Problem 3577: Count the Number of Computer Unlocking Permutations

Problem Information

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

Problem Description

You are given an array

complexity

of length

n

.

There are

n

locked

computers in a room with labels from 0 to

n - 1

, each with its own

unique

password. The password of the computer

i

has a complexity

complexity[i]

.

The password for the computer labeled 0 is

already

decrypted and serves as the root. All other computers must be unlocked using it or another previously unlocked computer, following this information:

You can decrypt the password for the computer

i

using the password for computer

j

, where

j

is

any

integer less than

i

with a lower complexity. (i.e.

j < i

and

complexity[j] < complexity[i]

)

To decrypt the password for computer

i

, you must have already unlocked a computer

j

such that

j < i

and

complexity[j] < complexity[i]

.

Find the number of

permutations

of

[0, 1, 2, ..., (n - 1)]

that represent a valid order in which the computers can be unlocked, starting from computer 0 as the only initially unlocked one.

Since the answer may be large, return it

modulo

10

9

+ 7.

Note

that the password for the computer

with label

0 is decrypted, and

not

the computer with the first position in the permutation.

Example 1:

Input:

complexity = [1,2,3]

Output:

2

Explanation:

The valid permutations are:

[0, 1, 2]

Unlock computer 0 first with root password.

Unlock computer 1 with password of computer 0 since

complexity[0] < complexity[1]

.

Unlock computer 2 with password of computer 1 since

complexity[1] < complexity[2]

.

[0, 2, 1]

Unlock computer 0 first with root password.

Unlock computer 2 with password of computer 0 since

complexity[0] < complexity[2]

.

Unlock computer 1 with password of computer 0 since

complexity[0] < complexity[1]

.

Example 2:

Input:

complexity = [3,3,3,4,4,4]

Output:

0

Explanation:

There are no possible permutations which can unlock all computers.

Constraints:

2 <= complexity.length <= 10

5

1 <= complexity[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int countPermutations(vector<int>& complexity) {

}
};
```

Java:

```
class Solution {
public int countPermutations(int[] complexity) {

}
}
```

Python3:

```
class Solution:
def countPermutations(self, complexity: List[int]) -> int:
```

Python:

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

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

JavaScript:

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

};
```

TypeScript:

```
function countPermutations(complexity: number[]): number {

};
```

C#:

```
public class Solution {
public int CountPermutations(int[] complexity) {

}
}
```

C:

```
int countPermutations(int* complexity, int complexitySize) {

}
```

Go:

```
func countPermutations(complexity []int) int {

}
```

Kotlin:

```
class Solution {
fun countPermutations(complexity: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countPermutations(_ complexity: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_permutations(complexity: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} complexity
# @return {Integer}
def count_permutations(complexity)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $complexity
* @return Integer
*/
function countPermutations($complexity) {

}
}
```

Dart:

```
class Solution {
int countPermutations(List<int> complexity) {

}
}
```

Scala:

```
object Solution {
def countPermutations(complexity: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_permutations(complexity :: [integer]) :: integer
def count_permutations(complexity) do

end
end
```

Erlang:

```
-spec count_permutations(Complexity :: [integer()]) -> integer().
count_permutations(Complexity) ->
.
```

Racket:

```
(define/contract (count-permutations complexity)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Computer Unlocking Permutations
* Difficulty: Medium
* Tags: array, math
*
* 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 countPermutations(vector<int>& complexity) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Computer Unlocking Permutations
* Difficulty: Medium
* Tags: array, math
*
* 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 countPermutations(int[] complexity) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Computer Unlocking Permutations
Difficulty: Medium
Tags: array, math
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count the Number of Computer Unlocking Permutations
* Difficulty: Medium
* Tags: array, math
*
* 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[]} complexity
* @return {number}
*/
var countPermutations = function(complexity) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Computer Unlocking Permutations
* Difficulty: Medium
* Tags: array, math
*
* 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 countPermutations(complexity: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count the Number of Computer Unlocking Permutations
* Difficulty: Medium
* Tags: array, math
*
* 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 CountPermutations(int[] complexity) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Computer Unlocking Permutations
* Difficulty: Medium
* Tags: array, math
*
* 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 countPermutations(int* complexity, int complexitySize) {

}
```

Go Solution:

```
// Problem: Count the Number of Computer Unlocking Permutations
// Difficulty: Medium
// Tags: array, math
//
// 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 countPermutations(complexity []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPermutations(complexity: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countPermutations(_ complexity: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count the Number of Computer Unlocking Permutations
// Difficulty: Medium
// Tags: array, math
```

```
//
// 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 count_permutations(complexity: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} complexity
# @return {Integer}
def count_permutations(complexity)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $complexity
* @return Integer
*/
function countPermutations($complexity) {

}
}
```

Dart Solution:

```
class Solution {
int countPermutations(List<int> complexity) {

}
}
```

Scala Solution:

```
object Solution {
def countPermutations(complexity: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_permutations(complexity :: [integer]) :: integer
def count_permutations(complexity) do

end
end
```

Erlang Solution:

```
-spec count_permutations(Complexity :: [integer()]) -> integer().
count_permutations(Complexity) ->
.
```

Racket Solution:

```
(define/contract (count-permutations complexity)
(-> (listof exact-integer?) exact-integer?)
)
```