Climbing Stairs(VERY IMP)

Difficulty	Easy
Category	1-DP
Question	https://leetcode.com/problems/climbing-stairs/
Solution	https://youtu.be/Y0lT9Fck7qI
Status	Done

Question

> You are climbing a staircase. It takes `n` steps to reach the top.
>
> Each time you can either climb `1` or `2` steps. In how many distinct ways can you climb to the top?

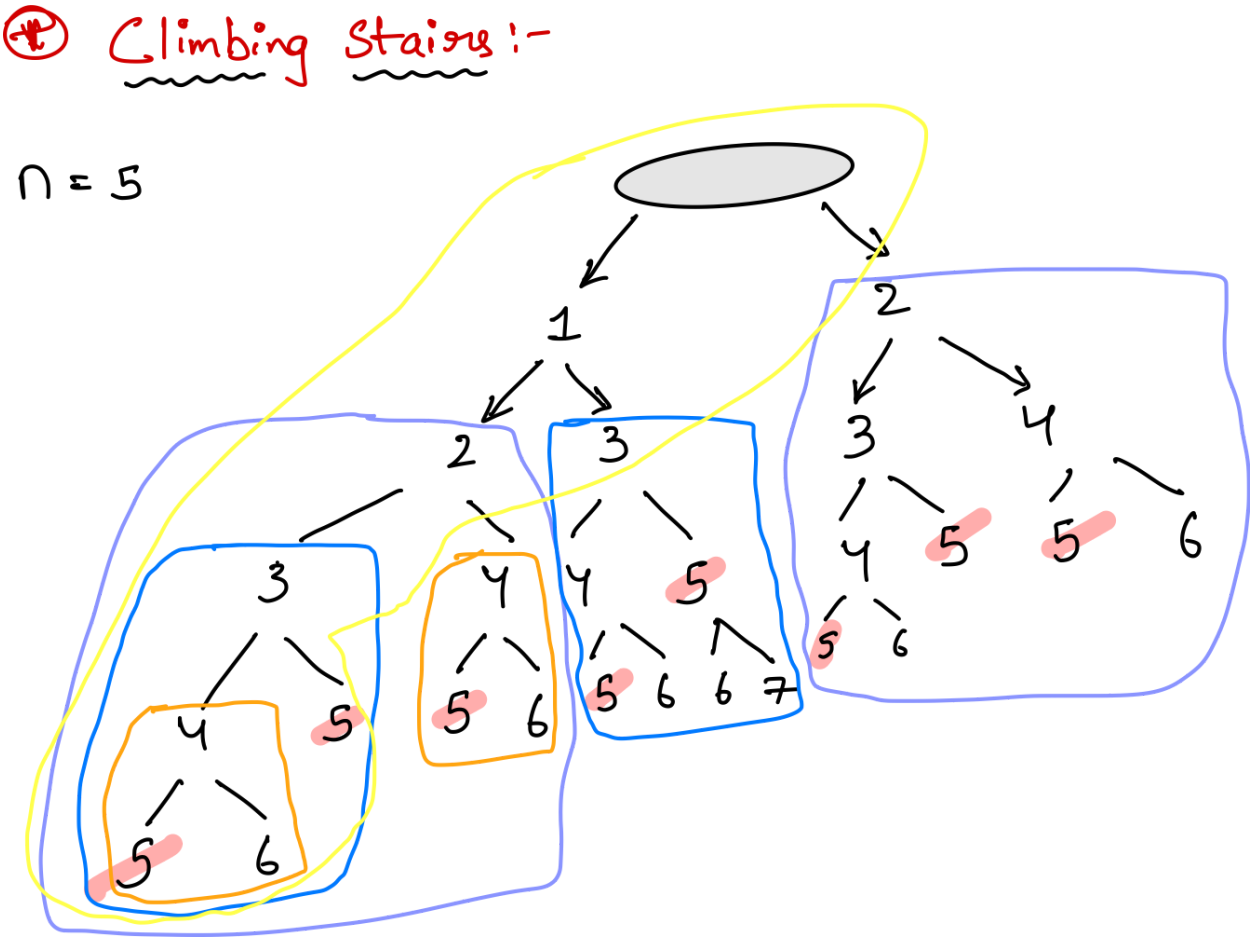
Example

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```



Idea

> 💡 Distinct ways to reach step n is the sum of ways to reach step n-1 and step n-2, since we can step 1 or 2 each time

