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

**QUESTIONS:**

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

**Answer -** Big O notation is a mathematical concept used to describe the performance or complexity of an algorithm in terms of time or space (memory) as the input size grows. It helps analyze how efficient an algorithm is, especially when dealing with large inputs. It allows you to compare different algorithms based on scalability.

**How It Helps in Analysis:**

* Best-case, average-case, and worst-case performance can be described using Big O.
* It gives a language-independent way to evaluate efficiency.
* Helps in choosing the most optimal algorithm for large inputs.

**Common Time Complexities**

|  |  |  |
| --- | --- | --- |
| **Big O** | **Name** | **Example** |
| O(1) | |  | | --- | |  |   Constant time | Accessing an array element |
| O(log n) | Logarithmic time | Binary search |
| O(n) | |  | | --- | |  |   Linear time | Linear Search |
| O(n log n) | Log-linear time | Merge sort, Quick sort (avg) |
| O(n2) | Quadratic time | Nested loops |
| O(2n) | Exponential time | Recursive codes |

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

**Answer -** When analyzing an algorithm like searching, you consider different scenarios depending on the data and input:

**Linear Search (example):** Scans each element in a list until it finds the target.

|  |  |  |
| --- | --- | --- |
| **Case** | **Description** | **Time Complexity** |
| Best Case | Target is at the beginning | O(1) |
| Average Case | Target is in the middle | O(n) |
| Worst Case | Target is at the end or absent | O(n) |

**Binary Search (on sorted data):** Repeatedly divides the search space in half.

|  |  |  |
| --- | --- | --- |
| **Case** | **Description** | **Time complexity** |
| Best Case | Target is in the middle | O(1) |
| Average Case | Target is somewhere in the list | O(log n) |
| Worst Case | Target is at end or absent | O(log n) |

**SCENARIO:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**CODE:**

*Product.java –*

package com.example.search;

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 "Product [productId=" + productId + ", productName=" + productName + ", category=" + category + "]";

}

}

*Search.java –*

package com.example.search;

import java.util.Arrays;

import java.util.Comparator;

public class Search {

//Linear Search

public static Product linearSearch(Product[] product, String name) {

for(Product p: product) {

if(p.productName.equalsIgnoreCase(name)) return p;

}

return null;

}

//Sort the Array for Binary search implementation

//We used custom sort

public static void customSort(Product[] products) {

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

}

//Binary Search

public static Product binarySearch(Product[] product, String name) {

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

while(left <= right) {

int mid = (left + right) / 2;

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

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

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

else right = mid - 1;

}

return null;

}

}

*Main.java -*

package com.example.search;

public class Main {

public static void main(String[] args) {

Product[] products = {

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

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

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

new Product(104, "Book", "Education")

};

// Linear Search

Product result1 = Search.*linearSearch*(products, "Mouse");

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

// Binary Search (sort first)

Search.*customSort*(products);

