E-commerce Platform Search Function

1. **Understanding Notation**

# Big O Notation

# Big O notation describes how the time or space complexity of an algorithm grows as the input size increases. It helps us compare algorithms based on performance and scalability — regardless of hardware or programming language.

# Search Case Scenarios

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

* + **Best Case**: When the item is found in the first attempt.
  + **Average Case**: When the item is found in the middle.
  + **Worst Case**: When the item is at the end or not found at all.

**1.Main Class : Program.cs**

using System;

using System.Linq;

using EcommerceSearchFunction.Models;

namespace EcommerceSearchFunction

{

    class Program

    {

        // Linear Search

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

        {

            foreach (var product in products)

            {

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

                    return product;

            }

            return null;

        }

        // Binary Search

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

        {

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

            while (left <= right)

            {

                int mid = (left + right) / 2;

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

                if (comparison == 0)

                    return products[mid];

                else if (comparison < 0)

                    right = mid - 1;

                else

                    left = mid + 1;

            }

            return null;

        }

        static void Main(string[] args)

        {

            // sample product data

            Product[] products = new Product[]

            {

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

                new Product(2, "Shoes", "Footwear"),

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

                new Product(4, "Phone", "Electronics"),

                new Product(5, "Bag", "Accessories"),

            };

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

            string input = Console.ReadLine() ?? "";

            // Linear Search

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

            var result1 = LinearSearch(products, input);

            if (result1 != null)

                Console.WriteLine($"Found: {result1.ProductName} (Category: {result1.Category})");

            else

                Console.WriteLine("Product not found.");

            // Binary Search

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

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

            var result2 = BinarySearch(sortedProducts, input);

            if (result2 != null)

                Console.WriteLine($"Found: {result2.ProductName} (Category: {result2.Category})");

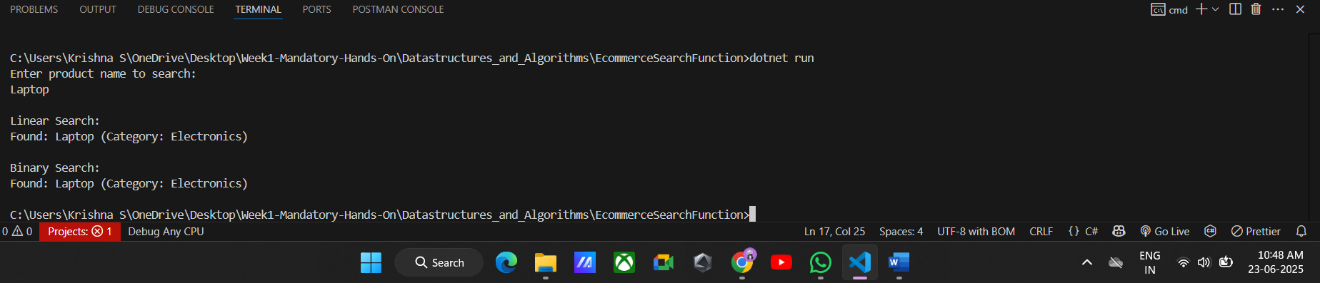
            else

                Console.WriteLine("Product not found.");

        }

    }

**2.Output**



**3.Time Complexity Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | **Space Complexity** | **Sorting Required** |
| Linear Search | O(n) | O(1) | No |
| Binary Search | O(log n) | O(1) | Yes |

**4.Analysis**

**Time Complexity Comparison:**

* **Linear Search**: O(n) – slower on large datasets; checks each item one by one.
* **Binary Search**: O(log n) – much faster, but requires data to be **sorted**.

**Which is More Suitable?**

* For an **e-commerce platform**, **binary search is better** for speed and efficiency if product data is sorted or indexed.