Problem 693: Binary Number with Alternating Bits

Problem Information

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

Problem Description

Given a positive integer, check whether it has alternating bits: namely, if two adjacent bits will always have different values.

Example 1:

Input:

n = 5

Output:

true

Explanation:

The binary representation of 5 is: 101

Example 2:

Input:

n = 7

Output:

false

Explanation:

The binary representation of 7 is: 111.

Example 3:

Input:

n = 11

Output:

false

Explanation:

The binary representation of 11 is: 1011.

Constraints:

1 <= n <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
bool hasAlternatingBits(int n) {

}
};
```

Java:

```
class Solution {
public boolean hasAlternatingBits(int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function hasAlternatingBits(n: number): boolean {

};
```

C#:

```
public class Solution {
public bool HasAlternatingBits(int n) {
```

```
}
}
```

C:

```
bool hasAlternatingBits(int n) {

}
```

Go:

```
func hasAlternatingBits(n int) bool {

}
```

Kotlin:

```
class Solution {
fun hasAlternatingBits(n: Int): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool hasAlternatingBits(int n) {

}
}
```

Scala:

```
object Solution {
def hasAlternatingBits(n: Int): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec has_alternating_bits(N :: integer()) -> boolean().
has_alternating_bits(N) ->
.
```

Racket:

```
(define/contract (has-alternating-bits n)
(-> exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Binary Number with Alternating 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:
bool hasAlternatingBits(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Binary Number with Alternating 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 boolean hasAlternatingBits(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Binary Number with Alternating 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 hasAlternatingBits(self, n: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Binary Number with Alternating 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 {boolean}
*/
var hasAlternatingBits = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Binary Number with Alternating 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 hasAlternatingBits(n: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Binary Number with Alternating 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 bool HasAlternatingBits(int n) {

}
}
```

C Solution:

```
/*
* Problem: Binary Number with Alternating 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
*/

bool hasAlternatingBits(int n) {

}
```

Go Solution:

```
// Problem: Binary Number with Alternating 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 hasAlternatingBits(n int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun hasAlternatingBits(n: Int): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Binary Number with Alternating 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 has_alternating_bits(n: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $n
* @return Boolean
*/
function hasAlternatingBits($n) {

}
}
```

Dart Solution:

```
class Solution {
bool hasAlternatingBits(int n) {

}
}
```

Scala Solution:

```
object Solution {
def hasAlternatingBits(n: Int): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec has_alternating_bits(N :: integer()) -> boolean().
has_alternating_bits(N) ->
.
```

Racket Solution:

```
(define/contract (has-alternating-bits n)
(-> exact-integer? boolean?)
)
```