Exercise 2: E-commerce Platform Search Function

**Step 2: Setup — Define the Product Class**

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;

}

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

**Step 3: Implement Linear and Binary Search**

import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearch {

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

for (Product p : products) {

if (p.productName.equalsIgnoreCase(targetName))

return p;

}

}

return null;

}

public static Product binarySearch(Product[] products, String targetName) {

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

while (left <= right) {

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

int cmp = targetName.compareToIgnoreCase(products[mid].productName);

if (cmp == 0)

return products[mid];

else if (cmp < 0)

right = mid - 1;

else

left = mid + 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shirt", "Clothing"),

new Product(103, "Phone", "Electronics"),

new Product(104, "Shoes", "Footwear"),

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

};

// Linear Search Test

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

Product foundLinear = linearSearch(products, "Phone");

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

// Binary Search requires sorted array by productName

Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

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

Product foundBinary = binarySearch(products, "Phone");

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

}

}

**Step 4: Analysis**

**Time Complexity:**

* **Linear Search**:
  + Best: O(1)
  + Worst: O(n)
  + Simple but inefficient on large datasets.
* **Binary Search**:
  + Best: O(1)
  + Worst: O(log n)
  + Efficient, but requires sorted data.

Exercise 7: Financial Forecasting

**Step 2: Setup Method Signature**

public static double forecastValue(int year, double currentValue, double growthRate)

**Step 3: Recursive Implementation**

public class FinancialForecast {

public static double forecastValue(int years, double currentValue, double growthRate) {

if (years == 0) {

return currentValue;

}

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

}

public static void main(String[] args) {

double startingValue = 1000.0; // Initial amount

double annualGrowthRate = 0.07; // 7% annual growth

int years = 5;

double futureValue = forecastValue(years, startingValue, annualGrowthRate);

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

}

}