**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.*

// EcommerceSearch.java

import java.util.Arrays;

public class EcommerceSearch {

// Step 2: Product class

static class Product implements Comparable<Product> {

int productId;

String productName;

String category;

public Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

@Override

public int compareTo(Product other) {

return Integer.compare(this.productId, other.productId);

}

@Override

public String toString() {

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

}

}

// Step 3: Linear Search

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

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

// Step 3: Binary Search (array must be sorted by productId)

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

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

while (left <= right) {

int mid = (left + right) / 2;

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

return products[mid];

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

left = mid + 1;

else

right = mid - 1;

}

return null;

}

// Step 1: Big O Explanation (Console based)

static void explainBigO() {

System.out.println("Big O notation expresses algorithm complexity:");

System.out.println("- Linear Search: O(n) - searches each element one by one.");

System.out.println("- Binary Search: O(log n) - divides the array in half each time.");

System.out.println("Best Case: Target found at beginning/middle.");

System.out.println("Average/Worst Case: Dependent on data position.");

}

// Step 4: Comparison

static void compareSearches() {

System.out.println("Linear Search: Slower for large unsorted datasets.");

System.out.println("Binary Search: Faster, but requires sorted data.");

}

public static void main(String[] args) {

explainBigO();

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Shoes", "Footwear"),

new Product(103, "Watch", "Accessories"),

new Product(104, "Phone", "Electronics"),

new Product(105, "Book", "Education")

};

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

Product result1 = linearSearch(products, 104);

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

Arrays.sort(products); // For binary search

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

Product result2 = binarySearch(products, 104);

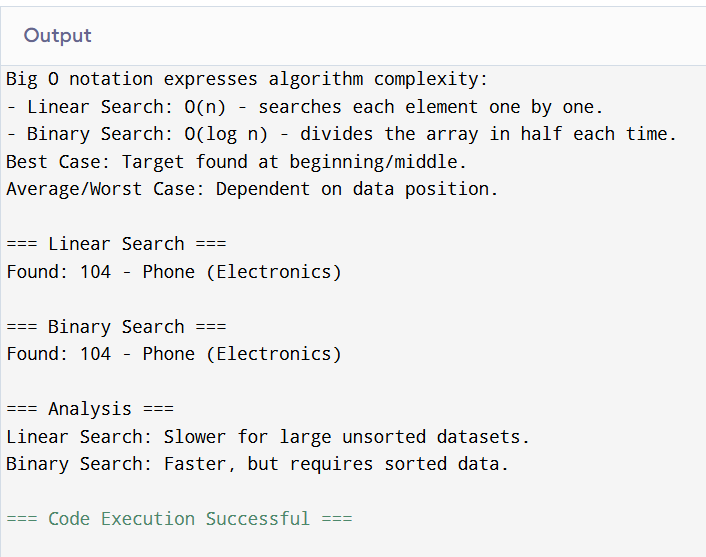
System.out.println(result2 != null ? "Found: " + result2 : "Product not found");

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

compareSearches();

}

}



**Exercise 7: Financial Forecasting**

**Scenario:**

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

*// FinancialForecast.java*

*public class FinancialForecast {*

*// Step 1: Recursion Explanation*

*static void explainRecursion() {*

*System.out.println("Recursion is when a method calls itself to solve sub-problems.");*

*System.out.println("Useful for problems like Fibonacci, tree traversal, forecasting, etc.");*

*}*

*// Step 2-3: Recursive method to predict future value*

*// Formula: futureValue = currentValue \* (1 + growthRate) ^ years*

*static double forecastRecursive(double currentValue, double growthRate, int years) {*

*if (years == 0) return currentValue;*

*return forecastRecursive(currentValue, growthRate, years - 1) \* (1 + growthRate);*

*}*

*// Step 4: Time complexity*

*static void analyzeForecast() {*

*System.out.println("Recursive Forecast Time Complexity: O(n)");*

*System.out.println("Each year involves one recursive call.");*

*System.out.println("Optimizing with iteration/memoization can reduce function overhead.");*

*}*

*public static void main(String[] args) {*

*explainRecursion();*

*double currentValue = 10000;*

*double growthRate = 0.1; // 10%*

*int years = 5;*

*System.out.println("\n=== Forecasting ===");*

*double futureValue = forecastRecursive(currentValue, growthRate, years);*

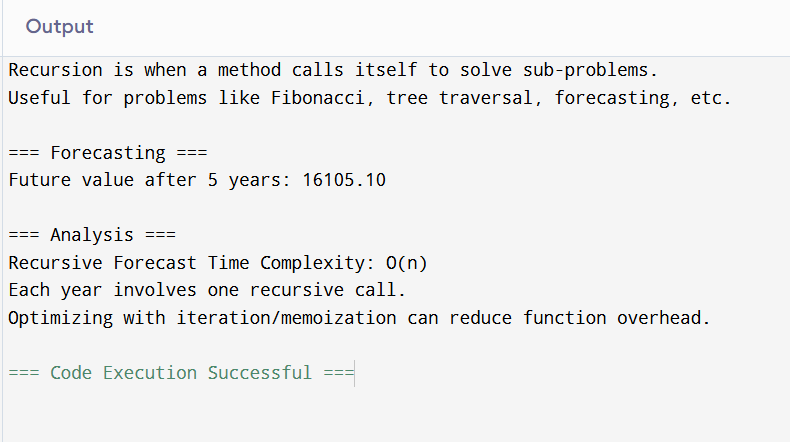
*System.out.printf("Future value after %d years: %.2f\n", years, futureValue);*

*System.out.println("\n=== Analysis ===");*

*analyzeForecast();*

*}*

*}*

**