Problem 3743: Maximize Cyclic Partition Score

Problem Information

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

Problem Description

You are given a

cyclic

array

nums

and an integer

k

.

Partition

nums

into

at most

k

subarrays

. As

nums

is cyclic, these subarrays may wrap around from the end of the array back to the beginning.

The

range

of a subarray is the difference between its

maximum

and

minimum

values. The

score

of a partition is the sum of subarray

ranges

.

Return the

maximum

possible

score

among all cyclic partitions.

Example 1:

Input:

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

Output:

3

Explanation:

Partition

nums

into

[2, 3]

and

[3, 1]

(wrapped around).

The range of

[2, 3]

is

max(2, 3) - min(2, 3) = 3 - 2 = 1

.

The range of

[3, 1]

is

max(3, 1) - min(3, 1) = 3 - 1 = 2

.

The score is

1 + 2 = 3

.

Example 2:

Input:

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

Output:

2

Explanation:

Partition

nums

into

[1, 2, 3, 3]

.

The range of

[1, 2, 3, 3]

is

max(1, 2, 3, 3) - min(1, 2, 3, 3) = 3 - 1 = 2

.

The score is 2.

Example 3:

Input:

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

Output:

3

Explanation:

Identical to Example 1, we partition

nums

into

[2, 3]

and

[3, 1]

. Note that

nums

may be partitioned into fewer than

k

subarrays.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

9

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maximumScore(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximize Cyclic Partition Score
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Maximize Cyclic Partition Score
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Cyclic Partition Score
```

```
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Cyclic Partition Score
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

TypeScript Solution:

```
/**
* Problem: Maximize Cyclic Partition Score
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Maximize Cyclic Partition Score
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Maximize Cyclic Partition Score
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long maximumScore(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximize Cyclic Partition Score
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maximumScore(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Cyclic Partition Score
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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