Week-1 Hands-On Exercise (Algorithms- Data Structures)

Exercise 2: E-commerce Platform Search Function

1) **Big O Notation** is a mathematical notation used to describe the time complexity or space complexity of an algorithm in terms of input size n. It provides an upper bound on the **growth rate of an algorithm's runtime or memory usage,** helping to understand how the algorithm will perform as the size of input data increases. Big O focuses only on the dominant term, ignoring constants and less significant factors, to represent the worst-case scenario of an algorithm's performance.

   - Big O describes how fast an algorithm grows relative to input size.
   - It helps compare algorithm efficiency and scalability.
   - It is commonly used in algorithm analysis and optimization.

   When analysing the performance of search algorithms, we consider three primary scenarios

1. Best Case

Definition: The most optimal situation where the desired element is found immediately or with minimal effort.

- **Linear Search**: The target element is found at the **first position.** Time complexity O(1).
- **Binary Search**: The target element is the **middle element** of the sorted array. Time complexity O(1).

2. Average Case

Definition: The expected performance of the algorithm over a typical or random set of inputs.

- **Linear Search**: The target element is found **somewhere in the middle** of the list. Time complexity O(n).
- **Binary Search**: The search continues by dividing the array until the target is found. Time complexity O(n).

3. Worst Case

Definition : The least favourable scenario where the algorithm performs the maximum number of steps.

- **Linear Search**: The element is at the **last position** or **not present** at all. Time complexity O(n).
- **Binary Search**: The algorithm divides the array until all possibilities are exhausted (target not found or found at the last step). Time complexity O(log n).

2) **Setup**

C# Program for creating class product

```
using System;
using System.Linq;
```

```
namespace ECommerceSearch
{
  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 override string ToString()
    {
      return $"[{ProductId}, {ProductName}, {Category}]";
    }
  }
```

3) Implementation

Implementing Linear search and Binary Search

```
using System;
using System.Linq;

namespace ECommerceSearch
{
public class Search
  {
    // Linear Search
    public static Product LinearSearch(Product[] products, string targetName)
    {
      foreach (var product in products)
      {
        if (product.ProductName.Equals(targetName, StringComparison.OrdinalIgnoreCase))
        {
          return product;
        }
      }
      return null;
    }

    // Binary Search (requires sorted array)
    public static Product BinarySearch(Product[] products, string targetName)
    {
      int left = 0, right = products.Length - 1;

      while (left <= right)
      {
```

```
            int mid = (left + right) / 2;
            int cmp = string.Compare(products[mid].ProductName, targetName, StringComparison.OrdinalIgnoreCase);

            if (cmp == 0)
                return products[mid];
            else if (cmp < 0)
                left = mid + 1;
            else
                right = mid - 1;
        }

        return null;
    }
}
```

4) Analysis

Result

```
Linear Search Result: [102, Chair, Furniture]
Time taken by Linear Search: 0.02 ms
Binary Search Result: [102, Chair, Furniture]
Time taken by Binary Search: 0.01 ms
```

Aspect	Linear Search	Binary Search
Best Case	O(1)	O(1)
Average Case	O(n)	O(log n)
Worst Case	O(n)	O(log n)
Data Requirement	Works on unsorted data	Requires sorted data
Scalability	Not efficient for large n	Very efficient for large n
Ease of Use	Very simple to implement	Slightly more complex

Which algorithm is suitable for our E-commerce platform

Binary Search is More Suitable for an e-commerce platform

Reason for binary search is suitable

Large Dataset Handling:

- E-commerce platforms usually have thousands to millions of products.
- Binary search performs well on large datasets due to its O(log n) time complexity.

Fast Search Performance:

- Users expect instant results when searching for products.
- Binary search reduces the number of comparisons significantly, improving response time.

Sorted or Indexed Data:

- Product catalogs can be easily sorted by product name or ID.
- Platforms often use database indexing, which supports binary-search-like performance.

Scalability:

- As the number of products grows, binary search continues to perform efficiently.
- Linear search becomes too slow with large product counts (O(n)).