**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:

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.

Setup:

Create a class Product with attributes for searching, such as productId, productName, and category.

Implementation:

Implement linear search and binary search algorithms.

Store products in an array for linear search and a sorted array for binary search.

Analysis:

Compare the time complexity of linear and binary search algorithms.

Discuss which algorithm is more suitable for your platform and why.

**Answer:**

#### **Product.java**

package search;

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;

}

}

**SearchEngine.java**

package search;

import java.util.Arrays;

import java.util.Comparator;

public class SearchEngine {

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 low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void sortProducts(Product[] products) {

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

}

}

**SearchTest.java**

package search;

import java.util.Scanner;

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shampoo", "Personal Care"),

new Product(103, "T-Shirt", "Apparel"),

new Product(104, "Blender", "Home Appliances")

};

Scanner scanner = new Scanner(System.in);

System.out.print("Enter product name to search: ");

String searchQuery = scanner.nextLine();

// Linear Search

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

Product result1 = SearchEngine.linearSearch(products, searchQuery);

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

// Binary Search

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

SearchEngine.sortProducts(products);

Product result2 = SearchEngine.binarySearch(products, searchQuery);

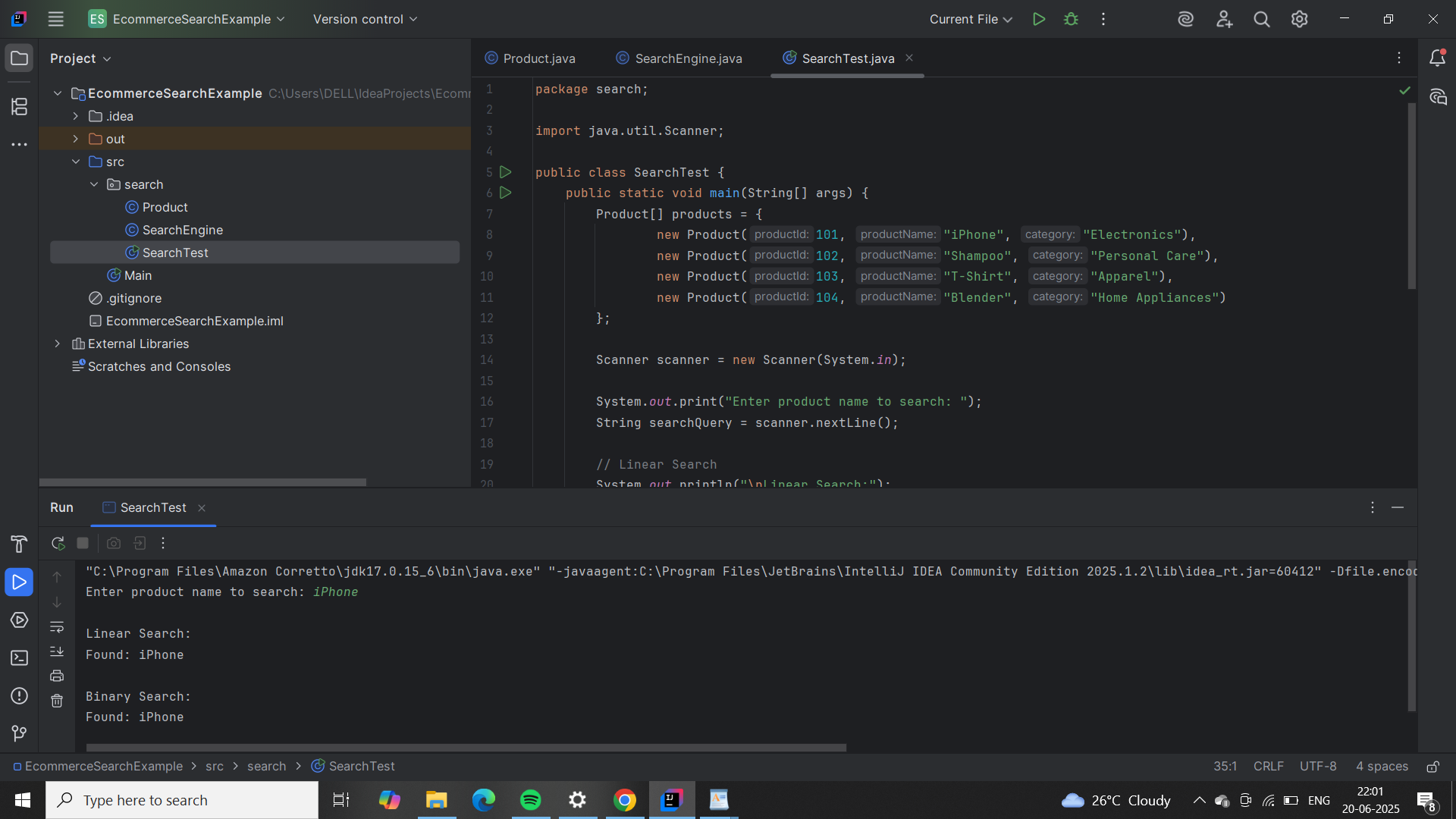
System.out.println(result2 != null ? "Found: " + result2.productName : "Product not found");

scanner.close();

}

}

**Output:**

****

**Exercise 7: Financial Forecasting**

Scenario:

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

Steps:

Understand Recursive Algorithms:

Explain the concept of recursion and how it can simplify certain problems.

