Problem 3077: Maximum Strength of K Disjoint Subarrays

Problem Information

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

Problem Description

You are given an array of integers

nums

with length

n

, and a positive

odd

integer

k

.

Select exactly

k

disjoint

subarrays

sub

1

, sub

2

, ..., sub

k

from

nums

such that the last element of

sub

i

appears before the first element of

sub

{i+1}

for all

1 <= i <= k-1

. The goal is to maximize their combined strength.

The strength of the selected subarrays is defined as:

strength = k * sum(sub

1

)- (k - 1) * sum(sub

2

) + (k - 2) * sum(sub

3

) - ... - 2 * sum(sub

{k-1}

) + sum(sub

k

)

where

sum(sub

i

)

is the sum of the elements in the

i

-th subarray.

Return the

maximum

possible strength that can be obtained from selecting exactly

k

disjoint subarrays from

nums

.

Note

that the chosen subarrays

don't

need to cover the entire array.

Example 1:

Input:

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

Output:

22

Explanation:

The best possible way to select 3 subarrays is: nums[0..2], nums[3..3], and nums[4..4]. The strength is calculated as follows:

strength = 3 * (1 + 2 + 3) - 2 * (-1) + 2 = 22

Example 2:

Input:

nums = [12,-2,-2,-2,-2], k = 5

Output:

64

Explanation:

The only possible way to select 5 disjoint subarrays is: nums[0..0], nums[1..1], nums[2..2], nums[3..3], and nums[4..4]. The strength is calculated as follows:

strength = 5 * 12 - 4 * (-2) + 3 * (-2) - 2 * (-2) + (-2) = 64

Example 3:

Input:

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

1

Output:

-1

Explanation:

The best possible way to select 1 subarray is: nums[0..0]. The strength is -1.

Constraints:

1 <= n <= 10

4

-10

9

<= nums[i] <= 10

9

1 <= k <= n

1 <= n * k <= 10

6

k

is odd.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Strength of K Disjoint Subarrays
* 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 maximumStrength(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Strength of K Disjoint Subarrays
* 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 maximumStrength(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Strength of K Disjoint Subarrays
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Strength of K Disjoint Subarrays
* 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 maximumStrength = function(nums, k) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Strength of K Disjoint Subarrays
* 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 maximumStrength(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Strength of K Disjoint Subarrays
* 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 MaximumStrength(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Strength of K Disjoint Subarrays
```

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

}
```

Go Solution:

```
// Problem: Maximum Strength of K Disjoint Subarrays
// 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 maximumStrength(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Strength of K Disjoint Subarrays
// 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_strength(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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