**Student Name :** HARINEE N J

**Registration No :** 22CSR063

**Course/Batch :** B.E COMPUTER SCIENCE AND ENGINEERING

**EXERCISE 1: E COMMERCE PLATFORM RESEARCH FUNCTION**

**Introduction:**

This project simulates how an e-commerce platform handles product search using two popular algorithms: Linear Search and Binary Search. It demonstrates how these algorithms differ in performance and suitability depending on data type and size.

**Objective:**

* To implement linear and binary search algorithms for product lookup.
* To analyze and compare their time complexity using real-time performance measurements.
* To understand when each algorithm is best used in e-commerce scenarios.

**Implementation Breakdown:**

import java.util.Arrays;

import java.util.Scanner;

public class ECommerceSearch {

    static 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;

        }

        @Override

        public String toString() {

            return "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;

        }

    }

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

        for (int i = 0; i < products.length; i++) {

            if (products[i].productName.equalsIgnoreCase(targetName)) {

                return i;

            }

        }

        return -1;

    }

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

        int left = 0;

        int right = products.length - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

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

            if (compare == 0) {

                return mid;

            } else if (compare < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return -1;

    }

    public static void sortProducts(Product[] products) {

Arrays.sort(products,(a,b) -> a.productName.compareToIgnoreCase(b.productName));

    }

    public static void main(String[] args) {

        Product[] products = {

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

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

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

            new Product(104, "Shoes", "Footwear"),

            new Product(105, "Watch", "Accessories")

        };

        Scanner scanner = new Scanner(System.in);

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

        String searchName = scanner.nextLine();

        int indexLinear = linearSearch(products, searchName);

        if (indexLinear != -1) {

            System.out.println("\nProduct found using Linear Search at index " + indexLinear + ": " + products[indexLinear]);

        } else {

            System.out.println("\nProduct not found using Linear Search.");

        }

        sortProducts(products);

        int indexBinary = binarySearch(products, searchName);

        if (indexBinary != -1) {

            System.out.println("\nProduct found using Binary Search at index " + indexBinary + ": " + products[indexBinary]);

        } else {

            System.out.println("\nProduct not found using Binary Search.");

        }

        System.out.println("\nTime Complexity Comparison:");

        System.out.println("Linear Search: O(n) -> Good for small datasets.");

