**CTS DIGITAL NURTURE - 4.0 JAVA FSE**

**WEEK 1 – ALGORITHMS\_DATA STRUCTURES**

MANDATORY HANDS-ON 1:

QUESTION:

**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.

1. **What is Big O Notation?**

* Big O notation describes the upper bound of an algorithm’s running time as the input size increases.
* It helps to estimate how fast or slow an algorithm performs independently of the hardware.

|  |  |
| --- | --- |
| **O(1)** | Constant time (fastest) |
| **O(log n)** | Logarithmic (e.g., binary search) |
| **O(n)** | Linear time (e.g., linear search) |
| **O(n log n)** | Log-linear (e.g., merge sort) |
| **O(n²)** | Quadratic (e.g., nested loops) |

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

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Linear Search** | O(1) – item is first | O(n/2) ≈ O(n) | O(n) – item is last/not found |
| **Binary Search** | O(1) – middle match | O(log n) | O(log n) – keep halving |

PROGRAM:

**Product.java**

package eCommerceSearchFunction;

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;

}

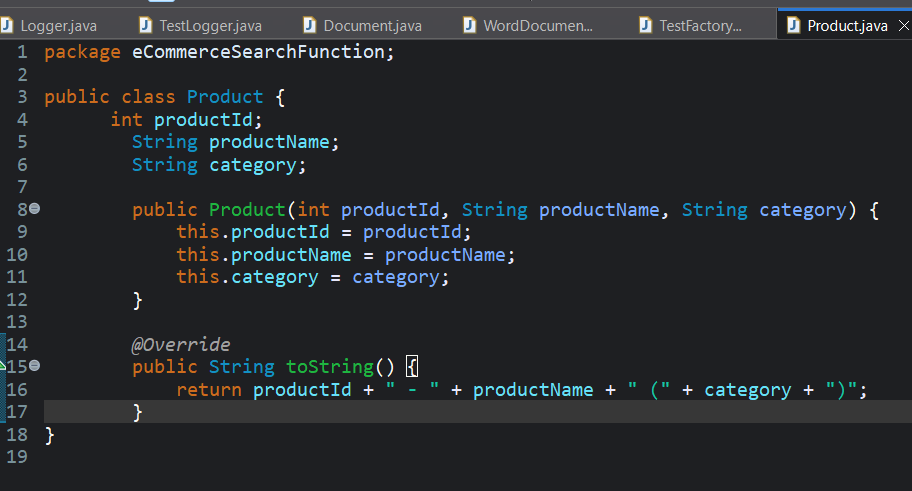
*@Override*

public String toString() {

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

}

}



**LinearSearch.java**

package eCommerceSearchFunction;

public class LinearSearch {

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

for (Product p : products) {

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

return p;

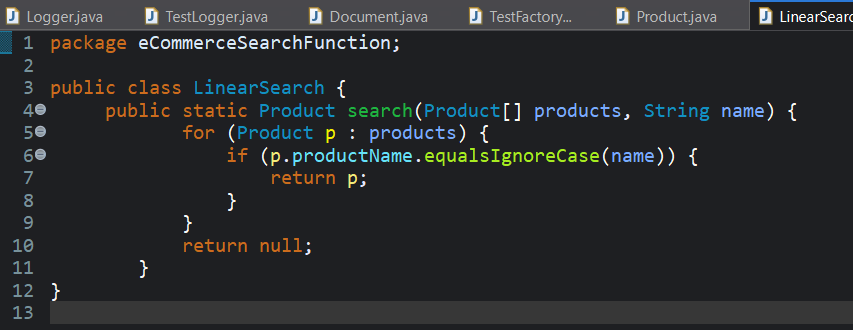
}

}

return null;

}

}

****

**BinarySearch.java**

package eCommerceSearchFunction;

public class BinarySearch {

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

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

while (left <= right) {

int mid = left + (right - left) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(name);

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

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

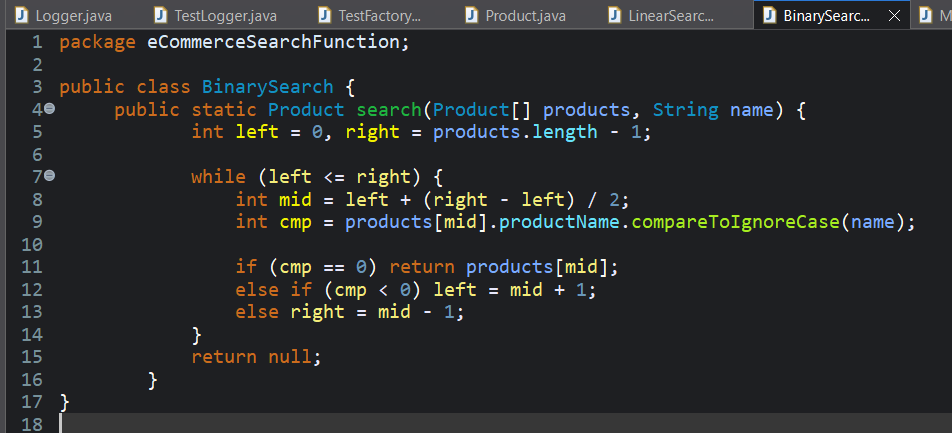
else right = mid - 1;

