**1. 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 :-**

import java.util.\*;

public class ECommerceSearch {

static class Product {

int productId;

String productName;

String category;

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

this.productId = id;

this.productName = name;

this.category = category;

}

@Override

public String toString() {

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

}

}

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

for (Product p : products) {

if (p.productId == targetId) {

return p;

}

}

return null;

}

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

return products[mid];

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

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(101, "Shirt", "Apparel"),

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

new Product(102, "Phone", "Electronics"),

new Product(103, "Shoes", "Footwear")

};

// Linear Search (unsorted)

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

Product result1 = linearSearch(products, 105);

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

// Binary Search (sorted by productId)

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

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

Product result2 = binarySearch(products, 105);

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

}

}

**Output :-**

**A black rectangular object with white text

AI-generated content may be incorrect.**

**2.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 :-**

public class FinancialForecast {

public static double futureValue(double initialValue, double growthRate, int years) {

if (years == 0) {

return initialValue;

}

return futureValue(initialValue, growthRate, years - 1) \* (1 + growthRate);

}

public static void main(String[] args) {

double initialInvestment = 10000;

double annualGrowthRate = 0.08; // 8%

int forecastYears = 5;

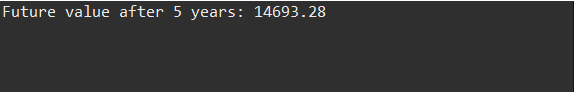
double result = futureValue(initialInvestment, annualGrowthRate, forecastYears);

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

}

}

**Output :-**

****