

Exercise 7: Financial Forecasting

Scenario:

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

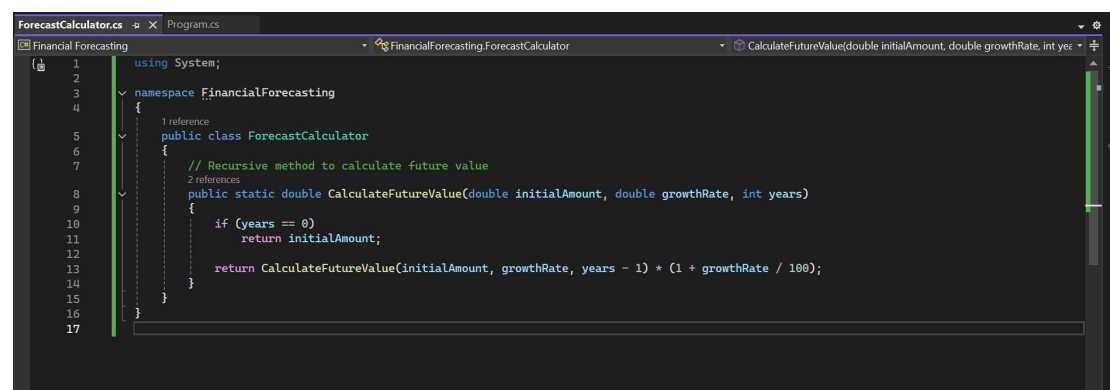
1. Understand Recursive Algorithms:

Explain the concept of recursion and how it can simplify certain problems.

Recursion is a programming concept where a function calls itself to solve smaller instances of the same problem. It continues breaking the problem into simpler sub-problems until it reaches a base case, which stops the recursion. Recursion can simplify complex problems like tree traversal, factorial calculation, or solving mazes because it naturally mirrors the problem's structure. Instead of writing long and repetitive code, recursion allows for cleaner, more readable solutions by focusing on solving one small part at a time. However, it must be used carefully to avoid infinite loops and stack overflow errors.

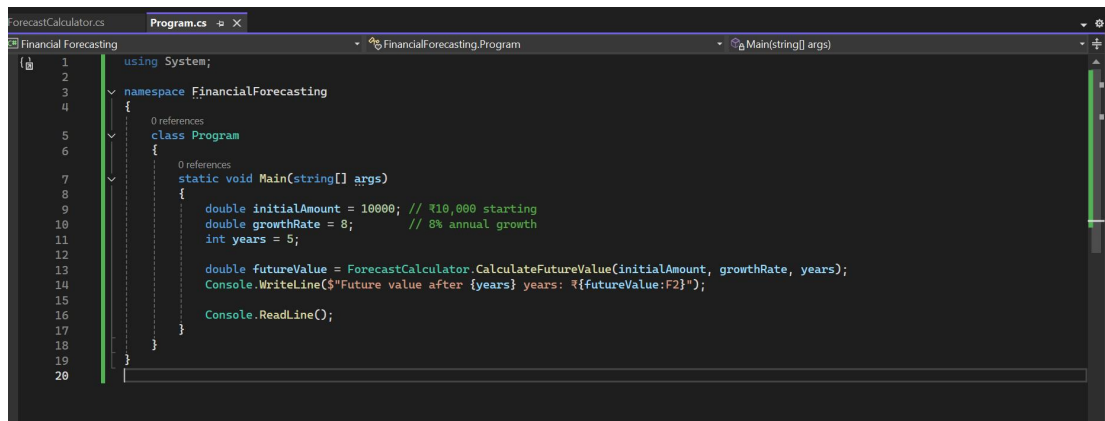
Implementation:

ForecastCalculator.cs



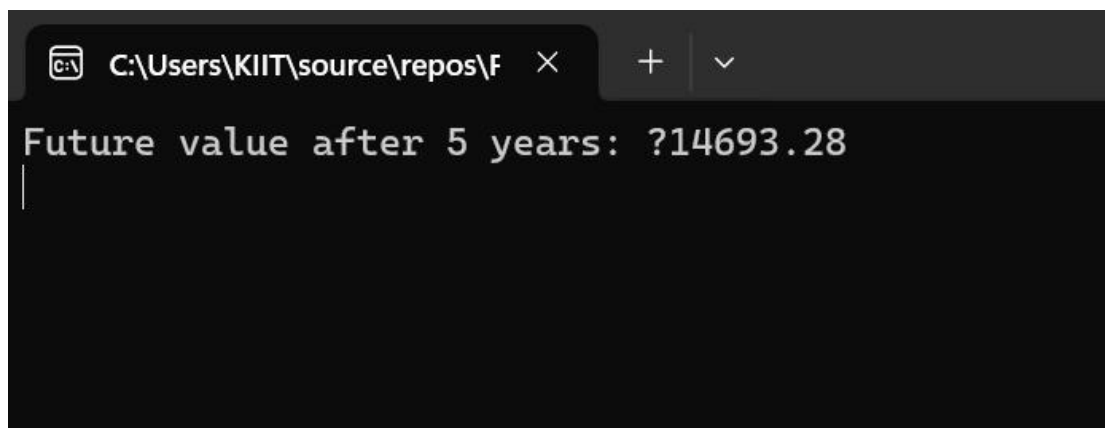
```
1 using System;
2
3 namespace FinancialForecasting
4 {
5     1 reference
6     public class ForecastCalculator
7     {
8         // Recursive method to calculate future value
9         2 references
10        public static double CalculateFutureValue(double initialAmount, double growthRate, int years)
11        {
12            if (years == 0)
13                return initialAmount;
14            return CalculateFutureValue(initialAmount, growthRate, years - 1) * (1 + growthRate / 100);
15        }
16    }
17 }
```

Program.cs



```
1 using System;
2
3 namespace FinancialForecasting
4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main(string[] args)
10        {
11            double initialAmount = 10000; // ₹10,000 starting
12            double growthRate = 8;        // 8% annual growth
13            int years = 5;
14
15            double futureValue = ForecastCalculator.CalculateFutureValue(initialAmount, growthRate, years);
16            Console.WriteLine($"Future value after {years} years: ₹{futureValue:F2}");
17
18            Console.ReadLine();
19        }
20    }
```

Output:



```
C:\Users\KIIT\source\repos\F >
Future value after 5 years: ?14693.28
```

Analysis:

Discuss the time complexity of your recursive algorithm.

The time complexity of the recursive algorithm is **$O(n)$** , where n is the number of years, because the function makes one recursive call for each year until it reaches the base case. Each call performs a constant-time multiplication and stores a stack frame, leading to a space complexity of **$O(n)$** as well. While this is manageable for small values of n , it can lead to stack overflow or performance issues for large inputs. To optimize, the recursive method can be rewritten iteratively to reduce the space complexity to **$O(1)$** .

Explain how to optimize the recursive solution to avoid excessive computation.

To optimize the recursive solution and avoid excessive computation or stack overflow, you can convert it to an **iterative approach**, which uses a simple loop instead of

recursive calls. This reduces the **space complexity from $O(n)$ to $O(1)$** since it avoids the overhead of the call stack. Alternatively, if the recursion had overlapping subproblems (not the case here), you could use **memoization** to store previously computed values and avoid redundant calculations. However, for this straightforward financial growth calculation, an **iterative solution is the most efficient and clean way to optimize the recursion.**