**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.

**What is Big O?**

Big O notation is used to analyze how the performance of an algorithm changes with the size of the input. It helps evaluate time and space complexity.

**CODE SECTION:**  
  
**Product.java**  
public class Product {

private int productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int getProductId() { return productId; }

public String getProductName() { return productName; }

public String getCategory() { return category; }

}  
  
**SearchUtility.java**import java.util.Arrays;

import java.util.Comparator;

public class SearchUtility {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

Arrays.sort(products, Comparator.comparing(Product::getProductName));

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

while (left <= right) {

int mid = (left + right) / 2;

int compare = products[mid].getProductName().compareToIgnoreCase(targetName);

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

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

else right = mid - 1;

}

return null;

}

}  
**SearchTest.java**  
import java.util.Arrays;

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shoes", "Fashion"),

new Product(3, "Smartphone", "Electronics"),

new Product(4, "Book", "Education"),

new Product(5, "Watch", "Accessories")

};

String searchName = "Smartphone";

long startLinear = System.nanoTime();

Product resultLinear = SearchUtility.linearSearch(products, searchName);

long endLinear = System.nanoTime();

Product[] productsForBinary = Arrays.copyOf(products, products.length);

long startBinary = System.nanoTime();

Product resultBinary = SearchUtility.binarySearch(productsForBinary, searchName);

long endBinary = System.nanoTime();

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

System.out.println("Time Taken (Linear Search): " + (endLinear - startLinear) + " ns");

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

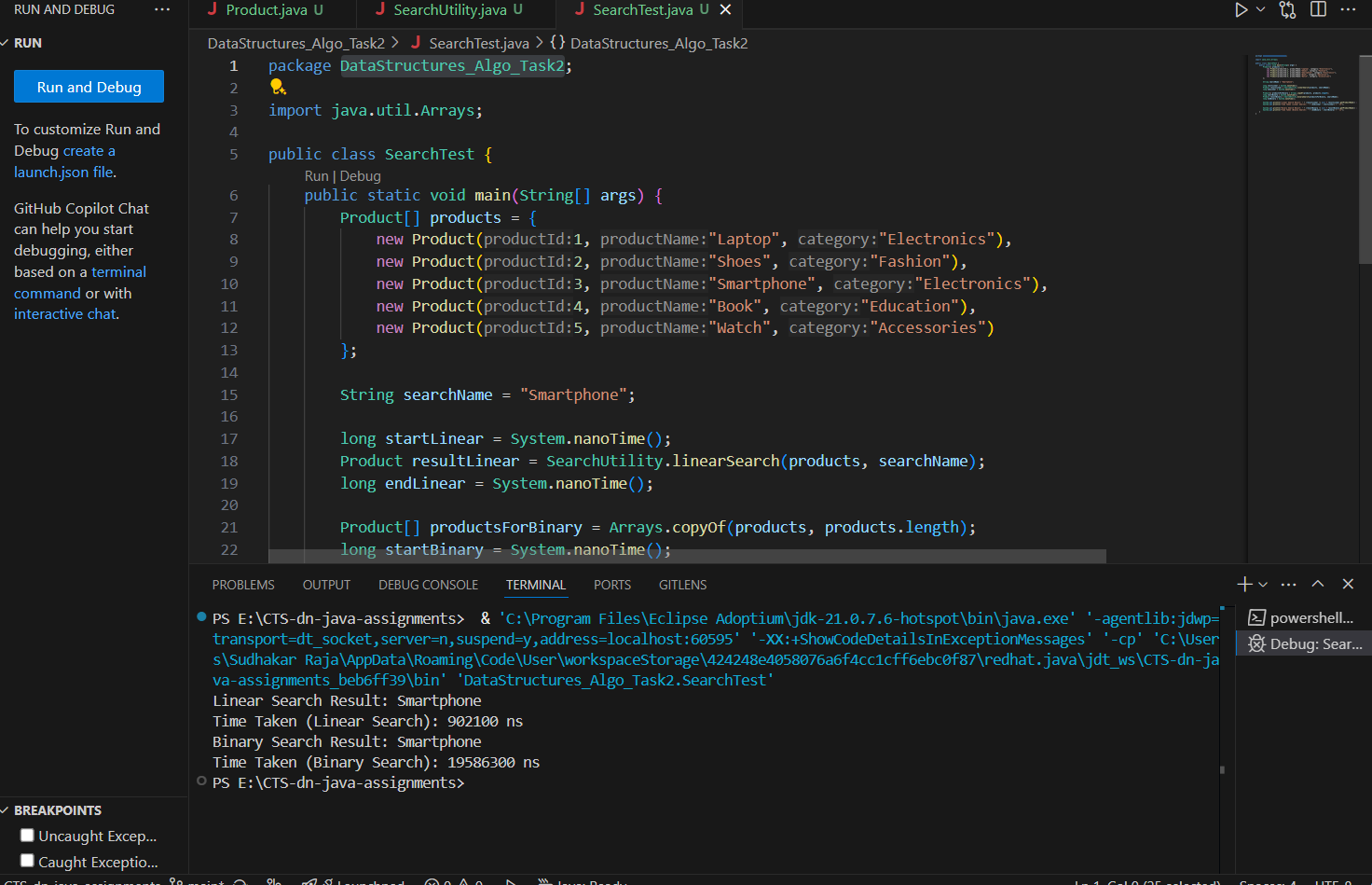
System.out.println("Time Taken (Binary Search): " + (endBinary - startBinary) + " ns");

}

}  
  
**Analysis**

**🔹 Time Complexity:**

* Linear Search: O(n)
* Binary Search: O(log n), but sorting takes O(n log n)

**OUTPUT SCREENSHOT:**  
  
  
**Conclusion:**

For a real-world e-commerce platform with large datasets, **binary search** is faster, but it requires sorted data. If the product list changes frequently or is small, **linear search** might be simpler and still effective.