**Exercise 2: E-commerce Platform Search Function**

**1. Understand Asymptotic Notation**

**What is the Big O notation:**

Big O notation describes the upper bound of the time complexity of an algorithm in the worst-case scenario. It helps in understanding the scalability and performance of the algorithm as the input size grows.

**Best, average, and worst-case scenarios for search operations:**

* **Best-case:** The item is found at the first position.
* **Average-case:** The item is found somewhere in the middle.
* **Worst-case:** The item is not found or is found at the last position.

**2. Setup**

**Create a Product class for searching:**

public class Product {

private int productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

// Getters and Setters

public int getProductId() {

return productId;

}

public void setProductId(int productId) {

this.productId = productId;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

}

**3. Implementation**

**Implement linear search and binary search algorithms:**

import java.util.Arrays;

public class SearchAlgorithms {

// Linear search

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

for (Product product : products) {

if (product.getProductName().equals(productName)) {

return product;

}

}

return null;

}

// Binary search

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

int comparison = products[mid].getProductName().compareTo(productName);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Smartphone", "Electronics"),

new Product(3, "Book", "Books"),

new Product(4, "Pen", "Stationery")

};

// Linear Search

System.out.println("Linear Search: " + linearSearch(products, "Book"));

// Sorting products by name for binary search

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

// Binary Search

System.out.println("Binary Search: " + binarySearch(products, "Book"));

}

}

**4. Analysis**

**Time complexity comparison:**

* **Linear Search:** O(n)
* **Binary Search:** O(log n)

**Suitable algorithm:**

Binary search is more suitable for large, sorted datasets due to its O(log n) time complexity. However, it requires the data to be sorted. For unsorted or small datasets, linear search is simpler and can be more efficient.