Problem 3233: Find the Count of Numbers Which Are Not Special

Problem Information

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

Problem Description

You are given 2

positive

integers

l

and

r

. For any number

x

, all positive divisors of

x

except

x

are called the

proper divisors

of

x

.

A number is called

special

if it has exactly 2

proper divisors

. For example:

The number 4 is

special

because it has proper divisors 1 and 2.

The number 6 is

not special

because it has proper divisors 1, 2, and 3.

Return the count of numbers in the range

[l, r]

that are

not

special

.

Example 1:

Input:

l = 5, r = 7

Output:

3

Explanation:

There are no special numbers in the range

[5, 7]

.

Example 2:

Input:

l = 4, r = 16

Output:

11

Explanation:

The special numbers in the range

[4, 16]

are 4 and 9.

Constraints:

1 <= l <= r <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int nonSpecialCount(int l, int r) {

}
};
```

Java:

```
class Solution {
public int nonSpecialCount(int l, int r) {

}
}
```

Python3:

```
class Solution:
def nonSpecialCount(self, l: int, r: int) -> int:
```

Python:

```
class Solution(object):
def nonSpecialCount(self, l, r):
"""
:type l: int
:type r: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} l
* @param {number} r
* @return {number}
*/
var nonSpecialCount = function(l, r) {

};
```

TypeScript:

```
function nonSpecialCount(l: number, r: number): number {

};
```

C#:

```
public class Solution {
public int NonSpecialCount(int l, int r) {

}
}
```

C:

```
int nonSpecialCount(int l, int r) {

}
```

Go:

```
func nonSpecialCount(l int, r int) int {

}
```

Kotlin:

```
class Solution {
fun nonSpecialCount(l: Int, r: Int): Int {

}
```

```
}
```

Swift:

```
class Solution {
func nonSpecialCount(_ l: Int, _ r: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn non_special_count(l: i32, r: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def non_special_count(l, r)

end
```

PHP:

```
class Solution {

/**
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function nonSpecialCount($l, $r) {

}
}
```

Dart:

```
class Solution {
int nonSpecialCount(int l, int r) {

}
}
```

Scala:

```
object Solution {
def nonSpecialCount(l: Int, r: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec non_special_count(l :: integer, r :: integer) :: integer
def non_special_count(l, r) do

end
end
```

Erlang:

```
-spec non_special_count(L :: integer(), R :: integer()) -> integer().
non_special_count(L, R) ->
.
```

Racket:

```
(define/contract (non-special-count l r)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Count of Numbers Which Are Not Special
```

```
* 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 nonSpecialCount(int l, int r) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Count of Numbers Which Are Not Special
* 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 nonSpecialCount(int l, int r) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Count of Numbers Which Are Not Special
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 nonSpecialCount(self, l: int, r: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def nonSpecialCount(self, l, r):
"""
:type l: int
:type r: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Count of Numbers Which Are Not Special
* 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} l
* @param {number} r
* @return {number}
*/
var nonSpecialCount = function(l, r) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Count of Numbers Which Are Not Special
* 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 nonSpecialCount(l: number, r: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Count of Numbers Which Are Not Special
* 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 NonSpecialCount(int l, int r) {

}
}
```

C Solution:

```
/*
* Problem: Find the Count of Numbers Which Are Not Special
* 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 nonSpecialCount(int l, int r) {

}
```

Go Solution:

```
// Problem: Find the Count of Numbers Which Are Not Special
// 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 nonSpecialCount(l int, r int) int {

}
```

Kotlin Solution:

```
class Solution {
fun nonSpecialCount(l: Int, r: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func nonSpecialCount(_ l: Int, _ r: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Count of Numbers Which Are Not Special
// 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 non_special_count(l: i32, r: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def non_special_count(l, r)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function nonSpecialCount($l, $r) {

}
}
```

Dart Solution:

```
class Solution {
int nonSpecialCount(int l, int r) {

}
}
```

Scala Solution:

```
object Solution {
def nonSpecialCount(l: Int, r: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec non_special_count(l :: integer, r :: integer) :: integer
def non_special_count(l, r) do

end
end
```

Erlang Solution:

```
-spec non_special_count(L :: integer(), R :: integer()) -> integer().
non_special_count(L, R) ->
.
```

Racket Solution:

```
(define/contract (non-special-count l r)
(-> exact-integer? exact-integer? exact-integer?)
)
```