**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:**

**Asymptotic Notation:**

**\* Big O Notation:**

Big O notation is used to describe the **performance or complexity** of an algorithm in terms of input size (n). It helps predict how the algorithm behaves as data grows.

**\*Big O for Search Operations**

| **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) |

* **Linear Search**: Scans each element.
* **Binary Search**: Requires a sorted array. Divides the array into halves.

**Code:**

public class ECommerceSearch {

    static 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;

        }

        public void display() {

            System.out.println("ID: " + productId + ", Name: " + productName + ", Category: " + category);

        }

    }

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

        for (Product product : products) {

            if (product.productId == idToFind) {

                return product;

            }

        }

        return null;

    }

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

        int left = 0;

        int right = products.length - 1;

        while (left <= right) {

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

            if (products[mid].productId == idToFind) {

                return products[mid];

            } else if (products[mid].productId < idToFind) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] unsortedProducts = {

                new Product(304, "Backpack", "Travel"),

                new Product(301, "Notebook", "Stationery"),

                new Product(305, "Jacket", "Fashion"),

                new Product(302, "Camera", "Electronics")

        };

        Product[] sortedProducts = {

                new Product(301, "Notebook", "Stationery"),

                new Product(302, "Camera", "Electronics"),

                new Product(304, "Backpack", "Travel"),

                new Product(305, "Jacket", "Fashion")

        };

        int searchId1 = 305;

        Product resultLinear = linearSearch(unsortedProducts, searchId1);

        System.out.println("\nLinear Search Result:");

        if (resultLinear != null) {

            resultLinear.display();

        } else {

            System.out.println("Product with ID " + searchId1 + " not found.");

        }

        int searchId2 = 302;

        Product resultBinary = binarySearch(sortedProducts, searchId2);

        System.out.println("\nBinary Search Result:");

        if (resultBinary != null) {

            resultBinary.display();

        } else {

            System.out.println("Product with ID " + searchId2 + " not found.");

        }

    }

}

**Analysis:**

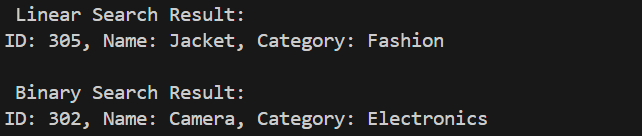
**Time Complexity Comparison**

| **Algorithm** | **Time Complexity** |
| --- | --- |
| Linear Search | O(n) |
| Binary Search | O(log n) |

* Linear Search is simple and works on unsorted data.
* Binary Search is much faster for large datasets but requires sorting (O(n log n) to sort initially).

Conclusion : For real-time optimized e-commerce search, use Binary Search with a sorted dataset or indexed data structures (like trees or hash maps) for scalability.

**Output:**

****