**Data structures and Algorithms:**

**QUESTION 1:**

**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.

**SOLUTION:**

**Product.java:**

public 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;

}

public String toString() {

return "[" + productId + ", " + productName + ", " + category + "]";

}

}

**SearchDemo.java:**

import java.util.Arrays;

import java.util.Comparator;

public class SearchDemo {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

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

while (left <= right) {

int mid = (left + right) / 2;

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

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", "Footwear"),

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

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

};

Product foundLinear = linearSearch(products, "Shoes");

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

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

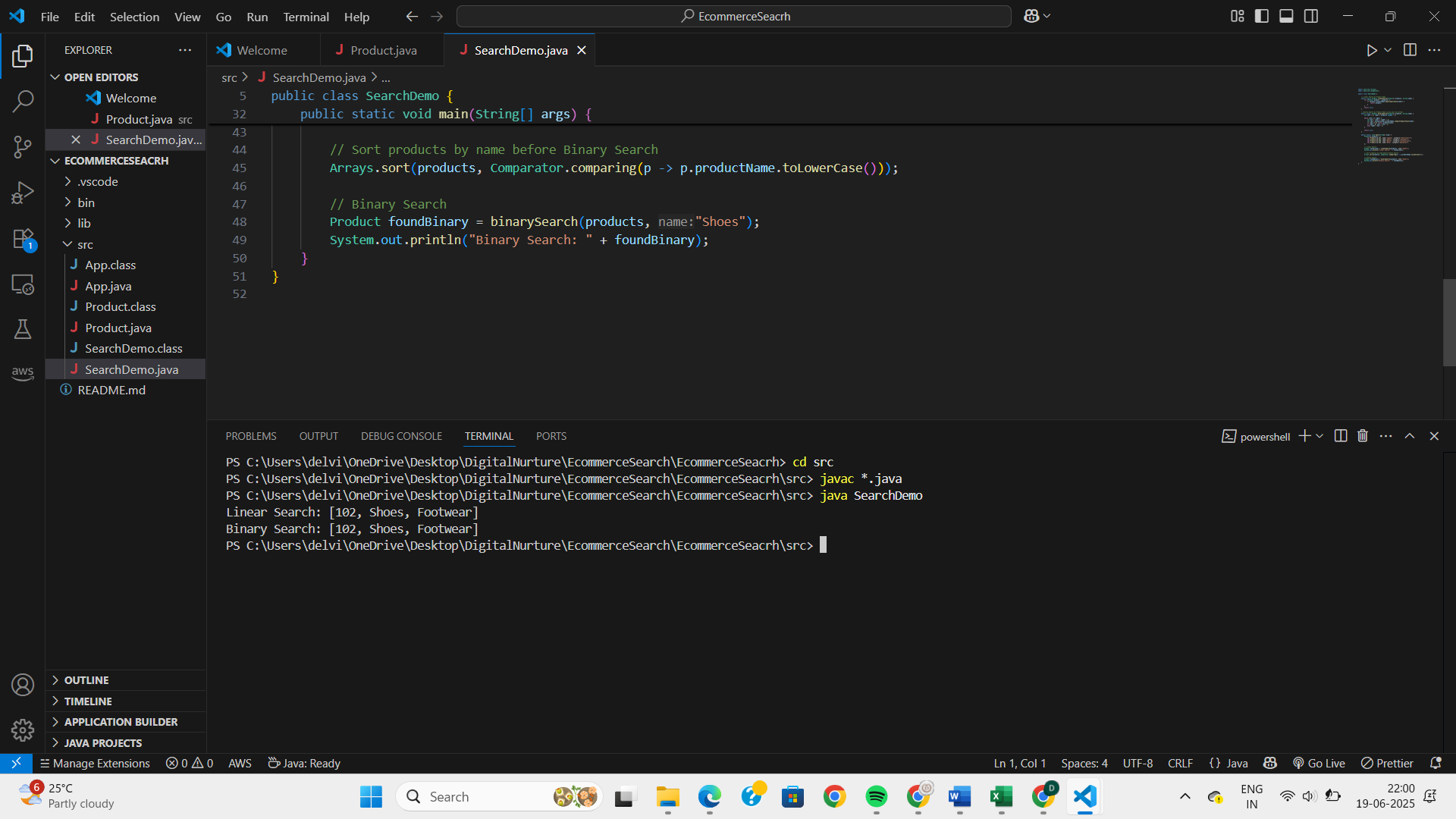
Product foundBinary = binarySearch(products, "Shoes");

System.out.println("Binary Search: " + foundBinary);

}

}

**OUTPUT:**



**QUESTION 2:**

**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**SOLUTION:**

**Forecasting.java:**

public class Forecasting {

public static double futureValue(double principal, double rate, int periods) {

if (periods == 0) return principal;

return (1 + rate) \* futureValue(principal, rate, periods - 1);

}

public static void main(String[] args) {

double P = 10000;

double r = 0.05;

int n = 5;

double result = futureValue(P, r, n);

System.out.printf("Future Value after %d years: %.2f\n", n, result);

}

}

**OUTPUT:**

