**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.

**1. Understanding Asymptotic Notation**

**Big O Notation and Its Importance**

* **Big O Notation:** Describes the upper bound of an algorithm's time complexity, indicating how the runtime grows with input size. It's crucial for predicting algorithm performance and scalability, especially with large datasets.

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

* **Best-Case:** The scenario where the algorithm does the least work, such as finding a search target at the start of the list.
* **Average-Case:** Represents typical performance across all inputs, offering a practical measure of efficiency.
* **Worst-Case:** The scenario where the algorithm does the most work, like searching through the entire list without finding the target or finding it at the last position.
* **2. Setup**
* **Create a Class Product with Attributes for Searching**
* We'll start by defining a Product class with attributes that are relevant for search operations, such as productId, productName, and category.

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;

}

@Override

public String toString() {

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

}

}

**3. Implementation**

**Implement Linear Search and Binary Search Algorithms**

* **Linear Search**: This algorithm searches for an element by iterating through the array one by one. It doesn't require the array to be sorted.
* **Binary Search**: This algorithm works on sorted arrays and repeatedly divides the search interval in half. If the target value is less than the middle element, the search continues in the left half; otherwise, it continues in the right half.

import java.util.Arrays;

public class ECommercePlatformSearch {

// Linear Search

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

for (Product product : products) {

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

return product;

}

}

return null; // Product not found

}

// Binary Search (requires sorted array)

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].productName.compareToIgnoreCase(productName);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null; // Product not found

}

public static void main(String[] args) {

// Creating an array of products

Product[] products = {

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

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

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

new Product(4, "Tablet", "Electronics"),

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

};

// Sorting the array by productName for binary search

Arrays.sort(products, (p1, p2) -> p1.productName.compareToIgnoreCase(p2.productName));

// User input for searching

String searchName = "Tablet";

// Perform Linear Search

Product linearResult = linearSearch(products, searchName);

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

// Perform Binary Search

Product binaryResult = binarySearch(products, searchName);

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

}

}

**4. Analysis**

**Compare the Time Complexity of Linear and Binary Search Algorithms**

* **Linear Search**:
  + **Time Complexity**: O(n), where n is the number of products in the array.
  + **Best-Case**: O(1) (when the target element is at the first position).
  + **Worst-Case**: O(n) (when the target element is at the last position or not present).
* **Binary Search**:
  + **Time Complexity**: O(log n), where n is the number of products in the sorted array.
  + **Best-Case**: O(1) (when the target element is the middle element).
  + **Worst-Case**: O(log n) (when the target element is found at the extremes or not present).

**Discuss Which Algorithm Is More Suitable for Your Platform and Why**

* **Linear Search** is simple to implement and works on unsorted data, making it versatile. However, its O(n) time complexity can lead to slow performance for large datasets.
* **Binary Search** is more efficient with O(log n) time complexity, but it requires the data to be sorted. This makes it suitable for platforms where search performance is critical and the data can be maintained in a sorted order.