Exercise 2: E-commerce Platform Search Function

Code:

using System;

using System.Linq;

namespace Ecomm

{

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 class ProductSearch

{

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

{

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

{

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

return products[i];

}

return null;

}

public static Product BinarySearch(Product[] products, string targetName)

{

int left = 0;

int right = products.Length - 1;

while (left <= right)

{

int mid = (left + right) / 2;

int comparison = string.Compare(products[mid].ProductName, targetName, true);

if (comparison == 0)

return products[mid];

else if (comparison < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

}

internal class Program

{

static void Main(string[] args)

{

Product[] products = new Product[]

{

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

new Product(2, "Backpack", "Accessories"),

new Product(3, "Washing Machine", "Home Appliances"),

new Product(4, "Sunglasses", "Fashion"),

new Product(5, "Office Chair", "Furniture")

};

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

string searchName = Console.ReadLine();

// Linear Search

Product linearResult = ProductSearch.LinearSearch(products, searchName);

Console.WriteLine("\nUsing Linear Search:");

Console.WriteLine(linearResult != null ? $"Found: {linearResult.ProductName} (Category: {linearResult.Category})" : "Product not found.");

// Binary Search (requires sorting)

Product[] sortedProducts = products.OrderBy(p => p.ProductName).ToArray();

Product binaryResult = ProductSearch.BinarySearch(sortedProducts, searchName);

Console.WriteLine("\nUsing Binary Search:");

Console.WriteLine(binaryResult != null ? $"Found: {binaryResult.ProductName} (Category: {binaryResult.Category})" : "Product not found.");

Console.WriteLine("\nPress any key to exit.");

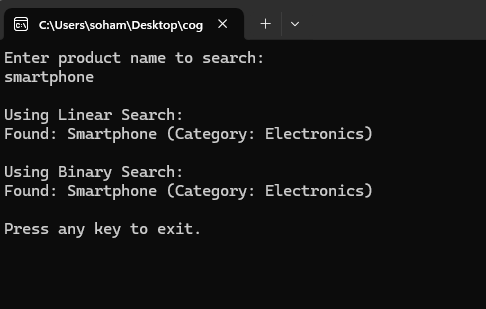
Console.ReadKey();

}

}

}

OUTPUT:



ANALYSIS:

Linear Search checks each element one by one (O(n)), while Binary Search (O(log n)) works only after sorting the array alphabetically by ProductName. Binary Search is more efficient for large, sorted datasets, but Linear Search is simpler and better for unsorted or small lists.

Exercise 7: Financial Forecasting

CODE:

using System;

namespace Financial\_Forecasting

{

internal class Program

{

static void Main(string[] args)

{

double initialValue = 10000;

double rate = 0.10;

int years = 5;

double futureValue = CalculateFutureValue(initialValue, rate, years);

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

double futureValueIterative = CalculateFutureValueIterative(initialValue, rate, years);

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

Console.ReadLine();

}

static double CalculateFutureValue(double initial, double rate, int years)

{

if (years == 0)

return initial;

return CalculateFutureValue(initial, rate, years - 1) \* (1 + rate);

}

static double CalculateFutureValueIterative(double initial, double rate, int years)

{

double result = initial;

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

{

result \*= (1 + rate);

}

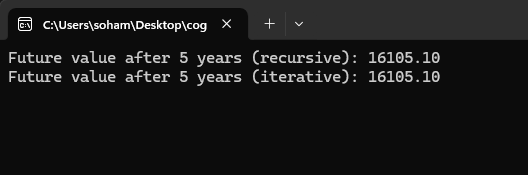
return result;

}

}

}

OUTPUT:



ANALYSIS:

The recursive algorithm has a time complexity of **O(n)**, where n is the number of years, due to one recursive call per year. This can lead to **stack overflow** for large values of n. To optimize, use an **iterative approach** which avoids recursion overhead. An even better solution is to use the **direct formula** with Math.Pow() for **O(1)** time