**What is Recursion?**

**Recursion** is a programming technique where a function **calls itself** to solve a smaller instance of the same problem.

* It continues until it reaches a **base case** (which doesn't make a recursive call).
* Recursion is often used to solve **problems that have a natural hierarchical or repetitive structure**, like:
  + Factorial
  + Fibonacci numbers
  + Tree traversal
  + Backtracking problems (e.g., maze solving)

**Example: Factorial using recursion**

python

Copy code

def factorial(n):

if n == 0:

return 1 # base case

return n \* factorial(n - 1) # recursive call

**How Recursion Simplifies Problems**

* Breaks down a complex problem into **smaller, more manageable subproblems**.
* Leads to **cleaner and shorter code**, especially for problems like:
  + Tree/graph traversal
  + Divide and conquer algorithms (e.g., Merge Sort)
  + Dynamic programming

**Time Complexity of Recursive Algorithms**

Time complexity depends on:

* **How many times** the function is called
* **Work done** in each call

**Example 1: Factorial**

python

Copy code

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

* Time Complexity: **O(n)**
* Space Complexity (due to call stack): **O(n)**

**Example 2: Naive Fibonacci**

python

Copy code

def fib(n):

if n <= 1:

return n

return fib(n - 1) + fib(n - 2)

* Time Complexity: **O(2^n)** (exponential!)
* Because it **recomputes** the same subproblems repeatedly.

**How to Optimize Recursive Solutions**

1. **Memoization (Top-down Dynamic Programming)**
   * Store the results of previous computations in a **cache**.
   * Prevents redundant work in recursive calls.

python

Copy code

memo = {}

def fib(n):

if n in memo:

return memo[n]

if n <= 1:

return n

memo[n] = fib(n - 1) + fib(n - 2)

return memo[n]

* + Time Complexity: **O(n)**
  + Space Complexity: **O(n)**

1. **Dynamic Programming (Bottom-up)**
   * Use iteration instead of recursion.
   * Example: Iterative Fibonacci uses **O(n)** time and **O(1)** space.
2. **Tail Recursion** (where supported)
   * Some languages optimize **tail-recursive** calls to avoid extra stack usage.
   * Python doesn’t support tail recursion optimization natively.