**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.

**Step 1: Understand Asymptotic Notation**

Big O notation is a mathematical representation used to describe the upper bound of an algorithm's runtime or space requirements in terms of the size of the input (n). It helps in analyzing the efficiency and scalability of algorithms by focusing on their growth rates as input size increases. Big O notation simplifies the comparison between algorithms by abstracting away constants and lower-order terms, providing a high-level understanding of their performance.

1. Best Case:

The scenario where the search operation performs the minimum number of steps.

For example, in a linear search, the best case occurs when the target element is the first element in the array (O(1)).

2. Average Case:

The scenario where the search operation performs an average number of steps.

For linear search, this would be when the target element is in the middle of the array (O(n/2), simplified to O(n)).

3. Worst Case:

The scenario where the search operation performs the maximum number of steps.

For linear search, the worst case occurs when the target element is the last element or not present at all (O(n)).

**Step 2: Setup**

Create a class Product with attributes for searching, such as productId, productName, and category.

**Step 3: Implementation**

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

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;

    }

    // Getters

    public String getProductId() {

        return productId;

    }

    public String getProductName() {

        return productName;

    }

    public String getCategory() {

        return category;

    }

    @Override

    public String toString() {

        return "Product{" +

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

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

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

                '}';

    }

}

class Library {

    // Linear search by product name

    public Product linearSearchByName(Product[] products, String productName) {

        for (Product product : products) {

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

                return product;

            }

        }

        return null;

    }

    // Binary search by product name

    public Product binarySearchByName(Product[] products, String productName) {

        // Ensure the array is sorted by product name

        Arrays.sort(products, Comparator.comparing(Product::getProductName));

        int left = 0;

        int right = products.length - 1;

        while (left <= right) {

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

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

            if (comparison == 0) {

                return products[mid];

            } else if (comparison < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

}

public class ECommerce {

    public static void main(String[] args) {

        Product[] products = {

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

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

            new Product("3", "Tablet", "Electronics"),

            new Product("4", "Chair", "Furniture"),

            new Product("5", "Desk", "Furniture")

        };

        Library library = new Library();

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter product name to search:");

        String targetName = scanner.nextLine();

        // Perform linear search

        Product foundProductLinear = library.linearSearchByName(products, targetName);

        if (foundProductLinear != null) {

            System.out.println("Product found using linear search: " + foundProductLinear);

        } else {

            System.out.println("Product not found using linear search.");

        }

        // Perform binary search

        Product foundProductBinary = library.binarySearchByName(products, targetName);

        if (foundProductBinary != null) {

            System.out.println("Product found using binary search: " + foundProductBinary);

        } else {

            System.out.println("Product not found using binary search.");

        }

        scanner.close();

    }

}

**Step 4: Analysis**

Time Complexity Comparison

**• Linear Search:**

Best Case: O(1)

Average Case: O(n)

Worst Case: O(n)

**• Binary Search:**

Best Case: O(1)

Average Case: O(log n)

Worst Case: O(log n)

**• Linear Search:**

Suitable for small datasets or unsorted data.

Simple to implement and does not require pre-sorting.

**• Binary Search:**

Suitable for large datasets where the data is sorted.

More efficient for large datasets due to its logarithmic time complexity.

Requires maintaining the sorted order of data, which can add overhead during insertion and deletion operations.

Conclusion: For an e-commerce platform, binary search is generally more suitable due to its superior performance with large datasets. However, maintaining a sorted dataset is crucial, and the additional overhead should be considered. Linear search might be used for smaller datasets or when the data is frequently changing and cannot be maintained in a sorted order.