Exercise 2: E-commerce Platform Search Function

Program :

using System;

using System.Diagnostics;

namespace ECommerceSearch

{

    public class Product : IComparable<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 int CompareTo(Product other)

        {

            return ProductId.CompareTo(other.ProductId);

        }

        public override string ToString()

        {

            return $"ID: {ProductId}, Name: {ProductName}, Category: {Category}";

        }

    }

    public class SearchManager

    {

        private Product[] productsArray;

        private Product[] sortedProductsArray;

        public SearchManager(Product[] products)

        {

            productsArray = (Product[])products.Clone();

            sortedProductsArray = (Product[])products.Clone();

            Array.Sort(sortedProductsArray);

        }

        public Product LinearSearchById(int productId)

        {

            foreach (var product in productsArray)

            {

                if (product.ProductId == productId)

                {

                    return product;

                }

            }

            return null;

        }

        public Product BinarySearchById(int productId)

        {

            int left = 0;

            int right = sortedProductsArray.Length - 1;

            while (left <= right)

            {

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

                if (sortedProductsArray[middle].ProductId == productId)

                {

                    return sortedProductsArray[middle];

                }

                if (sortedProductsArray[middle].ProductId < productId)

                {

                    left = middle + 1;

                }

                else

                {

                    right = middle - 1;

                }

            }

            return null;

        }

        public void PrintAllProducts()

        {

            Console.WriteLine("All Products (Original Order):");

            foreach (var product in productsArray)

            {

                Console.WriteLine(product);

            }

            Console.WriteLine("\nAll Products (Sorted by ID):");

            foreach (var product in sortedProductsArray)

            {

                Console.WriteLine(product);

            }

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Product[] products = new Product[]

            {

                new Product(102, "Wireless Mouse", "Electronics"),

                new Product(105, "Bluetooth Headphones", "Electronics"),

                new Product(101, "Mechanical Keyboard", "Electronics"),

                new Product(104, "Smart Watch", "Wearables"),

                new Product(103, "USB-C Cable", "Accessories")

            };

            SearchManager searchManager = new SearchManager(products);

            searchManager.PrintAllProducts();

            TestSearch(searchManager, 101, "Existing product (first in sorted)");

            TestSearch(searchManager, 103, "Existing product (middle in sorted)");

            TestSearch(searchManager, 105, "Existing product (last in sorted)");

            TestSearch(searchManager, 999, "Non-existing product");

        }

        static void TestSearch(SearchManager searchManager, int productId, string scenario)

        {

            Console.WriteLine($"\nScenario: {scenario} (ID: {productId})");

            var stopwatch = Stopwatch.StartNew();

            var productLinear = searchManager.LinearSearchById(productId);

            stopwatch.Stop();

            Console.WriteLine($"Linear Search - Time: {stopwatch.ElapsedTicks} ticks");

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

            stopwatch.Restart();

            var productBinary = searchManager.BinarySearchById(productId);

            stopwatch.Stop();

            Console.WriteLine($"Binary Search - Time: {stopwatch.ElapsedTicks} ticks");

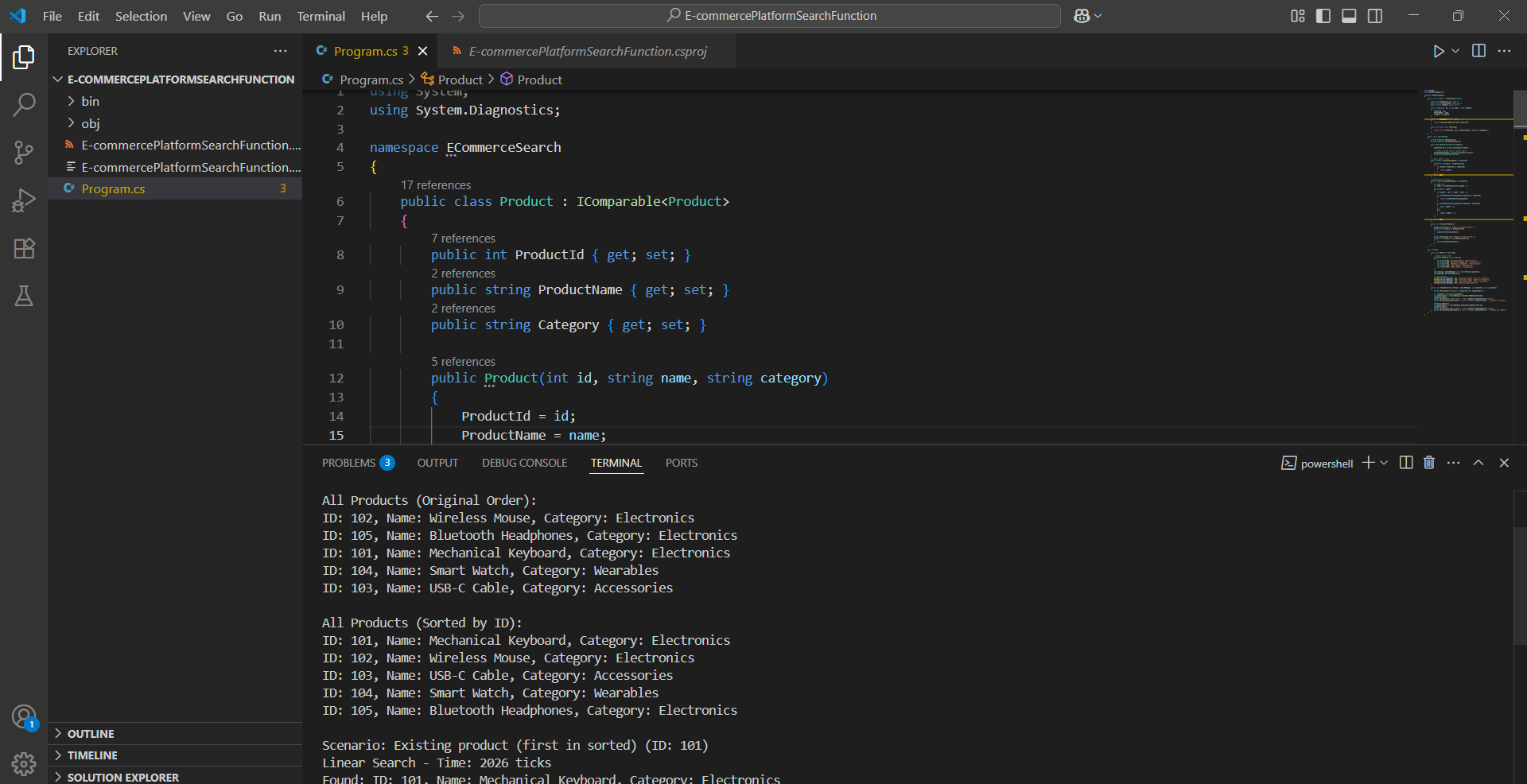
            Console.WriteLine(productBinary != null ? $"Found: {productBinary}" : "Product not found");

