**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.

**Source code:**

import java.util.Arrays;

import java.util.Comparator;

class Product {

int productId;

String productName;

String category;

Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;

}}

public class Main {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

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

if (cmp == 0) return products[mid];

else if (cmp < 0) left = mid + 1;

else right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Notebook", "Stationery"),

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

new Product(104, "Pen", "Stationery"),

new Product(105, "Tablet", "Electronics")

};

String searchItem = "Smartphone";

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

Product result1 = linearSearch(products, searchItem);

if (result1 != null)

System.out.println(result1);

else

System.out.println("Product not found.");

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

Product result2 = binarySearch(products, searchItem);

if (result2 != null)

System.out.println(result2);

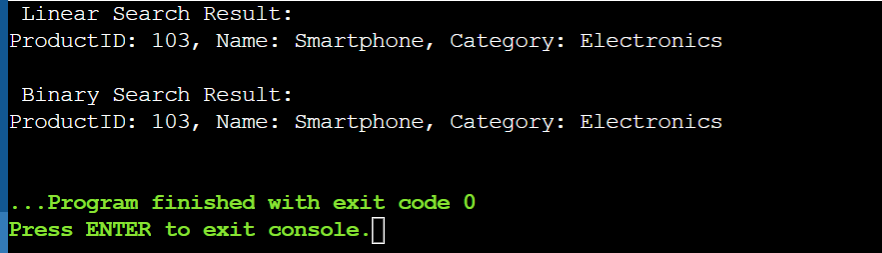
else

System.out.println("Product not found.");

}

}

**Output:**

****

**Linear search Time Complexity**:O(n) [sort is not required but not suitable for large data]

**Binary seach Time Complexity:**O(log n) [sort is required and suitable for large data]

**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.

**Source code:**

public class Main {

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

if (years == 0) {

return initialValue;

}

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

}

public static void main(String[] args) {

double initialValue = 10000;

double growthRate = 0.08;

int years = 5;

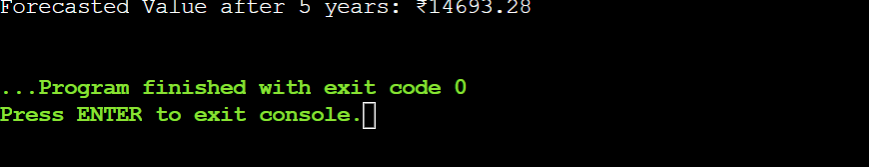
double finalAmount = forecastValue(initialValue, growthRate, years);

System.out.printf("Forecasted Value after %d years: ₹%.2f\n", years, finalAmount);

}

}

**Output:**



**Time Complexity (Recursive)**: Since this function calls itself once per year, it's time complexity is **O(n)** where *n* is the number of years.