Exercise 2: E-commerce Platform Search Function

Scenario:

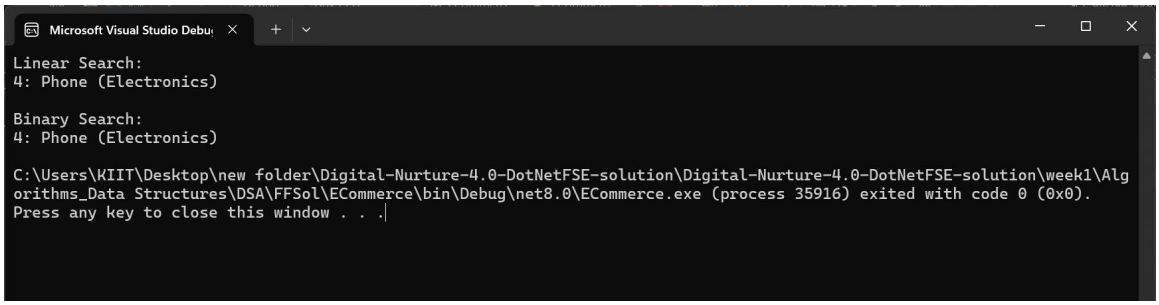
You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

Steps:

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

```
namespace ECommerce
{
    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})";
        }
    }

    class Search
    {
        // 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;
        }

        static void Main()
        {
            List<Product> products = new List<Product>
            {
                new Product(1, "Laptop", "Electronics"),
                new Product(2, "Shoes", "Fashion"),
                new Product(3, "Book", "Education"),
                new Product(4, "Phone", "Electronics"),
                new Product(5, "Watch", "Accessories")
            };

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

            // Binary Search
            products.Sort((x, y) => x.ProductName.CompareTo(y.ProductName));
            Console.WriteLine("\nBinary Search:");
            var result2 = BinarySearch(products, "Phone");
            if (result2 != null)
                Console.WriteLine(result2);
            else
                Console.WriteLine("Product not found");

        }
    }
}
```



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

```
namespace FF
{
    class Forecast
    {
        // Recursive method to calculate future value
        public static double PredictFutureValue(double initial, double
growthRate, int years)
        {
            if (years == 0)
                return initial;

            return PredictFutureValue(initial, growthRate, years - 1) * (1 +
growthRate);
        }

        static void Main()
        {
```

```
            double initialValue = 10000; // Initial amount
            double growthRate = 0.10;    // 10% annual growth
            int years = 5;

            double futureValue = PredictFutureValue(initialValue, growthRate,
years);
            Console.WriteLine($"Predicted Value after {years} years:
{futureValue:F2}");
        }
    }
}
```

ANALYSIS:

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

Drawback: Can cause stack overflow for very large n

Optimization: Use iteration or memoization