Problem 2968: Apply Operations to Maximize Frequency Score

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and an integer

k

.

You can perform the following operation on the array

at most

k

times:

Choose any index

i

from the array and

increase

or

decrease

nums[i]

by

1

.

The score of the final array is the

frequency

of the most frequent element in the array.

Return

the

maximum

score you can achieve

.

The frequency of an element is the number of occurences of that element in the array.

Example 1:

Input:

nums = [1,2,6,4], k = 3

Output:

3

Explanation:

We can do the following operations on the array: - Choose i = 0, and increase the value of nums[0] by 1. The resulting array is [2,2,6,4]. - Choose i = 3, and decrease the value of nums[3] by 1. The resulting array is [2,2,6,3]. - Choose i = 3, and decrease the value of nums[3] by 1. The resulting array is [2,2,6,2]. The element 2 is the most frequent in the final array so our score is 3. It can be shown that we cannot achieve a better score.

Example 2:

Input:

nums = [1,4,4,2,4], k = 0

Output:

3

Explanation:

We cannot apply any operations so our score will be the frequency of the most frequent element in the original array, which is 3.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

0 <= k <= 10

14

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maxFrequencyScore(int* nums, int numsSize, long long k) {

}
```

Go:

```
func maxFrequencyScore(nums []int, k int64) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_frequency_score(nums: Vec<i32>, k: i64) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Apply Operations to Maximize Frequency Score
* Difficulty: Hard
```

```
* Tags: array, 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 maxFrequencyScore(vector<int>& nums, long long k) {

}
};
```

Java Solution:

```
/**
* Problem: Apply Operations to Maximize Frequency Score
* Difficulty: Hard
* Tags: array, 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 maxFrequencyScore(int[] nums, long k) {

}
}
```

Python3 Solution:

```
"""
Problem: Apply Operations to Maximize Frequency Score
Difficulty: Hard
Tags: array, 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 maxFrequencyScore(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Apply Operations to Maximize Frequency Score
* Difficulty: Hard
* Tags: array, 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 maxFrequencyScore = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Apply Operations to Maximize Frequency Score
* Difficulty: Hard
* Tags: array, 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 maxFrequencyScore(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Apply Operations to Maximize Frequency Score
* Difficulty: Hard
* Tags: array, 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 MaxFrequencyScore(int[] nums, long k) {

}
}
```

C Solution:

```
/*
* Problem: Apply Operations to Maximize Frequency Score
* Difficulty: Hard
* Tags: array, 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 maxFrequencyScore(int* nums, int numsSize, long long k) {

}
```

Go Solution:

```
// Problem: Apply Operations to Maximize Frequency Score
// Difficulty: Hard
// Tags: array, 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 maxFrequencyScore(nums []int, k int64) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Apply Operations to Maximize Frequency Score
// Difficulty: Hard
// Tags: array, 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_score(nums: Vec<i32>, k: i64) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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