        }

    }

}

Output :



Exercise 7: Financial Forecasting

Program :

using System;

using System.Collections.Generic;

using System.Diagnostics;

namespace FinancialForecasting

{

    public class FinancialForecaster

    {

        public decimal CalculateFutureValueRecursive(decimal currentValue, decimal annualGrowthRate, int years)

        {

            if (years <= 0)

            {

                return currentValue;

            }

            decimal nextValue = currentValue \* (1 + annualGrowthRate / 100m);

            return CalculateFutureValueRecursive(nextValue, annualGrowthRate, years - 1);

        }

        private Dictionary<(decimal, decimal, int), decimal> memoCache = new Dictionary<(decimal, decimal, int), decimal>();

        public decimal CalculateFutureValueMemoized(decimal currentValue, decimal annualGrowthRate, int years)

        {

            var key = (currentValue, annualGrowthRate, years);

            if (memoCache.TryGetValue(key, out decimal cachedValue))

            {

                return cachedValue;

            }

            if (years <= 0)

            {

                memoCache[key] = currentValue;

                return currentValue;

            }

            decimal nextValue = currentValue \* (1 + annualGrowthRate / 100m);

            decimal result = CalculateFutureValueMemoized(nextValue, annualGrowthRate, years - 1);

            memoCache[key] = result;

            return result;

        }

        public decimal CalculateFutureValueIterative(decimal currentValue, decimal annualGrowthRate, int years)

        {

            decimal result = currentValue;

            for (int i = 0; i < years; i++)

            {

                result \*= (1 + annualGrowthRate / 100m);

            }

            return result;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            FinancialForecaster forecaster = new FinancialForecaster();

            decimal initialValue = 10000m;

            decimal growthRate = 5m;

            int years = 10;

            var stopwatch = Stopwatch.StartNew();

            decimal recursiveResult = forecaster.CalculateFutureValueRecursive(initialValue, growthRate, years);

            stopwatch.Stop();

            Console.WriteLine($"Recursive Result: {recursiveResult:C2}");

            Console.WriteLine($"Time taken: {stopwatch.ElapsedTicks} ticks");

            stopwatch.Restart();

            decimal memoizedResult = forecaster.CalculateFutureValueMemoized(initialValue, growthRate, years);

            stopwatch.Stop();

            Console.WriteLine($"\nMemoized Result: {memoizedResult:C2}");

            Console.WriteLine($"Time taken (first run): {stopwatch.ElapsedTicks} ticks");

            stopwatch.Restart();

            decimal iterativeResult = forecaster.CalculateFutureValueIterative(initialValue, growthRate, years);

            stopwatch.Stop();

            Console.WriteLine($"\nIterative Result: {iterativeResult:C2}");

            Console.WriteLine($"Time taken: {stopwatch.ElapsedTicks} ticks");

            Console.WriteLine("\nTesting memoization with repeated calculations...");

            stopwatch.Restart();

            for (int i = 0; i < 1000; i++)

            {

                forecaster.CalculateFutureValueMemoized(initialValue, growthRate, years);

            }

            stopwatch.Stop();

            Console.WriteLine($"1000 memoized calls: {stopwatch.ElapsedTicks} ticks");

            stopwatch.Restart();

            for (int i = 0; i < 1000; i++)

            {

                forecaster.CalculateFutureValueRecursive(initialValue, growthRate, years);

            }

            stopwatch.Stop();

            Console.WriteLine($"1000 recursive calls: {stopwatch.ElapsedTicks} ticks");

        }

    }

}

Output :

