CODE:-

//using System;

public class Product

{

public int ProductId { get; set; }

public string ProductName { get; set; }

public string Category { get; set; }

public Product(int productId, string productName, string category)

{

ProductId = productId;

ProductName = productName;

Category = category;

}

public void Display()

{

Console.WriteLine($"ID: {ProductId}, Name: {ProductName}, Category: {Category}");

}

}

class Program

{

// Linear Search

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

{

foreach (Product p in products)

{

if (p.ProductId == targetId)

return p;

}

return null;

}

// Binary Search (array must be sorted by ProductId)

static Product? BinarySearch(Product[] products, int targetId)

{

int low = 0, high = products.Length - 1;

while (low <= high)

{

int mid = (low + high) / 2;

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

return products[mid];

else if (products[mid].ProductId < targetId)

low = mid + 1;

else

high = mid - 1;

}

return null;

}

static void Main(string[] args)

{

Product[] products = new Product[]

{

new Product(103, "Laptop", "Electronics"),

new Product(101, "Shirt", "Clothing"),

new Product(105, "Book", "Stationery"),

new Product(102, "Headphones", "Electronics"),

new Product(104, "Shoes", "Footwear")

};

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

int searchId = 105;

Product? result1 = LinearSearch(products, searchId);

if (result1 != null)

{

Console.Write("Product Found: ");

result1.Display();

}

else

{

Console.WriteLine("Product not found.");

}

Console.WriteLine("\n=== Binary Search (sorted by ID) ===");

Array.Sort(products, (a, b) => a.ProductId.CompareTo(b.ProductId)); // sort for binary search

Product? result2 = BinarySearch(products, searchId);

if (result2 != null)

{

Console.Write("Product Found: ");

result2.Display();

}

else

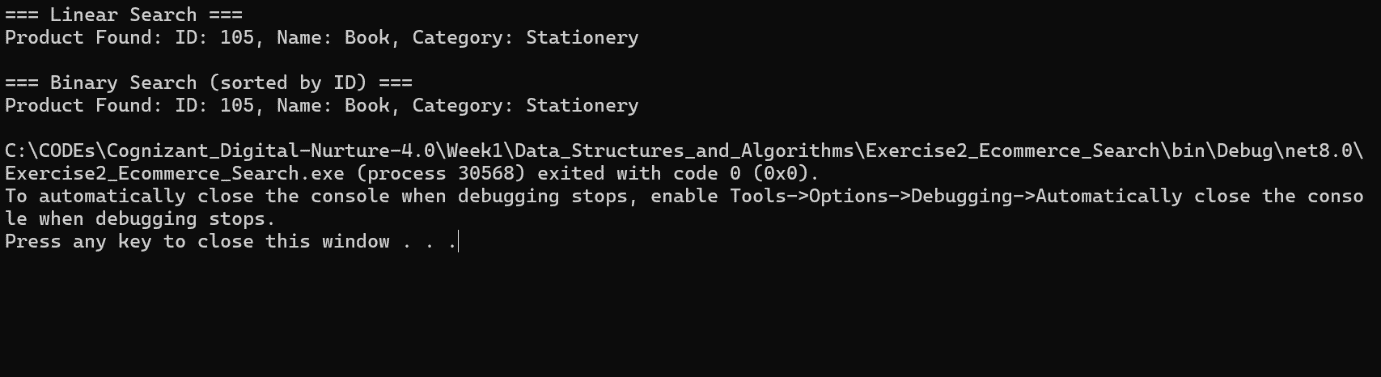
{

Console.WriteLine("Product not found.");

}

}

}



1. Understand Asymptotic Notation:

o Explain Big O notation and how it helps in analyzing algorithms.

o Describe the best, average, and worst-case scenarios for search operations.

Answer:-

1. Understand Asymptotic Notation

a) Big O Notation and Its Importance

Big O notation is used to describe the upper bound of an algorithm’s running time or space requirement

as a function of the input size n. It provides a high-level understanding of how an algorithm performs

as the input grows larger and helps compare the efficiency of different algorithms.

It ignores constant factors and lower-order terms, focusing only on the

dominant term that impacts scalability.

Helps developers choose the most efficient algorithm for large data sets.

Common Big O examples:

O(1): Constant time (e.g., accessing an array element)

O(log n): Logarithmic time (e.g., binary search)

O(n): Linear time (e.g., linear search)

O(n log n): Linearithmic (e.g., merge sort)

O(n²): Quadratic time (e.g., bubble sort)

b) Best, Average, and Worst-Case Scenarios for Search Operations

In algorithm analysis, it's important to consider different input conditions:

Scenario Explanation

Best Case: The search finds the target in the first few comparisons.

Average Case: The target is located somewhere in the middle, or equally likely anywhere.

Worst Case: The target is not found or found at the last comparison.

///////////////////////////////////////////////////////////////////////////////////////////////////////

4. Analysis:

o Compare the time complexity of linear and binary search algorithms.

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

Answer:-

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

Example: Linear Search on an array of n elements

Best case: O(1) → Target is at index 0.

Average case: O(n) → On average, checked n/2 elements.

Worst case: O(n) → Target is at the end or not present at all.

Example: Binary Search (on sorted array)

Best case: O(1) → Target is the middle element.

Average case: O(log n)

Worst case: O(log n) → Keep dividing search space until one element remains.

b) Suitable Algorithm for the Platform

Which algorithm is more suitable depends on the context:

If the data is unsorted and you're doing a single or few searches, Linear Search is

suitable because it requires no preprocessing.

If the data is sorted or can be pre-sorted and you perform frequent searches, Binary Search

is better due to its logarithmic time complexity.

For an e-commerce search platform, binary search (or even more advanced techniques like hash

indexing or trie trees) would be more appropriate, provided the product listings are sorted or

indexed, since performance and response time are critical.

Conclusion:

Binary Search is more efficient for sorted data, especially in systems where performance and

scalability are essential.

Use Linear Search for small or unsorted datasets or where sorting is not feasible due to time

or resource constraints.