🔹**Exercise 2: E-commerce Platform Search Function**

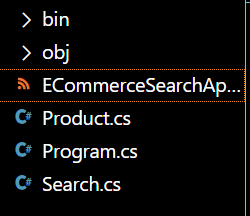
**Scenario:**

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:**
   * 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.

#### FILE STRUCTURE



#### 1. Understand Asymptotic Notation

**Big O Notation** describes the upper bound of an algorithm’s runtime:

**Best case**: minimum time the algorithm takes

(e.g., O(1) if the match is first).

**Average case**: expected time, assuming inputs are random

(e.g., O(n/2) for linear).

**Worst case**: maximum time, often used for upper bound analysis (e.g., O(n) for linear,

O(log n) for binary search).

Code

**product.cs**

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;

}

}

**program.cs**

using System;

class Program

{

static void Main(string[] args)

{

Product[] products = {

new Product(1, "Shoes", "Footwear"),

new Product(2, "Laptop", "Electronics"),

new Product(3, "Shirt", "Clothing"),

new Product(4, "Watch", "Accessories")

};

// Sort by product name for binary search

Array.Sort(products, (x, y) => string.Compare(x.ProductName, y.ProductName));

string searchTerm = "Laptop";

var linearResult = Search.LinearSearch(products, searchTerm);

Console.WriteLine(linearResult != null ? $"Linear Search Found: {linearResult.ProductName}" : "Linear Search: Not found");

var binaryResult = Search.BinarySearch(products, searchTerm);

Console.WriteLine(binaryResult != null ? $"Binary Search Found: {binaryResult.ProductName}" : "Binary Search: Not found");

}

}

**search.cs**

using System;

public static class Search

{

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

{

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

{

if (products[i].ProductName.Equals(name, StringComparison.OrdinalIgnoreCase))

return i;

}

return -1;

}

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

**OUTPUT**



**Exercise 7: Financial Forecasting**

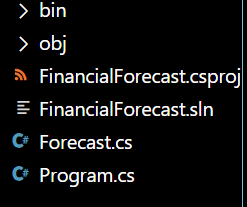
**Scenario:**

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

**Steps:**

1. **Understand Recursive Algorithms:**
   * 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.

**FILE STRUCTURE**



**CODE**

**Forcast.cs**

using System;

public class Forecast

{

// Recursive method to calculate future value

public static double PredictFutureValue(double presentValue, double growthRate, int years)

{

if (years == 0)

return presentValue;

return PredictFutureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

}

// Optimized version using iteration (optional)

public static double PredictFutureValueIterative(double presentValue, double growthRate, int years)

{

double futureValue = presentValue;

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

{

futureValue \*= (1 + growthRate);

}

return futureValue;

}

}

**program.cs**

using System;

class Program

{

static void Main()

{

Console.WriteLine("=== Financial Forecasting Tool ===");

double presentValue = 10000;

double growthRate = 0.05; // 5%

int years = 5;

double futureValue = Forecast.PredictFutureValue(presentValue, growthRate, years);

Console.WriteLine($"\nFuture value after {years} years (recursive): ₹{futureValue:F2}");

double futureValueIter = Forecast.PredictFutureValueIterative(presentValue, growthRate, years);

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

}

}

**OUTPUT**