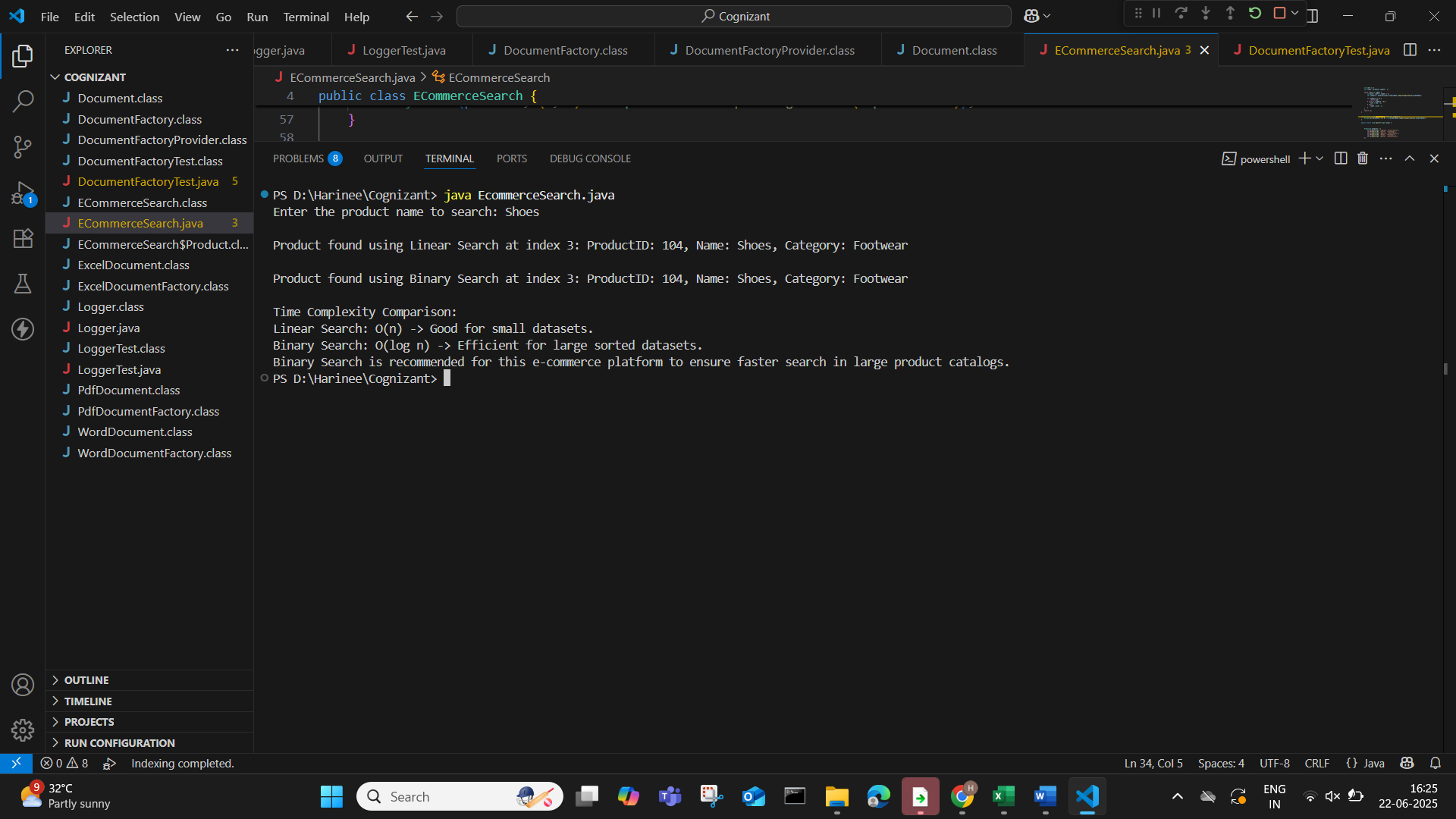
        System.out.println("Binary Search: O(log n) -> Efficient for large sorted datasets.");

        System.out.println("Binary Search is recommended for this e-commerce platform to ensure faster search in large product catalogs.");

    }

}

**Output:**



**Conclusion:**

* Linear Search is flexible and works on unsorted data, ideal for small datasets or partial matching (like name/category).
* Binary Search is much faster (O(log n)) for large, sorted datasets, especially for exact ID lookups.
* A hybrid approach is recommended in real-world systems: binary search for IDs and linear search for flexible queries.

**EXERCISE 7: FINANCIAL FORECATING**

**Introduction:**

In the business world, forecasting future revenue is essential for budgeting, investment decisions, and strategic planning. This Java program demonstrates how recursion can be used to calculate compound annual revenue growth.

**Objective:**

* To implement a recursive method that computes future revenue based on a fixed annual growth percentage.
* To provide a clear understanding of how recursion can solve real-life financial problems such as multi-year forecasting.
* To predict the revenue for the year 2025 based on the 2023 revenue and a fixed growth rate of 8%.

**Implementation Breakdown:**

import java.util.Scanner;

public class FinancialForecasting {

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

        if (years == 0) {

            return currentValue;

        }

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

    }

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

        double futureValue = currentValue;

        for (int i = 0; i < years; i++) {

            futureValue \*= (1 + growthRate);

        }

        return futureValue;

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        double currentValue = scanner.nextDouble();

        System.out.print("Enter annual growth rate (in decimal, e.g., 0.05 for 5%): ");

        double growthRate = scanner.nextDouble();

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

        int years = scanner.nextInt();

        double futureValueRecursive = predictFutureValue(currentValue, growthRate, years);

        System.out.println("\nPredicted future value using recursion: " + futureValueRecursive);

        double futureValueOptimized = predictFutureValueOptimized(currentValue, growthRate, years);

        System.out.println("Predicted future value using optimized approach: " + futureValueOptimized);

        System.out.println("\nTime Complexity:");

