**Digital Nurture 4.0 – Week 1**

**2.Data structures and Algorithms**

**Exercise 2:** E-commerce Platform Search Function

**Program:**

The program defines a Product class with productId, productName, and category.Two search methods are implemented: linearSearch (checks each item) and binarySearch (divides the list).  
Products are stored in an array, and the array is sorted by name before applying binary search.The main method tests both search types by searching for a product named "Phone".The result is displayed if found, showing how each search method works on the product list.

**1.Setup: 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;

}

@Override

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

}

**2.Implementation: 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 product : products) {

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

return product;

}

}

return null;

}

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

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

new Product(103, "Book", "Stationery"),

new Product(104, "Phone", "Electronics"),

new Product(105, "Pen", "Stationery")

};

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

System.out.println("Linear Search for 'Phone':");

Product result1 = linearSearch(products, "Phone");

System.out.println(result1 != null ? result1 : "Not Found");

System.out.println("\nBinary Search for 'Phone':");

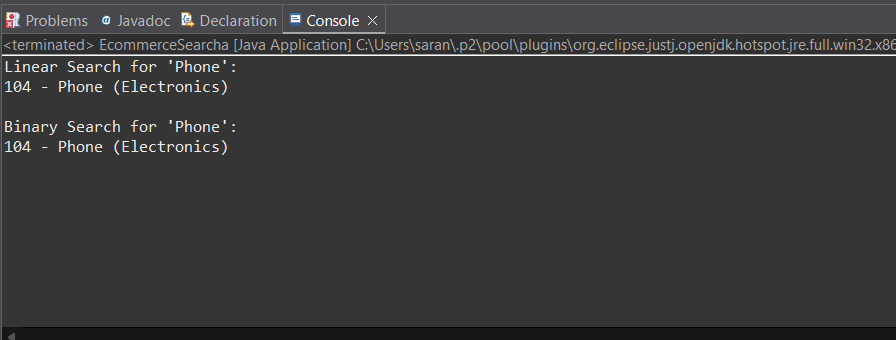
Product result2 = binarySearch(products, "Phone");

System.out.println(result2 != null ? result2 : "Not Found");

}

}

**Output:**



**Analysis: Time Complexity**

| **Feature** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Time Complexity** | O(n) | O(log n) |
| **Best Case** | O(1) | O(1) |
| **Worst Case** | O(n) | O(log n) |
| **Data Requirement** | Unsorted | Sorted |
| **Performance** | Slower for large data | Faster and more efficient |

**Discussion:**

Linear search is simple and works on **unsorted data**, but it's slow for large datasets.Binary search is faster with **O(log n)** time but needs the data to be **sorted**.While linear search checks each item one by one, binary search divides the search range.For an e-commerce platform with large product lists, binary search is more efficient.  
Therefore, sorting the product list upfront and using binary search improves search performance.

**Exercise 7:** Financial Forecasting

**Program:**

I created a recursive method to calculate future value based on initial amount, growth rate, and years.The basic futureValue method multiplies previous year's value with (1 + growthRate) recursively.To optimize it, I added futureValueMemo which uses an array to store and reuse results (memoization).The main method tests both versions and prints the forecasted value after 5 years.

**FinancialForecasting.java**

public class FinancialForecast {

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

if (years == 0) {

return initialValue;

}

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

}

public static double futureValueMemo(double initialValue, double growthRate, int years, double[] memo) {

if (years == 0) {

return initialValue;

}

if (memo[years] != 0) {

return memo[years];

}

memo[years] = futureValueMemo(initialValue, growthRate, years - 1, memo) \* (1 + growthRate);

return memo[years];

}

public static void main(String[] args) {

double initialValue = 1000.0;

double growthRate = 0.10; // 10% growth rate

int years = 5;

double result = futureValue(initialValue, growthRate, years);

System.out.println("Future Value (Recursion): $" + result);

double[] memo = new double[years + 1];

double resultMemo = futureValueMemo(initialValue, growthRate, years, memo);

System.out.println("Future Value (Memoized): $" + resultMemo);

}

}

**Output:**

