**Data Structures And Algorithm**

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

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

**Big O Notation:**

* Big O notation describes **how fast or slow an algorithm grows** with respect to input size (n).
* It helps in evaluating **efficiency** (time or space).

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

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Linear** | O(1) – First item | O(n/2) ≈ O(n) | O(n) – Last or not found |
| **Binary** | O(1) – Middle item | O(log n) | O(log n) |

Binary Search only works on **sorted data**  
Linear Search works on **unsorted or sorted**

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

| **Search Type** | **Time Complexity** | **Use Case** |
| --- | --- | --- |
| Linear Search | O(n) | Small or unsorted data |
| Binary Search | O(log n) | Large sorted datasets (better choice) |

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

**Binary Search** is more suitable for **e-commerce platforms**:

* Fast lookup in large product databases.
* Products can be **indexed or sorted once**, then searched repeatedly.

Use **Linear Search** only when:

* Dataset is small,
* Or not sorted (and sorting overhead is not worth it).

**PROGRAM:  
product.java**

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;

}

}

**SearchUtils.java**

import java.util.Arrays;

import java.util.Comparator;

public class SearchUtils {

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;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

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

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

else high = mid - 1;

}

return null;

}

public static void sortByName(Product[] products) {

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

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shoes", "Fashion"),

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

new Product(104, "T-Shirt", "Clothing"),

};

SearchUtils.sortByName(products);

Product found1 = SearchUtils.linearSearch(products, "Shoes");

Product found2 = SearchUtils.binarySearch(products, "Laptop");

if (found1 != null) {

System.out.println(found1.productName + " found (Linear) - ID: " + found1.productId + ", Category: " + found1.category);

} else {

System.out.println("Not Found (Linear)");

}

if (found2 != null) {

System.out.println(found2.productName + " found (Binary) - ID: " + found2.productId + ", Category: " + found2.category);

} else {

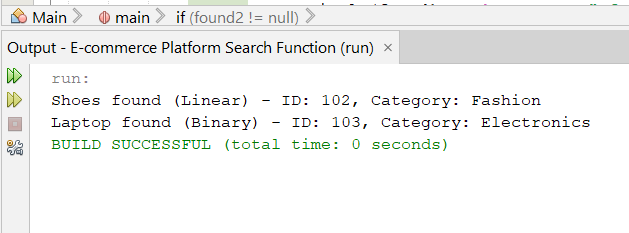
System.out.println("Not Found (Binary)");

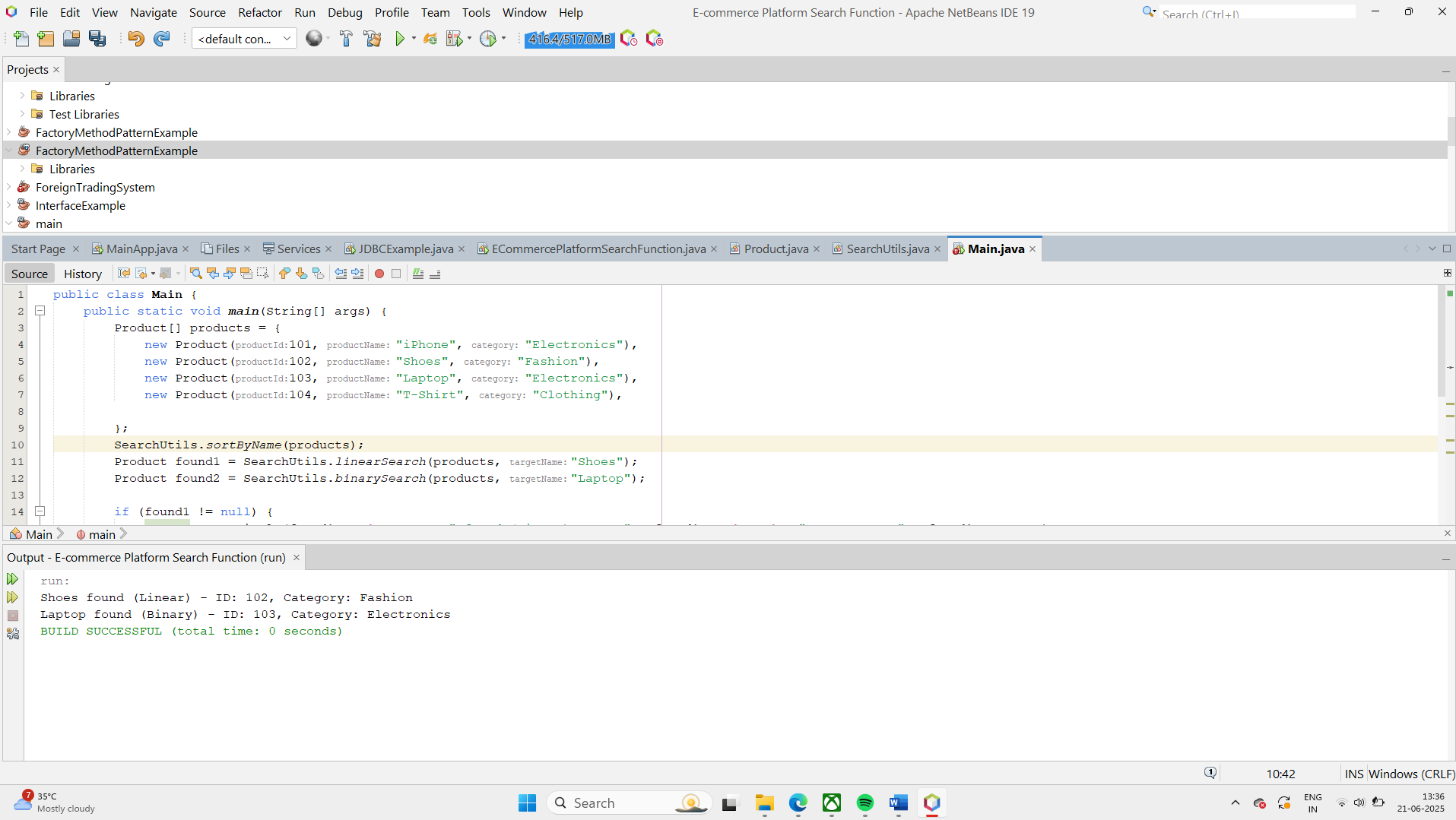
}

}

}

**OUTPUT:**

****

****