**DATA STRUCTURES AND ALGORITHMS**

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

**What is Big O Notation?**

The upper bound of an algorithm's time or space complexity as input size increases is expressed in Big O notation.

O(1) Constant time: quick, doesn't increase with input

Time increases with input size in O(n) linear time.

For big inputs, O(log n) logarithmic is incredibly efficient.

O(n2) Quadratic: ineffective with high inputs

**Best, Average, Worst Cases for Search**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |

**Product.cs**

namespace ECommerceSearch

{

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 $"{ProductId}: {ProductName} ({Category})";

}

}

}

**SearchEngine.cs**

**using System;**

**namespace ECommerceSearch**

**{**

**public static class SearchEngine**

**{**

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

**{**

**foreach (var product in products)**

**{**

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

**{**

**return product;**

**}**

**}**

**return null;**

**}**

**public static Product? BinarySearch(Product[] products, string name)**

**{**

**int low = 0;**

**int high = products.Length - 1;**

**while (low <= high)**

**{**

**int mid = (low + high) / 2;**

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

**if (comparison == 0)**

**return products[mid];**

**else if (comparison < 0)**

**low = mid + 1;**

**else**

**high = mid - 1;**

**}**

**return null;**

**}**

**}**

**}**

**Program.cs**

**using System;**

**using ECommerceSearch;**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Product[] products = new Product[]**

**{**

**new Product(101, "Phone", "Electronics"),**

**new Product(102, "Laptop", "Electronics"),**

**new Product(103, "Shoes", "Footwear"),**

**new Product(104, "Keyboard", "Accessories"),**

**new Product(105, "Watch", "Wearables")**

**};**

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

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

**string? name = Console.ReadLine();**

**if (string.IsNullOrEmpty(name))**

**{**

**Console.WriteLine("Invalid input.");**

**return;**

**}**

**Product? resultLinear = SearchEngine.LinearSearch(products, name);**

**Product? resultBinary = SearchEngine.BinarySearch(products, name);**

**Console.WriteLine("\n--- Linear Search Result ---");**

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

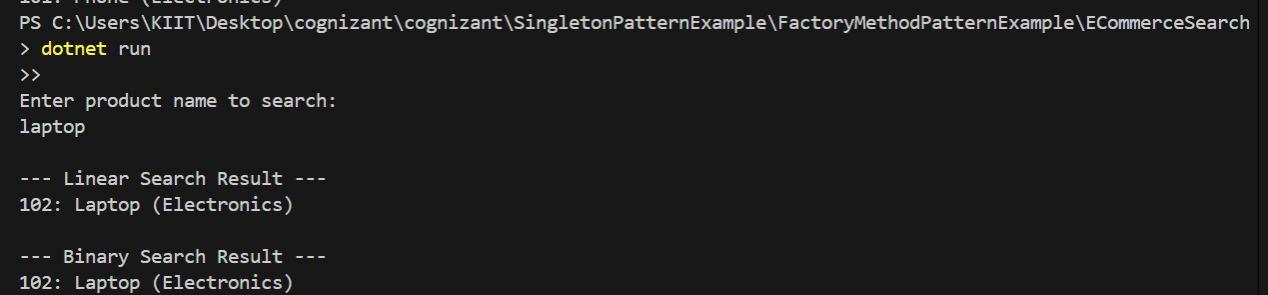
**Console.WriteLine("\n--- Binary Search Result ---");**

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

**}**

**}**

**OUTPUT:-**

****

**Exercise 7: Financial Forecasting**

**Understand Recursive Algorithms:**

Recursion is the technique via which a method solves smaller versions of the same problem by calling itself.

Example: Using the growth rate across several periods to determine the future worth of an investment.

**FutureValue(n) = FutureValue(n-1) \* (1 + growthRate)**

**Forecast.cs**

**namespace FinancialForecast**

**{ public static class Forecast**

**{**

**public static decimal PredictFutureValue(decimal currentValue, double growthRate, int years)**

**{**

**// Base case: no more years to forecast**

**if (years == 0)**

**return currentValue;**

**return PredictFutureValue(currentValue, growthRate, years - 1) \* (decimal)(1 + growthRate);**

**}**

**}**

**}**

**Program.cs**

**using System;**

**using FinancialForecast;**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Console.WriteLine("Enter current value:");**

**decimal currentValue = Convert.ToDecimal(Console.ReadLine());**

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

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

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

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

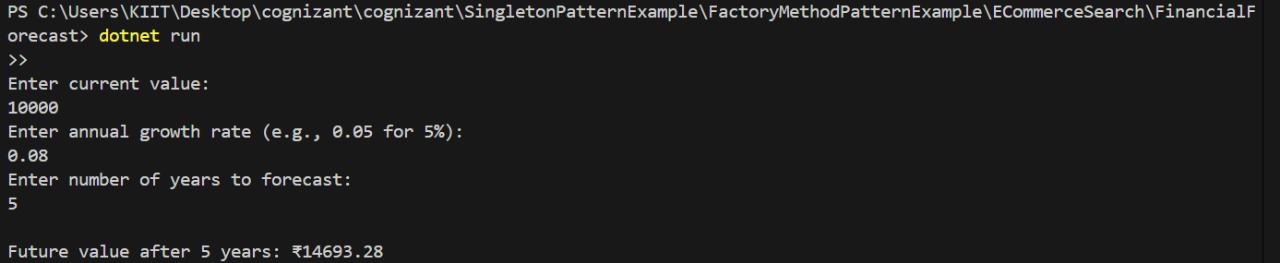
**decimal futureValue = Forecast.PredictFutureValue(currentValue, growthRate, years);**

**Console.WriteLine($"\nFuture value after {years} years: ₹{Math.Round(futureValue, 2)}");**

**}**

**}**

**OUTPUT:-**

****