Problem 2300: Successful Pairs of Spells and Potions

Problem Information

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

Problem Description

You are given two positive integer arrays

spells

and

potions

, of length

n

and

m

respectively, where

spells[i]

represents the strength of the

i

th

spell and

potions[j]

represents the strength of the

j

th

potion.

You are also given an integer

success

. A spell and potion pair is considered

successful

if the

product

of their strengths is

at least

success

.

Return

an integer array

pairs

of length

n

where

pairs[i]

is the number of

potions

that will form a successful pair with the

i

th

spell.

Example 1:

Input:

spells = [5,1,3], potions = [1,2,3,4,5], success = 7

Output:

[4,0,3]

Explanation:

- 0

th

spell: 5 * [1,2,3,4,5] = [5,

10

,

15

,

20

,

25

]. 4 pairs are successful. - 1

st

spell: 1 * [1,2,3,4,5] = [1,2,3,4,5]. 0 pairs are successful. - 2

nd

spell: 3 * [1,2,3,4,5] = [3,6,

9

,

12

,

15

]. 3 pairs are successful. Thus, [4,0,3] is returned.

Example 2:

Input:

spells = [3,1,2], potions = [8,5,8], success = 16

Output:

[2,0,2]

Explanation:

- 0

th

spell: 3 * [8,5,8] = [

24

,15,

24

]. 2 pairs are successful. - 1

st

spell: 1 * [8,5,8] = [8,5,8]. 0 pairs are successful. - 2

nd

spell: 2 * [8,5,8] = [

16

,10,

16

]. 2 pairs are successful. Thus, [2,0,2] is returned.

Constraints:

n == spells.length

m == potions.length

1 <= n, m <= 10

5

1 <= spells[i], potions[i] <= 10

5

1 <= success <= 10

10

Code Snippets

C++:

```
class Solution {
public:
vector<int> successfulPairs(vector<int>& spells, vector<int>& potions, long
long success) {

}
};
```

Java:

```
class Solution {
public int[] successfulPairs(int[] spells, int[] potions, long success) {

}
}
```

Python3:

```
class Solution:
def successfulPairs(self, spells: List[int], potions: List[int], success:
int) -> List[int]:
```

Python:

```
class Solution(object):
def successfulPairs(self, spells, potions, success):
"""
:type spells: List[int]
:type potions: List[int]
:type success: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} spells
* @param {number[]} potions
* @param {number} success
* @return {number[]}
*/
var successfulPairs = function(spells, potions, success) {

};
```

TypeScript:

```
function successfulPairs(spells: number[], potions: number[], success:
number): number[] {

};
```

C#:

```
public class Solution {
public int[] SuccessfulPairs(int[] spells, int[] potions, long success) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* successfulPairs(int* spells, int spellsSize, int* potions, int
potionsSize, long long success, int* returnSize) {

}
```

Go:

```
func successfulPairs(spells []int, potions []int, success int64) []int {

}
```

Kotlin:

```
class Solution {
fun successfulPairs(spells: IntArray, potions: IntArray, success: Long):
IntArray {

}
}
```

Swift:

```
class Solution {
func successfulPairs(_ spells: [Int], _ potions: [Int], _ success: Int) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn successful_pairs(spells: Vec<i32>, potions: Vec<i32>, success: i64) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} spells
# @param {Integer[]} potions
# @param {Integer} success
# @return {Integer[]}
def successful_pairs(spells, potions, success)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $spells
* @param Integer[] $potions
* @param Integer $success
* @return Integer[]
*/
function successfulPairs($spells, $potions, $success) {

}
}
```

Dart:

```
class Solution {
List<int> successfulPairs(List<int> spells, List<int> potions, int success) {

}
}
```

Scala:

```
object Solution {
def successfulPairs(spells: Array[Int], potions: Array[Int], success: Long):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec successful_pairs(spells :: [integer], potions :: [integer], success ::
integer) :: [integer]
def successful_pairs(spells, potions, success) do

end
end
```

Erlang:

```
-spec successful_pairs(Spells :: [integer()], Potions :: [integer()], Success
:: integer()) -> [integer()].
successful_pairs(Spells, Potions, Success) ->
.
```

Racket:

```
(define/contract (successful-pairs spells potions success)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Successful Pairs of Spells and Potions
* Difficulty: Medium
* Tags: array, sort, search
*
* 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:
vector<int> successfulPairs(vector<int>& spells, vector<int>& potions, long
long success) {
}
};
```

Java Solution:

```
/**
* Problem: Successful Pairs of Spells and Potions
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[] successfulPairs(int[] spells, int[] potions, long success) {

}
}
```

Python3 Solution:

```
"""
Problem: Successful Pairs of Spells and Potions
Difficulty: Medium
Tags: array, sort, search

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 successfulPairs(self, spells: List[int], potions: List[int], success:
int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def successfulPairs(self, spells, potions, success):
"""
:type spells: List[int]
:type potions: List[int]
:type success: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Successful Pairs of Spells and Potions
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[]} spells
* @param {number[]} potions
* @param {number} success
* @return {number[]}
*/
var successfulPairs = function(spells, potions, success) {

};
```

TypeScript Solution:

```
/**
* Problem: Successful Pairs of Spells and Potions
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 successfulPairs(spells: number[], potions: number[], success:
number): number[] {

};
```

C# Solution:

```
/*
* Problem: Successful Pairs of Spells and Potions
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[] SuccessfulPairs(int[] spells, int[] potions, long success) {

}
}
```

C Solution:

```
/*
* Problem: Successful Pairs of Spells and Potions
* Difficulty: Medium
* Tags: array, sort, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* successfulPairs(int* spells, int spellsSize, int* potions, int
potionsSize, long long success, int* returnSize) {
}
```

Go Solution:

```
// Problem: Successful Pairs of Spells and Potions
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 successfulPairs(spells []int, potions []int, success int64) []int {

}
```

Kotlin Solution:

```
class Solution {
fun successfulPairs(spells: IntArray, potions: IntArray, success: Long):
IntArray {

}
}
```

Swift Solution:

```
class Solution {
func successfulPairs(_ spells: [Int], _ potions: [Int], _ success: Int) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Successful Pairs of Spells and Potions
// Difficulty: Medium
// Tags: array, sort, search
//
```

```
// 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 successful_pairs(spells: Vec<i32>, potions: Vec<i32>, success: i64) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} spells
# @param {Integer[]} potions
# @param {Integer} success
# @return {Integer[]}
def successful_pairs(spells, potions, success)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $spells
* @param Integer[] $potions
* @param Integer $success
* @return Integer[]
*/
function successfulPairs($spells, $potions, $success) {

}
}
```

Dart Solution:

```
class Solution {
List<int> successfulPairs(List<int> spells, List<int> potions, int success) {
}
}
```

Scala Solution:

```
object Solution {
def successfulPairs(spells: Array[Int], potions: Array[Int], success: Long):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec successful_pairs(spells :: [integer], potions :: [integer], success ::
integer) :: [integer]
def successful_pairs(spells, potions, success) do

end
end
```

Erlang Solution:

```
-spec successful_pairs(Spells :: [integer()], Potions :: [integer()], Success
:: integer()) -> [integer()].
successful_pairs(Spells, Potions, Success) ->
.
```

Racket Solution:

```
(define/contract (successful-pairs spells potions success)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
exact-integer?))
)
```