Solution

```
# MEMOIZATION - TOPDOWN APPROACH

class Solution:
    def climbStairs(self, n: int) -> int:
        # Create a memoization table to store results of subproblems.
        memo = [-1] * (n + 1)

        def climb(n):
            if n <= 1:
                # Base case: If there are 0 or 1 steps left, there's only one way to climb.
                return 1
            if memo[n] != -1:
                # If we have already calculated the number of ways for 'n', return it from the memoization table.
```

```
                return memo[n]

            # Calculate the number of ways recursively by considering both climbing 1 step and 2 steps at a time.
            memo[n] = climb(n - 1) + climb(n - 2)
            return memo[n]

        # Start the recursive climb with 'n' steps.
        return climb(n)
```

```
# TABULATION - BOTTOMUP APPROACH

def climbStairsTabulation(n):
    if n <= 1:
        return 1

    # Create an array to store the number of ways to climb each step.
    dp = [0] * (n + 1)
    dp[0] = dp[1] = 1

    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]

    return dp[n]
```

💡 Difference between Memoization and Tabulation

> ### Memoization (Top-Down) Approach for Climbing Stairs:
>
> In the memoization approach for the "Climbing Stairs" problem, you start with a recursive algorithm that calculates the number of ways to climb 'n' steps. You then add memoization to avoid redundant calculations.
>
> 1. You begin with the top problem, which is finding the number of ways to climb 'n' steps.
> 2. As you go through the recursive calculations, you check if the result for a particular number of steps ('n') is already stored in the `memo` table. If it's there, you return the result immediately. This avoids recalculating the same subproblem multiple times.
> 3. If the result for 'n' is not found in the `memo` table, you calculate it recursively by breaking it down into smaller subproblems, such as 'n-1' and 'n-2'. You also

store the result in the `memo` table for future reference.

4. This approach is called "top-down" because you start from the top problem (climbing 'n' steps) and break it down into smaller subproblems (climbing 'n-1' and 'n-2' steps). You use recursion and cache results along the way.

Here's how it relates to the "Climbing Stairs" problem:

- When you calculate the number of ways to climb 'n' steps using memoization, you avoid redundant calculations by checking the `memo` table, which stores results for subproblems like climbing 'n-1' and 'n-2' steps.

Tabulation (Bottom-Up) Approach for Climbing Stairs:

In the tabulation approach for the "Climbing Stairs" problem, you start with the smallest subproblems (climbing 0 or 1 step) and build your way up to the top problem.

1. You begin with the smallest subproblems, where there is only one way to climb - either zero steps or one step.
2. You create an array or table (e.g., `dp`) to store the results for all subproblems. In this array, `dp[i]` represents the number of ways to climb 'i' steps.
3. You use iteration (typically a loop) to calculate the number of ways to climb 'i' steps based on the results of smaller subproblems. For example, to calculate `dp[2]`, you use the results of `dp[0]` and `dp[1]`, and so on.
4. You build up to the top problem by iteratively calculating the number of ways to climb increasing numbers of steps until you reach 'n'.

This approach is called "bottom-up" because you start from the smallest subproblems and use iteration to build up to the top problem without any recursion involved.

Here's how it relates to the "Climbing Stairs" problem:

- When you calculate the number of ways to climb 'n' steps using tabulation, you start with the smallest subproblems (climbing 0 and 1 step) and use an array (`dp`) to iteratively calculate the number of ways to climb 'n' steps based on the

> results of smaller subproblems. This avoids the need for recursion and stores results in a table.
>
> In summary, both memoization and tabulation are used to optimize the solution to the "Climbing Stairs" problem by avoiding redundant calculations. Memoization adds caching to a recursive algorithm, while tabulation uses iteration to build up results from smaller subproblems. The choice between the two approaches depends on the problem and personal preference.