Problem 1838: Frequency of the Most Frequent Element

Problem Information

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

Problem Description

The

frequency

of an element is the number of times it occurs in an array.

You are given an integer array

nums

and an integer

k

. In one operation, you can choose an index of

nums

and increment the element at that index by

1

.

Return

the

maximum possible frequency

of an element after performing

at most

k

operations

.

Example 1:

Input:

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

Output:

3

Explanation:

Increment the first element three times and the second element two times to make nums = [4,4,4]. 4 has a frequency of 3.

Example 2:

Input:

nums = [1,4,8,13], k = 5

Output:

2

Explanation:

There are multiple optimal solutions: - Increment the first element three times to make nums = [4,4,8,13]. 4 has a frequency of 2. - Increment the second element four times to make nums = [1,8,8,13]. 8 has a frequency of 2. - Increment the third element five times to make nums = [1,4,13,13]. 13 has a frequency of 2.

Example 3:

Input:

nums = [3,9,6], k = 2

Output:

1

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

1 <= k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int maxFrequency(vector<int>& nums, int k) {



}
};
```

Java:

```
class Solution {
public int maxFrequency(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxFrequency(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxFrequency(int[] nums, int k) {

}
}
```

C:

```
int maxFrequency(int* nums, int numsSize, int k) {

}
```

Go:

```
func maxFrequency(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxFrequency(nums: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxFrequency(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_frequency(nums: Vec<i32>, k: i32) -> i32 {
}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_frequency(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxFrequency($nums, $k) {

}
}
```

Dart:

```
class Solution {
int maxFrequency(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def maxFrequency(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_frequency(nums :: [integer], k :: integer) :: integer
def max_frequency(nums, k) do

end
end
```

Erlang:

```
-spec max_frequency(Nums :: [integer()], K :: integer()) -> integer().
max_frequency(Nums, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Frequency of the Most Frequent Element
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int maxFrequency(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Frequency of the Most Frequent Element
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maxFrequency(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Frequency of the Most Frequent Element
Difficulty: Medium
Tags: array, greedy, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxFrequency(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Frequency of the Most Frequent Element
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var maxFrequency = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Frequency of the Most Frequent Element
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxFrequency(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Frequency of the Most Frequent Element
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaxFrequency(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Frequency of the Most Frequent Element
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int maxFrequency(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Frequency of the Most Frequent Element
// Difficulty: Medium
// Tags: array, greedy, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func maxFrequency(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxFrequency(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxFrequency(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Frequency of the Most Frequent Element
// Difficulty: Medium
// Tags: array, greedy, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_frequency(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
```

```
def max_frequency(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxFrequency($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxFrequency(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxFrequency(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_frequency(nums :: [integer], k :: integer) :: integer
def max_frequency(nums, k) do

end
```

```
end
```

Erlang Solution:

```
-spec max_frequency(Nums :: [integer()], K :: integer()) -> integer().
max_frequency(Nums, K) ->
.
```

Racket Solution:

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