# Exercise 2: E-commerce Platform Search Function (C#)

Algorithms and Data Structures

Scenario:  
You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

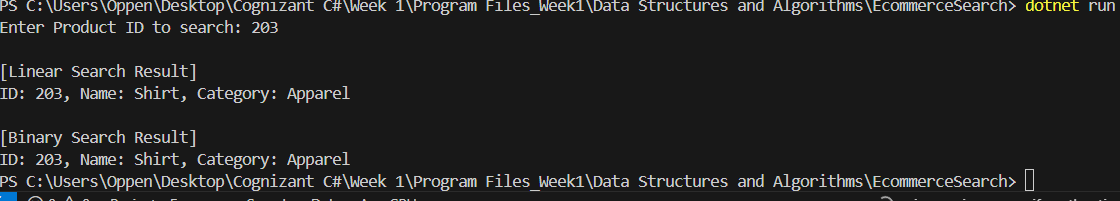
## Code: EcommerceSearch.cs

using System;  
  
public class Product  
{  
 public int ProductId;  
 public string ProductName;  
 public string Category;  
  
 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}";  
 }  
}  
  
public class Program  
{  
 public static Product LinearSearch(Product[] products, int id)  
 {  
 foreach (var product in products)  
 {  
 if (product.ProductId == id)  
 return product;  
 }  
 return null;  
 }  
  
 public static Product BinarySearch(Product[] products, int id)  
 {  
 int low = 0, high = products.Length - 1;  
 while (low <= high)  
 {  
 int mid = (low + high) / 2;  
 if (products[mid].ProductId == id) return products[mid];  
 if (products[mid].ProductId < id) low = mid + 1;  
 else high = mid - 1;  
 }  
 return null;  
 }  
  
 public static void Main(string[] args)  
 {  
 Product[] products = {  
 new Product(305, "Coffee Maker", "Home"),  
 new Product(203, "Shirt", "Apparel"),  
 new Product(101, "Laptop", "Electronics"),  
 new Product(401, "Book", "Stationery")  
 };  
  
 Array.Sort(products, (a, b) => a.ProductId.CompareTo(b.ProductId));  
  
 Console.Write("Enter Product ID to search: ");  
 int id = int.Parse(Console.ReadLine());  
  
 var result1 = LinearSearch(products, id);  
 Console.WriteLine("[Linear Search Result] " + (result1 != null ? result1.ToString() : "Product not found."));  
  
 var result2 = BinarySearch(products, id);  
 Console.WriteLine("[Binary Search Result] " + (result2 != null ? result2.ToString() : "Product not found."));  
 }  
}

## Analysis:

- Linear Search Time Complexity: O(n)  
 It checks each element one by one and is simple but inefficient for large datasets.  
- Binary Search Time Complexity: O(log n)  
 It divides the search space in half each time and is much faster for sorted arrays.  
- Suitability:  
 Binary Search is more suitable for performance-optimized platforms, but it requires data to be sorted. For unsorted or small datasets, Linear Search might be acceptable due to its simplicity.

## Output Screenshot



# Exercise 7: Financial Forecasting (C#)

Scenario:  
You are developing a financial forecasting tool that predicts future values based on past data.

## Code: FinancialForecast.cs

using System;  
  
public class FinancialForecast  
{  
 public static double ForecastRecursive(double value, double rate, int years)  
 {  
 if (years == 0) return value;  
 return ForecastRecursive(value \* (1 + rate), rate, years - 1);  
 }  
  
 public static double ForecastIterative(double value, double rate, int years)  
 {  
 for (int i = 0; i < years; i++)  
 {  
 value \*= (1 + rate);  
 }  
 return value;  
 }  
  
 public static void Main(string[] args)  
 {  
 Console.Write("Enter initial amount: ");  
 double initial = double.Parse(Console.ReadLine());  
  
 Console.Write("Enter growth rate (e.g., 0.10 for 10%): ");  
 double rate = double.Parse(Console.ReadLine());  
  
 Console.Write("Enter number of years: ");  
 int years = int.Parse(Console.ReadLine());  
  
 double rec = ForecastRecursive(initial, rate, years);  
 double iter = ForecastIterative(initial, rate, years);  
  
 Console.WriteLine($"Recursive Forecast: ₹{rec:F2}");  
 Console.WriteLine($"Iterative Forecast: ₹{iter:F2}");  
 }  
}

## Analysis:

- Recursive Time Complexity: O(n)  
 One recursive call per year, which grows linearly with the number of years.  
- Optimization:  
 Convert recursion to an iterative loop to avoid stack overflow and reduce overhead. This is implemented in the Forecast Iterative method, which is more memory efficient and performs better in large datasets.

## Output Screenshot

