Problem 1770: Maximum Score from Performing Multiplication Operations

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

nums

and

multipliers

of size

n

and

m

respectively, where

n >= m

.

You begin with a score of

0

. You want to perform

exactly

m

operations. On the

i

th

operation (

0-indexed

) you will:

Choose one integer

x

from

either the start or the end

of the array

nums

.

Add

multipliers[i] * x

to your score.

Note that

multipliers[0]

corresponds to the first operation,

multipliers[1]

to the second operation, and so on.

Remove

x

from

nums

.

Return

the

maximum

score after performing

m

operations.

Example 1:

Input:

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

Output:

14

Explanation:

An optimal solution is as follows: - Choose from the end, [1,2,

3

], adding 3 * 3 = 9 to the score. - Choose from the end, [1,

2

], adding 2 * 2 = 4 to the score. - Choose from the end, [

1

], adding 1 * 1 = 1 to the score. The total score is 9 + 4 + 1 = 14.

Example 2:

Input:

nums = [-5,-3,-3,-2,7,1], multipliers = [-10,-5,3,4,6]

Output:

102

Explanation:

An optimal solution is as follows: - Choose from the start, [

-5

,-3,-3,-2,7,1], adding -5 * -10 = 50 to the score. - Choose from the start, [

-3

,-3,-2,7,1], adding -3 * -5 = 15 to the score. - Choose from the start, [

-3

,-2,7,1], adding -3 * 3 = -9 to the score. - Choose from the end, [-2,7,

1

], adding 1 * 4 = 4 to the score. - Choose from the end, [-2,

7

], adding 7 * 6 = 42 to the score. The total score is 50 + 15 - 9 + 4 + 42 = 102.

Constraints:

n == nums.length

m == multipliers.length

1 <= m <= 300

m <= n <= 10

5

-1000 <= nums[i], multipliers[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
```

```
int maximumScore(vector<int>& nums, vector<int>& multipliers) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maximumScore(int* nums, int numsSize, int* multipliers, int
multipliersSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score from Performing Multiplication Operations
* 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:
```

```
int maximumScore(vector<int>& nums, vector<int>& multipliers) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score from Performing Multiplication Operations
* 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 int maximumScore(int[] nums, int[] multipliers) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score from Performing Multiplication Operations
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], multipliers: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Score from Performing Multiplication Operations
* 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[]} multipliers
* @return {number}
*/
var maximumScore = function(nums, multipliers) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score from Performing Multiplication Operations
* 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[], multipliers: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Maximum Score from Performing Multiplication Operations
* 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 int MaximumScore(int[] nums, int[] multipliers) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score from Performing Multiplication Operations
* 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
*/

int maximumScore(int* nums, int numsSize, int* multipliers, int
multipliersSize) {

}
```

Go Solution:

```
// Problem: Maximum Score from Performing Multiplication Operations
// 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, multipliers []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Score from Performing Multiplication Operations
// 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>, multipliers: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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