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

**CODE**

using System;

using System.Diagnostics;

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public Product(int id, string name, string category)

{

Id = id;

Name = name;

Category = category;

}

public override string ToString()

{

return $"ID: {Id}, Name: {Name}, Category: {Category}";

}

}

class Program

{

static (Product result, int comparisons) LinearSearch(Product[] products, string targetName)

{

int count = 0;

foreach (var product in products)

{

count++;

if (product.Name.Equals(targetName, StringComparison.OrdinalIgnoreCase))

{

return (product, count);

}

}

return (null, count);

}

static (Product result, int comparisons) BinarySearch(Product[] products, string targetName)

{

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

int count = 0;

while (left <= right)

{

count++;

int mid = (left + right) / 2;

int cmp = string.Compare(products[mid].Name, targetName, StringComparison.OrdinalIgnoreCase);

if (cmp == 0)

return (products[mid], count);

else if (cmp < 0)

left = mid + 1;

else

right = mid - 1;

}

return (null, count);

}

static (Product result, long timeTicks, int comparisons) MeasureExecutionTime(

Func<(Product result, int comparisons)> searchFunc)

{

Stopwatch sw = Stopwatch.StartNew();

var (result, comparisons) = searchFunc();

sw.Stop();

return (result, sw.ElapsedTicks, comparisons);

}

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 searchName = Console.ReadLine();

var (linearResult, linearTime, linearComparisons) = MeasureExecutionTime(

() => LinearSearch(products, searchName)

);

Product[] sortedProducts = (Product[])products.Clone();

Array.Sort(sortedProducts, (a, b) => a.Name.CompareTo(b.Name));

var (binaryResult, binaryTime, binaryComparisons) = MeasureExecutionTime(

() => BinarySearch(sortedProducts, searchName)

);

Console.WriteLine("\n--- Search Results ---");

Console.WriteLine(linearResult != null

? $"Linear Search: Found - {linearResult}"

: "Linear Search: Not Found");

Console.WriteLine($"Time: {linearTime} ticks, Comparisons: {linearComparisons}");

Console.WriteLine(binaryResult != null

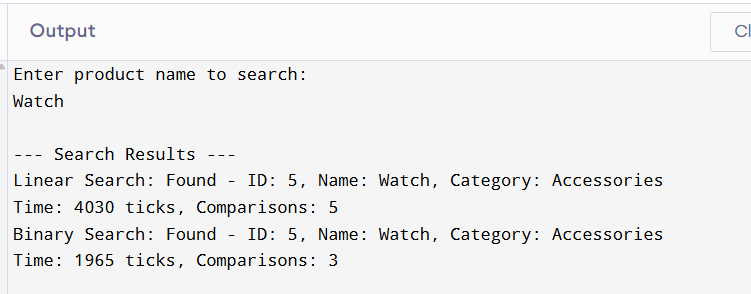
? $"Binary Search: Found - {binaryResult}"

: "Binary Search: Not Found");

Console.WriteLine($"Time: {binaryTime} ticks, Comparisons: {binaryComparisons}");

}

}



**Exercise 7: Financial Forecasting**

**CODE**

using System;

using System.Collections.Generic;

class Program

{

static double CalculateForecast(double principal, double annualRate, int period)

{

if (period == 0)

return principal;

return CalculateForecast(principal, annualRate, period - 1) \* (1 + annualRate);

}

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

static double CalculateForecastMemoized(double principal, double annualRate, int period)

{

if (period == 0)

return principal;

if (forecastCache.ContainsKey(period))

return forecastCache[period];

double value = CalculateForecastMemoized(principal, annualRate, period - 1) \* (1 + annualRate);

forecastCache[period] = value;

return value;

}

static void Main()

{

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

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

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

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

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

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

double futureRecursive = CalculateForecast(initialAmount, growthRate, numberOfYears);

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

double futureMemoized = CalculateForecastMemoized(initialAmount, growthRate, numberOfYears);

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

}

}

