Problem 190: Reverse Bits

Problem Information

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

Problem Description

Reverse bits of a given 32 bits signed integer.

Example 1:

Input:

n = 43261596

Output:

964176192

Explanation:

Integer

Binary

43261596

00000010100101000001111010011100

964176192

00111001011110000010100101000000

Example 2:

Input:

n = 2147483644

Output:

1073741822

Explanation:

Integer

Binary

2147483644

01111111111111111111111111111100

1073741822

00111111111111111111111111111110

Constraints:

0 <= n <= 2

31

- 2

n

is even.

Follow up:

If this function is called many times, how would you optimize it?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int reverseBits(int n) {

}
```

Go:

```
func reverseBits(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (reverse-bits n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Bits
* 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 reverseBits(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Bits
* 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 reverseBits(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse Bits
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 reverseBits(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reverse Bits
* 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 reverseBits = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Bits
* 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 reverseBits(n: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Reverse Bits
* 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 ReverseBits(int n) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Bits
* 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 reverseBits(int n) {

}
```

Go Solution:

```
// Problem: Reverse Bits
// 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 reverseBits(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reverse Bits
// 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 reverse_bits(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (reverse-bits n)
(-> exact-integer? exact-integer?)
)
```