**Week1- 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:**

o Explain Big O notation and how it helps in analyzing algorithms.

o Describe the best, average, and worst-case scenarios for search operations.

**2. Setup:**

o Create a class Product with attributes for searching, such as productId, productName, and category.

**3. Implementation:**

o Implement linear search and binary search algorithms.

o Store products in an array for linear search and a sorted array for binary search.

**4. Analysis:**

o Compare the time complexity of linear and binary search algorithms.

o Discuss which algorithm is more suitable for your platform and why.

**Product.java**

package ECommerce;

public 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 "[" + productId +"," + productName + "," + "category" +"]";

}

}

**linearSearch.java**

package ECommerce;

public class linearSearch {

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

for (Product p : products) {

if (p.productName.equalsIgnoreCase(targetName)) {

return p;

}

}

return null;

}

}

**BinearySearch.java**

package ECommerce;

import java.util.Arrays;

import java.util.Comparator;

public class BinearySearch {

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

// Sort by productName

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

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

while (left <= right) {

int mid = (left + right) / 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;

}

}

**Main.java**

package ECommerce;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Shoes", "Footwear"),

new Product(2, "Phone", "Electronics"),

new Product(3, "T-shirt", "Clothing"),

new Product(4, "Laptop", "Electronics")

};

// Test linear search

Product found1 = linearSearch.*search*(products, "Phone");

System.***out***.println(found1 != null ? "Linear: " + found1.productName : "Not found");

// Test binary search

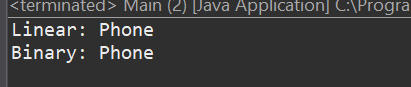
Product found2 = BinearySearch.*search*(products, "Phone");

System.***out***.println(found2 != null ? "Binary: " + found2.productName : "Not found");

}

}

**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:**

o Explain the concept of recursion and how it can simplify certain problems.

**2. Setup:**

o Create a method to calculate the future value using a recursive approach.

**3. Implementation:**

o Implement a recursive algorithm to predict future values based on past growth rates.

**4. Analysis:**

o Discuss the time complexity of your recursive algorithm.

o Explain how to optimize the recursive solution to avoid excessive computation.

**Forecast.java**

package FinancialForecasting;

public class Forecast {

public static double futureValue(double amount, double rate, int years) {

if (years == 0)

return amount;

return *futureValue*(amount \* (1 + rate), rate, years - 1);

}

public static void main(String[] args) {

double initialAmount = 2000;

double growthRate = 0.05;

int years = 5;

double result = *futureValue*(initialAmount, growthRate, years);

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

}

}

**Output:**