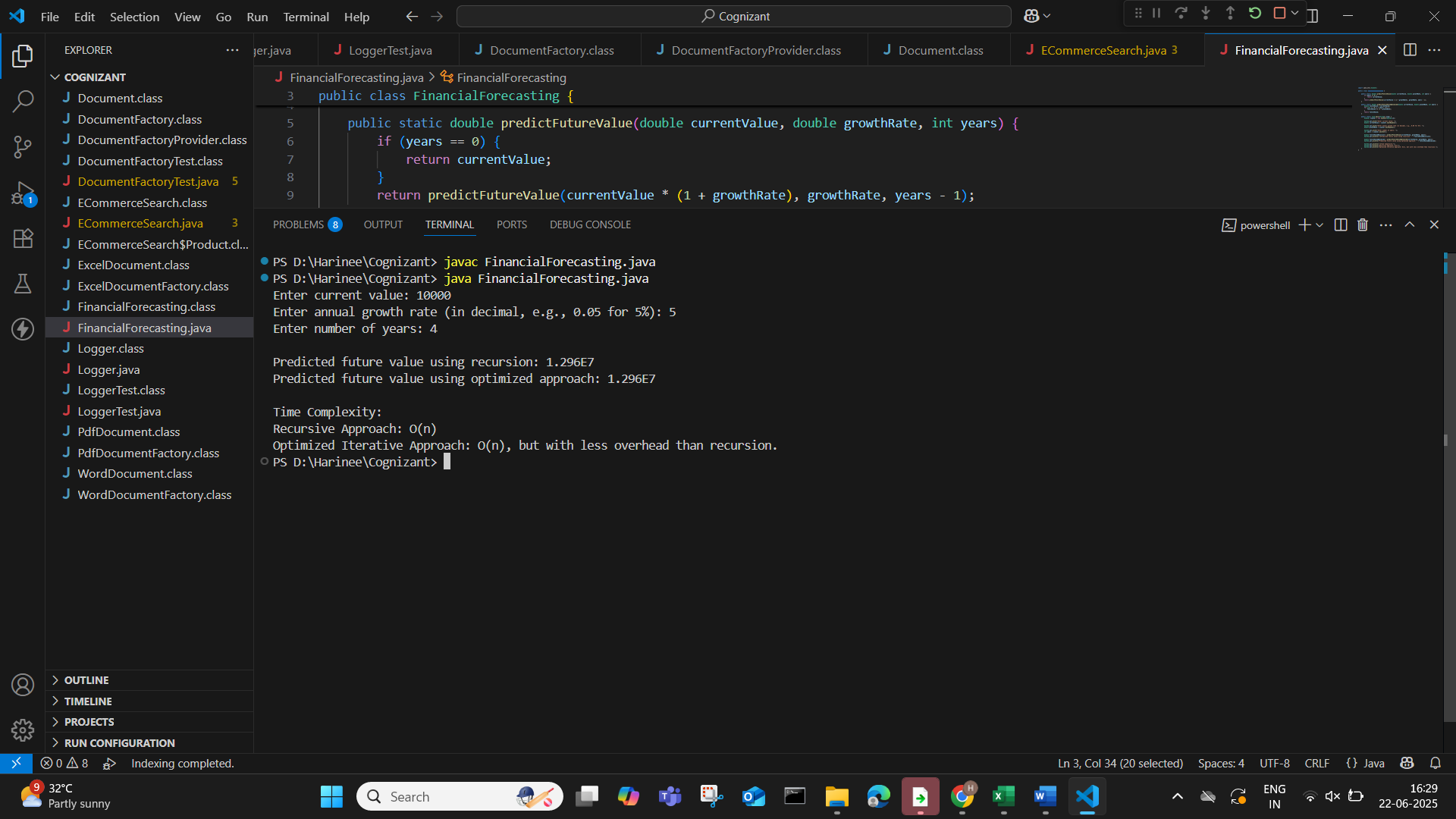
        System.out.println("Recursive Approach: O(n)");

        System.out.println("Optimized Iterative Approach: O(n), but with less overhead than recursion.");

    }

}

**Output:**



**Conclusion:**

The recursive function effectively models compound revenue growth over multiple years. This approach not only simplifies the logic but also helps visualize how revenue grows step-by-step annually. The program correctly forecasts the revenue for 2024 and 2025, reinforcing the use of recursion in practical scenarios like financial planning.