**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.

**Answer:**

**1) Explain Big O notation and how it helps in analyzing algorithms.**

Big O measures how the time or space complexity of an algorithm grows as the input size n increases. It helps determine the scalability and efficiency of your search algorithm.

**2) Describe the best, average, and worst-case scenarios for search operations.**

**3) Compare the time complexity of linear and binary search algorithms.**

Same answer for question 2 and question 3

| **Scenario** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| Best Case | O(1) – first item | O(1) – middle item |
| Average Case | O(n) | O(log n) |
| Worst Case | O(n) – last item | O(log n) |

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

If the product array is sorted and has less insertion and deletion of product binary search is suitable because it works with o(log n) time complexity.

If the product array is unsorted and there are more insertions and deletion of product then linear search is suited because for binary search we need to sort which will cause overhead of o(n logn).

**Code:**

Product.java:

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 getProductName() {

return productName;

}

@Override

public String toString() {

return "Product{" +

"productId=" + productId +

", productName='" + productName + '\'' +

", category='" + category + '\'' +

'}';

}

}

ProductSearch.java(Main Class):

public class ProductSearch {

public static void bubbleSort(Product[] products) {

int n = products.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (products[j].getProductName().compareToIgnoreCase(products[j + 1].getProductName()) > 0) {

Product temp = products[j];

products[j] = products[j + 1];

products[j + 1] = temp;

}

}

}

}

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

for (Product product : products) {

if (product.getProductName().equalsIgnoreCase(targetName)) {

return product;

}

}

return null;

}

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

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

while (low <= high) {

int mid = (low + high) / 2;

int comparison = products[mid].getProductName().compareToIgnoreCase(targetName);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(1, "Shoes", "Fashion"),

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

new Product(3, "Shampoo", "Personal Care"),

new Product(4, "Phone", "Electronics"),

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

};

String searchItem1 = "Shampoo";

Product resultLinear = *linearSearch*(products, searchItem1);

System.*out*.println("Linear Search Result for \"" + searchItem1 + "\": " + resultLinear);

*bubbleSort*(products);

String searchItem2 = "Phone";

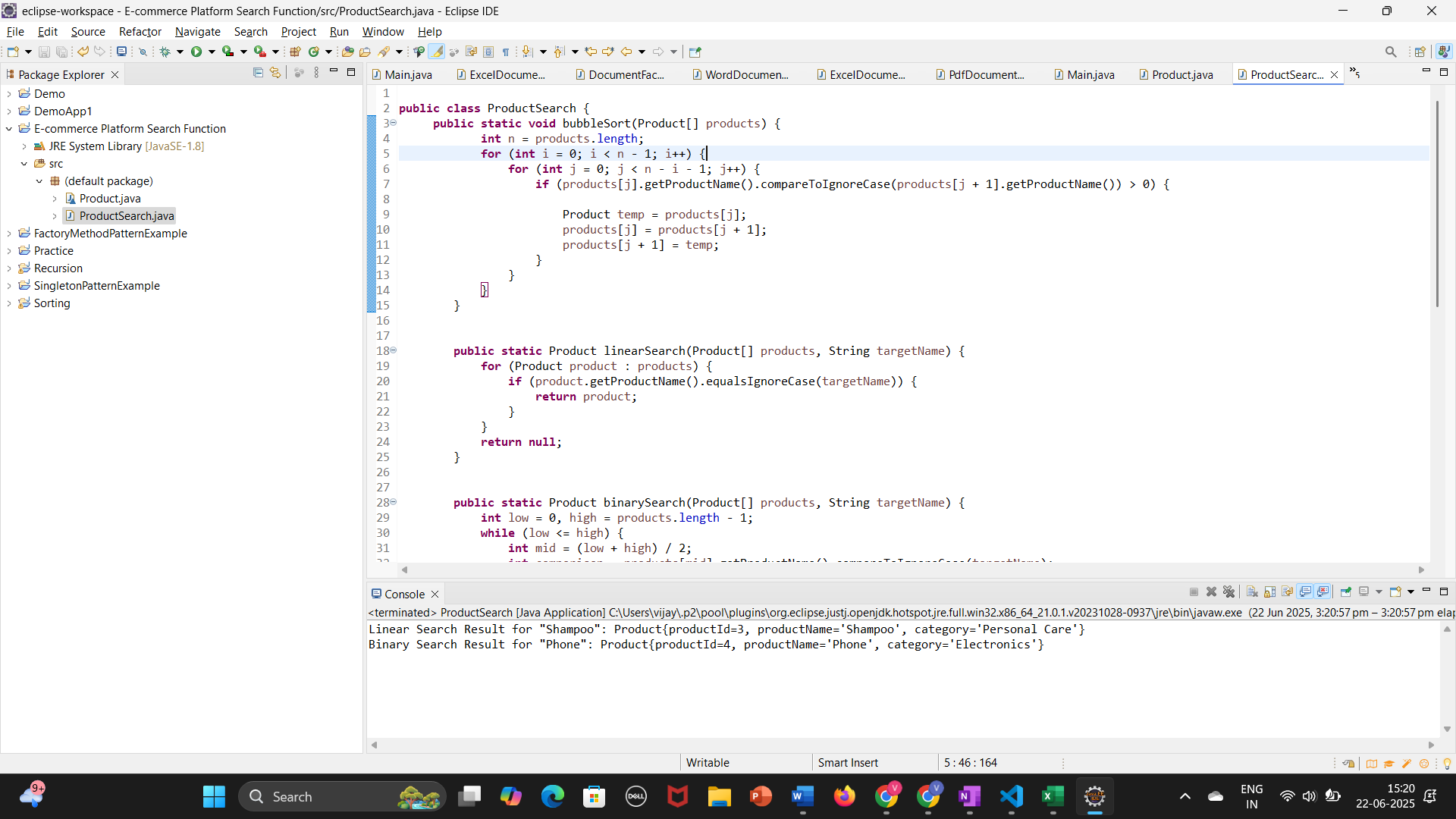
Product resultBinary = *binarySearch*(products, searchItem2);

System.*out*.println("Binary Search Result for \"" + searchItem2 + "\": " + resultBinary);

}

}

**Output:**



**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.

Answer:

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

Recursion is when a method calls itself to solve a problem. It's often used to break down complex problems into smaller, manageable sub-problems. Ideal for problems that are naturally repetitive or follow a pattern (like forecasting future values year by year).

1. **Discuss the time complexity of your recursive algorithm.**

The time complexity is O(n), where n is the number of years. Each recursive call decreases years by 1 until it reaches 0.

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

Recursive calls may result in stack overflow or redundant computations for large n. For simple compounding, recursion is not necessary — an iterative solution is more efficient.

**Code:**

Financial Forecast(Main class):

public class FinancialForecast {

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

if (years == 0) {

return presentValue;

}

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

}

public static void main(String[] args) {

double presentValue = 10000.0; // Initial investment

double annualGrowthRate = 0.08; // 8% annual growth

int years = 5;

double futureValue = calculateFutureValue(presentValue, annualGrowthRate, years);

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

}

}

**Output:**

