Problem 2119: A Number After a Double Reversal

Problem Information

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

Problem Description

Reversing

an integer means to reverse all its digits.

For example, reversing

2021

gives

1202

. Reversing

12300

gives

321

as the

leading zeros are not retained

.

Given an integer

num

,

reverse

num

to get

reversed1

,

then reverse

reversed1

to get

reversed2

. Return

true

if

reversed2

equals

num

. Otherwise return

false

.

Example 1:

Input:

num = 526

Output:

true

Explanation:

Reverse num to get 625, then reverse 625 to get 526, which equals num.

Example 2:

Input:

num = 1800

Output:

false

Explanation:

Reverse num to get 81, then reverse 81 to get 18, which does not equal num.

Example 3:

Input:

num = 0

Output:

true

Explanation:

Reverse num to get 0, then reverse 0 to get 0, which equals num.

Constraints:

0 <= num <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def isSameAfterReversals(self, num):
```

```
"""
:type num: int
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
bool isSameAfterReversals(int num) {

}
```

Go:

```
func isSameAfterReversals(num int) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-same-after-reversals num)
(-> exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: A Number After a Double Reversal
* 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 isSameAfterReversals(int num) {

}
};
```

Java Solution:

```
/**
* Problem: A Number After a Double Reversal
* 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 isSameAfterReversals(int num) {

}
}
```

Python3 Solution:

```
"""
Problem: A Number After a Double Reversal
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 isSameAfterReversals(self, num: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: A Number After a Double Reversal
* 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 isSameAfterReversals = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: A Number After a Double Reversal
* 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 isSameAfterReversals(num: number): boolean {

};
```

C# Solution:

```
/*
* Problem: A Number After a Double Reversal
* 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 IsSameAfterReversals(int num) {

}
}
```

C Solution:

```
/*
* Problem: A Number After a Double Reversal
* 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 isSameAfterReversals(int num) {

}
```

Go Solution:

```
// Problem: A Number After a Double Reversal
// 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 isSameAfterReversals(num int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: A Number After a Double Reversal
// 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_same_after_reversals(num: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-same-after-reversals num)
(-> exact-integer? boolean?)
)
```