Week-1 Hands-On Exercise (Algorithms- Data Structures)

Exercise 7: Financial Forecasting

1) Recursion

Recursion is a programming technique where a function calls itself to solve a smaller instance of the same problem. It continues breaking the problem down until it reaches a base case (a condition where the function stops calling itself)

Recursion helps break complex problems into smaller, manageable parts. It makes code more elegant and logical, especially for problems that are naturally recursive in structure. However, it must be used carefully to avoid excessive stack usage or infinite loops

i.e.. Example Pseudocode for recursion

```
FUNCTION Factorial(n)
IF n == 0 THEN
    RETURN 1       // Base case
ELSE
    RETURN n * Factorial(n - 1)  // Recursive case
END FUNCTION
```

2) Setup

Formula for finding Future Value

Future Value (FV) = Present Value (PV) × (1 + Growth Rate) ^ n

Code for Implementing recursive method to find future value

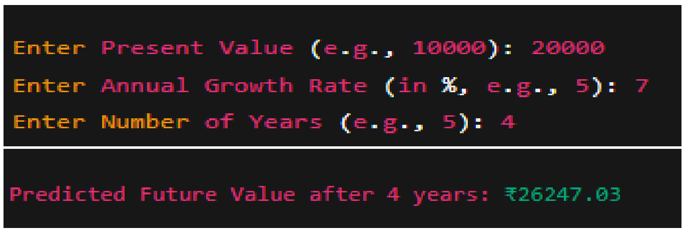
```
using System;
namespace futurevalue{
public static double PredictFutureValue(double presentValue, double growthRate, int years)
{
   if (years == 0)
      return presentValue;
   return PredictFutureValue(presentValue, growthRate, years - 1) * (1 + growthRate);
}
```

3) Implementation

```
// main code
using System;

class Program
{
  static void Main()
  {
    // Get input from user
    Console.Write("Enter Present Value (e.g., 10000): ");
    double presentValue = Convert.ToDouble(Console.ReadLine());

    Console.Write("Enter Annual Growth Rate (in %, e.g., 5): ");
    double annualGrowthRatePercent = Convert.ToDouble(Console.ReadLine());
    double growthRate = annualGrowthRatePercent / 100; // Convert to decimal

    Console.Write("Enter Number of Years (e.g., 5): ");
    int years = Convert.ToInt32(Console.ReadLine());

    // Predict future value using recursion
    double futureValue = PredictFutureValue(presentValue, growthRate, years);

    // Display result
    Console.WriteLine($"\nPredicted Future Value after {years} years: ₹{futureValue:F2}");
  }

  // Recursive function
  public static double PredictFutureValue(double presentValue, double growthRate, int years)
  {
    if (years == 0)
      return presentValue;
    return PredictFutureValue(presentValue, growthRate, years - 1) * (1 + growthRate);
  }
}
```

4) Analysis

OUTPUT :



```
Enter Present Value (e.g., 10000): 20000
Enter Annual Growth Rate (in %, e.g., 5): 7
Enter Number of Years (e.g., 5): 4

Predicted Future Value after 4 years: ₹26247.03
```

Time Complexity of above code

- The recursive method calls itself once for each year, reducing years by 1 each time.
- It performs one multiplication per call, without overlapping subproblems.
- Time Complexity: O(n) - Where n is the number of years.
- Space Complexity: O(n) - Due to the call stack storing n recursive calls.

Optimizing the algorithm/Code to avoid excessive computation

1. **Convert Recursion to Iteration:**
   Use a simple for loop to avoid stack usage. Iterative version uses constant space: **Space Complexity: O(1)**

   Example code for iteration

```
      public static double PredictFutureValueIterative(double presentValue, double growthRate, int years)
{
    double result = presentValue;
    for (int i = 0; i < years; i++)
    {
        result *= (1 + growthRate);
    }
    return result;
}
```

2. **Memoization :**
   For problems with overlapping subproblems (e.g., Fibonacci), memoization stores results. This problem doesn't need it because each step depends only on the previous one.

3. **Use Built-in Power Function :**

   Use Math.Pow() for direct calculation:
   FV = presentValue × Math.Pow(1 + rate, years)