

Problem 1780: Check if Number is a Sum of Powers of Three

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given an integer

n

, return

true

if it is possible to represent

n

as the sum of distinct powers of three.

Otherwise, return

false

.

An integer

y

is a power of three if there exists an integer

x

such that

$y == 3$

x

.

Example 1:

Input:

$n = 12$

Output:

true

Explanation:

$12 = 3$

1

+ 3

2

Example 2:

Input:

$n = 91$

Output:

true

Explanation:

91 = 3

0

+ 3

2

+ 3

4

Example 3:

Input:

n = 21

Output:

false

Constraints:

1 <= n <= 10

7

Code Snippets

C++:

```
class Solution {  
public:
```

```
bool checkPowersOfThree(int n) {  
  
}  
};
```

Java:

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

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function checkPowersOfThree(n: number): boolean {  
  
};
```

C#:

```
public class Solution {  
    public bool CheckPowersOfThree(int n) {  
  
    }  
}
```

C:

```
bool checkPowersOfThree(int n) {  
  
}
```

Go:

```
func checkPowersOfThree(n int) bool {  
  
}
```

Kotlin:

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

Swift:

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

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

    }

}
```

Dart:

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

  }
}
```

Scala:

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

  }
}
```

Elixir:

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

```
end  
end
```

Erlang:

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

Racket:

```
(define/contract (check-powers-of-three n)  
  (-> exact-integer? boolean?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Check if Number is a Sum of Powers of Three  
 * Difficulty: Medium  
 * 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 checkPowersOfThree(int n) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Check if Number is a Sum of Powers of Three
```

```

* Difficulty: Medium
* 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 checkPowersOfThree(int n) {

}
}

```

Python3 Solution:

```

"""
Problem: Check if Number is a Sum of Powers of Three
Difficulty: Medium
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 checkPowersOfThree(self, n: int) -> bool:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

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

```

JavaScript Solution:


```

/**
 * Problem: Check if Number is a Sum of Powers of Three
 * Difficulty: Medium
 * 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 checkPowersOfThree = function(n) {

};

```

TypeScript Solution:

```

/**
 * Problem: Check if Number is a Sum of Powers of Three
 * Difficulty: Medium
 * 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 checkPowersOfThree(n: number): boolean {

};

```

C# Solution:

```

/*
 * Problem: Check if Number is a Sum of Powers of Three
 * Difficulty: Medium
 * 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 CheckPowersOfThree(int n) {

}

}

```

C Solution:

```

/*
* Problem: Check if Number is a Sum of Powers of Three
* Difficulty: Medium
* 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 checkPowersOfThree(int n) {

}

```

Go Solution:

```

// Problem: Check if Number is a Sum of Powers of Three
// Difficulty: Medium
// 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 checkPowersOfThree(n int) bool {

}

```

Kotlin Solution:

```

class Solution {
    fun checkPowersOfThree(n: Int): Boolean {

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```

// Problem: Check if Number is a Sum of Powers of Three
// Difficulty: Medium
// 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_powers_of_three(n: i32) -> bool {

    }
}

```

Ruby Solution:

```

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

end

```

PHP Solution:

```

class Solution {

```

```

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

}

}

```

Dart Solution:

```

class Solution {
  bool checkPowersOfThree(int n) {

  }
}

```

Scala Solution:

```

object Solution {
  def checkPowersOfThree(n: Int): Boolean = {

  }
}

```

Elixir Solution:

```

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

  end
end

```

Erlang Solution:

```

-spec check_powers_of_three(N :: integer()) -> boolean().
check_powers_of_three(N) ->
.

```

Racket Solution:

```
(define/contract (check-powers-of-three n)
  (-> exact-integer? boolean?)
)
```