**COGNIZANT Deep\_Skilling**

**Data Structures and Algorithms**

Question 2: E-commerce Platform Search Function

Code:

using System;

using System.Linq;

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 class ECommerceSearch

{

    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 left = 0;

        int right = products.Length - 1;

        while (left <= right)

        {

            int mid = (left + right) / 2;

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

            if (compare == 0)

                return products[mid];

            else if (compare < 0)

                left = mid + 1;

            else

                right = mid - 1;

        }

        return null;

    }

}

class Program

{

    static void Main()

    {

        Product[] products = new Product[]

        {

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

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

            new Product(3, "Smartphone", "Electronics"),

            new Product(4, "Book", "Education")

        };

        var foundLinear = ECommerceSearch.LinearSearch(products, "Shoes");

        Console.WriteLine(foundLinear != null ? $"Linear: Found {foundLinear.ProductName}" : "Linear: Not Found");

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

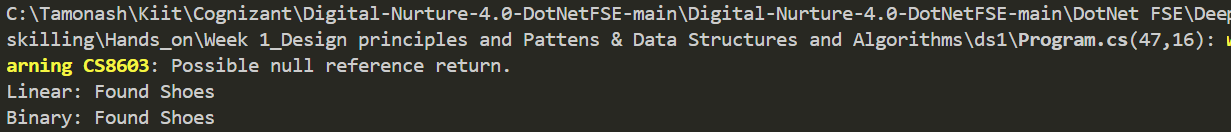
        var foundBinary = ECommerceSearch.BinarySearch(sorted, "Shoes");

        Console.WriteLine(foundBinary != null ? $"Binary: Found {foundBinary.ProductName}" : "Binary: Not Found");

    }

}

**Output:**



**Understanding the problem questions:**

1. Explain why data structures and algorithms are essential in handling large inventories.

Answer :

Data structures and algorithms are essential in handling large inventories because they directly impact the speed and efficiency of core operations like searching, updating, and deleting inventory items. In a warehouse environment, time-sensitive decisions depend on quickly retrieving accurate information about products such as quantity or location. Without the use of efficient algorithms and data structures, these operations would become increasingly slow as the inventory grows. By applying optimal data structures (such as hash tables or trees) and algorithms, the system can ensure that performance remains fast and reliable even as the volume of data increases dramatically.

1. Discuss the types of data structures suitable for this problem.

Answer:

The most suitable data structures for an inventory management system are those that provide fast lookup, insertion, and deletion — primarily hash-based structures like HashMap or Dictionary. These allow quick access to product records using a unique key like productId. In scenarios where you need to maintain order or perform range-based queries (e.g., finding all products within a price range), tree-based structures like binary search trees or AVL trees could also be useful. Additionally, using arrays or lists may work for small datasets or simple applications but are generally less efficient for search-heavy workloads due to their linear time complexity.

**Analysis Questions**

1. What is the time complexity of each operation?

Answer:

When using a Dictionary<int, Product> (or a HashMap), the time complexity for adding, updating, and deleting items is O(1) on average. This means these operations are performed in constant time regardless of the inventory size, which is ideal for systems with frequent updates and lookups. However, searching based on other fields like product name or category requires iterating through all items, resulting in O(n) time complexity. Therefore, while key-based operations are fast, attribute-based searches without additional indexing are slower and become a bottleneck as inventory size increases.

4)How can these operations be optimized?

Answer:

To optimize performance, especially for non-key lookups, the system can introduce secondary indexes — for example, an additional Dictionary indexed by productName or category. This enables near-instant lookups for those fields as well. For frequently accessed items, implementing caching (such as an LRU cache) can significantly reduce the load on the main data structure. Batch processing of updates can also minimize the overhead caused by repeated operations. Additionally, structuring the data with normalization and avoiding redundancy can enhance both memory usage and integrity, ensuring the system remains responsive and scalable under heavy use.