**Concept of Recursion**

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

A recursive function typically consists of:

**Base case** – a condition to stop recursion

**Recursive case** – the function calling itself to work toward the base case

Recursion **breaks down complex problems** into simpler, identical subproblems. It is especially useful for problems that are naturally hierarchical or repetitive, such as:

* Mathematical sequences (factorial, Fibonacci)
* Tree and graph traversal
* File system navigation
* Backtracking algorithms (e.g., sudoku, maze solving)
* Divide and conquer (e.g., merge sort)

Let’s compare **iteration** and **recursion** for the **factorial** function.

**Factorial Definition**

**🔁 Using Iteration**

int FactorialIterative(int n)

{

int result = 1;

for (int i = 2; i <= n; i++)

result \*= i;

return result;

}

**🔄 Using Recursion**

int FactorialRecursive(int n)

{

if (n == 0 || n == 1)

return 1;

return n \* FactorialRecursive(n - 1);

}

**Explanation:**

In FactorialRecursive, we are saying:

* "To compute n!, multiply n by (n-1)!"
* Eventually, we reach 1!, which is 1 — this is the **base case**

So:

FactorialRecursive(4)

= 4 \* FactorialRecursive(3)

= 4 \* 3 \* FactorialRecursive(2)

= 4 \* 3 \* 2 \* FactorialRecursive(1)

= 4 \* 3 \* 2 \* 1

= 24

In finance, many problems follow patterns that are naturally recursive — for example, **compound growth**:

FV(n)=FV(n−1)×(1+r)

Where:

* FV(n) = Future value at year n
* r = Growth rate
* FV(0) = Base value (initial investment, revenue, etc.)

Instead of using a loop to calculate this over n years, we can express it **recursively** as:

double Forecast(int n)

{

if (n == 0)

return baseValue;

return Forecast(n - 1) \* (1 + growthRate);

}

We create a method that calculates future values using a **recursive formula** based on past values and a growth rate.

double ForecastRecursive(double baseValue, double growthRate, int years)

**Time Complexity of the Recursive Algorithm**

In this implementation:

* For ForecastRecursive(n), we make one recursive call to ForecastRecursive(n - 1)
* The call chain goes all the way down to n = 0
* So the number of calls = n + 1

**Time Complexity:**

* **O(n)** — Linear time
* Each function call depends on the previous one

The recursive function **recomputes** values even when they’ve already been calculated. For example:

* To compute Forecast(5), we compute Forecast(4)
* To compute Forecast(4), we again compute Forecast(3) — and so on

This creates **overlapping subproblems**.

**Solution: Memoization**

Memoization means **storing previously computed results** in a cache (e.g., dictionary). So, when the function is called again for the same input, the result is **retrieved from cache** instead of recomputed.