Problem 509: Fibonacci Number

Problem Information

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

Problem Description

The

Fibonacci numbers

, commonly denoted

F(n)

form a sequence, called the

Fibonacci sequence

, such that each number is the sum of the two preceding ones, starting from

0

and

1

. That is,

F(0) = 0, F(1) = 1 F(n) = F(n - 1) + F(n - 2), for n > 1.

Given

n

, calculate

F(n)

.

Example 1:

Input:

n = 2

Output:

1

Explanation:

F(2) = F(1) + F(0) = 1 + 0 = 1.

Example 2:

Input:

n = 3

Output:

2

Explanation:

F(3) = F(2) + F(1) = 1 + 1 = 2.

Example 3:

Input:

n = 4

Output:

3

Explanation:

F(4) = F(3) + F(2) = 2 + 1 = 3.

Constraints:

0 <= n <= 30

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```



int fib(int n){

}
```

Go:

```
func fib(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Fibonacci Number
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int fib(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Fibonacci Number
* Difficulty: Easy
* Tags: dp, math
```

```
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int fib(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Fibonacci Number
Difficulty: Easy
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def fib(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Fibonacci Number
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Fibonacci Number
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function fib(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Fibonacci Number
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int Fib(int n) {

}
}
```

C Solution:

```
/*
* Problem: Fibonacci Number
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/



int fib(int n){

}
```

Go Solution:

```
// Problem: Fibonacci Number
// Difficulty: Easy
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func fib(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Fibonacci Number
// Difficulty: Easy
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```