Problem 476: Number Complement

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

num

, return

its complement

.

Example 1:

Input:

num = 5

Output:

2

Explanation:

The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

Example 2:

Input:

num = 1

Output:

0

Explanation:

The binary representation of 1 is 1 (no leading zero bits), and its complement is 0. So you need to output 0.

Constraints:

1 <= num < 2

31

Note:

This question is the same as 1009:

https://leetcode.com/problems/complement-of-base-10-integer/

Code Snippets

C++:

```
class Solution {
public:
int findComplement(int num) {
```

```
}
};
```

Java:

```
class Solution {
public int findComplement(int num) {

}
}
```

Python3:

```
class Solution:
def findComplement(self, num: int) -> int:
```

Python:

```
class Solution(object):
def findComplement(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} num
* @return {number}
*/
var findComplement = function(num) {

};
```

TypeScript:

```
function findComplement(num: number): number {

};
```

C#:

```
public class Solution {
public int FindComplement(int num) {

}
}
```

C:

```
int findComplement(int num) {

}
```

Go:

```
func findComplement(num int) int {

}
```

Kotlin:

```
class Solution {
fun findComplement(num: Int): Int {

}
}
```

Swift:

```
class Solution {
func findComplement(_ num: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_complement(num: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num
# @return {Integer}
def find_complement(num)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num
* @return Integer
*/
function findComplement($num) {

}
}
```

Dart:

```
class Solution {
int findComplement(int num) {

}
}
```

Scala:

```
object Solution {
def findComplement(num: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_complement(num :: integer) :: integer
def find_complement(num) do

end
end
```

Erlang:

```
-spec find_complement(Num :: integer()) -> integer().
find_complement(Num) ->
.
```

Racket:

```
(define/contract (find-complement num)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number Complement
* 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 findComplement(int num) {

}
};
```

Java Solution:

```
/**
* Problem: Number Complement
* 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 findComplement(int num) {

}
}
```

Python3 Solution:

```
"""
Problem: Number Complement
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 findComplement(self, num: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findComplement(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number Complement
* 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} num
* @return {number}
*/
var findComplement = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Number Complement
* 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 findComplement(num: number): number {

};
```

C# Solution:

```
/*
* Problem: Number Complement
* 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 FindComplement(int num) {

}
}
```

C Solution:

```
/*
* Problem: Number Complement
* 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 findComplement(int num) {

}
```

Go Solution:

```
// Problem: Number Complement
// 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 findComplement(num int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findComplement(num: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findComplement(_ num: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number Complement
// 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 find_complement(num: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num
# @return {Integer}
def find_complement(num)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $num
* @return Integer
*/
function findComplement($num) {

}
}
```

Dart Solution:

```
class Solution {
int findComplement(int num) {

}
}
```

Scala Solution:

```
object Solution {
def findComplement(num: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_complement(num :: integer) :: integer
def find_complement(num) do

end
end
```

Erlang Solution:

```
-spec find_complement(Num :: integer()) -> integer().
find_complement(Num) ->
.
```

Racket Solution:

```
(define/contract (find-complement num)
(-> exact-integer? exact-integer?)
)
```