Setup:

Create a method to calculate the future value using a recursive approach.

Implementation:

Implement a recursive algorithm to predict future values based on past growth rates.

Analysis:

Discuss the time complexity of your recursive algorithm.

Explain how to optimize the recursive solution to avoid excessive computation.

**Answer:**

**Forecast.java**

import java.util.Scanner;

public class Forecast {

// Recursive method to calculate future value

public static double calculateFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

return calculateFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter initial value: ");

double initialValue = scanner.nextDouble();

System.out.print("Enter annual growth rate (in %): ");

double growthRatePercent = scanner.nextDouble();

double growthRate = growthRatePercent / 100.0;

System.out.print("Enter number of years: ");

int years = scanner.nextInt();

double futureValue = calculateFutureValue(initialValue, growthRate, years);

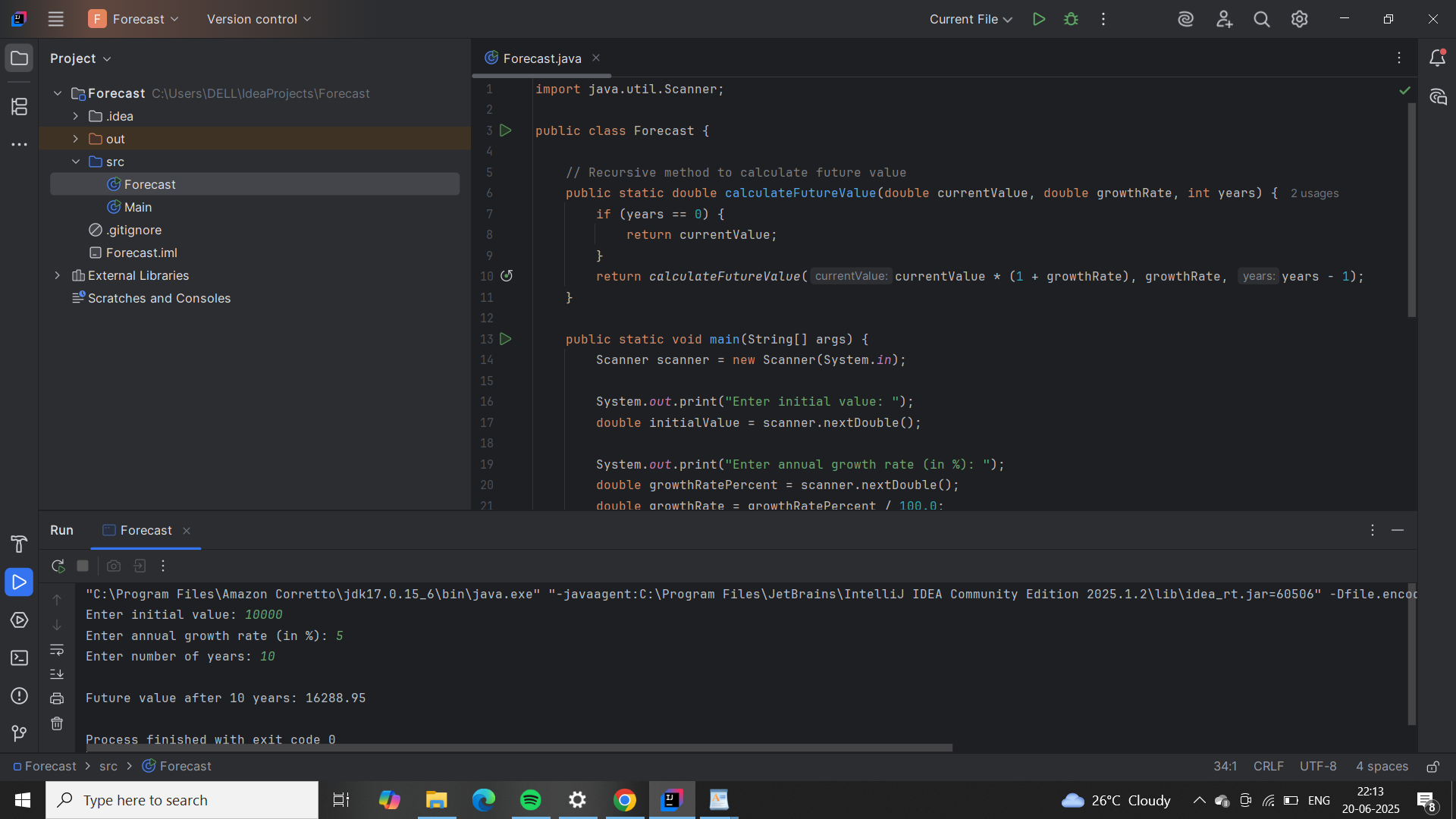
System.out.printf("\nFuture value after %d years: %.2f\n", years, futureValue);

scanner.close();

}

}

**Output:**

****