Problem 201: Bitwise AND of Numbers Range

Problem Information

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

Problem Description

Given two integers

left

and

right

that represent the range

[left, right]

, return

the bitwise AND of all numbers in this range, inclusive

.

Example 1:

Input:

left = 5, right = 7

Output:

4

Example 2:

Input:

left = 0, right = 0

Output:

0

Example 3:

Input:

left = 1, right = 2147483647

Output:

0

Constraints:

0 <= left <= right <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
int rangeBitwiseAnd(int left, int right) {
```

```
}
};
```

Java:

```
class Solution {
public int rangeBitwiseAnd(int left, int right) {

}
}
```

Python3:

```
class Solution:
def rangeBitwiseAnd(self, left: int, right: int) -> int:
```

Python:

```
class Solution(object):
def rangeBitwiseAnd(self, left, right):
"""
:type left: int
:type right: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} left
* @param {number} right
* @return {number}
*/
var rangeBitwiseAnd = function(left, right) {

};
```

TypeScript:

```
function rangeBitwiseAnd(left: number, right: number): number {

};
```

C#:

```
public class Solution {
public int RangeBitwiseAnd(int left, int right) {

}
}
```

C:

```
int rangeBitwiseAnd(int left, int right) {

}
```

Go:

```
func rangeBitwiseAnd(left int, right int) int {

}
```

Kotlin:

```
class Solution {
fun rangeBitwiseAnd(left: Int, right: Int): Int {

}
}
```

Swift:

```
class Solution {
func rangeBitwiseAnd(_ left: Int, _ right: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn range_bitwise_and(left: i32, right: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} left
# @param {Integer} right
# @return {Integer}
def range_bitwise_and(left, right)

end
```

PHP:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function rangeBitwiseAnd($left, $right) {

}
}
```

Dart:

```
class Solution {
int rangeBitwiseAnd(int left, int right) {

}
}
```

Scala:

```
object Solution {
def rangeBitwiseAnd(left: Int, right: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec range_bitwise_and(left :: integer, right :: integer) :: integer
```

```
def range_bitwise_and(left, right) do

end
end
```

Erlang:

```
-spec range_bitwise_and(Left :: integer(), Right :: integer()) -> integer().
range_bitwise_and(Left, Right) ->
.
```

Racket:

```
(define/contract (range-bitwise-and left right)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Bitwise AND of Numbers Range
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int rangeBitwiseAnd(int left, int right) {

}
};
```

Java Solution:

```
/**
* Problem: Bitwise AND of Numbers Range
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int rangeBitwiseAnd(int left, int right) {

}
}
```

Python3 Solution:

```
"""
Problem: Bitwise AND of Numbers Range
Difficulty: Medium
Tags: general

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def rangeBitwiseAnd(self, left: int, right: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def rangeBitwiseAnd(self, left, right):
"""
:type left: int
:type right: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Bitwise AND of Numbers Range
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} left
* @param {number} right
* @return {number}
*/
var rangeBitwiseAnd = function(left, right) {

};
```

TypeScript Solution:

```
/**
* Problem: Bitwise AND of Numbers Range
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function rangeBitwiseAnd(left: number, right: number): number {

};
```

C# Solution:

```
/*
* Problem: Bitwise AND of Numbers Range
* Difficulty: Medium
```

```
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int RangeBitwiseAnd(int left, int right) {

}
}
```

C Solution:

```
/*
* Problem: Bitwise AND of Numbers Range
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int rangeBitwiseAnd(int left, int right) {

}
```

Go Solution:

```
// Problem: Bitwise AND of Numbers Range
// Difficulty: Medium
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func rangeBitwiseAnd(left int, right int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun rangeBitwiseAnd(left: Int, right: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func rangeBitwiseAnd(_ left: Int, _ right: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Bitwise AND of Numbers Range
// Difficulty: Medium
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn range_bitwise_and(left: i32, right: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} left
# @param {Integer} right
# @return {Integer}
def range_bitwise_and(left, right)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function rangeBitwiseAnd($left, $right) {

}
}
```

Dart Solution:

```
class Solution {
int rangeBitwiseAnd(int left, int right) {

}
}
```

Scala Solution:

```
object Solution {
def rangeBitwiseAnd(left: Int, right: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec range_bitwise_and(left :: integer, right :: integer) :: integer
def range_bitwise_and(left, right) do

end
end
```

Erlang Solution:

```
-spec range_bitwise_and(Left :: integer(), Right :: integer()) -> integer().
range_bitwise_and(Left, Right) ->
.
```

Racket Solution:

```
(define/contract (range-bitwise-and left right)
(-> exact-integer? exact-integer? exact-integer?)
)
```