Exercise 1: E-commerce Platform Search Function

import java.util.Arrays;

import java.util.Comparator;

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;

}

}

static Product linearSearch(Product[] products, int targetId) {

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

static Product binarySearch(Product[] products, int targetId) {

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

while (left <= right) {

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

if (products[mid].productId == targetId) {

return products[mid];

} else if (products[mid].productId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(3, "Keyboard", "Electronics"),

new Product(1, "Laptop", "Electronics"),

new Product(4, "Chair", "Furniture"),

new Product(2, "Book", "Stationery")

};

System.out.println("=== Asymptotic Notation (Big O) ===");

System.out.println("Linear Search: O(n) - Best: O(1), Worst: O(n)");

System.out.println("Binary Search: O(log n) - Best: O(1), Worst: O(log n)");

System.out.println();

System.out.println("=== Linear Search ===");

Product result1 = linearSearch(products, 4);

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

Arrays.sort(products, Comparator.comparingInt(p -> p.productId));

System.out.println("=== Binary Search ===");

Product result2 = binarySearch(products, 4);

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

System.out.println();

System.out.println("=== Analysis ===");

System.out.println("Linear search is simple but inefficient for large datasets.");

System.out.println("Binary search is much faster, but the data must be sorted.");

System.out.println("For large sorted datasets, binary search is more suitable.");

}

}

Exercise 2: Financial Forecasting

public class FinancialForecast {

static void explainRecursion() {

System.out.println("=== Understanding Recursion ===");

System.out.println("Recursion is a method where the solution to a problem depends on smaller instances of the same problem.");

System.out.println("Example: FutureValue(n) = FutureValue(n-1) \* (1 + growthRate)");

System.out.println();

}

static double futureValueRecursive(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

}

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

}

static double futureValueMemo(double presentValue, double growthRate, int years, Double[] memo) {

if (years == 0) return presentValue;

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

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

return memo[years];

}

public static void main(String[] args) {

explainRecursion();

double presentValue = 10000.0;

double growthRate = 0.10;

int years = 5;

System.out.println("=== Forecast Using Recursive Approach ===");

double forecastRecursive = futureValueRecursive(presentValue, growthRate, years);

System.out.printf("Predicted Value after %d years (Recursive): ₹%.2f%n", years, forecastRecursive);

System.out.println();

System.out.println("=== Forecast Using Optimized Recursive (Memoization) ===");

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

double forecastMemo = futureValueMemo(presentValue, growthRate, years, memo);

System.out.printf("Predicted Value after %d years (Memoized): ₹%.2f%n", years, forecastMemo);

System.out.println();

System.out.println("=== Time Complexity Analysis ===");

System.out.println("Recursive: O(n) calls, each with O(1) computation → Total O(n)");

System.out.println("Memoized: O(n) calls with saved results → Still O(n), but avoids recomputation");

System.out.println("In cases with overlapping subproblems, memoization greatly improves efficiency.");

}

}