Financial Forecasting using Recursive Algorithm in C#

# 1. Problem Statement

You are developing a financial forecasting tool that predicts future values based on past data. This implementation uses recursion to calculate the future value based on a given growth rate over a number of years.

# 2. Understanding Recursion

Recursion is a programming concept where a function calls itself to solve smaller instances of the same problem. Each recursive function must have a base case to avoid infinite calls. In financial forecasting, recursion can simplify the repeated application of a growth formula.

# 3. Code Implementation (C#)

Below is the recursive and iterative implementation in C#:

using System;  
  
class FinancialForecast  
{  
 // Recursive method  
 static double CalculateFutureValue(double presentValue, double growthRate, int years)  
 {  
 // Base case: if no more years, return the present value  
 if (years == 0)  
 {  
 return presentValue;  
 }  
 // Recursive case: calculate for 1 year and recurse  
 return CalculateFutureValue(presentValue \* (1 + growthRate), growthRate, years - 1);  
 }  
  
 // Iterative method (Optimized)  
 static double CalculateFutureValueIterative(double presentValue, double growthRate, int years)  
 {  
 for (int i = 0; i < years; i++)  
 {  
 presentValue \*= (1 + growthRate);  
 }  
 return presentValue;  
 }  
  
 static void Main(string[] args)  
 {  
 Console.WriteLine("Enter Present Value (e.g., 10000): ");  
 double presentValue = Convert.ToDouble(Console.ReadLine());  
  
 Console.WriteLine("Enter Annual Growth Rate (e.g., 0.10 for 10%): ");  
 double growthRate = Convert.ToDouble(Console.ReadLine());  
  
 Console.WriteLine("Enter Number of Years: ");  
 int years = Convert.ToInt32(Console.ReadLine());  
  
 double futureValueRecursive = CalculateFutureValue(presentValue, growthRate, years);  
 double futureValueIterative = CalculateFutureValueIterative(presentValue, growthRate, years);  
  
 Console.WriteLine($"  
Future Value using Recursion after {years} years: ₹{futureValueRecursive:F2}");  
 Console.WriteLine($"Future Value using Iteration after {years} years: ₹{futureValueIterative:F2}");  
 }  
}

# 4. Analysis

Time Complexity: O(n), where n is the number of years.

Recursion is elegant but can be inefficient for large inputs due to stack depth. An iterative solution is more memory-efficient and faster for large datasets.

# 5. Optimization

To avoid performance issues, the recursive approach can be replaced by an iterative one. This prevents excessive function calls and is easier to debug.

