Problem 1009: Complement of Base 10 Integer

Problem Information

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

Problem Description

The

complement

of an integer is the integer you get when you flip all the

0

's to

1

's and all the

1

's to

0

's in its binary representation.

For example, The integer

5

is

"101"

in binary and its

complement

is

"010"

which is the integer

2

.

Given an integer

n

, return

its complement

.

Example 1:

Input:

n = 5

Output:

2

Explanation:

5 is "101" in binary, with complement "010" in binary, which is 2 in base-10.

Example 2:

Input:

n = 7

Output:

0

Explanation:

7 is "111" in binary, with complement "000" in binary, which is 0 in base-10.

Example 3:

Input:

n = 10

Output:

5

Explanation:

10 is "1010" in binary, with complement "0101" in binary, which is 5 in base-10.

Constraints:

0 <= n < 10

9

Note:

This question is the same as 476:

https://leetcode.com/problems/number-complement/

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int bitwiseComplement(int n) {

}
```

Go:

```
func bitwiseComplement(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (bitwise-complement n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Complement of Base 10 Integer
* Difficulty: Easy
* 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 bitwiseComplement(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Complement of Base 10 Integer
* Difficulty: Easy
* 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 bitwiseComplement(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Complement of Base 10 Integer
Difficulty: Easy
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 bitwiseComplement(self, n: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Complement of Base 10 Integer
* Difficulty: Easy
* 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} n
* @return {number}
*/
var bitwiseComplement = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Complement of Base 10 Integer
* Difficulty: Easy
* 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 bitwiseComplement(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Complement of Base 10 Integer
* Difficulty: Easy
* 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 BitwiseComplement(int n) {

}
}
```

C Solution:

```
/*
* Problem: Complement of Base 10 Integer
* Difficulty: Easy
* 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 bitwiseComplement(int n) {

}
```

Go Solution:

```
// Problem: Complement of Base 10 Integer
// Difficulty: Easy
// 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 bitwiseComplement(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Complement of Base 10 Integer
// Difficulty: Easy
// 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 bitwise_complement(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (bitwise-complement n)
(-> exact-integer? exact-integer?)
)
```