**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.

**Answers:**

1. Understand Asymptotic Notation:

1.1 Big O notation:

* **Big O** describes how the runtime or space requirements of an algorithm grow with input size n.
* It helps us understand **scalability** and choose efficient algorithms.

|  |  |
| --- | --- |
| **Case** | **Description** |
| **Best** | The minimum time an algorithm takes (ideal input). |
| **Average** | Expected time over all inputs. |
| **Worst** | The maximum time it will take (most unoptimized input). |

1.2 the best, average, and worst-case scenarios for search operations

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| **Linear** | O(1) | O(n) | O(n) |
| **Binary** | O(1) | O(log n) | O(log n) |

2**.Setup – Product Class:**

//file name Product.java

public class Product {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

}

3. **Implementation:**

//SearchOperations.java

Import java.util.Arrays;

public class SearchOperations {

// Linear Search

public static Product linearSearch(Product[] products, int targetId) {

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

// Binary Search (requires sorted input)

public 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;

}

// Print product info manually

public static void printProduct(Product p) {

if (p == null) {

System.out.println("Product not found.");

} else {

System.out.println("Product ID: " + p.productId + ", Name: " + p.productName + ", Category: " + p.category);

}

}

}

// file name CheckProductMain.java

public class CheckProductMain {

public static void main(String[] args) {

Product[] products = {

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

new Product(101, "T-shirt", "Clothing"),

new Product(104, "Headphones", "Electronics"),

new Product(103, "Shoes", "Footwear")

};

// Sort products by productId (for binary search)

Arrays.sort(products, java.util.Comparator.comparingInt(p -> p.productId));

// Linear Search

Product p1 = SearchOperations.linearSearch(products, 103);

System.out.println("Linear Search Result: " + format(p1));

// Binary Search

Product p2 = SearchOperations.binarySearch(products, 103);

System.out.println("Binary Search Result: " + format(p2));

}

// Simple formatter method

static String format(Product p) {

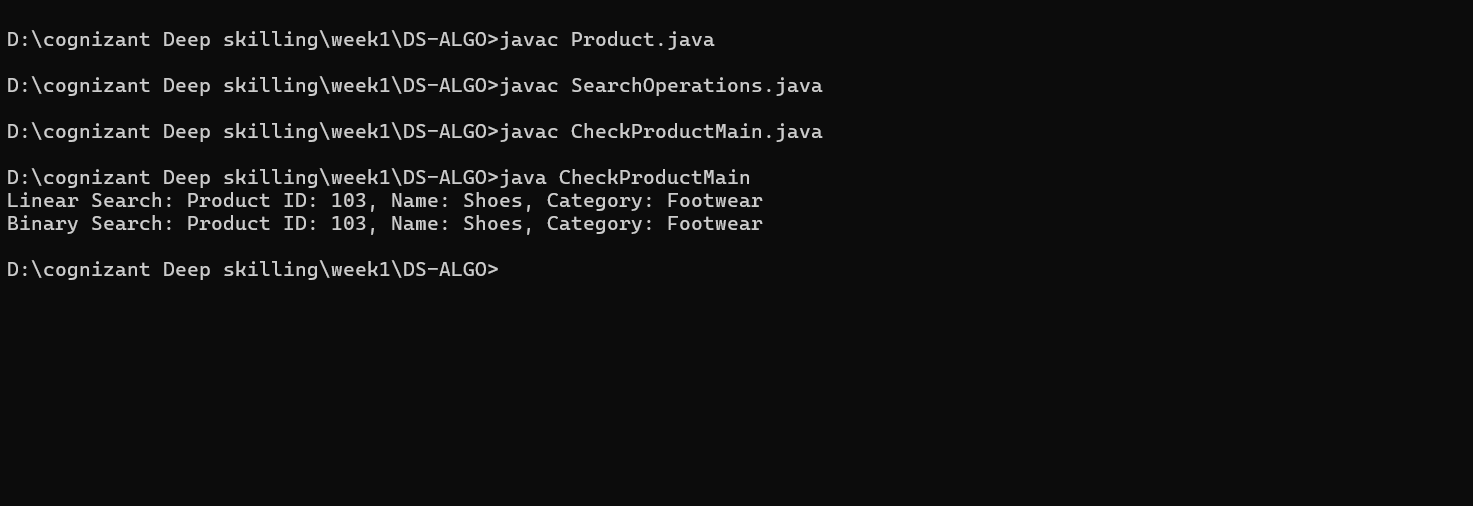
if (p == null) return "Product not found.";

return p.productId + " - " + p.productName + " (" + p.category + ")";

}

}

**OUTPUT:**

****

4. **Analysis:**

**Linear Search:**

**Time Complexity:**

* **Best case**: O(1) — match at the first element.
* **Average case**: O(n/2) → O(n) — match somewhere in the middle.
* **Worst case**: O(n) — match at the end or not found.

**Space Complexity**: O(1)

**When to use**: Best when you have a **small number of products** or when the list is not sorted.

**Binary Search:**

**Time Complexity:**

* **Best case:** O(1) — found at the middle.
* **Average/Worst case:** O(log n)

**Space Complexity:** O(1)

**When to use**: Best when you have a large, sorted product list, and you need quick searches**.**

**Which One is Better for E-commerce?**

* In real-world e-commerce platforms, product catalogs are huge — often containing thousands or millions of items.
* Users expect instant search results, so using binary search.
* Linear search is too slow for large systems, though it’s fine for learning or small apps.