}

return null;

}

}



**MainApp.java**

package eCommerceSearchFunction;

import java.util.\*;

public class MainApp {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Shoes", "Footwear"),

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

new Product(103, "Watch", "Accessories"),

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

new Product(105, "Bag", "Accessories")

};

Product result1 = LinearSearch.*search*(products, "Watch");

System.***out***.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));

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

Product result2 = BinarySearch.*search*(products, "Watch");

System.***out***.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));

System.***out***.println("\n--- Search Algorithm Analysis ---");

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

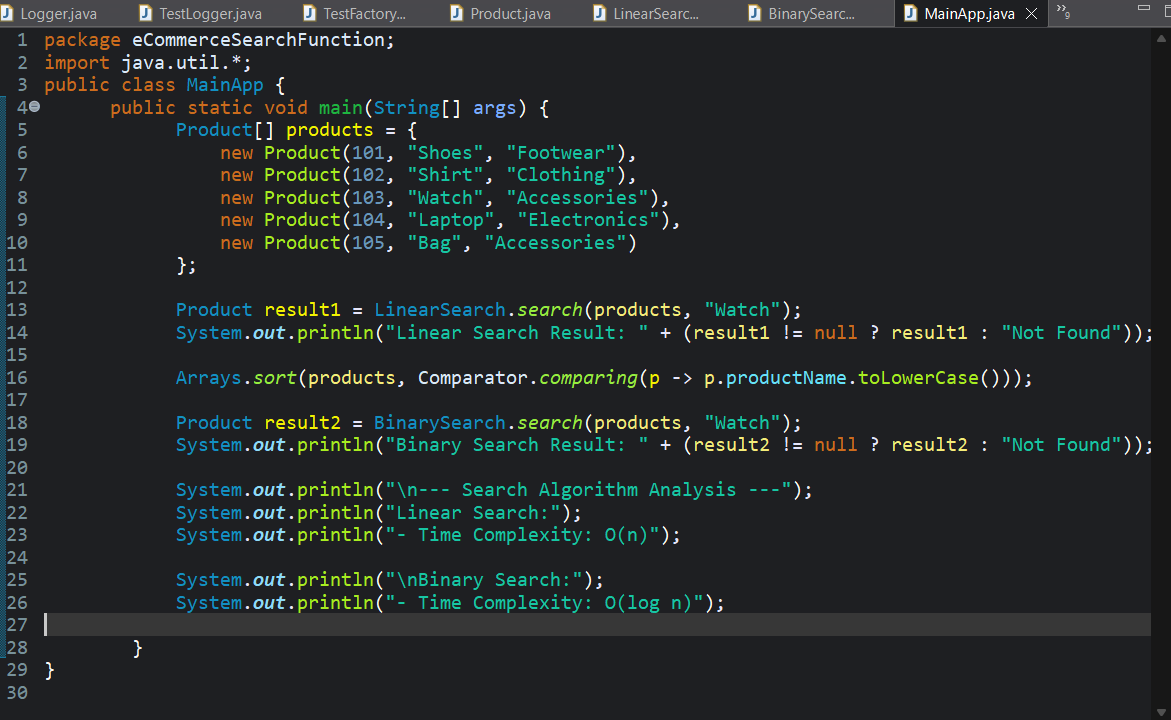
System.***out***.println("- Time Complexity: O(n)");

System.***out***.println("\nBinary Search:");

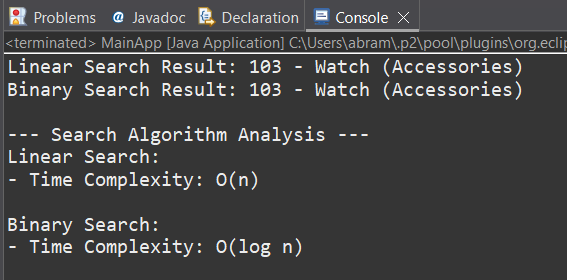
System.***out***.println("- Time Complexity: O(log n)");

}

}



OUTPUT:



**4. Compare the time complexity of linear and binary search algorithms.**

Linear Search has a time complexity of O(n). This means that in the worst case, the algorithm may have to check every product in the list before finding a match. It works well for small or unsorted datasets, but becomes inefficient as the number of products increases.

Binary Search has a time complexity of O(log n). It is much faster because it repeatedly divides the search space in half. However, it requires the product list to be sorted by product name or ID before searching.

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

Binary Search is more suitable.  
It offers faster performance due to its logarithmic time complexity and is ideal for searching through sorted product data.

However, if the product list is small or changes very frequently which makes sorting difficult in real-time, Linear Search can be used as a temporary or fallback approach.