Problem 231: Power of Two

Problem Information

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

Problem Description

Given an integer

n

, return

true

if it is a power of two. Otherwise, return

false

.

An integer

n

is a power of two, if there exists an integer

x

such that

n == 2

x

.

Example 1:

Input:

n = 1

Output:

true

Explanation:

2

0

= 1

Example 2:

Input:

n = 16

Output:

true

Explanation:

2

4

= 16

Example 3:

Input:

n = 3

Output:

false

Constraints:

-2

31

<= n <= 2

31

- 1

Follow up:

Could you solve it without loops/recursion?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
bool isPowerOfTwo(int n) {

}
```

Go:

```
func isPowerOfTwo(n int) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-power-of-two n)
(-> exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Power of Two
* 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:
bool isPowerOfTwo(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Power of Two
* 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 boolean isPowerOfTwo(int n) {
}
}
```

Python3 Solution:

```
"""
Problem: Power of Two
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 isPowerOfTwo(self, n: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Power of Two
* 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} n
* @return {boolean}
*/
var isPowerOfTwo = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Power of Two
* 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 isPowerOfTwo(n: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Power of Two
* 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 bool IsPowerOfTwo(int n) {

}

}
```

C Solution:

```
/*
* Problem: Power of Two
* 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
*/

bool isPowerOfTwo(int n) {

}
```

Go Solution:

```
// Problem: Power of Two
// 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 isPowerOfTwo(n int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Power of Two
// 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 is_power_of_two(n: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-power-of-two n)
(-> exact-integer? boolean?)
)
```