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

* **Step-1: Asymptotic Notation**
* **Big O Notation:**

Big O notation represents the **upper bound** of an algorithm’s runtime. It focuses on how the algorithm time increase with increasing input, providing a general idea of its efficiency. Essentially, it tells you the worst-case scenario for an algorithm's performance

| **Notation** | **Meaning** |
| --- | --- |
| O(1) | Constant time |
| O(n) | Linear time |
| O(log n) | Logarithmic time |
| O(n²) | Quadratic time |

* **Search Operations:**

**1.Linear Search:**

**Best Case:** O(1)

**Average Case:** O(n)

**Worst Case:** O(n)

**2.Binary Search:**

**Best Case:** O(1)

**Average Case:** O(logn)

**Worst Case:** O(logn)

* **Step-2,3:Implementation**

**Product.java**

class Product {

    int productId;

    String productName;

    String category;

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

        this.productId = id;

        this.productName = name;

        this.category = category;

    }

}

**Searching\_Test.java**

import java.util.\*;

public class Searching\_Test {

    public static Product linearSearch(Product[] products, String k) {  //we are returning Product class type

        for (Product p : products) {

            if (p.productName.equals(k)) {

                return p;

            }

        }

        return null;

    }

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

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

        int low = 0, high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

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

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

            else if (cmp < 0) high = mid - 1;

            else low = mid + 1;

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(1, "Laptop", "Electronics"),

            new Product(2, "Phone", "Electronics"),

            new Product(3, "Coffee", "Grocery"),

            new Product(4, "Tomatoes", "Grocery"),

            new Product(5, "CognizantHoody", "Fashion")

        };

        String s = "CognizantHoody";

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

        Product lr = linearSearch(products, s);

        if(lr!=null) System.out.println(lr+"    "+"Product Found");

        else System.out.println("Product Not Found");

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

        Product br = binarySearch(products, s);

        if(br!=null) System.out.println(lr+"    "+"Product Found");

        else System.out.println("Product Not Found");

    }

}

**Step-4:analysis**

* As said above the best case is same for both algorithms -0(1), but when comes to avg,worst cases binary Search algorithm is more efficient with 0(logn).
* So I prefer **Binary Search for larger Search Area**,

else I use **Linear Search for Smaller data sets** or Search area as its simpler to implement**.**

