact-integer?)
)
```