**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.

#1Creating a Product Class

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;

}

public override string ToString()

{

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

}

}

#2 Implementing Linear and binary search

using System;

public static class ProductSearch

{

// Linear Search - O(n)

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

{

foreach (var product in products)

{

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

return product;

}

return null;

}

// Binary Search - O(log n)

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

{

int left = 0, right = sortedProducts.Length - 1;

while (left <= right)

{

int mid = (left + right) / 2;

int cmp = string.Compare(sortedProducts[mid].ProductName, name, StringComparison.OrdinalIgnoreCase);

if (cmp == 0)

return sortedProducts[mid];

else if (cmp < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

}

#3 Analysing the Search

using System;

using System.Linq;

class Program

{

static void Main()

{

// Create a list of products

Product[] products = new Product[]

{

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

new Product(2, "Shirt", "Clothing"),

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

new Product(4, "Shoes", "Footwear"),

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

};

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

var linearResult = ProductSearch.LinearSearch(products, "Phone");

Console.WriteLine(linearResult != null ? linearResult.ToString() : "Product not found");

Console.WriteLine("\n=== Binary Search ===");

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

var binaryResult = ProductSearch.BinarySearch(sortedProducts, "Phone");

Console.WriteLine(binaryResult != null ? binaryResult.ToString() : "Product not found");

}

}

#4 Time Complexity Analysis of Algorithms

**#Linear Search**

Time Complexity**:** O(n)

Pros**:** No need to sort the list

Cons**:** Slower on large datasets

**#Binary Search**

TimeComplexity**:** O(log n)

Pros**:** Fast for large, sorted datasets

Cons**:** Requires pre-sorted list (adds O(n log n) for sorting)

#Final Analysis of Algorithm

Binary Search is more suitable in working with a sorted list and performance is important.  
However, in a real-world e-commerce platform, using Binary Search would be even more efficient than binary or linear search.

**Exercise 7 : Financial Forecasting**

**Scenario:**

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

**#1 Recursion**

Recursion is when a method calls itself to solve a smaller part of the problem.

In financial forecasting, recursion can simplify problems like compound interest or repeated growth over time.

**#2 Setting up and Implementing Recursion Algorithm**

using System;

class FinancialForecast

{

// Recursive method to calculate future value

public static double CalculateFutureValue(double presentValue, double rate, int periods)

{

// Base case: no more periods

if (periods == 0)

return presentValue;

// Recursive case: apply growth to the result of previous period

return (1 + rate) \* CalculateFutureValue(presentValue, rate, periods - 1);

}

static void Main()

{

double presentValue = 1000; // initial investment

double annualGrowthRate = 0.05; // 5% annual growth

int years = 5;

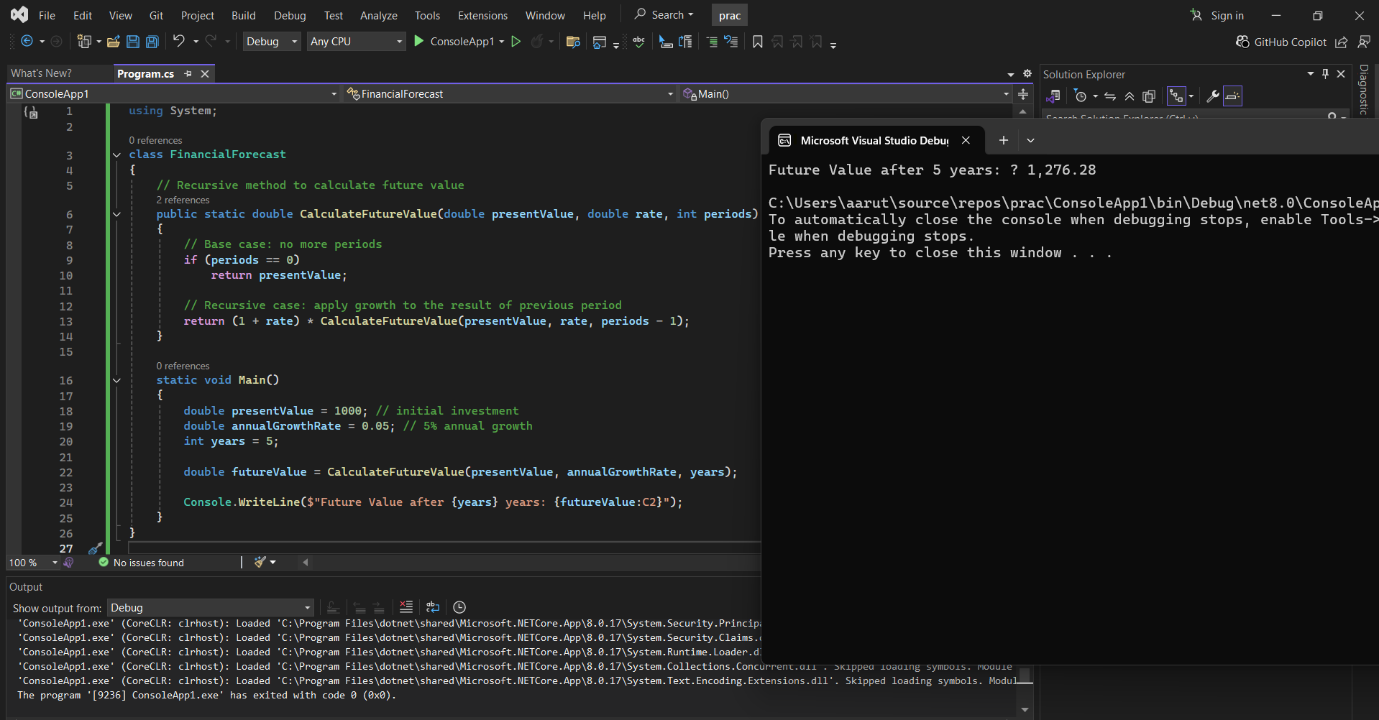
double futureValue = CalculateFutureValue(presentValue, annualGrowthRate, years);

Console.WriteLine($"Future Value after {years} years: {futureValue:C2}");

}

}

**#Output**



**#Analysis**

#Time Complexity

Recursive calls = n, where n is the number of periods

Time Complexity: O(n)

Space Complexity: O(n) (due to call stack)

#Optimization

Recursion can cause stack overflow for large n.