Problem 1006: Clumsy Factorial

Problem Information

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

Problem Description

The

factorial

of a positive integer

n

is the product of all positive integers less than or equal to

n

.

For example,

factorial(10) = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1

.

We make a

clumsy factorial

using the integers in decreasing order by swapping out the multiply operations for a fixed rotation of operations with multiply

'*'

, divide

'/'

, add

'+'

, and subtract

'-'

in this order.

For example,

clumsy(10) = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1

.

However, these operations are still applied using the usual order of operations of arithmetic. We do all multiplication and division steps before any addition or subtraction steps, and multiplication and division steps are processed left to right.

Additionally, the division that we use is floor division such that

10 * 9 / 8 = 90 / 8 = 11

.

Given an integer

n

, return

the clumsy factorial of

n

.

Example 1:

Input:

n = 4

Output:

7

Explanation:

7 = 4 * 3 / 2 + 1

Example 2:

Input:

n = 10

Output:

12

Explanation:

12 = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1

Constraints:

1 <= n <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int clumsy(int n) {

}
};
```

Java:

```
class Solution {
public int clumsy(int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function clumsy(n: number): number {

};
```

C#:

```
public class Solution {
public int Clumsy(int n) {

}
}
```

C:

```
int clumsy(int n) {

}
```

Go:

```
func clumsy(n int) int {

}
```

Kotlin:

```
class Solution {
fun clumsy(n: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int clumsy(int n) {

}
}
```

Scala:

```
object Solution {
def clumsy(n: Int): Int = {

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec clumsy(N :: integer()) -> integer().
clumsy(N) ->
.
```

Racket:

```
(define/contract (clumsy n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Clumsy Factorial
* Difficulty: Medium
* Tags: math, stack
*
* 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:
int clumsy(int n) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Clumsy Factorial
* Difficulty: Medium
* Tags: math, stack
*
* 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 int clumsy(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Clumsy Factorial
Difficulty: Medium
Tags: math, stack

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 clumsy(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Clumsy Factorial
* Difficulty: Medium
* Tags: math, stack
*
* 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 {number}
*/
var clumsy = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Clumsy Factorial
* Difficulty: Medium
* Tags: math, stack
*
* 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 clumsy(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Clumsy Factorial
* Difficulty: Medium
* Tags: math, stack
*
* 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 int Clumsy(int n) {

}
}
```

C Solution:

```
/*
* Problem: Clumsy Factorial
* Difficulty: Medium
* Tags: math, stack
*
* 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
*/

int clumsy(int n) {

}
```

Go Solution:

```
// Problem: Clumsy Factorial
// Difficulty: Medium
// Tags: math, stack
//
// 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 clumsy(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun clumsy(n: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Clumsy Factorial
// Difficulty: Medium
// Tags: math, stack
//
// 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 clumsy(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int clumsy(int n) {

}
}
```

Scala Solution:

```
object Solution {
def clumsy(n: Int): Int = {

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

```
-spec clumsy(N :: integer()) -> integer().
clumsy(N) ->
.
```

Racket Solution:

```
(define/contract (clumsy n)
(-> exact-integer? exact-integer?)
)
```