**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.

**Answer**

**Big O Notation**

Big O Notation is a mathematical notation that describes the upper bound of an algorithm’s time or space complexity. It tells you how the runtime or space requirements of an algorithm grow relative to input size n.

**Time Complexities of Search Operations**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** | **Sorted Data Needed?** |
| --- | --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) | No |
| Binary Search | O(1) | O(log n) | O(log n) | Yes |

* **Linear Search**: Good for small or unsorted data.
* **Binary Search**: Much faster for large, sorted data sets.

**Program**

import java.util.\*;

class Product implements Comparable<Product> {

int productId;

String productName;

String category;

Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

public int compareTo(Product other) {

return this.productId - other.productId;

}

}

public class EcommerceSearch {

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

for (Product p : products) {

if (p.productId == targetId) {

return p;

}

}

return null;

}

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

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

return products[mid];

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(103, "Headphones", "Electronics"),

new Product(101, "Laptop", "Electronics"),

new Product(105, "T-shirt", "Clothing"),

new Product(102, "Coffee Mug", "Kitchen"),

new Product(104, "Book", "Stationery")

};

// Sort for binary search

Product[] sortedProducts = products.clone();

Arrays.sort(sortedProducts);

int targetId = 104;

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

Product result1 = linearSearch(products, targetId);

System.out.println(result1 != null ? "Found: " + result1 : "Product not found");

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

Product result2 = binarySearch(sortedProducts, targetId);

System.out.println(result2 != null ? "Found: " + result2 : "Product not found");

System.out.println("\n---- Time Complexity Analysis ----");

System.out.println("Linear Search: O(n) - checks each item one by one.");

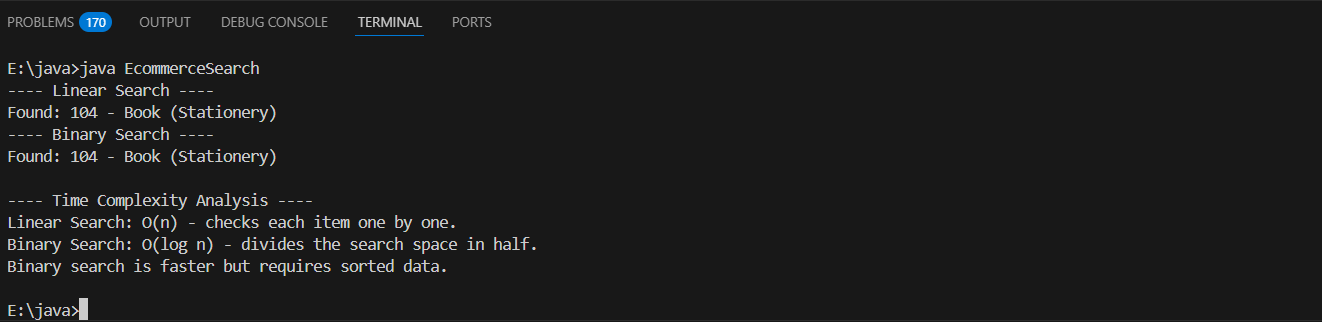
System.out.println("Binary Search: O(log n) - divides the search space in half.");

System.out.println("Binary search is faster but requires sorted data.");

}

}

**Output**

****

**Comparison:**

| **Feature** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| Works on unsorted data | Yes | No (requires sorted data) |
| Time complexity | O(n) | O(log n) |
| Suitable for large datasets | Slow | Fast |
| Code simplicity | Very simple | Slightly complex |
| Use case | Small or infrequent searches | Large-scale frequent searches |

For an e-commerce platform, Binary Search is the better choice for the following reasons:

**1. Performance with Large Data**

E-commerce platforms often manage thousands or millions of products.

* Binary Search performs in O(log n) time, making it highly scalable.
* Linear Search takes O(n) time, which becomes inefficient for large datasets.

**2. Search Speed Improves User Experience**

Fast search is essential to keep customers engaged. A delay in results may lead to user frustration and abandonment.

* Binary Search ensures faster search results when the product list is sorted by product ID or name.

**3. Sorted Data is Common in E-commerce**

Product catalogs are typically sorted (e.g., by price, name, popularity), either in databases or in-memory.

* Sorting once allows repeated fast searches.
* You can use efficient sort algorithms (O(n log n)) or maintain sorted order with balanced trees or search indexes.

**4. Modern Search Systems Use Hybrid/Indexed Search**

In real-world systems, search is often handled via search engines (like Elasticsearch) or inverted indexes, but for in-memory operations:

* Binary Search or Hash-based lookups (for exact ID search) are preferred.

**Conclusion**

Binary Search is more suitable for an e-commerce platform because it provides faster performance, handles large data efficiently, and fits naturally into a system where products are sorted or indexed.