# Problem 2843: Count Symmetric Integers

## Problem Information

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

## Problem Description

You are given two positive integers

low

and

high

.

An integer

x

consisting of

2 * n

digits is

symmetric

if the sum of the first

n

digits of

x

is equal to the sum of the last

n

digits of

x

. Numbers with an odd number of digits are never symmetric.

Return

the

number of symmetric

integers in the range

[low, high]

.

Example 1:

Input:

low = 1, high = 100

Output:

9

Explanation:

There are 9 symmetric integers between 1 and 100: 11, 22, 33, 44, 55, 66, 77, 88, and 99.

Example 2:

Input:

low = 1200, high = 1230

Output:

4

Explanation:

There are 4 symmetric integers between 1200 and 1230: 1203, 1212, 1221, and 1230.

Constraints:

1 <= low <= high <= 10

4

## Code Snippets

**C++:**

```
class Solution {
public:
int countSymmetricIntegers(int low, int high) {

}
};
```

**Java:**

```
class Solution {
public int countSymmetricIntegers(int low, int high) {

}
}
```

### Python3:

```
class Solution:
def countSymmetricIntegers(self, low: int, high: int) -> int:
```

### Python:

```
class Solution(object):
def countSymmetricIntegers(self, low, high):
"""
:type low: int
:type high: int
:rtype: int
"""
```

### JavaScript:

```
/**
* @param {number} low
* @param {number} high
* @return {number}
*/
var countSymmetricIntegers = function(low, high) {

};
```

### TypeScript:

```
function countSymmetricIntegers(low: number, high: number): number {

};
```

### C#:

```
public class Solution {
public int CountSymmetricIntegers(int low, int high) {

}
}
```

### C:

```
int countSymmetricIntegers(int low, int high) {

}
```

**Go:**

```
func countSymmetricIntegers(low int, high int) int {

}
```

**Kotlin:**

```
class Solution {
fun countSymmetricIntegers(low: Int, high: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func countSymmetricIntegers(_ low: Int, _ high: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn count_symmetric_integers(low: i32, high: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer} low
# @param {Integer} high
# @return {Integer}
def count_symmetric_integers(low, high)

end
```

### PHP:

```
class Solution {

/**
* @param Integer $low
* @param Integer $high
* @return Integer
*/
function countSymmetricIntegers($low, $high) {

}
}
```

### Dart:

```
class Solution {
int countSymmetricIntegers(int low, int high) {

}
}
```

### Scala:

```
object Solution {
def countSymmetricIntegers(low: Int, high: Int): Int = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec count_symmetric_integers(low :: integer, high :: integer) :: integer
def count_symmetric_integers(low, high) do

end
end
```

### Erlang:

```
-spec count_symmetric_integers(Low :: integer(), High :: integer()) ->
integer().
```

```
count_symmetric_integers(Low, High) ->
.
```

### Racket:

```
(define/contract (count-symmetric-integers low high)
(-> exact-integer? exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Count Symmetric Integers
* Difficulty: Easy
* Tags: math
*
* 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 countSymmetricIntegers(int low, int high) {

}
};
```

### Java Solution:

```
/**
* Problem: Count Symmetric Integers
* Difficulty: Easy
* Tags: math
*
* 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 countSymmetricIntegers(int low, int high) {

}
}
```

### Python3 Solution:

```
"""
Problem: Count Symmetric Integers
Difficulty: Easy
Tags: math

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 countSymmetricIntegers(self, low: int, high: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def countSymmetricIntegers(self, low, high):
"""
:type low: int
:type high: int
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Count Symmetric Integers
* Difficulty: Easy
* Tags: math
```

```
*
* 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} low
* @param {number} high
* @return {number}
*/
var countSymmetricIntegers = function(low, high) {

};
```

### TypeScript Solution:

```
/**
* Problem: Count Symmetric Integers
* Difficulty: Easy
* Tags: math
*
* 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 countSymmetricIntegers(low: number, high: number): number {

};
```

### C# Solution:

```
/*
* Problem: Count Symmetric Integers
* Difficulty: Easy
* Tags: math
*
* 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 CountSymmetricIntegers(int low, int high) {

}
}
```

### C Solution:

```
/*
* Problem: Count Symmetric Integers
* Difficulty: Easy
* Tags: math
*
* 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 countSymmetricIntegers(int low, int high) {

}
```

### Go Solution:

```
// Problem: Count Symmetric Integers
// Difficulty: Easy
// Tags: math
//
// 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 countSymmetricIntegers(low int, high int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun countSymmetricIntegers(low: Int, high: Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func countSymmetricIntegers(_ low: Int, _ high: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Count Symmetric Integers
// Difficulty: Easy
// Tags: math
//
// 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 count_symmetric_integers(low: i32, high: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer} low
# @param {Integer} high
# @return {Integer}
def count_symmetric_integers(low, high)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer $low
* @param Integer $high
* @return Integer
*/
function countSymmetricIntegers($low, $high) {

}
}
```

### Dart Solution:

```
class Solution {
int countSymmetricIntegers(int low, int high) {

}
}
```

### Scala Solution:

```
object Solution {
def countSymmetricIntegers(low: Int, high: Int): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec count_symmetric_integers(low :: integer, high :: integer) :: integer
def count_symmetric_integers(low, high) do

end
end
```

### Erlang Solution:

```
-spec count_symmetric_integers(Low :: integer(), High :: integer()) ->
integer().
count_symmetric_integers(Low, High) ->
```

```
.
```

## Racket Solution:

```
(define/contract (count-symmetric-integers low high)
(-> exact-integer? exact-integer? exact-integer?)
)
```