**Algorithms\_Data Structures:**

**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.

**Code Snippet:**

import java.util.Arrays;

class Item {

int id;

String name;

String type;

public Item(int id, String name, String type) {

this.id = id;

this.name = name;

this.type = type;

}

@Override

public String toString() {

return "ID: " + id + " | Name: " + name + " | Category: " + type;

}

}

// Search manager class that handles product storage and lookup

class CatalogSearch {

Item[] items;

Item[] sortedItems;

int count = 0;

public CatalogSearch() {

items = new Item[100]; // Let's keep it simple

sortedItems = new Item[100];

}

// Add a product to the catalog

public void addItem(Item product) {

items[count] = product;

count++;

System.arraycopy(items, 0, sortedItems, 0, count);

Arrays.sort(sortedItems, 0, count, (a, b) -> a.id - b.id); // sort by ID

}

// Basic linear search

public Item linearLookup(int searchId) {

int checks = 0;

for (int i = 0; i < count; i++) {

checks++;

if (items[i].id == searchId) {

System.out.println("Linear Search took " + checks + " checks");

return items[i];

}

}

System.out.println("Linear Search took " + checks + " checks (not found)");

return null;

}

//Binary Search

public Item binaryLookup(int searchId) {

int checks = 0;

int left = 0, right = count - 1;

while (left <= right) {

checks++;

int mid = (left + right) / 2;

if (sortedItems[mid].id == searchId) {

System.out.println("Binary Search took " + checks + " checks");

return sortedItems[mid];

}

if (sortedItems[mid].id < searchId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

System.out.println("Binary Search took " + checks + " checks (not found)");

return null;

}

public void showItems() {

System.out.println("\nCatalog (" + count + " items):");

for (int i = 0; i < count; i++) {

System.out.println((i + 1) + ". " + items[i]);

}

}

}

public class EcommerceSearchExample {

public static void main(String[] args) {

CatalogSearch searchTool = new CatalogSearch();

searchTool.addItem(new Item(105, "Smartphone", "Electronics"));

searchTool.addItem(new Item(102, "Laptop", "Electronics"));

searchTool.addItem(new Item(108, "Headphones", "Electronics"));

searchTool.addItem(new Item(101, "T-Shirt", "Clothing"));

searchTool.addItem(new Item(107, "Jeans", "Clothing"));

// Display current catalog

searchTool.showItems();

System.out.println("\n SEARCH DEMO ");

// First test - should be found easily

System.out.println("\n1. Searching for ID 105:");

Item found1 = searchTool.linearLookup(105);

System.out.println("Found (Linear): " + found1);

Item found2 = searchTool.binaryLookup(105);

System.out.println("Found (Binary): " + found2);

// Now test a missing ID

System.out.println("\n2. Searching for ID 999 (non-existent):");

searchTool.linearLookup(999);

searchTool.binaryLookup(999);

// Compare performance a bit

System.out.println("\n PERFORMANCE SNAPSHOT ");

int n = searchTool.count;

int binaryWorst = (int)Math.ceil(Math.log(n) / Math.log(2));

System.out.println("Catalog size: " + n + " items");

System.out.println("\nLinear Search:");

System.out.println("- Best case: 1 check");

System.out.println("- Worst case: " + n + " checks");

System.out.println("\nBinary Search:");

System.out.println("- Best case: 1 check");

System.out.println("- Worst case: " + binaryWorst + " checks");

System.out.println("\n QUICK THOUGHTS ");

System.out.println("Binary is faster when IDs are sorted");