**Output:**

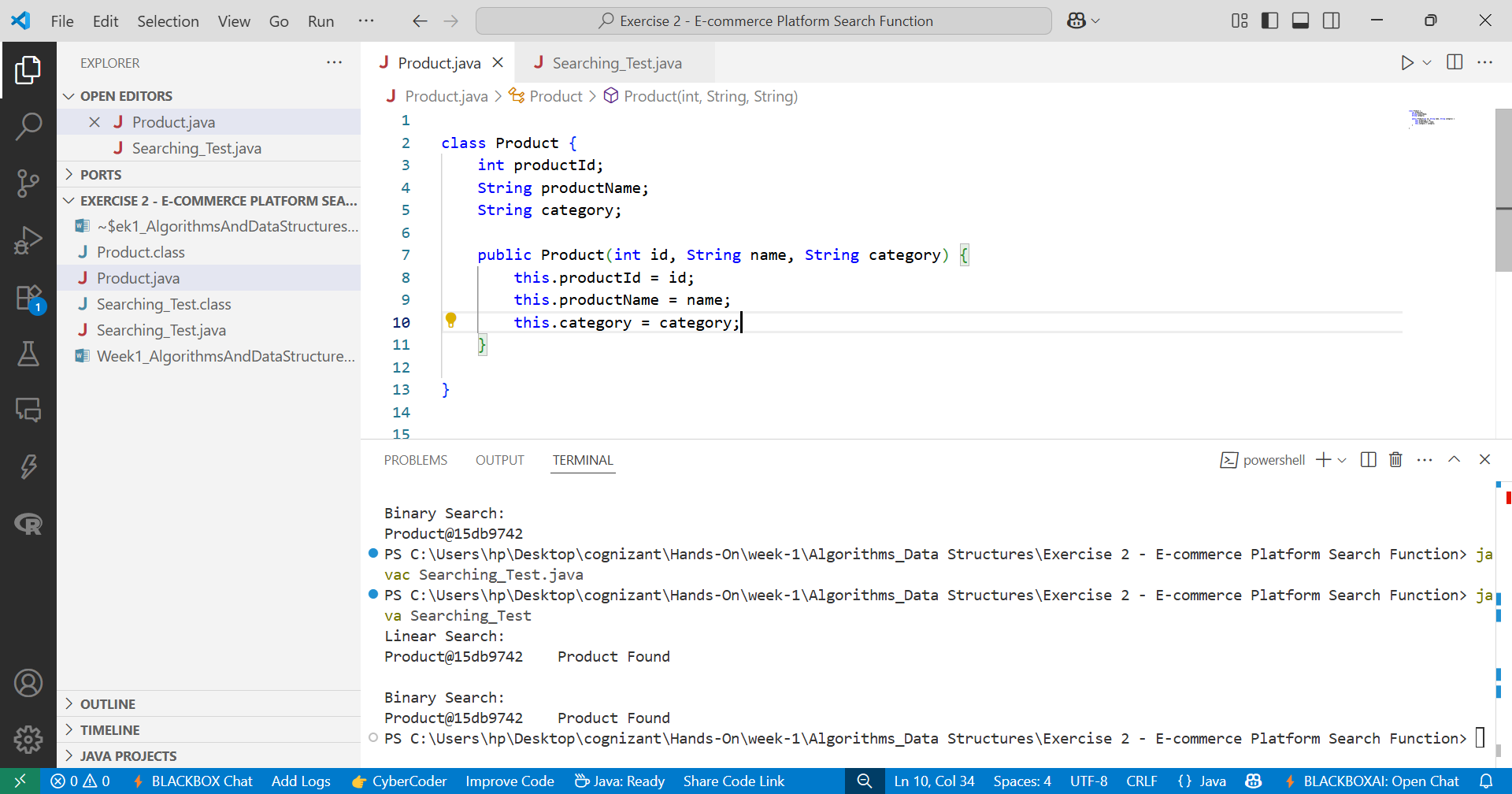
****

Fig-1:Product class

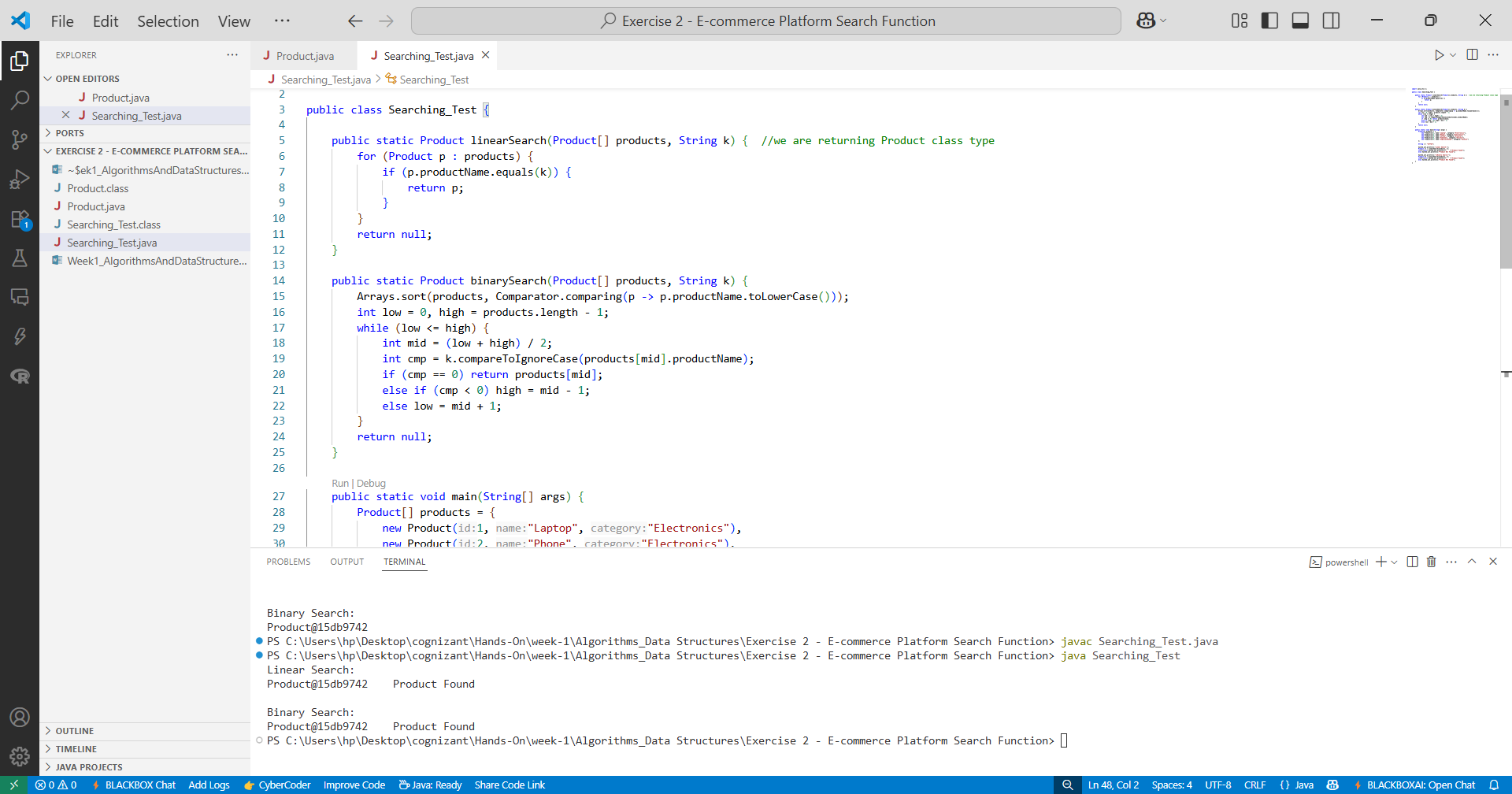


Fig-2:Searching\_Test class with output