Problem 3282: Reach End of Array With Max Score

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

.

Your goal is to start at index

0

and reach index

n - 1

. You can only jump to indices

greater

than your current index.

The score for a jump from index

i

to index

j

is calculated as

(j - i) * nums[i]

.

Return the

maximum

possible

total score

by the time you reach the last index.

Example 1:

Input:

nums = [1,3,1,5]

Output:

7

Explanation:

First, jump to index 1 and then jump to the last index. The final score is

1 * 1 + 2 * 3 = 7

.

Example 2:

Input:

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

Output:

16

Explanation:

Jump directly to the last index. The final score is

4 * 4 = 16

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long findMaximumScore(vector<int>& nums) {
}
};
```

Java:

```
class Solution {
public long findMaximumScore(List<Integer> nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findMaximumScore(nums: number[]): number {

};
```

C#:

```
public class Solution {
public long FindMaximumScore(IList<int> nums) {

}
}
```

C:

```
long long findMaximumScore(int* nums, int numsSize) {

}
```

Go:

```
func findMaximumScore(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun findMaximumScore(nums: List<Int>): Long {

}
}
```

Swift:

```
class Solution {
func findMaximumScore(_ nums: [Int]) -> Int {

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def find_maximum_score(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function findMaximumScore($nums) {

}
}
```

Dart:

```
class Solution {
int findMaximumScore(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def findMaximumScore(nums: List[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_maximum_score(nums :: [integer]) :: integer
def find_maximum_score(nums) do

end
end
```

Erlang:

```
-spec find_maximum_score(Nums :: [integer()]) -> integer().
find_maximum_score(Nums) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Reach End of Array With Max Score
* Difficulty: Medium
* Tags: array, greedy
*
* 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:
long long findMaximumScore(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Reach End of Array With Max Score
* Difficulty: Medium
* Tags: array, greedy
*
* 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 long findMaximumScore(List<Integer> nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Reach End of Array With Max Score
Difficulty: Medium
Tags: array, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reach End of Array With Max Score
* Difficulty: Medium
* Tags: array, greedy
*
* 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
* @return {number}
*/
var findMaximumScore = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Reach End of Array With Max Score
* Difficulty: Medium
* Tags: array, greedy
*
* 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 findMaximumScore(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Reach End of Array With Max Score
* Difficulty: Medium
* Tags: array, greedy
*
* 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 long FindMaximumScore(IList<int> nums) {

}
}
```

C Solution:

```
/*
* Problem: Reach End of Array With Max Score
* Difficulty: Medium
* Tags: array, greedy
*
* 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
*/

long long findMaximumScore(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Reach End of Array With Max Score
// Difficulty: Medium
// Tags: array, greedy
//
// 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 findMaximumScore(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun findMaximumScore(nums: List<Int>): Long {

}
}
```

Swift Solution:

```
class Solution {
func findMaximumScore(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Reach End of Array With Max Score
// Difficulty: Medium
// Tags: array, greedy
//
// 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 find_maximum_score(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def find_maximum_score(nums)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Integer
*/
function findMaximumScore($nums) {

}
}
```

Dart Solution:

```
class Solution {
int findMaximumScore(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def findMaximumScore(nums: List[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_maximum_score(nums :: [integer]) :: integer
def find_maximum_score(nums) do

end
end
```

Erlang Solution:

```
-spec find_maximum_score(Nums :: [integer()]) -> integer().
find_maximum_score(Nums) ->
.
```

Racket Solution:

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