E-commerce Platform Search Function

1. Understanding Notation

Big O Notation

Big O notation describes the upper bound of an algorithm's runtime or space requirement with respect to the input size `n`. It helps developers predict scalability and efficiency.

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 somewhere in the middle.
- **Worst Case**: When the item is at the end or not found at all.

2. Setup: Product Class

```
namespace ECommercePlatformSearchFunction.Models
{
    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;
        }
    }
}
```

3. Implementation

```
using ECommercePlatformSearchFunction.Models;


namespace ECommercePlatformSearchFunction.Services
{
```

```
    public static class SearchService
    {
        public static Product? LinearSearch(Product[] products, string
productName)
        {
            foreach (var product in products)
            {
                if (product.ProductName.Equals(productName,
StringComparison.OrdinalIgnoreCase))
                    return product;
            }
            return null;
        }

        public static Product? BinarySearch(Product[] products, string
productName)
        {

             Array.Sort(products, (p1, p2) =>
                string.Compare(p1.ProductName, p2.ProductName,
StringComparison.OrdinalIgnoreCase));

            int low = 0, high = products.Length - 1;
            while (low <= high)
            {
                int mid = (low + high) / 2;
                int comparison = string.Compare(products[mid].ProductName,
productName, StringComparison.OrdinalIgnoreCase);

                if (comparison == 0) return products[mid];
                else if (comparison < 0) low = mid + 1;
                else high = mid - 1;
            }
            return null;
        }
    }
}
```

4. Program Execution (Main Class)

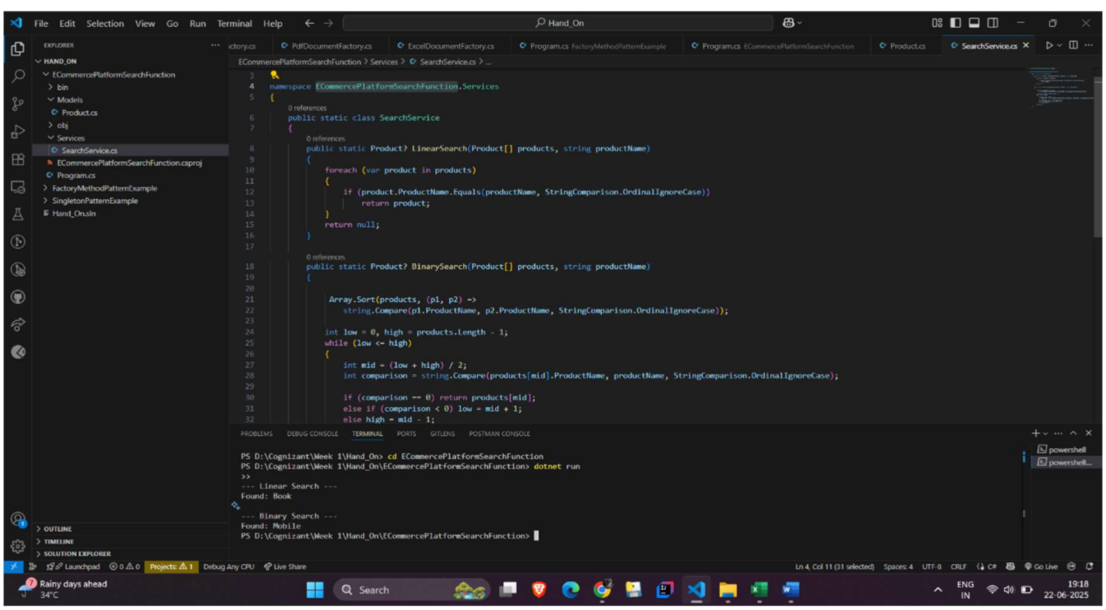
```
using ECommercePlatformSearchFunction.Models;
using ECommercePlatformSearchFunction.Services;

class Program
{
    static void Main()
    {
```

```
        Product[] products = new Product[]
        {
            new Product(1, "Laptop", "Electronics"),
            new Product(2, "Shoes", "Fashion"),
            new Product(3, "Book", "Education"),
            new Product(4, "Mobile", "Electronics")
        };

        Console.WriteLine("--- Linear Search ---");
        var result1 = SearchService.LinearSearch(products, "Book");
        Console.WriteLine(result1 != null ? $"Found: {result1.ProductName}" :
"Not Found");

        Console.WriteLine("\n--- Binary Search ---");
        var result2 = SearchService.BinarySearch(products, "Mobile");
        Console.WriteLine(result2 != null ? $"Found: {result2.ProductName}" :
"Not Found");
    }
}
```

5. Time Complexity Analysis

Algorithm	Time Complexity	Space Complexity	Sorted Required
Linear Search	O(n)	O(1)	No
Binary Search	O(log n)	O(1)	Yes

6.Output



7. Conclusion

- **Binary Search** is optimal for large, sorted datasets with frequent search operations due to its `O(log n)` complexity.
- **Linear Search** is simple and works without sorting but becomes inefficient for large datasets.
- **Recommendation**: Use Binary Search after sorting the product list once, especially if the product data is mostly read-only and searched frequently.