Problem 526: Beautiful Arrangement

Problem Information

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

Problem Description

Suppose you have

n

integers labeled

1

through

n

. A permutation of those

n

integers

perm

(

1-indexed

) is considered a

beautiful arrangement

if for every

i

(

1 <= i <= n

),

either

of the following is true:

perm[i]

is divisible by

i

.

i

is divisible by

perm[i]

.

Given an integer

n

, return

the

number

of the

beautiful arrangements

that you can construct

.

Example 1:

Input:

n = 2

Output:

2

Explanation:

The first beautiful arrangement is [1,2]: - perm[1] = 1 is divisible by i = 1 - perm[2] = 2 is divisible by i = 2 The second beautiful arrangement is [2,1]: - perm[1] = 2 is divisible by i = 1 - i = 2 is divisible by perm[2] = 1

Example 2:

Input:

n = 1

Output:

1

Constraints:

1 <= n <= 15

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int countArrangement(int n) {

}
```

Go:

```
func countArrangement(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-arrangement n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Beautiful Arrangement
* Difficulty: Medium
* Tags: array, dp
*
* 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 countArrangement(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Beautiful Arrangement
* Difficulty: Medium
* Tags: array, dp
*
* 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 countArrangement(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Beautiful Arrangement
Difficulty: Medium
Tags: array, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Beautiful Arrangement
* Difficulty: Medium
* Tags: array, dp
*
* 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 countArrangement = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Beautiful Arrangement
* Difficulty: Medium
* Tags: array, dp
*
* 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 countArrangement(n: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Beautiful Arrangement
* Difficulty: Medium
* Tags: array, dp
*
* 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 CountArrangement(int n) {

}
}
```

C Solution:

```
/*
* Problem: Beautiful Arrangement
* Difficulty: Medium
* Tags: array, dp
*
* 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 countArrangement(int n) {

}
```

Go Solution:

```
// Problem: Beautiful Arrangement
// Difficulty: Medium
```

```
// Tags: array, dp
//
// 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 countArrangement(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Beautiful Arrangement
// Difficulty: Medium
// Tags: array, dp
//
// 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 count_arrangement(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```php
class Solution {

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

}
}
```

Dart Solution:

```dart
class Solution {
int countArrangement(int n) {

}
}
```

Scala Solution:

```scala
object Solution {
def countArrangement(n: Int): Int = {

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-arrangement n)
(-> exact-integer? exact-integer?)
)
```