Problem 3005: Count Elements With Maximum Frequency

Problem Information

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

Problem Description

You are given an array

nums

consisting of

positive

integers.

Return

the

total frequencies

of elements in

nums

such that those elements all have the

maximum

frequency

.

The

frequency

of an element is the number of occurrences of that element in the array.

Example 1:

Input:

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

Output:

4

Explanation:

The elements 1 and 2 have a frequency of 2 which is the maximum frequency in the array. So the number of elements in the array with maximum frequency is 4.

Example 2:

Input:

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

Output:

5

Explanation:

All elements of the array have a frequency of 1 which is the maximum. So the number of elements in the array with maximum frequency is 5.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
 * @return {number}
 */
var maxFrequencyElements = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-frequency-elements nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Elements With Maximum Frequency
* 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 maxFrequencyElements(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Elements With Maximum Frequency
* 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 maxFrequencyElements(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Elements With Maximum Frequency
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 maxFrequencyElements(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Elements With Maximum Frequency
* 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 maxFrequencyElements = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Elements With Maximum Frequency
* 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 maxFrequencyElements(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Elements With Maximum Frequency
* 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 MaxFrequencyElements(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Elements With Maximum Frequency
* 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 maxFrequencyElements(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Elements With Maximum Frequency
// 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 maxFrequencyElements(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Elements With Maximum Frequency
// 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 max_frequency_elements(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-frequency-elements nums)
(-> (listof exact-integer?) exact-integer?)
)
```