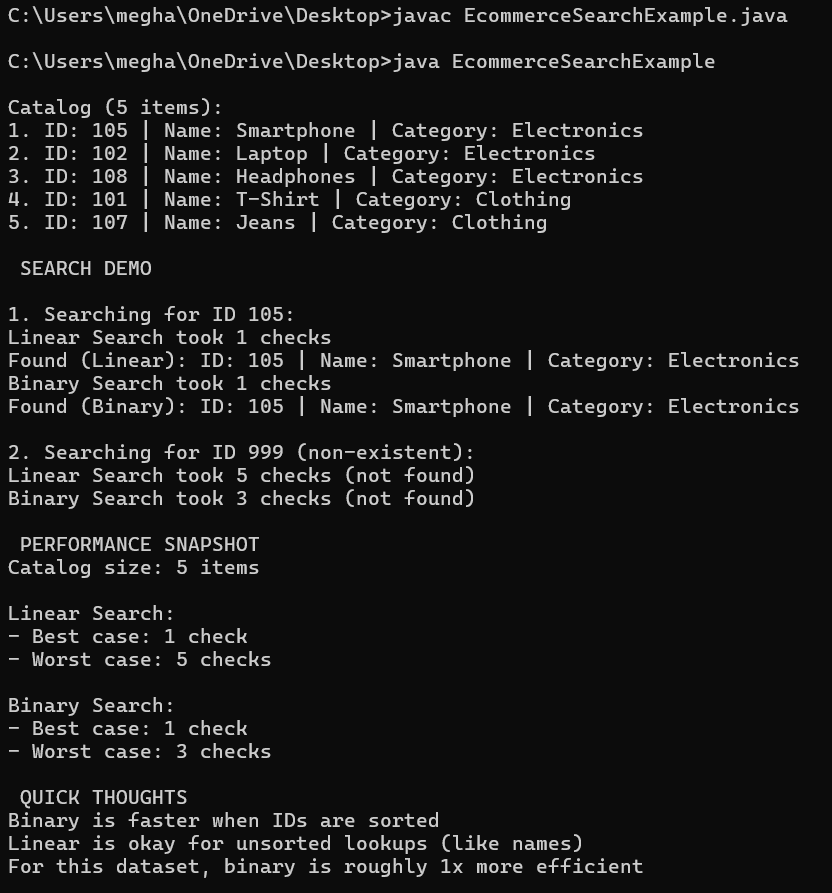
System.out.println("Linear is okay for unsorted lookups (like names)");

System.out.println("For this dataset, binary is roughly " + (n / binaryWorst) + "x more efficient");

}

}

**Output:**



**Exercise 7: 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.

**Code Snippet:**

public class FinancialForecast {

public static double estimateFutureValue(double baseAmount, double annualRate, int remainingYears) {

if (remainingYears == 0) {

return baseAmount; }

return estimateFutureValue(baseAmount, annualRate, remainingYears - 1) \* (1 + annualRate);

}

public static double estimateWithMemo(double baseAmount, double annualRate, int years, Double[] cache) {

if (years == 0) {

return baseAmount;

}

if (cache[years] != null) {

return cache[years];

}

cache[years] = estimateWithMemo(baseAmount, annualRate, years - 1, cache) \* (1 + annualRate);

return cache[years];

}

public static void main(String[] args) {

double startingValue = 1000.0; // starting cash (e.g., investment)

double interestRate = 0.07; // annual growth – 7%

int durationYears = 10; // we're forecasting 10 years into the future

System.out.println("Financial Forecasting Tool ");

System.out.println("Starting Investment: $" + startingValue);

System.out.println("Expected Growth: " + (interestRate \* 100) + "% annually");

System.out.println("Time Horizon: " + durationYears + " years\n");

double futureValueBasic = estimateFutureValue(startingValue, interestRate, durationYears);

System.out.printf("Basic Recursive Estimate: $%.2f after %d years\n", futureValueBasic, durationYears);

Double[] memoTable = new Double[durationYears + 1]; // +1 beacuse include year zero

double futureValueFast = estimateWithMemo(startingValue, interestRate, durationYears, memoTable);

System.out.printf("Memoized Estimate: $%.2f after %d years\n", futureValueFast, durationYears);

}

}

**Output:**

