# Problem 3677: Count Binary Palindromic Numbers

## Problem Information

**Difficulty:** Hard
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

You are given a

non-negative

integer

n

.

A

non-negative

integer is called

binary-palindromic

if its binary representation (written without leading zeros) reads the same forward and backward.

Return the number of integers

k

such that

0 <= k <= n

and the binary representation of

k

is a palindrome.

Note:

The number 0 is considered binary-palindromic, and its representation is

"0"

.

Example 1:

Input:

n = 9

Output:

6

Explanation:

The integers

k

in the range

[0, 9]

whose binary representations are palindromes are:

0 → "0"

1 → "1"

3 → "11"

5 → "101"

7 → "111"

9 → "1001"

All other values in

[0, 9]

have non-palindromic binary forms. Therefore, the count is 6.

Example 2:

Input:

n = 0

Output:

1

Explanation:

Since

"0"

is a palindrome, the count is 1.

Constraints:

0 <= n <= 10

15

## Code Snippets

**C++:**

```
class Solution {
public:
int countBinaryPalindromes(long long n) {

}
};
```

**Java:**

```
class Solution {
public int countBinaryPalindromes(long n) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

```
/**
* @param {number} n
```

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

};
```

**TypeScript:**

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

};
```

**C#:**

```
public class Solution {
public int CountBinaryPalindromes(long n) {

}
}
```

**C:**

```
int countBinaryPalindromes(long long n) {

}
```

**Go:**

```
func countBinaryPalindromes(n int64) int {

}
```

**Kotlin:**

```
class Solution {
fun countBinaryPalindromes(n: Long): Int {

}
}
```

**Swift:**

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

}
}
```

**Rust:**

```
impl Solution {
pub fn count_binary_palindromes(n: i64) -> i32 {

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

### Scala:

```
object Solution {
def countBinaryPalindromes(n: Long): Int = {

}
}
```

### Elixir:

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

end
end
```

### Erlang:

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

### Racket:

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

## Solutions

### C++ Solution:

```
/*
* Problem: Count Binary Palindromic Numbers
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int countBinaryPalindromes(long long n) {

}
};
```

### Java Solution:

```
/**
* Problem: Count Binary Palindromic Numbers
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countBinaryPalindromes(long n) {

}
}
```

### Python3 Solution:

```
"""
Problem: Count Binary Palindromic Numbers
Difficulty: Hard
Tags: string, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

```
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Count Binary Palindromic Numbers
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* 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 countBinaryPalindromes = function(n) {

};
```

### TypeScript Solution:

```
/**
* Problem: Count Binary Palindromic Numbers
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

function countBinaryPalindromes(n: number): number {

};
```

**C# Solution:**

```
/*
* Problem: Count Binary Palindromic Numbers
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int CountBinaryPalindromes(long n) {

}
}
```

**C Solution:**

```
/*
* Problem: Count Binary Palindromic Numbers
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int countBinaryPalindromes(long long n) {

}
```

**Go Solution:**

```
// Problem: Count Binary Palindromic Numbers
// Difficulty: Hard
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countBinaryPalindromes(n int64) int {

}
```

### Kotlin Solution:

```
class Solution {
fun countBinaryPalindromes(n: Long): Int {

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Count Binary Palindromic Numbers
// Difficulty: Hard
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_binary_palindromes(n: i64) -> i32 {

}
```

```
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

```
object Solution {
def countBinaryPalindromes(n: Long): Int = {

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

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