Exercise 2: 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

ANSWER Product.cs

```
namespace AlgorithAndDataStructures
{
    public class Product
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; }
        public string Category { get; set; }

        public Product(int id, string name, string category)
        {
            ProductId = id;
            ProductName = name;
            Category = category;
        }

        public override string ToString()
        {
            return $"{ProductId}: {ProductName} ({Category})";
        }
    }

    public class ProductSearch
    {
        // Linear Search
        public static Product LinearSearch(List<Product> products, string
productName)
        {
            foreach (var product in products)
            {
```

```
                if (product.ProductName.Equals(productName,
StringComparison.OrdinalIgnoreCase))
                    return product;
            }
            return null;
        }

        // Binary Search
        public static Product BinarySearch(List<Product> sortedProducts,
string productName)
        {
            int left = 0, right = sortedProducts.Count - 1;
            while (left <= right)
            {
                int mid = (left + right) / 2;
                int cmp = string.Compare(sortedProducts[mid].ProductName,
productName, true);
                if (cmp == 0) return sortedProducts[mid];
                else if (cmp < 0) left = mid + 1;
                else right = mid - 1;
            }
            return null;
        }
    }
}
```

ANALYSIS:

Linear Search is simple but slow for large data. Time: O(n). Binary Search is much faster but requires sorted data. Time: O(log n). We can use binary search with a sorted array or a more advanced structure like Trie or HashMap for real-world performance

Exercise 7: 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

ANSWER FinancialForecast.cs

```
namespace AlgorithAndDataStructures
{
    public class FinancialForecast
    {
        // Recursive method to forecast future value
        public static double ForecastRecursive(double amount, double rate,
int years)
        {
            if (years == 0)
                return amount;
            return ForecastRecursive(amount * (1 + rate), rate, years - 1);
        }

    }
}
```

ANALYSIS:

Time Complexity: O(n) due to n recursive calls

Drawback: Can cause stack overflow for very large n

Optimization: Use iteration or memoization

Program.cs

```
using AlgorithAndDataStructures;
using System;
using System.Collections.Generic;


class Program
{
    static void Main()
    {
        Console.WriteLine("===== Financial Forecasting =====");
        double initialAmount = 1000.0;
        double annualRate = 0.10;
        int years = 5;
        double forecast = FinancialForecast.ForecastRecursive(initialAmount,
annualRate, years);
        Console.WriteLine($"Recursive Forecast after {years} years:
Rs{forecast:F2}");


        Console.WriteLine("\n===== E-commerce Platform Search =====");
        List<Product> products = new List<Product>
        {
            new Product(1, "Laptop", "Electronics"),
            new Product(2, "Shoes", "Fashion"),
            new Product(3, "Book", "Education"),
```

```csharp
            new Product(4, "Phone", "Electronics"),
            new Product(5, "Watch", "Accessories")
        };

        Console.WriteLine("\nLinear Search:");
        var result1 = ProductSearch.LinearSearch(products, "Phone");
        if (result1 != null)
            Console.WriteLine(result1);
        else
            Console.WriteLine("Product not found");

        products.Sort((x, y) => x.ProductName.CompareTo(y.ProductName));

        Console.WriteLine("\nBinary Search:");
        var result2 = ProductSearch.BinarySearch(products, "Phone");
        if (result2 != null)
            Console.WriteLine(result2);
        else
            Console.WriteLine("Product not found");
    }
}
```

```
Microsoft Visual Studio Debu
===== Financial Forecasting =====
Recursive Forecast after 5 years: Rs1610.51

===== E-commerce Platform Search =====

Linear Search:
4: Phone (Electronics)

Binary Search:
4: Phone (Electronics)

C:\Users\KIIT\Documents\Digital-Nurture-4.0-DotNetFSE-solution\week1\Algorithms_Data Structures\Codes\Financial Forecast
ing\FinancialForecastSolution\FinancialForecast\bin\Debug\net8.0\AlgorithAndDataStructure.exe (process 62964) exited wit
h code 0 (0x0).
Press any key to close this window . . .
```