**Exercise 2: E-commerce Platform Search Function**

**Big O Notation**

* Big O notation describes the *upper bound* of an algorithm’s running time or space usage as input size n grows.
* It helps to analyze and compare algorithms independently of hardware or implementation details.
* We focus on the dominant term and drop constants → e.g., O(2n + 3) simplifies to O(n).

| **Scenario** | **Linear Search (Unsorted Data)** | **Binary Search (Sorted Data)** |
| --- | --- | --- |
| Best case | O(1) → Item found at first position | O(1) → Item found at mid-point |
| Average case | O(n/2) → Simplifies to O(n) | O(log n) |
| Worst case | O(n) → Item at last position or not found | O(log n) |

**Code:**

import java.util.Arrays;

import java.util.Comparator;

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;

    }

    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 + '\'' +

                '}';

    }

}

public class ECommerceSearch {

    private Product[] products;

    public ECommerceSearch(Product[] products) {

        this.products = products;

    }

    public Product linearSearch(String name) {

        for (Product product : products) {

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

                return product;

            }

        }

        return null;

    }

    public Product binarySearch(String name) {

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

        while (left <= right) {

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

            int cmp = products[mid].getProductName().compareToIgnoreCase(name);

            if (cmp == 0) {

                return products[mid];

            } else if (cmp < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public void sortProductsByName() {

        Arrays.sort(products, Comparator.comparing(Product::getProductName, String.CASE\_INSENSITIVE\_ORDER));

    }

    public void listProducts() {

        for (Product product : products) {

            System.out.println(product);

        }

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product("P001", "Laptop", "Electronics"),

            new Product("P002", "Mouse", "Electronics"),

            new Product("P003", "Shirt", "Clothing"),

            new Product("P004", "Book", "Stationery"),

            new Product("P005", "Phone", "Electronics")

        };

        ECommerceSearch search = new ECommerceSearch(products);

        System.out.println(" Linear Search (unsorted array):");

        Product result1 = search.linearSearch("Phone");

        if (result1 != null) {

            System.out.println("Found: " + result1);

        } else {

            System.out.println("Not found");

        }

        System.out.println("\n Sorting products for binary search...");

        search.sortProductsByName();

        search.listProducts();

        System.out.println("\n Binary Search (sorted array):");

        Product result2 = search.binarySearch("Phone");

        if (result2 != null) {

            System.out.println("Found: " + result2);

        } else {

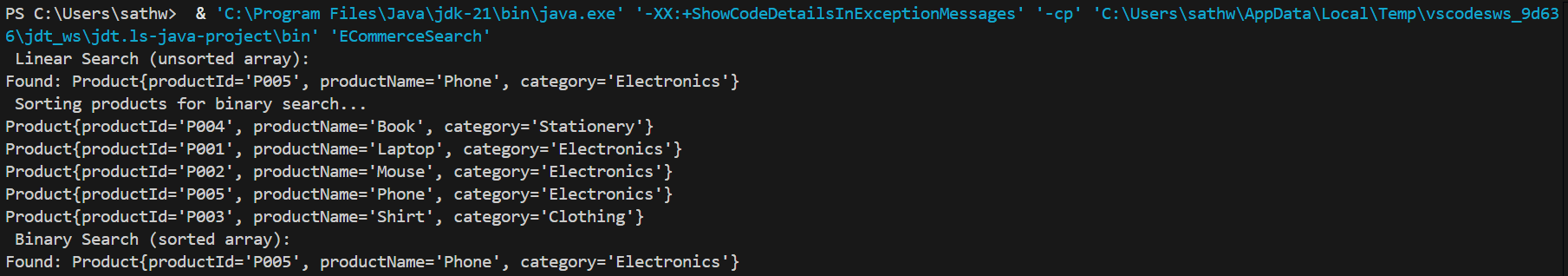
            System.out.println("Not found");

        }

    }

}

**Output:**



**Linear Search**

* Search each product one-by-one in the array.
* No sorting is required.

**Binary Search**

* The array must be **sorted** (e.g., by productId or productName).
* It divides the search space in half at each step.

**Analysis: Linear vs Binary Search**

| **Property** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| Time Complexity | O(n) | O(log n) |
| Data Requirement | Unsorted array | Sorted array |
| Implementation complexity | Very simple | Slightly more complex |
| Flexibility | Can work on any array | Needs sorted array |

**Which is better for e-commerce?**

**Binary search** is more efficient → O(log n) is much faster for large data (e.g., thousands of products).