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

**Big O Notation**

Big O notation describes how the runtime or space of an algorithm scales with the input size (n). It helps compare the efficiency of algorithms.

| **Big O** | **Meaning** | **Example** |
| --- | --- | --- |
| O(1) | Constant time | Accessing array element by index |
| O(log n) | Logarithmic time | Binary search |
| O(n) | Linear time | Linear search |
| O(n log n) | Linearithmic time | Merge sort |
| O(n²) | Quadratic time | Nested loops |

**Best, Average, Worst Case for Search**

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

**Implementation**

Main.java

Product.java

SearchService.java

**Product.java**

public 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 String toString() {

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

    }

}

**SearchSevice.java**

import java.util.Arrays;

import java.util.Comparator;

public class SearchService {

    // Linear search by productName

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

        for (Product product : products) {

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

                return product;

            }

        }

        return null;

    }

    // Binary search by productName

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

        int left = 0;

        int right = products.length - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

            int compare = products[mid].productName.compareToIgnoreCase(name);

            if (compare == 0) {

                return products[mid];

            } else if (compare < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    // Sort products by name for binary search

    public static void sortProductsByName(Product[] products) {

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

    }

}

**Main.java**

public class Main {

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(102, "Shirt", "Clothing"),

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

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

            new Product(105, "Shoes", "Footwear")

        };

        // Linear Search

        Product found1 = SearchService.linearSearch(products, "Phone");

        System.out.println("Linear Search Result: " + (found1 != null ? found1 : "Not Found"));

        // Prepare for binary search

        SearchService.sortProductsByName(products);

        // Binary Search

        Product found2 = SearchService.binarySearch(products, "Phone");

        System.out.println("Binary Search Result: " + (found2 != null ? found2 : "Not Found"));

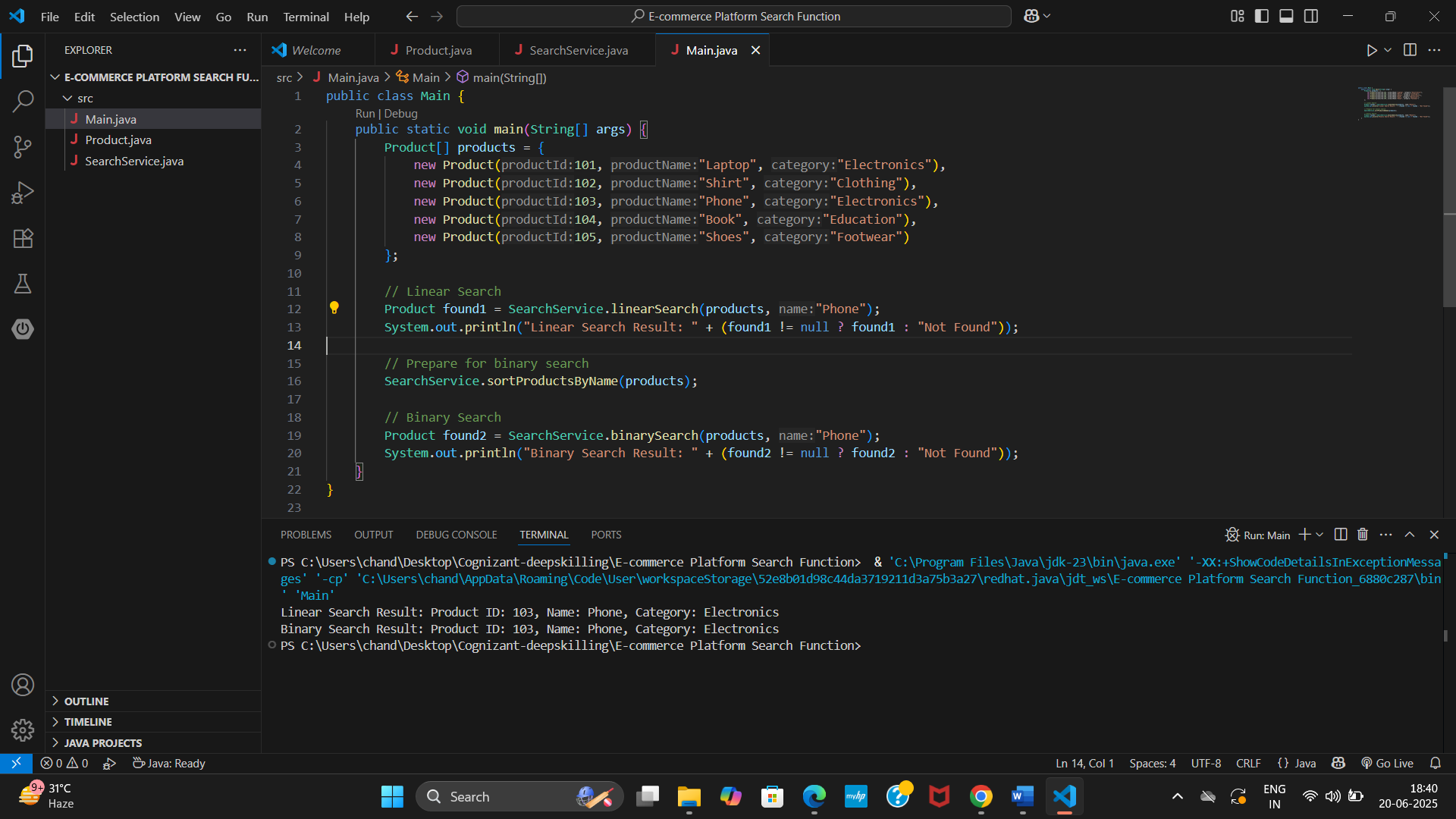
    }

}

**Output:**

Linear Search Result: Product ID: 103, Name: Phone, Category: Electronics

Binary Search Result: Product ID: 103, Name: Phone, Category: Electronics



**Analysis**

| **Search Type** | **Time Complexity** | **Sorted Required?** | **Efficiency (Large n)** |
| --- | --- | --- | --- |
| Linear Search | O(n) | No | Slower |
| Binary Search | O(log n) | Yes | Faster |

* We can use **Linear Search** for small datasets or unsorted data.
* We can use **Binary Search** for large sorted datasets and it’s significantly faster as it reduces the search space by half in each step.
* **Recommended** for e-commerce platforms:
* Maintain a sorted product list or use a database index to support **binary search** for optimal performance.