Problem 2708: Maximum Strength of a Group

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

representing the score of students in an exam. The teacher would like to form one

non-empty

group of students with maximal

strength

, where the strength of a group of students of indices

i

0

,

i

1

,

i

2

, ... ,

i

k

is defined as

nums[i

0

] * nums[i

1

] * nums[i

2

] * ... * nums[i

k

]

.

Return

the maximum strength of a group the teacher can create

.

Example 1:

Input:

nums = [3,-1,-5,2,5,-9]

Output:

1350

Explanation:

One way to form a group of maximal strength is to group the students at indices [0,2,3,4,5]. Their strength is 3 * (-5) * 2 * 5 * (-9) = 1350, which we can show is optimal.

Example 2:

Input:

nums = [-4,-5,-4]

Output:

20

Explanation:

Group the students at indices [0, 1] . Then, we’ll have a resulting strength of 20. We cannot achieve greater strength.

Constraints:

1 <= nums.length <= 13

-9 <= nums[i] <= 9

Code Snippets

C++:

```
class Solution {
public:
long long maxStrength(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public long maxStrength(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

```
}
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaxStrength(int[] nums) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun maxStrength(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def maxStrength(nums: Array[Int]): Long = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Strength of a Group
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* 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 maxStrength(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Strength of a Group
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* 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 maxStrength(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Strength of a Group
Difficulty: Medium
Tags: array, dp, greedy, sort

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Strength of a Group
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* 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
* @return {number}
*/
var maxStrength = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Strength of a Group
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* 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 maxStrength(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Strength of a Group
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* 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 MaxStrength(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Strength of a Group
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* 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 maxStrength(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Strength of a Group
// Difficulty: Medium
// Tags: array, dp, greedy, sort
//
// 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 maxStrength(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxStrength(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Strength of a Group
// Difficulty: Medium
// Tags: array, dp, greedy, sort
//
// 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 max_strength(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def maxStrength(nums: Array[Int]): Long = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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