***Week 1\_3: MANDATORY HANDSON EXERCISE (ALGORITHMS\_DATA\_STRUCTURE)***

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

**Scenario:**

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

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

***CODE:***

using System;

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;

    }

}

class Program

{

     public static int LinearSearch(Product[] products, int targetId)

    {

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

        {

            if (products[i].ProductId == targetId)

                return i;

        }

        return -1;

    }

    public static int BinarySearch(Product[] sortedProducts, int targetId)

{

    int left = 0;

    int right = sortedProducts.Length - 1;

    while (left <= right)

    {

        int mid = left + (right - left) / 2;

        int midId = sortedProducts[mid].ProductId;

        if (midId == targetId) return mid;

        if (midId < targetId) left = mid + 1;

        else right = mid - 1;

    }

    return -1;

}

    static void Main(string[] args)

    {

        Console.WriteLine("Hello, e-commerce search!");

        Product[] products = {

            new Product(101, "Keyboard", "Electronics"),

            new Product(205, "Desk Lamp", "Furniture"),

            new Product(307, "Mouse", "Electronics")

        };

        Array.Sort(products, (a, b) => a.ProductId.CompareTo(b.ProductId));

        int targetId = 205;

        int linearResult = LinearSearch(products, targetId);

        int binaryResult = BinarySearch(products, targetId);

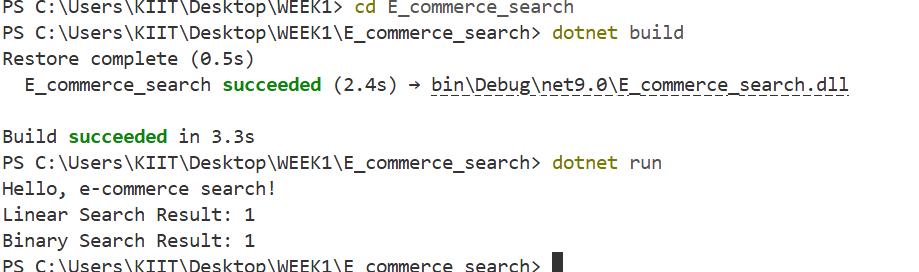
        Console.WriteLine($"Linear Search Result: {linearResult}");

        Console.WriteLine($"Binary Search Result: {binaryResult}");

    }

}

***OUTPUT:***



1. Asymptotic Notation and Search Scenarios

Big O notation measures algorithm scalability by describing how runtime grows relative to input size

n. It quantifies worst-case efficiency, ignoring constants and lower-order terms. For search operations:

Best-case: O(1)(target found immediately)

Average-case: Expected performance over random inputs.

Worst-case: O(n) for linear search (target missing/last element), O(logn) for binary search.

Complexity Analysis and Platform Recommendation

Time Complexity Comparison

Algorithm Best-case Worst-case

Linear Search O(1) O(n)

Binary Search O(1) O(log n)

Platform Suitability

Binary search is optimal for e-commerce platforms because:

Logarithmic scaling: For 1 million products, binary search needs ~20 comparisons vs. 500,000 average for linear search

Real-world efficiency: Product databases are typically indexed/sorted during ingestion

Large dataset performance: Reduces latency for high-traffic search operations.

Optimization Recommendations

Pre-sort data: Sort products during database ingestion

Hybrid approach: Use linear search for small inventories (<100 items), binary search for larger datasets

Indexing: Combine with hash tables for O(1) lookups when product IDs are knownBinary search provides superior scalability for

e-commerce platforms due to its logarithmic time complexity. The initial sorting overhead is negligible compared to ongoing

search efficiency gains, especially for catalogs exceeding 10,000 products.