**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Understanding and analyzing :**

Recursion is a programming technique where a method calls itself to solve a problem. It's useful for problems with a clear base case and repetitive structure, like calculating compound growth or computing Fibonacci sequences.

The recursive financial forecasting algorithm has a time complexity of O(n), where n is the number of years to forecast. Without memoization, each recursive call leads to another, creating a deep call stack that grows linearly with n. This can result in stack overflow for large input values.

By introducing memoization, we still maintain an O(n) time complexity, but avoid redundant calculations and reduce the stack depth, making the algorithm more efficient and scalable. To further optimize for performance-critical scenarios, recursion can be replaced with an iterative approach which eliminates the overhead of function calls entirely and provides better control over execution.

**Code :**

using System;

namespace FinancialForecasting

{

class Program

{

static void Main()

{

double presentValue = 1000.0;

double annualRate = 0.05;

int years = 5;

double futureValue = ForecastRecursive(presentValue, annualRate, years);

Console.WriteLine($"Future value after {years} years: {futureValue:F2}");

double[] memo = new double[years + 1];

double optimizedFutureValue = ForecastWithMemo(presentValue, annualRate, years, memo);

Console.WriteLine($"Optimized future value after {years} years: {optimizedFutureValue:F2}");

}

static double ForecastRecursive(double pv, double rate, int years)

{

if (years == 0)

return pv;

return ForecastRecursive(pv, rate, years - 1) \* (1 + rate);

}

static double ForecastWithMemo(double pv, double rate, int years, double[] memo)

{

if (years == 0)

return pv;

if (memo[years] != 0)

return memo[years];

memo[years] = ForecastWithMemo(pv, rate, years - 1, memo) \* (1 + rate);

return memo[years];

}

}

}

**ImageOutput:**