Problem 507: Perfect Number

Problem Information

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

Problem Description

A

perfect number

is a

positive integer

that is equal to the sum of its

positive divisors

, excluding the number itself. A

divisor

of an integer

x

is an integer that can divide

x

evenly.

Given an integer

n

, return

true

if

n

is a perfect number, otherwise return

false

.

Example 1:

Input:

num = 28

Output:

true

Explanation:

28 = 1 + 2 + 4 + 7 + 14 1, 2, 4, 7, and 14 are all divisors of 28.

Example 2:

Input:

num = 7

Output:

false

Constraints:

1 <= num <= 10

8

Code Snippets

C++:

```
class Solution {
public:
bool checkPerfectNumber(int num) {

}
};
```

Java:

```
class Solution {
public boolean checkPerfectNumber(int num) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function checkPerfectNumber(num: number): boolean {

};
```

C#:

```
public class Solution {
public bool CheckPerfectNumber(int num) {

}
}
```

C:

```
bool checkPerfectNumber(int num) {

}
```

Go:

```
func checkPerfectNumber(num int) bool {

}
```

Kotlin:

```
class Solution {
fun checkPerfectNumber(num: Int): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool checkPerfectNumber(int num) {

}
```

```
}
```

Scala:

```
object Solution {
def checkPerfectNumber(num: Int): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec check_perfect_number(Num :: integer()) -> boolean().
check_perfect_number(Num) ->
.
```

Racket:

```
(define/contract (check-perfect-number num)
(-> exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Perfect Number
* 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 checkPerfectNumber(int num) {

}
};
```

Java Solution:

```
/**
* Problem: Perfect Number
* 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 checkPerfectNumber(int num) {

}
}
```

Python3 Solution:

```
"""
Problem: Perfect Number
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 checkPerfectNumber(self, num: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Perfect Number
* 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} num
* @return {boolean}
*/
var checkPerfectNumber = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Perfect Number
* 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 checkPerfectNumber(num: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Perfect Number
* 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 CheckPerfectNumber(int num) {

}
}
```

C Solution:

```
/*
* Problem: Perfect Number
* 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 checkPerfectNumber(int num) {
```

```
}
```

Go Solution:

```
// Problem: Perfect Number
// 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 checkPerfectNumber(num int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkPerfectNumber(num: Int): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Perfect Number
// 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 check_perfect_number(num: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool checkPerfectNumber(int num) {

}
}
```

Scala Solution:

```
object Solution {
def checkPerfectNumber(num: Int): Boolean = {
```

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec check_perfect_number(Num :: integer()) -> boolean().
check_perfect_number(Num) ->
.
```

Racket Solution:

```
(define/contract (check-perfect-number num)
(-> exact-integer? boolean?)
)
```