**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.

import java.util.Arrays;

class Product {

private String productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

public String toString() {

return "Product{" +

"productId='" + productId + '\'' +

", productName='" + productName + '\'' +

", category='" + category + '\'' +

'}';

}

}

public class ECommerceSearch {

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

for (Product product : products) {

if (product.getProductId().equals(productId)) {

return product;

}

}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (products[mid].getProductId().equals(productId)) {

return products[mid];

} else if (products[mid].getProductId().compareTo(productId) < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product("P001", "Laptop", "Electronics"),

new Product("P002", "Smartphone", "Electronics"),

new Product("P003", "Tablet", "Electronics"),

new Product("P004", "Headphones", "Accessories"),

new Product("P005", "Smartwatch", "Wearables")

};

String searchId1 = "P003";

Product result1 = *linearSearch*(products, searchId1);

System.***out***.println("Linear Search Result for " + searchId1 + ": " + (result1 != null ? result1 : "Product not found"));

Arrays.*sort*(products, (p1, p2) -> p1.getProductId().compareTo(p2.getProductId()));

String searchId2 = "P002";

Product result2 = *binarySearch*(products, searchId2);

System.***out***.println("Binary Search Result for " + searchId2 + ": " + (result2 != null ? result2 : "Product not found"));

}

}

OUTPUT:

Linear Search Result for P003: Product{productId='P003', productName='Tablet', category='Electronics'}

Binary Search Result for P002: Product{productId='P002', productName='Smartphone', category='Electronics'}

A Java program that implements the search functionality for an e-commerce platform, including both linear search and binary search algorithms. The program defines a Product class and demonstrates how to use both search methods.

1. Understanding Asymptotic Notation:

Big O Notation:

Big O notation is a mathematical representation used to describe the upper limit of the time complexity of an algorithm. It helps in analyzing how the runtime of an algorithm grows relative to the input size. For example, O(n) indicates that the time complexity grows linearly with the input size, while O(log n) indicates logarithmic growth.

Best, Average, and Worst-Case Scenarios:

Linear Search:

- Best Case: O(1) (the element is found at the first position).

- Average Case: O(n) (the element is found somewhere in the middle).

- Worst Case: O(n) (the element is not found, requiring a full scan).

Binary Search:

- Best Case: O(1) (the middle element is the target).

- Average Case: O(log n) (the search space is halved each time).

- Worst Case: O(log n) (the element is not found, but the search space is reduced logarithmically).

2. Analysis:

Time Complexity Comparison:

- Linear Search: O(n)

- Binary Search: O(log n)

Suitability:

Binary search is more suitable for the e-commerce platform when the product list is large and sorted, as it significantly reduces the search time compared to linear search. However, if the product list is small or unsorted, linear search may be simpler and sufficient.

This program provides a basic framework for implementing search functionality in an e-commerce platform and can be expanded with additional features as needed.