Week 1: Data Structures And Algorithms

1. **Exercise 2**

Code:

namespace ECommerceSearch

{

    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;

        }

    }

}

using System;

using System.Linq;

namespace ECommerceSearch

{

    class Program

    {

        static void Main(string[] args)

        {

            Product[] products = {

                new Product(1, "Laptop", "Electronics"),

                new Product(2, "Chair", "Furniture"),

                new Product(3, "Keyboard", "Electronics"),

                new Product(4, "Notebook", "Stationery"),

                new Product(5, "Mouse", "Electronics")

            };

            Console.WriteLine("--- Linear Search ---");

            int index1 = SearchAlgorithms.LinearSearch(products, "Mouse");

            Console.WriteLine(index1 != -1 ? $"Found at index {index1}" : "Not found");

            Console.WriteLine("--- Binary Search ---");

            // Sort before binary search

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

            int index2 = SearchAlgorithms.BinarySearch(sortedProducts, "Mouse");

            Console.WriteLine(index2 != -1 ? $"Found at index {index2}" : "Not found");

            Console.WriteLine("\nLinear Search: O(n), Binary Search: O(log n)");

            Console.WriteLine("Binary Search is more suitable for sorted data and large product lists.");

        }

    }

}

using System;

namespace ECommerceSearch

{

    public class SearchAlgorithms

    {

        public static int LinearSearch(Product[] products, string name)

        {

            for (int i = 0; i < products.Length; i++)

            {

                if (products[i].ProductName == name)

                    return i;

            }

            return -1;

        }

        public static int BinarySearch(Product[] products, string name)

        {

            int left = 0;

            int right = products.Length - 1;

            while (left <= right)

            {

                int mid = left + (right - left) / 2;

                int comparison = string.Compare(products[mid].ProductName, name);

                if (comparison == 0)

                    return mid;

                else if (comparison < 0)

                    left = mid + 1;

                else

                    right = mid - 1;

            }

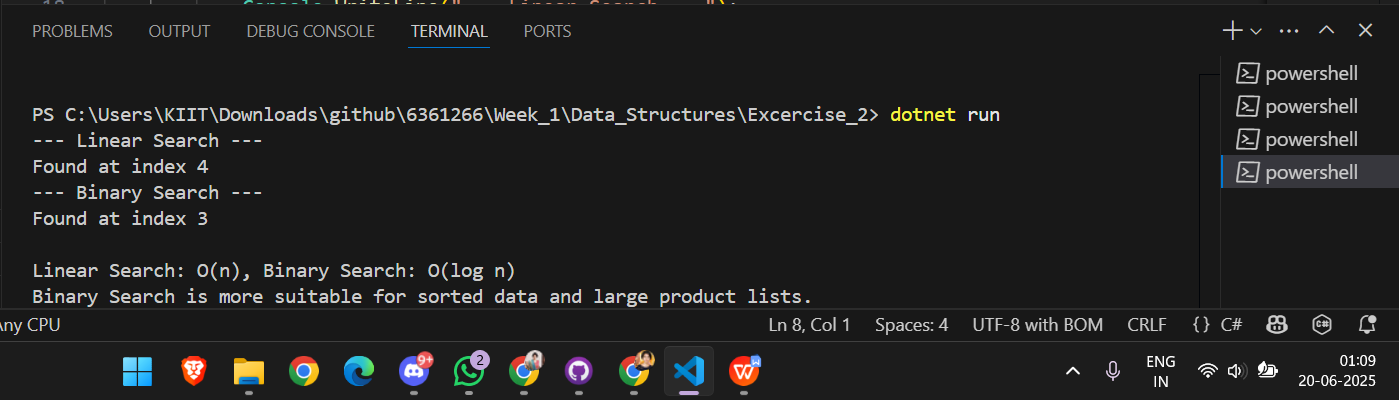
            return -1;

        }

    }

}

Output:



1. **Exercise 7**

Code:

namespace FinancialForecasting

{

    public class Forecast

    {

        public static double FutureValue(double presentValue, double rate, int years)

        {

            if (years == 0)

                return presentValue;

            return (1 + rate) \* FutureValue(presentValue, rate, years - 1);

        }

        public static double FutureValueMemo(double presentValue, double rate, int years, double[] memo)

        {

            if (years == 0)

                return presentValue;

            if (memo[years] != 0)

                return memo[years];

            memo[years] = (1 + rate) \* FutureValueMemo(presentValue, rate, years - 1, memo);

            return memo[years];

        }

    }

}

using System;

namespace FinancialForecasting

{

    class Program

    {

        static void Main(string[] args)

        {

            double presentValue = 1000.0;

            double rate = 0.05;

            int years = 5;

            Console.WriteLine("--- Recursive Forecast ---");

            double future = Forecast.FutureValue(presentValue, rate, years);

            Console.WriteLine($"Future Value after {years} years: {future:F2}");

            Console.WriteLine("--- Memoized Forecast ---");

            double[] memo = new double[years + 1];

            double futureMemo = Forecast.FutureValueMemo(presentValue, rate, years, memo);

            Console.WriteLine($"Memoized Future Value after {years} years: {futureMemo:F2}");

            Console.WriteLine("\nTime Complexity:");

            Console.WriteLine("- Recursive: O(n)");

            Console.WriteLine("- Optimized (memoized): O(n) with space O(n)");

        }

    }

}

Output:

