**ALGORITHMS AND DATA STRUCTURES**

**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 Class:**

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 productId + " - " + productName + " (" + category + ")";

    }

}

**Linear Search Class:**

public class LinearSearch {

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

        for (Product product : products) {

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

                return product;

            }

        }

        return null;

    }

}

**Binary Search Class:**

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

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

        // Binary search assumes the array is already sorted

        int left = 0;

        int 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];

            } else if (cmp < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

}

**Main Class:**

import java.util.Comparator;

import java.util.Arrays;

public class Main {

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Shoes", "Fashion"),

            new Product(102, "Laptop", "Electronics"),

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

            new Product(104, "Watch", "Accessories"),

            new Product(105, "Bag", "Fashion")

        };    // LINEAR SEARCH

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

        Product result1 = LinearSearch.search(products, "Mobile");

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

  // Sort for binary search

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

    // BINARY SEARCH

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

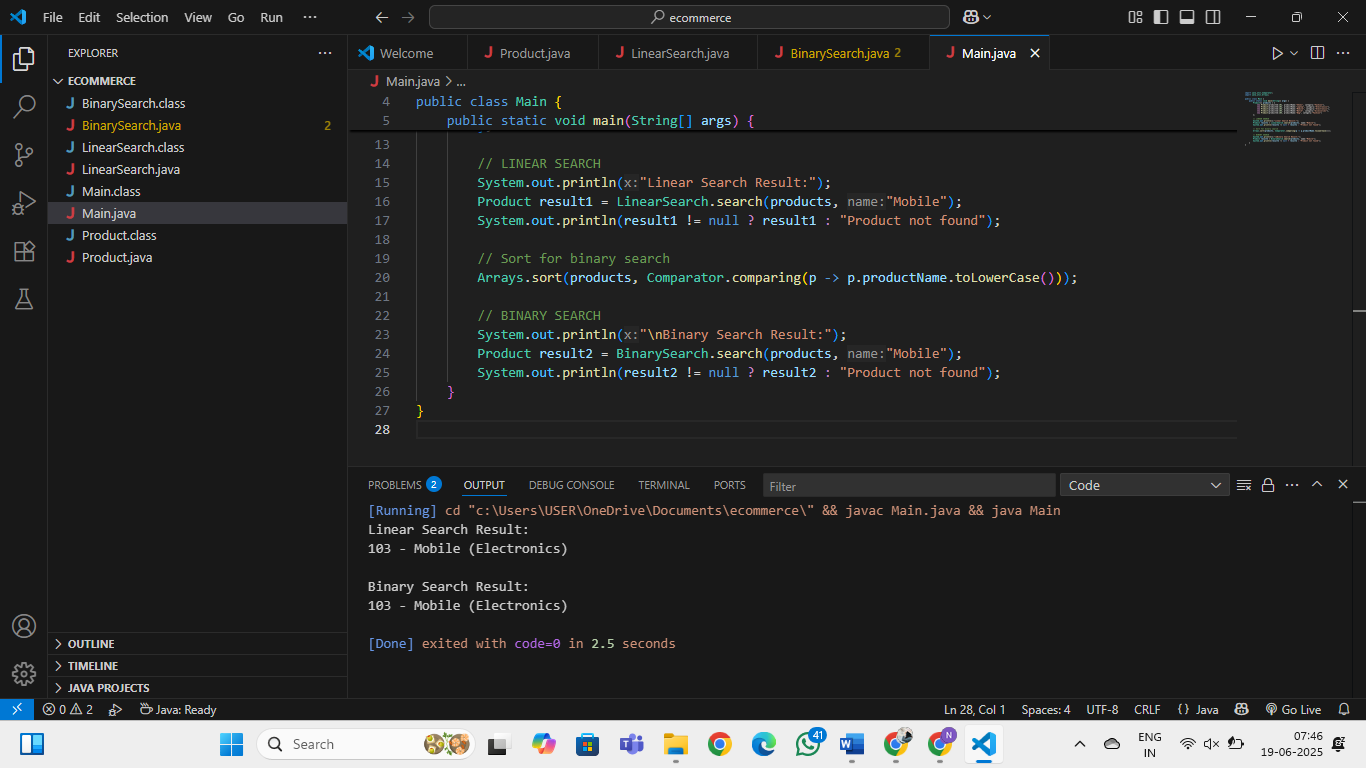
        Product result2 = BinarySearch.search(products, "Mobile");

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

  }

}

**OUTPUT:**



**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:**

**Forecast Class:**

public class Forecast {

    // Recursive method to calculate future value

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

        if (years == 0) {

            return presentValue;

        }

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

    }

    // Optimized version using memoization (optional)

    public static double calculateFutureValueMemo(double presentValue, double growthRate, int years, double[] memo) {

        if (years == 0) {

            return presentValue;

        }

        if (memo[years] != 0) {

            return memo[years];

        }

        memo[years] = calculateFutureValueMemo(presentValue, growthRate, years - 1, memo) \* (1 + growthRate);

        return memo[years];

    }

}

**Main Class:**

public class Main {

    public static void main(String[] args) {

        double presentValue = 1000.0;      // Starting investment

        double growthRate = 0.10;          // 10% growth

        int years = 5;

        double futureValue = Forecast.calculateFutureValue(presentValue, growthRate, years);

        System.out.printf("Future Value (recursive): %.2f\n", futureValue);

        double[] memo = new double[years + 1];

        double optimizedValue = Forecast.calculateFutureValueMemo(presentValue, growthRate, years, memo);

        System.out.printf("Future Value (memoized): %.2f\n", optimizedValue);

    }

}

**OUTPUT:**

