Problem 2524: Maximum Frequency Score of a Subarray

Problem Information

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

Problem Description

You are given an integer array

nums

and a

positive

integer

k

.

The

frequency score

of an array is the sum of the

distinct

values in the array raised to the power of their

frequencies

, taking the sum

modulo

10

9

+ 7

.

For example, the frequency score of the array

[5,4,5,7,4,4]

is

(4

3

+ 5

2

+ 7

1

) modulo (10

9

+ 7) = 96

.

Return

the

maximum

frequency score of a

subarray

of size

k

in

nums

. You should maximize the value under the modulo and not the actual value.

A

subarray

is a contiguous part of an array.

Example 1:

Input:

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

Output:

5

Explanation:

The subarray [2,1,2] has a frequency score equal to 5. It can be shown that it is the maximum frequency score we can have.

Example 2:

Input:

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

Output:

1

Explanation:

All the subarrays of length 4 have a frequency score equal to 1.

Constraints:

1 <= k <= nums.length <= 10

5

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_frequency_score(nums: Vec<i32>, k: i32) -> 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: Int): 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: Maximum Frequency Score of a Subarray
* Difficulty: Hard
* Tags: array, math, hash, stack
*
* 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 maxFrequencyScore(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Frequency Score of a Subarray
```

```
* Difficulty: Hard
* Tags: array, math, hash, stack
*
* 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 maxFrequencyScore(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Frequency Score of a Subarray
Difficulty: Hard
Tags: array, math, hash, stack

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 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: Maximum Frequency Score of a Subarray
* Difficulty: Hard
* Tags: array, math, hash, stack
*
* 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
* @param {number} k
* @return {number}
*/
var maxFrequencyScore = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Frequency Score of a Subarray
* Difficulty: Hard
* Tags: array, math, hash, stack
*
* 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 maxFrequencyScore(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Frequency Score of a Subarray
* Difficulty: Hard
* Tags: array, math, hash, stack
```

```
*
* 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 MaxFrequencyScore(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Frequency Score of a Subarray
* Difficulty: Hard
* Tags: array, math, hash, stack
*
* 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 maxFrequencyScore(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Frequency Score of a Subarray
// Difficulty: Hard
// Tags: array, math, hash, stack
//
// 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 maxFrequencyScore(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Frequency Score of a Subarray
// Difficulty: Hard
// Tags: array, math, hash, stack
//
// 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_score(nums: Vec<i32>, k: i32) -> 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: Int): 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?)
)
```