Problem 2992: Number of Self-Divisible Permutations

Problem Information

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

Problem Description

Given an integer

n

, return

the number of

permutations

of the

1-indexed

array

nums = [1, 2, ..., n]

, such that it's

self-divisible

.

A

1-indexed

array

a

of length

n

is

self-divisible

if for every

1 <= i <= n

,

gcd

(a[i], i) == 1

.

A

permutation

of an array is a rearrangement of the elements of that array, for example here are all of the permutations of the array

[1, 2, 3]

:

[1, 2, 3]

[1, 3, 2]

[2, 1, 3]

[2, 3, 1]

[3, 1, 2]

[3, 2, 1]

Example 1:

Input:

n = 1

Output:

1

Explanation:

The array [1] has only 1 permutation which is self-divisible.

Example 2:

Input:

n = 2

Output:

1

Explanation:

The array [1,2] has 2 permutations and only one of them is self-divisible: nums = [1,2]: This is not self-divisible since gcd(nums[2], 2) != 1. nums = [2,1]: This is self-divisible since gcd(nums[1], 1) == 1 and gcd(nums[2], 2) == 1.

Example 3:

Input:

n = 3

Output:

3

Explanation:

The array [1,2,3] has 3 self-divisble permutations: [1,3,2], [3,1,2], [2,3,1]. It can be shown that the other 3 permutations are not self-divisible. Hence the answer is 3.

Constraints:

1 <= n <= 12

Code Snippets

C++:

```
class Solution {
public:
int selfDivisiblePermutationCount(int n) {

}
};
```

Java:

```
class Solution {
public int selfDivisiblePermutationCount(int n) {

}
}
```

Python3:

```
class Solution:
def selfDivisiblePermutationCount(self, n: int) -> int:
```

Python:

```
class Solution(object):
def selfDivisiblePermutationCount(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var selfDivisiblePermutationCount = function(n) {

};
```

TypeScript:

```
function selfDivisiblePermutationCount(n: number): number {

};
```

C#:

```
public class Solution {
public int SelfDivisiblePermutationCount(int n) {

}
}
```

C:

```
int selfDivisiblePermutationCount(int n) {

}
```

Go:

```
func selfDivisiblePermutationCount(n int) int {

}
```

Kotlin:

```
class Solution {
fun selfDivisiblePermutationCount(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func selfDivisiblePermutationCount(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn self_divisible_permutation_count(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def self_divisible_permutation_count(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function selfDivisiblePermutationCount($n) {

}
}
```

Dart:

```
class Solution {
int selfDivisiblePermutationCount(int n) {

}
}
```

Scala:

```
object Solution {
def selfDivisiblePermutationCount(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec self_divisible_permutation_count(n :: integer) :: integer
def self_divisible_permutation_count(n) do

end
end
```

Erlang:

```
-spec self_divisible_permutation_count(N :: integer()) -> integer().
self_divisible_permutation_count(N) ->
.
```

Racket:

```
(define/contract (self-divisible-permutation-count n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Self-Divisible Permutations
* Difficulty: Medium
* Tags: array, dp, math
*
* 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:
int selfDivisiblePermutationCount(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Self-Divisible Permutations
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 int selfDivisiblePermutationCount(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Self-Divisible Permutations
Difficulty: Medium
Tags: array, dp, math

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 selfDivisiblePermutationCount(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def selfDivisiblePermutationCount(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Self-Divisible Permutations
* Difficulty: Medium
* Tags: array, dp, math
*
* 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} n
* @return {number}
*/
var selfDivisiblePermutationCount = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Self-Divisible Permutations
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 selfDivisiblePermutationCount(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Self-Divisible Permutations
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 int SelfDivisiblePermutationCount(int n) {

}
}
```

C Solution:

```
/*
* Problem: Number of Self-Divisible Permutations
* Difficulty: Medium
* Tags: array, dp, math
*
* 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
*/

int selfDivisiblePermutationCount(int n) {

}
```

Go Solution:

```
// Problem: Number of Self-Divisible Permutations
// Difficulty: Medium
// Tags: array, dp, math
//
// 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 selfDivisiblePermutationCount(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun selfDivisiblePermutationCount(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func selfDivisiblePermutationCount(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Self-Divisible Permutations
// Difficulty: Medium
// Tags: array, dp, math
//
// 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 self_divisible_permutation_count(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def self_divisible_permutation_count(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function selfDivisiblePermutationCount($n) {

}
}
```

Dart Solution:

```
class Solution {
int selfDivisiblePermutationCount(int n) {
}
}
```

Scala Solution:

```
object Solution {
def selfDivisiblePermutationCount(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec self_divisible_permutation_count(n :: integer) :: integer
def self_divisible_permutation_count(n) do

end
end
```

Erlang Solution:

```
-spec self_divisible_permutation_count(N :: integer()) -> integer().
self_divisible_permutation_count(N) ->
.
```

Racket Solution:

```
(define/contract (self-divisible-permutation-count n)
(-> exact-integer? exact-integer?)
)
```