ALGORITHM AND DATA STRUCTURES

Exercise 2: E-commerce Platform Search Function

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

Program.cs:

using System;

using System.Linq;

namespace EcommerceSearchApp

{

    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;

        }

        public override string ToString()

        {

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

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Product[] products = new Product[]

            {

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

                new Product(2, "Shoes", "Footwear"),

                new Product(3, "Watch", "Accessories"),

                new Product(4, "Mouse", "Electronics"),

                new Product(5, "T-Shirt", "Clothing")

            };

            Console.WriteLine("Enter product name to search:");

            string searchTerm = Console.ReadLine();

            Console.WriteLine("\n Linear Search:");

            var result1 = LinearSearch(products, searchTerm);

            Console.WriteLine(result1 != null ? $"Found: {result1}" : "Product not found.");

            Console.WriteLine("\n Binary Search (sorted by ProductName):");

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

            var result2 = BinarySearch(sortedProducts, searchTerm);

            Console.WriteLine(result2 != null ? $"Found: {result2}" : "Product not found.");

        }

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

        {

            foreach (var product in products)

            {

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

                {

                    return product;

                }

            }

            return null;

        }

        static Product BinarySearch(Product[] sortedProducts, string name)

        {

            int left = 0;

            int right = sortedProducts.Length - 1;

            while (left <= right)

            {

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

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

                if (comparison == 0)

                    return sortedProducts[mid];

                else if (comparison < 0)

                    left = mid + 1;

                else

                    right = mid - 1;

            }

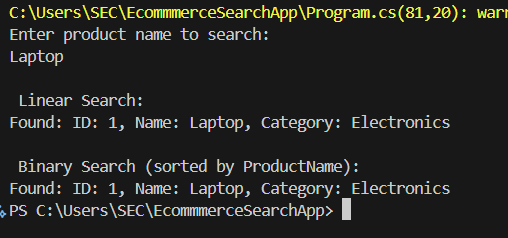
            return null;

        }

    }

}

Output:



which algorithm is more suitable for your platform and why:

For small scale platform both linear and binary is fine but for large scale platform binary search is efficient if the array is sorted due 0(log n) time complexity.

Exercise 7: Financial Forecasting

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

Program.cs:

using System;

namespace FinancialForecasting

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Enter current value:");

            double currentValue = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Enter annual growth rate (as percentage):");

            double rate = Convert.ToDouble(Console.ReadLine()) / 100;

            Console.WriteLine("Enter number of years to forecast:");

            int years = Convert.ToInt32(Console.ReadLine());

            double result = ForecastValue(currentValue, rate, years);

            Console.WriteLine($"\n Predicted value after {years} years: {result:F2}");

            Console.WriteLine("\n Optimized (with memoization):");

            double optimizedResult = ForecastValueMemo(currentValue, rate, years, new double?[years + 1]);

            Console.WriteLine($"Predicted value after {years} years: {optimizedResult:F2}");

        }

        static double ForecastValue(double value, double rate, int years)

        {

            if (years == 0)

                return value;

            return ForecastValue(value, rate, years - 1) \* (1 + rate);

        }

        static double ForecastValueMemo(double value, double rate, int years, double?[] memo)

        {

            if (years == 0)

                return value;

            if (memo[years] != null)

                return memo[years].Value;

            memo[years] = ForecastValueMemo(value, rate, years - 1, memo) \* (1 + rate);

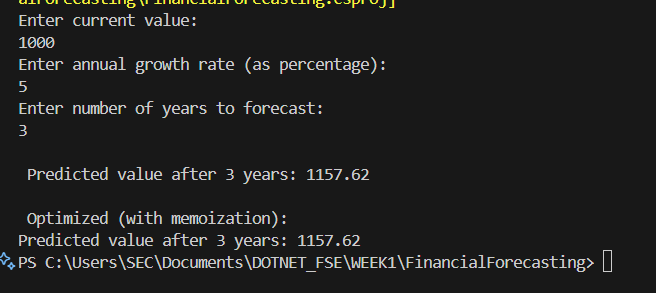
            return memo[years].Value;

        }

    }

}

Output :



how to optimize the recursive solution to avoid excessive computation:

To avoid excessive computation in recursion, use memoization to store and reuse previously computed results. This reduces time complexity from exponential to linear. Alternatively, use an iterative approach to improve both speed and memory efficiency.