**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. **Understanding :**

#### Big O Notation

Big O notation is a mathematical notation used to describe the upper bound of an algorithm's running time or space requirements in terms of input size. It provides a high-level understanding of the algorithm's efficiency and scalability. Big O notation helps in comparing different algorithms and understanding their behavior as the input size grows.

#### Best, Average, and Worst-Case Scenarios

* **Best Case**: The scenario where the algorithm performs the minimum number of operations. For search algorithms, this often means finding the target element on the first attempt.
* **Average Case**: The scenario that represents the expected performance over a large number of inputs. It considers all possible inputs and their probabilities.
* **Worst Case**: The scenario where the algorithm performs the maximum number of operations. This provides a guarantee on the upper bound of the running time.

1. **Setup and Implementation :**

public 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;

}

@Override

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

import java.util.Arrays;

public class Search {

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;

int comparison = products[mid].getProductId().compareTo(productId);

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", "Shoes", "Apparel"),

new Product("4", "Watch", "Accessories"),

new Product("5", "Bag", "Accessories")

};

// Ensure the array is sorted for binary search

Arrays.sort(products, (a, b) -> a.getProductId().compareTo(b.getProductId()));

// Linear search test

System.out.println("Linear Search:");

Product result = linearSearch(products, "3");

System.out.println(result != null ? result : "Product not found");

// Binary search test

System.out.println("Binary Search:");

result = binarySearch(products, "3");

System.out.println(result != null ? result : "Product not found");

}

}

1. **Analysis :**

#### Time Complexity

* **Linear Search:**
  + **Best Case**: O(1) (Product is at the first position)
  + **Average Case**: O(n/2) ≈ O(n) (Product is somewhere in the middle)
  + **Worst Case**: O(n) (Product is at the last position or not present)
* **Binary Search**:
  + **Best Case**: O(1) (Product is at the middle position)
  + **Average Case**: O(log n)
  + **Worst Case**: O(log n) (Product is at either end or not present)