**Cognizant - DN 4.0 I Deep Skilling**

**WEEK-1**

**Data Structures & Algorithms**

**Exercise 2: E-commerce Platform Search Function (in c#)**

**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.

**SOLUTION :**

**CODE -:**

using System;

using System.Diagnostics;

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)

    {

        if (other == null) return 1;

        return ProductId.CompareTo(other.ProductId);

    }

    public override string ToString()

    {

        return $"[{ProductId}] {ProductName} - {Category}";

    }

}

class Program

{

    // Linear Search: O(n)

    static Product? LinearSearch(Product[] products, int productId)

    {

        foreach (var product in products)

        {

            if (product.ProductId == productId)

                return product;

        }

        return null;

    }

    // Binary Search: O(log n) — requires sorted array

    static Product? BinarySearch(Product[] products, int productId)

    {

        int left = 0;

        int right = products.Length - 1;

        while (left <= right)

        {

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

            if (products[mid].ProductId == productId)

                return products[mid];

            else if (products[mid].ProductId < productId)

                left = mid + 1;

            else

                right = mid - 1;

        }

        return null;

    }

    static void Main(string[] args)

    {

        // Sample product list

        Product[] productList = {

        new Product(103, "Shoes", "Footwear"),

        new Product(101, "Laptop", "Electronics"),

        new Product(105, "Watch", "Accessories"),

        new Product(102, "Phone", "Electronics"),

        new Product(104, "T-shirt", "Clothing")

    };

        int searchId = 105;

        Console.WriteLine("🔍 Linear Search (Unsorted List):");

        var swLinear = Stopwatch.StartNew();

        var foundLinear = LinearSearch(productList, searchId);

        swLinear.Stop();

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

        Console.WriteLine($"Linear Search Time: {swLinear.Elapsed.TotalMilliseconds} ms");

        Console.WriteLine("Time Complexity: O(n)");

        Console.WriteLine("\n🔍 Binary Search (Sorted List):");

        Array.Sort(productList);

        var swBinary = Stopwatch.StartNew();

        var foundBinary = BinarySearch(productList, searchId);

        swBinary.Stop();

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

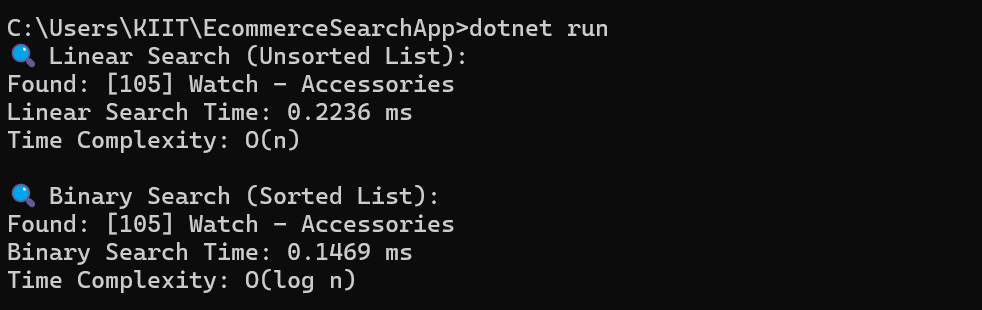
        Console.WriteLine($"Binary Search Time: {swBinary.Elapsed.TotalMilliseconds} ms");

        Console.WriteLine("Time Complexity: O(log n)");

    }

}

**OUTPUT -:**



**Exercise 7: Financial Forecasting (in c#)**

**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.

**SOLUTION :**

**CODE -:**

using System;

using System.Collections.Generic;

using System.Diagnostics;

class Program

{

    // Recursive version without memoization

    static double ForecastRecursive(int years, double currentValue, double growthRate)

    {

        if (years == 0)

            return currentValue;

        return ForecastRecursive(years - 1, currentValue, growthRate) \* (1 + growthRate);

    }

    // Optimized version using memoization (caching)

    static double ForecastMemo(int years, double currentValue, double growthRate, Dictionary<int, double> cache)

    {

        if (cache.ContainsKey(years))

            return cache[years];

        if (years == 0)

            return currentValue;

        double result = ForecastMemo(years - 1, currentValue, growthRate, cache) \* (1 + growthRate);

        cache[years] = result;

        return result;

    }

    static void Main(string[] args)

    {

        double initialAmount = 10000;     // Starting value

        double annualGrowthRate = 0.08;   // 8% annual growth

        int years = 30;

        Console.WriteLine("📈 Financial Forecasting using Recursion:");

        var swRecursive = Stopwatch.StartNew();

        double recursiveResult = ForecastRecursive(years, initialAmount, annualGrowthRate);

        swRecursive.Stop();

        Console.WriteLine($"Future value (recursive) after {years} years: ₹{recursiveResult:F2}");

        Console.WriteLine($"Execution Time: {swRecursive.Elapsed.TotalMilliseconds} ms");

        Console.WriteLine("Time Complexity: O(n)");

        Console.WriteLine("\n🚀 Financial Forecasting using Memoization:");

        var swMemo = Stopwatch.StartNew();

        var cache = new Dictionary<int, double>();

        double optimizedResult = ForecastMemo(years, initialAmount, annualGrowthRate, cache);

        swMemo.Stop();

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

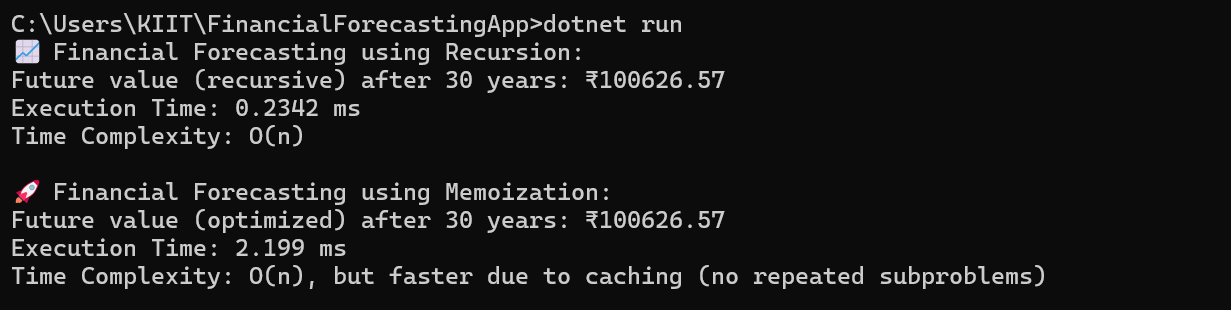
        Console.WriteLine($"Execution Time: {swMemo.Elapsed.TotalMilliseconds} ms");

        Console.WriteLine("Time Complexity: O(n), but faster due to caching (no repeated subproblems)");

    }

}

**OUTPUT -:**



-------------------------------------------------------------------------------