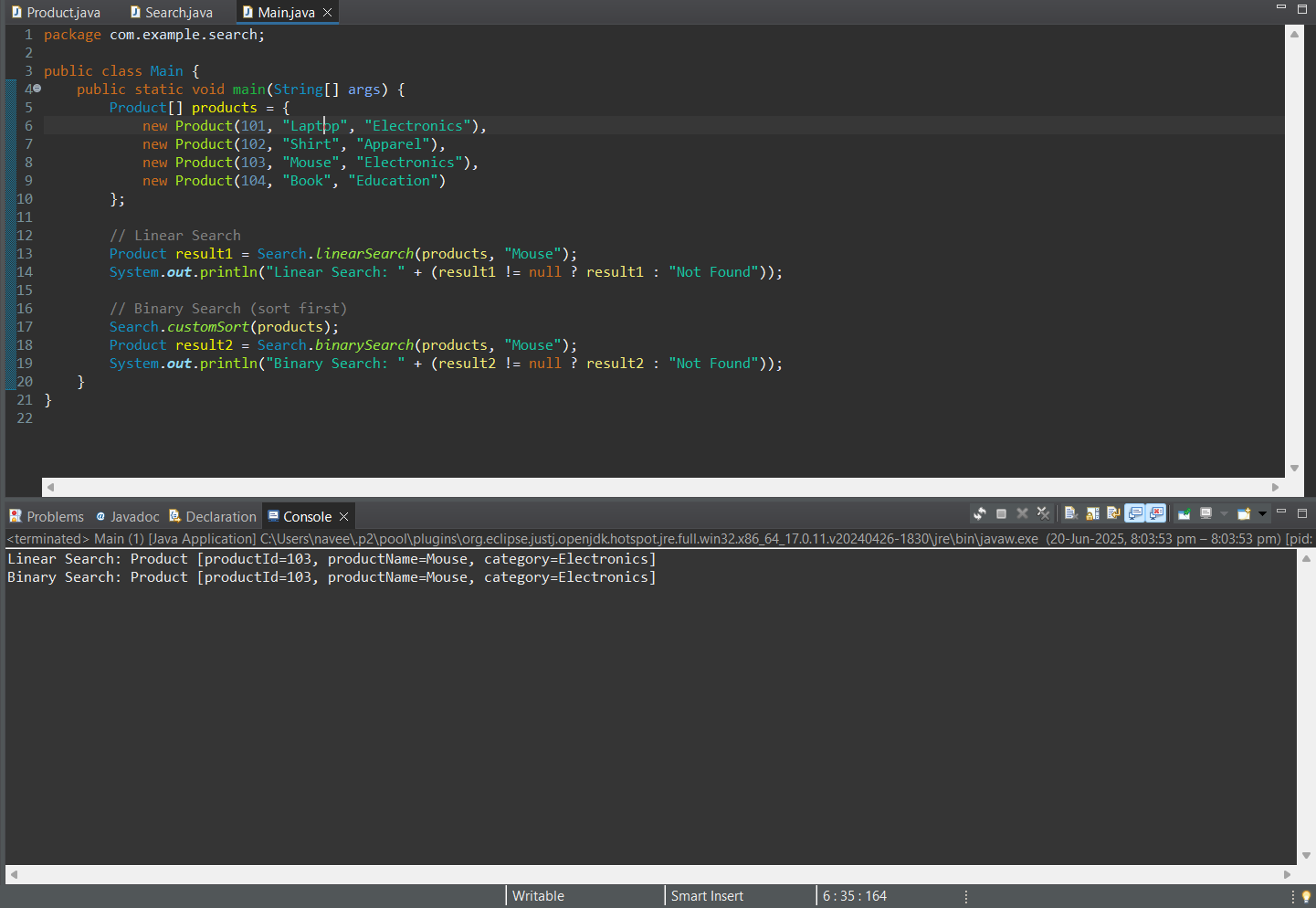
Product result2 = Search.*binarySearch*(products, "Mouse");

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

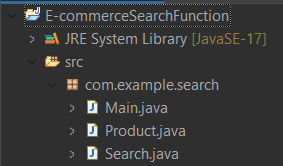
}

}

**OUTPUT:**

****

**OUTPUT: (Project Structure)**

****

**ANALYSIS:**

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

**Answer –**

**Comparison between Linear and Binary search Time complexities**

|  |  |  |
| --- | --- | --- |
| **Case** | **Linear Search** | **Binary Search** |
| Best Case | O(1) | O(1) |
| Average Case | O(n) | O(log n) |
| Worst Case | O(n) | O(log n) |

Explanation:

* Linear search scans each item one by one, so in the worst case, it checks all n products.
* Binary search is efficient, but only works on **sorted** arrays. Here, my customSort() uses Arrays.sort() → O(n log n).
* So, Time complexity for binary search becomes:
* **Sort + Search = O(n log n + log n) ≈ O(n log n)**

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

**Answer –** In the context of my platform, where users search for products by name, both linear search and binary search have their advantages. However, the suitability depends on the size of the dataset and the frequency of search operations.

**Linear Search Suitability:**

Linear search is more suitable when:

* The number of products is small.
* Search operations are infrequent.
* The product list changes often, making sorting less efficient.

Since linear search does not require any preprocessing or sorting, it is simple to implement and works well for smaller datasets. In such cases, the overhead of sorting (required for binary search) is not justified.

**Binary Search Suitability:**

Binary search is more suitable when:

* The dataset is large.
* The data remains relatively static.
* High performance is needed for frequent searches.

Binary search significantly reduces search time to O(log n), but it requires the data to be sorted. In my current implementation, the array is sorted using a custom comparator before applying binary search. This approach is beneficial when dealing with larger product catalogs, as it improves search efficiency.

**Conclusion:**

For small and dynamic datasets, **linear search** is more suitable due to its simplicity and low overhead. However, for larger and more static datasets, **binary search** becomes the better choice because of its faster search time. To maximize binary search performance, the product list should be sorted once and reused for subsequent searches.