# Fibonacci Numbers Pattern

This pattern covers problems where the answer for position `n` depends on the answers to **previous positions**—typically, the last one or two (sometimes more).
The classic example is the **Fibonacci sequence**:
`F(n) = F(n-1) + F(n-2)`

**Many dynamic programming problems fit this pattern:**

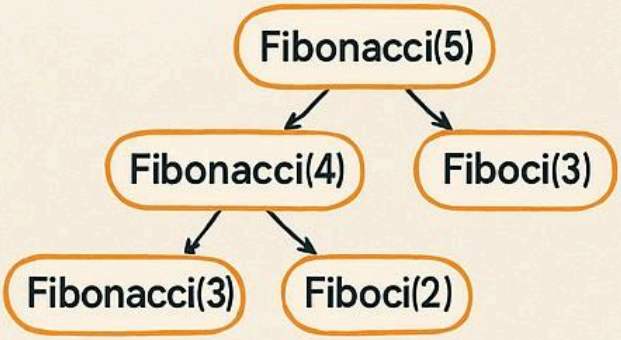
- Counting ways to climb stairs (each step can be 1 or 2 at a time)
- Tiling problems
- House thief/robber (can't rob adjacent houses)
- Jumping stairs with variable jumps

# Fibonacci Numbers Pattern:
## Fibonacci Sequence in Coding and DP

Given n, find n nth Fibonacci number, where
F(0) = 0, F(1) = 1, F(n) = F(n-1) + ((n-2)
Example: n = 7, Output; 13

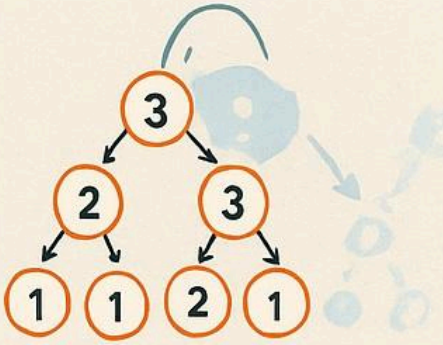
### 1. Naive Recursion

Break the problem into two smalier's subproblems




### 2. Efficient DP

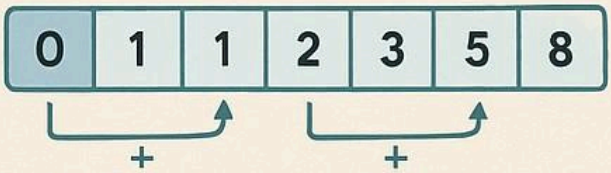
Build up solutions and reuse them!




= Fib [i] + Fib [i - 2]

### 3. Efficient DP

Build up solutions and reuse them!




Fib[j] = Fib[i-1] + Fib [i-2)

Bonus: This pattern appears in stair-climbing, tiling, and many DP problems where answers depend on previousr ıults

The Fibonacci pattern teaches recursion, overlapping

### 4. Iterative DP

Just keep the last two numbers for max efficiency


BONUS: This pattern appears in stair-climbing, tiling, and many DP problems where answers depend on previous results!

**Fib(7) = 13**

subproblems, and the power

## Classic Problem: Fibonacci Sequence

**Problem Statement:**
Given `n`, compute the `n`th Fibonacci number (`F(0) = 0`, `F(1) = 1`, `F(n) = F(n-1) + F(n-2)`).

## Explanation

1. **Naive Recursive Solution:**
   Just call the function for `n-1` and `n-2`, and add the results.
   (But this is slow! O(2^n))
2. **Efficient Dynamic Programming (Bottom-Up):**
   - Use an array (or two variables) to keep track of already computed results.
   - Build from 0 upwards to n.

## C Code Example: Iterative Fibonacci (Efficient)

```
#include <stdio.h>

int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;

    int prev2 = 0, prev1 = 1, curr;
    for (int i = 2; i <= n; i++) {
        curr = prev1 + prev2;
        prev2 = prev1;
        prev1 = curr;
    }
    return curr;
}

int main() {
    int n = 10;
    printf("Fibonacci number %d is %d\n", n, fibonacci(n));
    return 0;
}
```

## How does this work?

- Start with `0` and `1`.
- For each position up to `n`, compute the next number as the sum of the previous two.
- This is **O(n)** time and **O(1)** space.

## Where is this pattern useful?

- **Staircase Problems:**
  Count ways to climb `n` stairs, taking 1 or 2 at a time.
- **House Robber/Thief:**
  Maximum money if no two adjacent houses can be robbed.

- **Tiling:**
  Number of ways to tile a floor with tiles of certain lengths.
- **Anything where the answer depends on the last few solutions.**

## Practice Challenge

- Modify the code to **count the number of ways to climb n stairs** if you can take either 1 or 2 steps at a time.
- Try implementing the Fibonacci with **recursion + memoization** (top-down DP).