

Problem 342: Power of Four

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 four. Otherwise, return

false

.

An integer

n

is a power of four, if there exists an integer

x

such that

$n == 4^x$

x

.

Example 1:

Input:

n = 16

Output:

true

Example 2:

Input:

n = 5

Output:

false

Example 3:

Input:

n = 1

Output:

true

Constraints:

-2

31

$\leq n \leq 2$

31

- 1

Follow up:

Could you solve it without loops/recursion?

Code Snippets

C++:

```
class Solution {  
public:  
    bool isPowerOfFour(int n) {  
  
    }  
};
```

Java:

```
class Solution {  
    public boolean isPowerOfFour(int n) {  
  
    }  
}
```

Python3:

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

Python:

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

```
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

    }
}
```

C:

```
bool isPowerOfFour(int n) {

}
```

Go:

```
func isPowerOfFour(n int) bool {

}
```

Kotlin:

```
class Solution {  
    fun isPowerOfFour(n: Int): Boolean {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $n  
     * @return Boolean  
     */  
    function isPowerOfFour($n) {  
  
    }  
}
```

Dart:

```
class Solution {  
  bool isPowerOfFour(int n) {  
  
  }  
}
```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```

/*
 * Problem: Power of Four
 * 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 isPowerOfFour(int n) {

    }
};

```

Java Solution:

```

/**
 * Problem: Power of Four
 * 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 isPowerOfFour(int n) {

    }
}

```

Python3 Solution:

```

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

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Power of Four
 * 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 isPowerOfFour = function(n) {

};

```

TypeScript Solution:


```

/**
 * Problem: Power of Four
 * 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 isPowerOfFour(n: number): boolean {

};

```

C# Solution:

```

/*
 * Problem: Power of Four
 * 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 IsPowerOfFour(int n) {

    }
}

```

C Solution:

```

/*
 * Problem: Power of Four
 * 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 isPowerOfFour(int n) {

}
```

Go Solution:

```
// Problem: Power of Four
// 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 isPowerOfFour(n int) bool {

}
```

Kotlin Solution:

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

    }
}
```

Swift Solution:

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

    }
}
```

Rust Solution:

```
// Problem: Power of Four
// 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_four(n: i32) -> bool {

    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }
}
```

Dart Solution:

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

    }
}
```

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

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