***Exercise 1: 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.**

import java.util.Arrays;

public class ECommerceSearchDemo {

public static void main(String[] args) {

Product[] products = {

new Product(1001, "Laptop", "Electronics"),

new Product(1002, "Chair", "Furniture"),

new Product(1003, "Pen", "Stationery"),

new Product(1004, "Keyboard", "Electronics"),

new Product(1005, "Notebook", "Stationery")

};

System.out.println("=== LINEAR SEARCH ===");

Product result1 = SearchAlgorithms.linearSearch(products, "Notebook");

printResult(result1);

Arrays.sort(products); // Sort for binary search

System.out.println("\n=== BINARY SEARCH ===");

Product result2 = SearchAlgorithms.binarySearch(products, "Notebook");

printResult(result2);

System.out.println("\n=== ANALYSIS ===");

System.out.println("Linear Search: O(n)");

System.out.println("Binary Search: O(log n) — applicable only on sorted data");

System.out.println("Binary Search is faster on large sorted lists.");

}

public static void printResult(Product p) {

if (p != null)

System.out.println("Found: " + p);

else

System.out.println("Product not found.");

}

}

class Product implements Comparable<Product> {

private int productId;

private String productName;

private 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 int compareTo(Product other) {

return this.productName.compareToIgnoreCase(other.productName);

}

@Override

public String toString() {

return "[" + productId + "] " + productName + " (" + category + ")";

}

}

class SearchAlgorithms {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

public static Product binarySearch(Product[] sortedProducts, String name) {

int left = 0, right = sortedProducts.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int cmp = sortedProducts[mid].getProductName().compareToIgnoreCase(name);

if (cmp == 0) return sortedProducts[mid];

else if (cmp < 0) left = mid + 1;

else right = mid - 1;

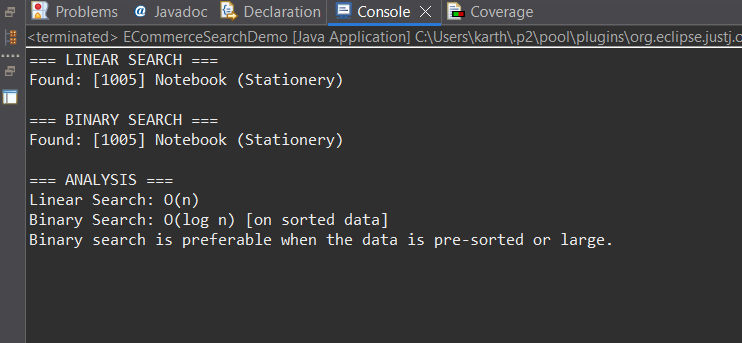
}

return null;

}

}

***Output:***

******

***Exercise 2: Financial Forecasting***

**Scenario:**

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

public class FinancialForecastDemo {

public static void main(String[] args) {

FinancialForecaster forecaster = new FinancialForecaster();

double initial = 10000;

double rate = 6.5;

int years = 5;

double basic = forecaster.forecast(initial, rate, years);

System.out.println("Recursive Forecast (No Memo): " + basic);

Double[] cache = new Double[years + 1];

double memoized = forecaster.forecastMemo(initial, rate, years, cache);

System.out.println("Recursive Forecast (Memoized): " + memoized);

}

}

class FinancialForecaster {

public double forecast(double currentValue, double annualRate, int years) {

if (years == 0) return currentValue;

double nextValue = currentValue \* (1 + annualRate / 100);

return forecast(nextValue, annualRate, years - 1);

}

public double forecastMemo(double currentValue, double annualRate, int years, Double[] cache) {

if (years == 0) return currentValue;

if (cache[years] != null) return cache[years];

double nextValue = currentValue \* (1 + annualRate / 100);

cache[years] = forecastMemo(nextValue, annualRate, years - 1, cache);

return cache[years];

}

}

***Output:***

