Q)Financial Forecasting

Scenario:

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

Steps:

1. Understand Recursive Algorithms:
   * Explain the concept of recursion and how it can simplify certain problems.
2. Setup:
   * Create a method to calculate the future value using a recursive approach.
3. Implementation:
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. Analysis:
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

using System;

class Program

{

    static void Main(string[] args)

    {

        double initialValue = 1000.0;

        double annualGrowthRate = 0.05; // 5%

        int years = 10;

        double forecast = ForecastRecursive(initialValue, annualGrowthRate, years);

        Console.WriteLine($"Forecasted value after {years} years: ₹{forecast:F2}");

    }

    static double ForecastRecursive(double value, double growthRate, int years)

    {

        if (years == 0)

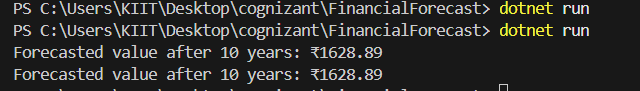
            return value;

        return ForecastRecursive(value, growthRate, years - 1) \* (1 + growthRate);

    }

}

OUTPUT



**Q) E-commerce Platform Search Function**

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

using System;

using System.Linq;

public class Product

{

    public int ProductId { get; set; }

    public string ProductName { get; set; }

    public string Category { get; set; }

    public Product(int productId, string productName, string category)

    {

        ProductId = productId;

        ProductName = productName;

        Category = category;

    }

}

class Program

{

    static void Main(string[] args)

    {

        Product[] products = new Product[]

        {

            new Product(101, "Shoes", "Fashion"),

            new Product(102, "Laptop", "Electronics"),

            new Product(103, "Book", "Stationery"),

            new Product(104, "Headphones", "Electronics"),

            new Product(105, "Watch", "Accessories")

        };

        // 🔍 Linear Search

        Console.WriteLine("Linear Search:");

        var found1 = LinearSearch(products, "Laptop");

        Console.WriteLine(found1 != null ? $"Found: {found1.ProductName}" : "Product not found");

        // 🔍 Binary Search

        var sortedProducts = products.OrderBy(p => p.ProductName).ToArray();

        Console.WriteLine("\nBinary Search (Sorted by name):");

        var found2 = BinarySearch(sortedProducts, "Laptop");

        Console.WriteLine(found2 != null ? $"Found: {found2.ProductName}" : "Product not found");

    }

    public static Product? LinearSearch(Product[] products, string target)

    {

        foreach (var product in products)

        {

            if (product.ProductName.Equals(target, StringComparison.OrdinalIgnoreCase))

                return product;

        }

        return null;

    }

    public static Product? BinarySearch(Product[] products, string target)

    {

        int left = 0;

        int right = products.Length - 1;

        while (left <= right)

        {

            int mid = (left + right) / 2;

            int comparison = string.Compare(products[mid].ProductName, target, StringComparison.OrdinalIgnoreCase);

            if (comparison == 0)

                return products[mid];

            else if (comparison < 0)

                left = mid + 1;

            else

                right = mid - 1;

        }

        return null;

    }

}

OUTPUT:

