Problem 1793: Maximum Score of a Good Subarray

Problem Information

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

Problem Description

You are given an array of integers

nums

(0-indexed)

and an integer

k

.

The

score

of a subarray

(i, j)

is defined as

min(nums[i], nums[i+1], ..., nums[j]) * (j - i + 1)

. A

good

subarray is a subarray where

i <= k <= j

.

Return

the maximum possible

score

of a

good

subarray.

Example 1:

Input:

nums = [1,4,3,7,4,5], k = 3

Output:

15

Explanation:

The optimal subarray is (1, 5) with a score of min(4,3,7,4,5) * (5-1+1) = 3 * 5 = 15.

Example 2:

Input:

nums = [5,5,4,5,4,1,1,1], k = 0

Output:

20

Explanation:

The optimal subarray is (0, 4) with a score of min(5,5,4,5,4) * (4-0+1) = 4 * 5 = 20.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 2 * 10

4

0 <= k < nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

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): Int = {

}
}
```

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: Maximum Score of a Good Subarray
* Difficulty: Hard
* Tags: array, search, stack
*
* 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 maximumScore(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score of a Good Subarray
* Difficulty: Hard
* Tags: array, search, stack
*
* 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 maximumScore(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score of a Good Subarray
Difficulty: Hard
Tags: array, search, stack

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 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: Maximum Score of a Good Subarray
* Difficulty: Hard
* Tags: array, search, stack
```

```
*
* 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 maximumScore = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score of a Good Subarray
* Difficulty: Hard
* Tags: array, search, stack
*
* 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 maximumScore(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Score of a Good Subarray
* Difficulty: Hard
* Tags: array, search, stack
*
* 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 MaximumScore(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score of a Good Subarray
* Difficulty: Hard
* Tags: array, search, stack
*
* 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 maximumScore(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Score of a Good Subarray
// Difficulty: Hard
// Tags: array, search, stack
//
// 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 maximumScore(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Score of a Good Subarray
// Difficulty: Hard
// Tags: array, search, stack
//
// 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 maximum_score(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

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): Int = {

}
}
```

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?)
)
```