Problem 2549: Count Distinct Numbers on Board

Problem Information

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

Problem Description

You are given a positive integer

n

, that is initially placed on a board. Every day, for

10

9

days, you perform the following procedure:

For each number

x

present on the board, find all numbers

1 <= i <= n

such that

x % i == 1

.

Then, place those numbers on the board.

Return

the number of

distinct

integers present on the board after

10

9

days have elapsed

.

Note:

Once a number is placed on the board, it will remain on it until the end.

%

stands for the modulo operation. For example,

14 % 3

is

2

.

Example 1:

Input:

n = 5

Output:

4

Explanation:

Initially, 5 is present on the board. The next day, 2 and 4 will be added since 5 % 2 == 1 and 5 % 4 == 1. After that day, 3 will be added to the board because 4 % 3 == 1. At the end of a billion days, the distinct numbers on the board will be 2, 3, 4, and 5.

Example 2:

Input:

n = 3

Output:

2

Explanation:

Since 3 % 2 == 1, 2 will be added to the board. After a billion days, the only two distinct numbers on the board are 2 and 3.

Constraints:

1 <= n <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
}
```

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

}
}
```

C:

```
int distinctIntegers(int n) {

}
```

Go:

```
func distinctIntegers(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (distinct-integers n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Distinct Numbers on Board
* Difficulty: Easy
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int distinctIntegers(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Count Distinct Numbers on Board
* Difficulty: Easy
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int distinctIntegers(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Distinct Numbers on Board
Difficulty: Easy
Tags: array, math, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def distinctIntegers(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Distinct Numbers on Board
* Difficulty: Easy
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Distinct Numbers on Board
* Difficulty: Easy
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function distinctIntegers(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Distinct Numbers on Board
* Difficulty: Easy
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public int DistinctIntegers(int n) {

}
}
```

C Solution:

```
/*
* Problem: Count Distinct Numbers on Board
* Difficulty: Easy
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int distinctIntegers(int n) {

}
```

Go Solution:

```
// Problem: Count Distinct Numbers on Board
// Difficulty: Easy
// Tags: array, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func distinctIntegers(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Distinct Numbers on Board
// Difficulty: Easy
// Tags: array, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (distinct-integers n)
(-> exact-integer? exact-integer?)
)
```