**Data structures and Algorithms**

Exercise 2: E-commerce Platform Search Function

1. **Understand Asymptotic Notation:**
2. **Explain Big O notation and how it helps in analyzing algorithms.**

Ans: **Big O notation** is a mathematical representation used to describe the upper bound of an algorithm's running time or space requirements in terms of input size.

* It focuses on the **worst-case scenario**, helping developers understand how the performance of an algorithm degrades as the input grows.
* Big O ignores constant factors and lower-order terms, allowing us to compare the scalability of algorithms at a high level.
* By analyzing algorithms using Big O notation, developers can make informed decisions about which algorithm is best suited for performance-critical applications, especially when working with large datasets.

**b) Describe the best, average, and worst-case scenarios for search operations.**

Ans: **Best Case**: The item is found at the first position·

**Average Case**: The item is found somewhere in the middle ·

**Worst Case**: The item is at the last position or not present

1. **Setup:**

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;

}

}

1. **Implementation:**

using System;

class SearchExample

{

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, right = products.Length - 1;

while (left <= right)

{

int mid = (left + right) / 2;

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

if (compare == 0)

return products[mid];

else if (compare < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

static void Main()

{

Product[] products = {

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

new Product(2, "Shirt", "Clothing"),

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

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

new Product(5, "Tablet", "Electronics")

};

Console.WriteLine("=== Linear Search ===");

var result1 = LinearSearch(products, "Phone");

Console.WriteLine(result1 != null ? $"Found: {result1.ProductName}" : "Product not found");

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

Console.WriteLine("=== Binary Search ===");

var result2 = BinarySearch(products, "Phone");

Console.WriteLine(result2 != null ? $"Found: {result2.ProductName}" : "Product not found");

}

}

1. **Analysis:**

**a) Compare the time complexity of linear and binary search algorithms.**

Ans :

* Linear search has a time complexity of **O(n)** because it checks each element one by one until the target is found or the list ends.
* Binary search, on the other hand, has a time complexity of **O(log n)**, as it repeatedly divides the sorted array in half to locate the target. This makes binary search significantly faster than linear search for large, sorted datasets.

1. **Discuss which algorithm is more suitable for your platform and why.**

Ans: Binary search is more suitable for an e-commerce platform because it offers much faster search performance with **O(log n)** time complexity, which is ideal for handling large product datasets. Although it requires the data to be **sorted**, this can be managed during data entry or updates. Its efficiency in delivering quick search results greatly enhances user experience, especially as the platform scales.

* OUTPUT

