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.

ANSWER

```
import java.util.Arrays;
import java.util.Comparator;
public class ProductSearchTest {
// Product class with attributes
static class Product {
int productId;
String productName;
String category;
public Product(int id, String name, String category) {
this.productId = id;
this.productName = name;
this.category = category;
}
public String toString() {
return productId + " - " + productName + " (" + category + ")";
}
}
// Linear Search
public static int linearSearch(Product[] products, String name) {
for (int i = 0; i < products.length; i++) {
if (products[i].productName.equalsIgnoreCase(name)) {
return i;
}
}
return -1;
```

```
}
// Binary Search (requires sorted array)
public static int binarySearch(Product[] products, String name) {
int left = 0, right = products.length - 1;
while (left <= right) {int mid = (left + right) / 2;
int cmp = name.compareToIgnoreCase(products[mid].productName);
if (cmp == 0)
return mid;
else if (cmp < 0)
right = mid - 1;
else
left = mid + 1;
}
return -1;
}
// Main method to test searches
public static void main(String[] args) {
Product[] products = {
new Product(101, "Shoes", "Footwear"),
new Product(102, "Laptop", "Electronics"),
new Product(103, "Watch", "Accessories"),
new Product(104, "Phone", "Electronics")
};
System.out.println(" Linear Search for 'Laptop':");
int linearIndex = linearSearch(products, "Laptop");
if (linearIndex != -1)
System.out.println("Found at index: " + linearIndex + " → " +
products[linearIndex]);
else
System.out.println("Not found.");
// Sort before binary search
Arrays.sort(products, Comparator.comparing(p ->
p.productName.toLowerCase()));
System.out.println("\n Binary Search for 'Laptop':");
int binaryIndex = binarySearch(products, "Laptop");
if (binaryIndex != -1)
System.out.println("Found at index: " + binaryIndex + " → " +
products[binaryIndex]);
else
System.out.println("Not found.");
}
}
```

```
[Running] cd "c:\Users\KIIT\Desktop\DotNetFSE\Design Principles and Patterns\.dist\"
ProductSearchTest
 Linear Search for 'Laptop':
Found at index: 1 ? 102 - Laptop (Electronics)

 Binary Search for 'Laptop':
Found at index: 0 ? 102 - Laptop (Electronics)

[Done] exited with code=0 in 0.897 seconds
```

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.

ANSWER

```
public class FinancialForecastTest {
// Recursive method to forecast future value
public static double forecastRecursive(double amount, double rate, int years)
{
if (years == 0)
return amount;
return forecastRecursive(amount * (1 + rate), rate, years - 1);
}
// Main method for testing
public static void main(String[] args) {
double initialAmount = 1000.0;
double annualRate = 0.10; // 10%
int years = 5;
// Using recursive method
double recursiveForecast = forecastRecursive(initialAmount, annualRate,
years);
System.out.printf("Recursive Forecast after %d years: Rs%.2f\n", years,
```

```
recursiveForecast);
}
}
```

```
[Running] cd "c:\Users\KIIT\Desktop\DotNetFSE\Design Principles and Patterns\" && javac Financia
FinancialForecastTest
Recursive Forecast after 5 years: Rs1610.51

[Done] exited with code=0 in 0.832 seconds
```