# Week1\_DSA4\_HandsOn

## 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.

Java Code:

import java.util.Arrays;  
  
public class ECommerceSearch {  
  
 static 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 + " - " + productName + " (" + category + ")";  
 }  
 }  
  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(1, "Laptop", "Electronics"),  
 new Product(2, "Shoes", "Fashion"),  
 new Product(3, "Keyboard", "Electronics"),  
 new Product(4, "Book", "Education")  
 };  
  
 System.out.println("Linear Search: Looking for 'Shoes'");  
 Product result1 = LinearSearch.search(products, "Shoes");  
 if (result1 != null) {  
 System.out.println("Found: " + result1);  
 } else {  
 System.out.println("Not found");  
 }  
  
 System.out.println("Binary Search: Looking for 'Book'");  
 Product result2 = BinarySearch.search(products, "Book");  
 if (result2 != null) {  
 System.out.println("Found: " + result2);  
 } else {  
 System.out.println("Not found");  
 }  
 }  
}  
  
class LinearSearch {  
 public static ECommerceSearch.Product search(ECommerceSearch.Product[] products, String name) {  
 for (int i = 0; i < products.length; i++) {  
 String currentName = products[i].productName.toLowerCase();  
 String targetName = name.toLowerCase();  
 if (currentName.equals(targetName)) {  
 return products[i];  
 }  
 }  
 return null;  
 }  
}  
  
class BinarySearch {  
 public static ECommerceSearch.Product search(ECommerceSearch.Product[] products, String name) {  
 Arrays.sort(products, (a, b) -> a.productName.toLowerCase().compareTo(b.productName.toLowerCase()));  
 int left = 0, right = products.length - 1;  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 String midName = products[mid].productName.toLowerCase();  
 String targetName = name.toLowerCase();  
 if (midName.equals(targetName)) {  
 return products[mid];  
 } else if (midName.compareTo(targetName) < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 return null;  
 }  
}

## 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.

Java Code:

public class FinancialForecast {  
  
 public static double calculateFutureValue(double presentValue, double growthRate, int years) {  
 if (years == 0) {  
 return presentValue;  
 } else {  
 return calculateFutureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);  
 }  
 }  
  
 public static void main(String[] args) {  
 double presentValue = 10000.0;  
 double growthRate = 0.05;  
 int years = 5;  
  
 double futureValue = calculateFutureValue(presentValue, growthRate, years);  
 System.out.println("Future value after " + years + " years: ₹" + futureValue);  
 }  
}