**Exercise: E-commerce Platform Search Function**

**1. Understanding Asymptotic Notation**

**Big O Notation:**  
Big O notation is a way to express the time or space complexity of an algorithm in terms of input size. It helps developers analyse and compare the performance of algorithms, especially as data size grows. For search operations, it allows us to evaluate how fast an algorithm can find a target element under different circumstances.

In the case of Linear Search, the best-case scenario occurs when the element is found at the very beginning of the array, which results in a time complexity of O(1). On average, linear search may need to traverse half the array, leading to O(n) time complexity. The worst-case scenario occurs when the element is not found or is at the end of the array, also resulting in O(n). On the other hand, Binary Search works on sorted arrays and has a best-case time complexity of O(1) if the target is in the middle. Both its average and worst-case complexities are O(log n), since the search space is halved with each step. Therefore, binary search is significantly more efficient for large, sorted data sets.

**2. Setup**

Created a Java class named Product with three attributes: productId, productName, and category. These attributes are used to store and search product-related data.

**3. Implementation**

The following code demonstrates both linear and binary search implementations on an array of Product objects:

import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearch {

static 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;

}

public void display() {

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

}

}

public static Product linearSearch(Product[] products, String targetName) {

for (Product product : products) {

if (product.productName.equalsIgnoreCase(targetName)) {

return product;

}

}

return null;

}

public static Product binarySearch(Product[] products, String targetName) {

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

while (low <= high) {

int mid = (low + high) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(targetName);

if (cmp == 0) {

return products[mid];

} else if (cmp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Shirt", "Clothing"),

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

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

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

};

System.out.println("Linear Search for 'Phone':");

Product result1 = linearSearch(products, "Phone");

if (result1 != null) result1.display();

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

Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

System.out.println("\nBinary Search for 'Phone':");

Product result2 = binarySearch(products, "Phone");

if (result2 != null) result2.display();

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

}

}

**4. Analysis**

When comparing the performance of linear and binary search, linear search has a time complexity of O(n), as it may need to check each element in the array. It does not require the data to be sorted, which makes it more flexible but slower, especially with large datasets. Binary search, however, has a time complexity of O(log n) since it divides the search space in half each time. This makes binary search far more efficient for large arrays, but it requires the array to be sorted beforehand. For an e-commerce platform with potentially thousands or millions of products, binary search is the better choice for performance, assuming the product list can be maintained in sorted order.