Problem 481: Magical String

Problem Information

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

Problem Description

A magical string

s

consists of only

'1'

and

'2'

and obeys the following rules:

The string s is magical because concatenating the number of contiguous occurrences of characters

'1'

and

'2'

generates the string

s

itself.

The first few elements of

s

is

s = "1221121221221121122……"

. If we group the consecutive

1

's and

2

's in

s

, it will be

"1 22 11 2 1 22 1 22 11 2 11 22 ....."

and the occurrences of

1

's or

2

's in each group are

"1 2 2 1 1 2 1 2 2 1 2 2 ....."

. You can see that the occurrence sequence is

s

itself.

Given an integer

n

, return the number of

1

's in the first

n

number in the magical string

s

.

Example 1:

Input:

n = 6

Output:

3

Explanation:

The first 6 elements of magical string s is "122112" and it contains three 1's, so return 3.

Example 2:

Input:

n = 1

Output:

1

Constraints:

1 <= n <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int magicalString(int n) {

}
```

Go:

```
func magicalString(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (magical-string n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Magical String
* Difficulty: Medium
* Tags: array, string
*
* 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 magicalString(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Magical String
* Difficulty: Medium
* Tags: array, string
*
* 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 magicalString(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Magical String
Difficulty: Medium
Tags: array, string
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Magical String
* Difficulty: Medium
* Tags: array, string
*
* 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} n
* @return {number}
*/
var magicalString = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Magical String
* Difficulty: Medium
* Tags: array, string
*
* 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 magicalString(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Magical String
* Difficulty: Medium
* Tags: array, string
*
* 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 MagicalString(int n) {

}
}
```

C Solution:

```
/*
* Problem: Magical String
* Difficulty: Medium
* Tags: array, string
*
* 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 magicalString(int n) {

}
```

Go Solution:

```
// Problem: Magical String
// Difficulty: Medium
// Tags: array, string
//
// 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 magicalString(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Magical String
// Difficulty: Medium
// Tags: array, string
```

```
//
// 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 magical_string(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (magical-string n)
(-> exact-integer? exact-integer?)
)
```