Problem 2281: Sum of Total Strength of Wizards

Problem Information

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

Problem Description

As the ruler of a kingdom, you have an army of wizards at your command.

You are given a

0-indexed

integer array

strength

, where

strength[i]

denotes the strength of the

i

th

wizard. For a

contiguous

group of wizards (i.e. the wizards' strengths form a subarray

of

strength

), the

total strength

is defined as the

product

of the following two values:

The strength of the

weakest

wizard in the group.

The

total

of all the individual strengths of the wizards in the group.

Return

the

sum

of the total strengths of

all

contiguous groups of wizards

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

A

subarray

is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

strength = [1,3,1,2]

Output:

44

Explanation:

The following are all the contiguous groups of wizards: - [1] from [

1

,3,1,2] has a total strength of min([1]) * sum([1]) = 1 * 1 = 1 - [3] from [1,

3

,1,2] has a total strength of min([3]) * sum([3]) = 3 * 3 = 9 - [1] from [1,3,

1

,2] has a total strength of min([1]) * sum([1]) = 1 * 1 = 1 - [2] from [1,3,1,

2

] has a total strength of min([2]) * sum([2]) = 2 * 2 = 4 - [1,3] from [

1,3

,1,2] has a total strength of min([1,3]) * sum([1,3]) = 1 * 4 = 4 - [3,1] from [1,

3,1

,2] has a total strength of min([3,1]) * sum([3,1]) = 1 * 4 = 4 - [1,2] from [1,3,

1,2

] has a total strength of min([1,2]) * sum([1,2]) = 1 * 3 = 3 - [1,3,1] from [

1,3,1

,2] has a total strength of min([1,3,1]) * sum([1,3,1]) = 1 * 5 = 5 - [3,1,2] from [1,

3,1,2

] has a total strength of min([3,1,2]) * sum([3,1,2]) = 1 * 6 = 6 - [1,3,1,2] from [

1,3,1,2

] has a total strength of min([1,3,1,2]) * sum([1,3,1,2]) = 1 * 7 = 7 The sum of all the total strengths is 1 + 9 + 1 + 4 + 4 + 4 + 3 + 5 + 6 + 7 = 44.

Example 2:

Input:

strength = [5,4,6]

Output:

213

Explanation:

The following are all the contiguous groups of wizards: - [5] from [

5

,4,6] has a total strength of min([5]) * sum([5]) = 5 * 5 = 25 - [4] from [5,

4

,6] has a total strength of min([4]) * sum([4]) = 4 * 4 = 16 - [6] from [5,4,

6

] has a total strength of min([6]) * sum([6]) = 6 * 6 = 36 - [5,4] from [

5,4

,6] has a total strength of min([5,4]) * sum([5,4]) = 4 * 9 = 36 - [4,6] from [5,

4,6

] has a total strength of min([4,6]) * sum([4,6]) = 4 * 10 = 40 - [5,4,6] from [

5,4,6

] has a total strength of min([5,4,6]) * sum([5,4,6]) = 4 * 15 = 60 The sum of all the total strengths is 25 + 16 + 36 + 36 + 40 + 60 = 213.

Constraints:

1 <= strength.length <= 10

5

1 <= strength[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int totalStrength(vector<int>& strength) {

}
};
```

Java:

```
class Solution {
public int totalStrength(int[] strength) {

}
}
```

Python3:

```
class Solution:
def totalStrength(self, strength: List[int]) -> int:
```

Python:

```
class Solution(object):
def totalStrength(self, strength):
"""
:type strength: List[int]
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function totalStrength(strength: number[]): number {

};
```

C#:

```
public class Solution {
public int TotalStrength(int[] strength) {

}
}
```

C:

```
int totalStrength(int* strength, int strengthSize) {

}
```

Go:

```
func totalStrength(strength []int) int {

}
```

Kotlin:

```
class Solution {
fun totalStrength(strength: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func totalStrength(_ strength: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn total_strength(strength: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} strength
# @return {Integer}
def total_strength(strength)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $strength
* @return Integer
*/
function totalStrength($strength) {

}
}
```

Dart:

```
class Solution {
int totalStrength(List<int> strength) {

}
}
```

Scala:

```
object Solution {
def totalStrength(strength: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec total_strength(strength :: [integer]) :: integer
def total_strength(strength) do

end
end
```

Erlang:

```
-spec total_strength(Strength :: [integer()]) -> integer().
total_strength(Strength) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sum of Total Strength of Wizards
* Difficulty: Hard
* Tags: array, 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 totalStrength(vector<int>& strength) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Total Strength of Wizards
* Difficulty: Hard
* Tags: array, 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 totalStrength(int[] strength) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Total Strength of Wizards
Difficulty: Hard
Tags: array, 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 totalStrength(self, strength: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Total Strength of Wizards
* Difficulty: Hard
* Tags: array, 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[]} strength
* @return {number}
*/
var totalStrength = function(strength) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Total Strength of Wizards
* Difficulty: Hard
* Tags: array, 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 totalStrength(strength: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Total Strength of Wizards
* Difficulty: Hard
* Tags: array, 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 TotalStrength(int[] strength) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Total Strength of Wizards
* Difficulty: Hard
* Tags: array, 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 totalStrength(int* strength, int strengthSize) {

}
```

Go Solution:

```
// Problem: Sum of Total Strength of Wizards
// Difficulty: Hard
// Tags: array, 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 totalStrength(strength []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun totalStrength(strength: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func totalStrength(_ strength: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Total Strength of Wizards
// Difficulty: Hard
// Tags: array, 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 total_strength(strength: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} strength
# @return {Integer}
def total_strength(strength)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $strength
* @return Integer
*/
function totalStrength($strength) {

}
}
```

Dart Solution:

```
class Solution {
int totalStrength(List<int> strength) {

}
}
```

Scala Solution:

```
object Solution {
def totalStrength(strength: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec total_strength(strength :: [integer]) :: integer
def total_strength(strength) do

end
end
```

Erlang Solution:

```
-spec total_strength(Strength :: [integer()]) -> integer().
total_strength(Strength) ->
.
```

Racket Solution:

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