**Digital Nurture Deep Skilling**

Week 1 - Algorithms\_Data Structures

**RAJAKUMAR S**

**727822TUEC144**

**SUPERSET ID: 6373997**

**Exercise 2: E-commerce Platform Search Function**

import java.util.\*;

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 + "]";

}

}

public class EcommerceSearchExample {

public static Product linearSearch(Product[] items, int targetId) {

for (Product item : items) {

if (item.productId == targetId) {

return item;

}

}

return null;

}

public static Product binarySearch(Product[] items, int targetId) {

int left = 0;

int right = items.length - 1;

while (left <= right) {

int middle = left + (right - left) / 2;

if (items[middle].productId == targetId) {

return items[middle];

} else if (items[middle].productId < targetId) {

left = middle + 1;

} else {

right = middle - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(105, "Smartphone", "Electronics"),

new Product(101, "Running Shoes", "Footwear"),

new Product(109, "Backpack", "Accessories"),

new Product(102, "Bluetooth Speaker", "Electronics"),

new Product(108, "Wrist Watch", "Accessories")

};

Product[] sortedProducts = products.clone();

Arrays.sort(sortedProducts, (a, b) -> Integer.compare(a.productId, b.productId));

int targetId = 102;

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

Product result1 = linearSearch(products, targetId);

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

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

Product result2 = binarySearch(sortedProducts, targetId);

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

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

System.out.println("Linear Search: O(n) — checks each product one by one.");

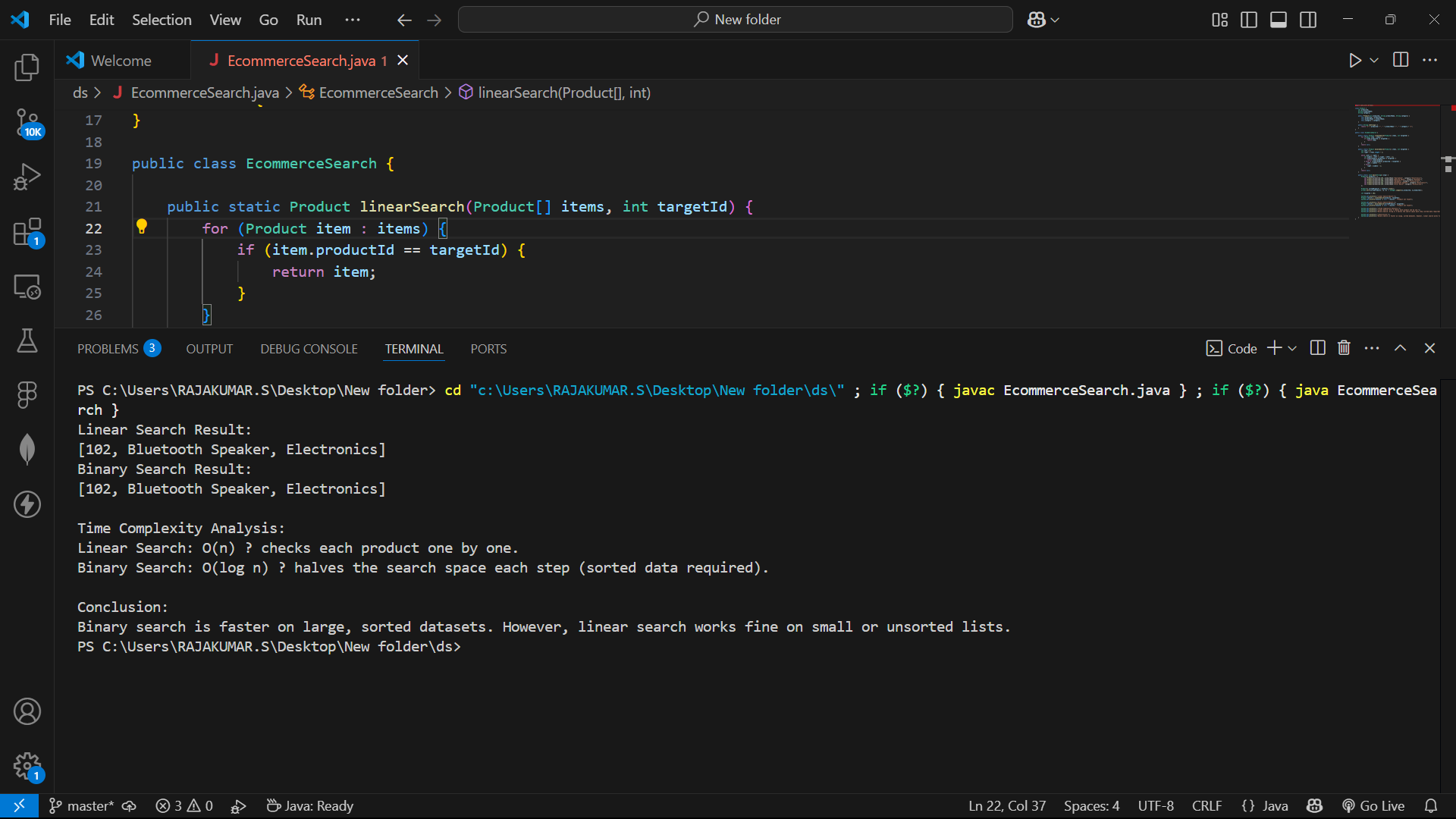
System.out.println("Binary Search: O(log n) — halves the search space each step (sorted data required).");

System.out.println("\nConclusion:");

System.out.println("Binary search is faster on large, sorted datasets. However, linear search works fine on small or unsorted lists.");

}

}



**Exercise 7: Financial Forecasting**

public class FinancialForecast {

public static double forecastValue(double currentAmount, double growthRate, int years) {

if (years == 0) {

return currentAmount;

}

return forecastValue(currentAmount \* (1 + growthRate), growthRate, years - 1);

}

public static double forecastMemoized(int years, double currentAmount, double growthRate, Double[] memo) {

if (years == 0) {

return currentAmount;

}

if (memo[years] != null) {

return memo[years];

}

memo[years] = forecastMemoized(years - 1, currentAmount, growthRate, memo) \* (1 + growthRate);

return memo[years];

}

public static void main(String[] args) {

double baseAmount = 10000.0;

double annualGrowth = 0.08;

int futurePeriod = 5;

System.out.println("Recursive Forecasting (Simple):");

double predicted1 = forecastValue(baseAmount, annualGrowth, futurePeriod);

System.out.printf("Estimated value after %d years: %.2f%n", futurePeriod, predicted1);

System.out.println("\nRecursive Forecasting (Memoized):");

Double[] memoTable = new Double[futurePeriod + 1];

double predicted2 = forecastMemoized(futurePeriod, baseAmount, annualGrowth, memoTable);

System.out.printf("Estimated value after %d years: %.2f%n", futurePeriod, predicted2);

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

System.out.println("Basic recursion: O(n) — each call waits for the next.");

System.out.println("With memoization: O(n) — avoids recalculating the same subproblems.");

System.out.println("\nOptimization Note:");

System.out.println("For very large time periods, memoization or iterative methods reduce function stack depth and boost performance.");

}

}

