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

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SearchFunctionExample

{

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 $"ProductId: {ProductId}, ProductName: {ProductName}, Category: {Category}";

}

}

class LinearSearch

{

public static Product SearchByName(Product[] products, string productName)

{

foreach (var product in products)

{

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

{

return product;

}

}

return null;

}

}

class BinarySearch

{

public static Product SearchByName(Product[] products, string productName)

{

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

while (left <= right)

{

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

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

if (comparison == 0)

{

return products[mid];

}

if (comparison < 0)

{

left = mid + 1;

}

else

{

right = mid - 1;

}

}

return null;

}

}

class Program

{

static void Main(string[] args)

{

Product[] products = {

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

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

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

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

new Product(5, "Headphones", "Audio"),

new Product(6, "Speaker", "Audio"),

new Product(7, "Tablet", "Electronics"),

new Product(8, "Charger", "Accessories"),

new Product(9, "Monitor", "Electronics"),

new Product(10, "Webcam", "Accessories")

};

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

Console.WriteLine("Linear Search:");

Product linearResult = LinearSearch.SearchByName(products, "Mouse");

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

Console.WriteLine("\nBinary Search:");

Product binaryResult = BinarySearch.SearchByName(sortedProducts, "Mouse");

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

Console.WriteLine("\nTime Complexity Analysis:");

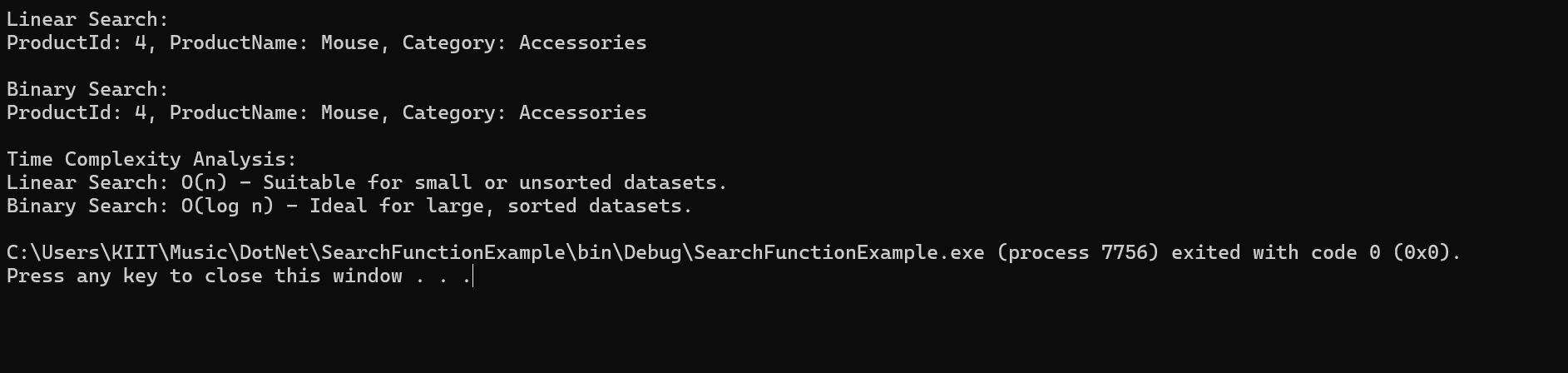
Console.WriteLine("Linear Search: O(n) - Suitable for small or unsorted datasets.");

Console.WriteLine("Binary Search: O(log n) - Ideal for large, sorted datasets.");

}

}

}



**Exercise 7: Financial Forecasting**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ForecastingExample

{

class Program

{

static double normal(double presentValue, double rate, int periods)

{

if (periods == 0)

return presentValue;

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

}

static double optim(double presentValue, double rate, int periods, double[] arr)

{

if (periods == 0)

return presentValue;

if (arr[periods] != -1)

return arr[periods];

arr[periods] =optim(presentValue \* (1 + rate), rate, periods - 1, arr);

return arr[periods];

}

static void Main(string[] args)

{

double presentValue = 1000;

double growthRate = 0.05;

int periods = 5;

Console.WriteLine("Recursive Calculation:");

double futureValue = normal(presentValue, growthRate, periods);

Console.WriteLine($"Future Value (Recursive): {futureValue:C}");

Console.WriteLine("\nOptimized Recursive Calculation with Memoization:");

double[] temp = new double[periods + 1];

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

{

temp[i] = -1;

}

double future = optim(presentValue, growthRate, periods, temp);

Console.WriteLine($"Future Value (Optimized): {future:C}");

}

}

}

