**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.

**SOLUTION:**  
  
using System;

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}";

}

}

class Program

{

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[] products, string name)

{

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

while (left <= right)

{

int mid = (left + right) / 2;

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

if (result == 0)

return products[mid];

else if (result < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

static void Main()

{

Product[] products = new Product[]

{

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

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

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

new Product(4, "T-Shirt", "Clothing"),

new Product(5, "Watch", "Accessories")

};

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

string input = Console.ReadLine();

var linearResult = LinearSearch(products, input);

Console.WriteLine(linearResult != null

? "Found using Linear Search: " + linearResult

: "Not found using Linear Search.");

Array.Sort(products, (p1, p2) => p1.ProductName.CompareTo(p2.ProductName));

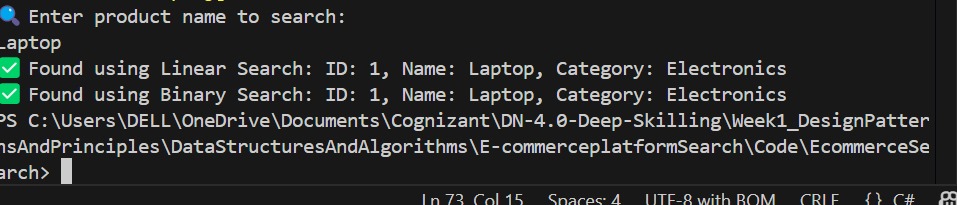
var binaryResult = BinarySearch(products, input);

Console.WriteLine(binaryResult != null

? "Found using Binary Search: " + binaryResult

: "Not found using Binary Search.");

}

}  
  
  
**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.

SOLUTION:  
  
using System;

using System.Collections.Generic;

class Program

{

static double ForecastValue(double initialAmount, double growthRate, int years)

{

if (years == 0)

return initialAmount;

return ForecastValue(initialAmount, growthRate, years - 1) \* (1 + growthRate);

}

static Dictionary<int, double> memo = new Dictionary<int, double>();

static double ForecastValueMemo(double initialAmount, double growthRate, int years)

{

if (years == 0)

return initialAmount;

if (memo.ContainsKey(years))

return memo[years];

double result = ForecastValueMemo(initialAmount, growthRate, years - 1) \* (1 + growthRate);

memo[years] = result;

return result;

}

static void Main()

{

Console.WriteLine("Enter current investment amount:");

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

Console.WriteLine("Enter annual growth rate (e.g., 0.05 for 5%):");

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

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

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

double future1 = ForecastValue(currentAmount, rate, years);

Console.WriteLine($"\nFuture Value (Recursive): {future1:F2}");

double future2 = ForecastValueMemo(currentAmount, rate, years);

Console.WriteLine($"Future Value (Memoized): {future2:F2}");

}

}  
  
  
**OUTPUT:**  
  
  
A computer screen with white text

AI-generated content may be incorrect.