**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.

**SOLUTION:**

**1. Understand Asymptotic Notation**

* Big O notation gives an upper bound on how the running time grows as the input size (n) increases. It lets us compare algorithms by their “growth rate” rather than machine-specific timings.
* Best case: the search finds the item immediately (e.g. first element);
* Average case: on average, it examines about half the elements;
* Worst case: it examines every element (or fails to find it).

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

**2. Setup  
We’ll define a simple Product class in Java to hold the searchable attributes:**

public class Product {

private int productId;

private String productName;

private String category;

public Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

// Getters

public int getProductId() { return productId; }

public String getProductName(){ return productName; }

public String getCategory() { return category; }

}

**3. Implementation**

import java.util.Arrays;

import java.util.Comparator;

public class SearchDemo {

// 2. Product class as a static inner class

static class Product {

private int productId;

private String productName;

private String category;

public Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

public int getProductId() { return productId; }

public String getProductName() { return productName; }

public String getCategory() { return category; }

}

// 3a. Linear Search

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

for (int i = 0; i < products.length; i++) {

if (products[i].getProductId() == targetId) {

return i;

}

}

return -1;

}

// 3b. Binary Search (array must be sorted by productId)

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

int left = 0, right = products.length - 1;

while (left <= right) {

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

int midId = products[mid].getProductId();

if (midId == targetId) {

return mid;

} else if (midId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

// 3c. Prepare a small catalogue

Product[] catalogue = {

new Product(101, "T-Shirt", "Apparel"),

new Product(205, "Laptop", "Electronics"),

new Product(309, "Coffee Mug", "Kitchen")

};

// Sort by productId for binary search

Arrays.sort(catalogue, Comparator.comparingInt(Product::getProductId));

int idToFind = 205;

System.out.println("Searching for product ID: " + idToFind);

System.out.println("→ Linear Search found it at index " + linearSearch(catalogue, idToFind));

System.out.println("→ Binary Search found it at index " + binarySearch(catalogue, idToFind));

}

}

**4. Analysis & Recommendation**

* Time Complexity
  + Linear search checks one by one: O(n) in average/worst cases.
  + Binary search repeatedly halves the search space: O(log n).
* Which to choose?
  + If your product list is small or unsorted and you rarely search, linear search is simplest.
  + For large catalogs (thousands+ items) where searches happen often, binary search (or even better, a hash-based lookup or database index) is far faster because it scales logarithmically**.**

**OUTPUT:**

A screen shot of a computer

AI-generated content may be incorrect.