**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.

**1. Understand Asymptotic Notation**

Big O Notation

Big O notation describes the upper bound of an algorithm’s runtime in terms of input size n. It helps compare algorithms by their growth rates:

* O(1): Constant time
* O(log n): Logarithmic time
* O(n): Linear time
* O(n log n): Log-linear time
* O(n²): Quadratic time, and so on.

Best, Average, and Worst Case for Search

| Search Type | Best Case | Average Case | Worst Case |
| --- | --- | --- | --- |
| Linear Search | O(1) (first element) | O(n/2) ≈ O(n) | O(n) (last element/not found) |
| Binary Search | O(1) (middle) | O(log n) | O(log n) |

**Code:-**

import java.util.Arrays;

import java.util.Comparator;

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

    }

}

public class ECommerceSearch {

    // Linear Search by product name

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

        for (Product p : products) {

            if (p.productName.equalsIgnoreCase(targetName)) {

                return p;

            }

        }

        return null;

    }

    // Binary Search by product name (assumes sorted array)

    public static Product binarySearch(Product[] products, String targetName) {

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

        while (left <= right) {

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

            int cmp = products[mid].productName.compareToIgnoreCase(targetName);

            if (cmp == 0) return products[mid];

            if (cmp < 0) left = mid + 1;

            else right = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(102, "Shoes", "Fashion"),

            new Product(103, "Watch", "Accessories"),

            new Product(104, "Mobile", "Electronics"),

            new Product(105, "Headphones", "Electronics")

        };

        // Perform Linear Search

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

        String searchTerm = "Watch";

        Product result1 = linearSearch(products, searchTerm);

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

        // Sort products by name before Binary Search

        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

        // Perform Binary Search

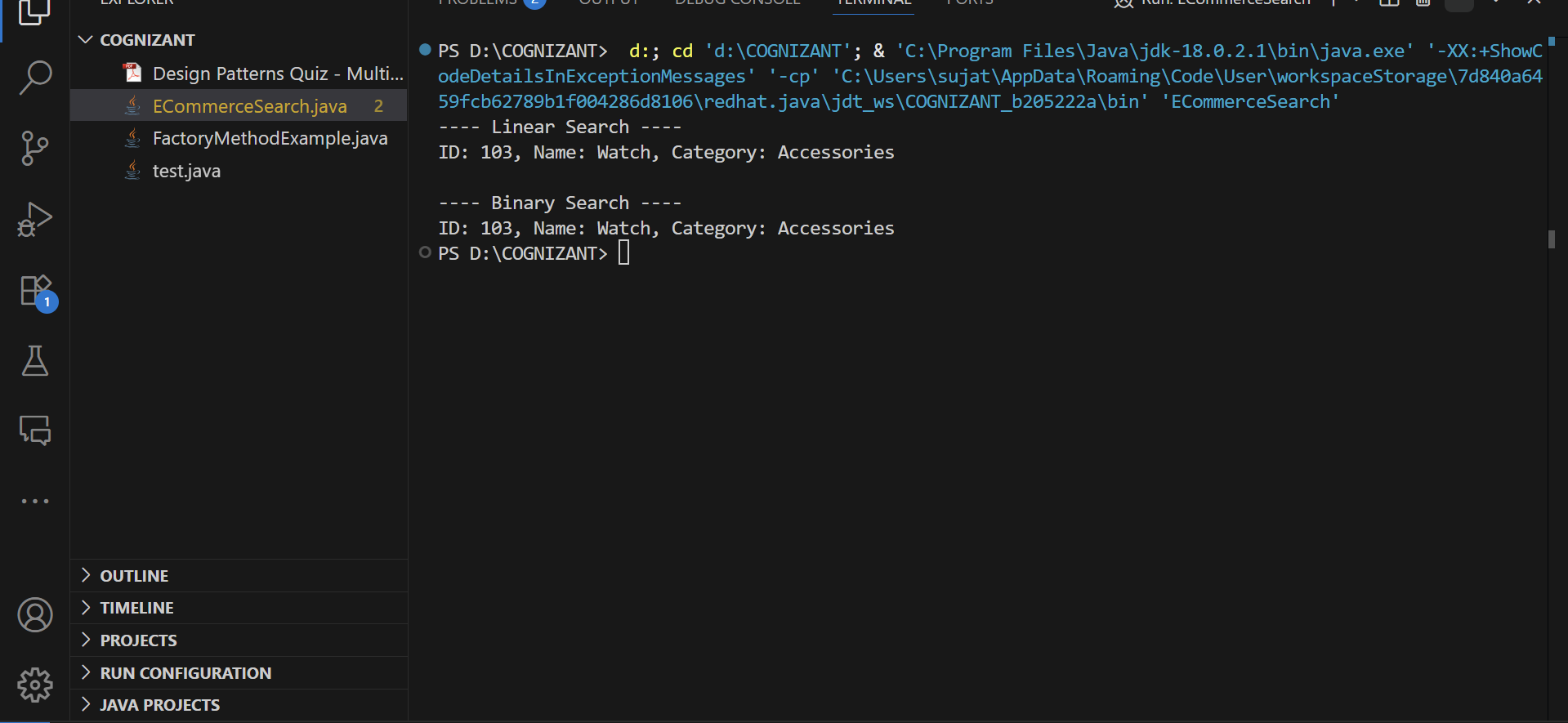
        System.out.println("\n---- Binary Search ----");

        Product result2 = binarySearch(products, searchTerm);

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

    }

}

**Output:-**

1. **Analysis: Linear vs Binary Search**

| **Algorithm** | **Time Complexity** | **Space Complexity** | **Sorted Requirement** | **Performance** |
| --- | --- | --- | --- | --- |
| **Linear** | O(n) | O(1) | ❌ No | Slower for large datasets |
| **Binary** | O(log n) | O(1) | ✅ Yes | Faster if array is sorted |

**Which Algorithm is More Suitable?**

* **Linear Search** is better when:
  + The dataset is small.
  + Data is unsorted.
  + Quick implementation is needed.
* **Binary Search** is better when:
  + The dataset is large and can be sorted.
  + High performance is required.
  + Search operation is frequent.

**✅ For an e-commerce platform, Binary Search is preferred** after sorting, especially when searching by product names or IDs in large inventories.