**1) 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.

**CODE:**

**ProductSearchDemo.java**

**import** java.util.\*;

**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;

}

**public** String toString() {

**return** "Product[ID=" + productId + ", Name=" + productName + ", Category=" + category + "]";

}

}

**public** **class** ProductSearchDemo {

//Linear search only for ID

**public** **static** Product linearSearchById(Product[] products, **int** targetId) {

**for** (Product p : products) {

**if** (p.productId == targetId) **return** p;

}

**return** **null**;

}

// Binary search by ID and Name using Recrusion

**public** **static** Product binarySearchByIdAndName(Product[] products, **int** left, **int** right, **int** targetId, String targetName) {

**if** (left > right) **return** **null**;

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

**if** (products[mid].productId == targetId) {

// Check name too

**if** (products[mid].productName.equalsIgnoreCase(targetName)) {

**return** products[mid];

} **else** {

**return** **null**;

}

}

**if** (products[mid].productId > targetId) {

**return** *binarySearchByIdAndName*(products, left, mid - 1, targetId, targetName);

} **else** {

**return** *binarySearchByIdAndName*(products, mid + 1, right, targetId, targetName);

}

}

**public** **static** **int** readInt(Scanner sc, String prompt) {

**while** (**true**) {

System.out.print(prompt);

**if** (sc.hasNextInt()) **return** sc.nextInt();

sc.next();

System.out.println("Please enter a number.");

}

}

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

**int** n = *readInt*(sc, "Enter number of products: ");

sc.nextLine();

Product[] products = **new** Product[n];

**for** (**int** i = 0; i < n; i++) {

System.out.println("\nProduct #" + (i+1));

**int** id = *readInt*(sc, "Product ID: ");

sc.nextLine();

System.out.print("Product Name: ");

String name = sc.nextLine();

System.out.print("Product Category: ");

String category = sc.nextLine();

products[i] = **new** Product(id, name, category);

}

System.out.println("\nChoose search option:");

System.out.println("1.Search by ID only for linear search");

System.out.println("2. Search by ID and Name for binary search");

**int** option = *readInt*(sc,"Enter your choice to perform the operation: ");

**if** (option == 1) {

**int** searchId = *readInt*(sc, "Enter product ID to search: ");

Product result = *linearSearchById*(products, searchId);

System.out.println("\nLinear Search Result: " + result);

} **else** **if** (option == 2) {

**int** searchId = *readInt*(sc, "Enter product ID to search: ");

sc.nextLine();

System.out.print("Enter product name to search: ");

String searchName = sc.nextLine();

Arrays.*sort*(products, Comparator.*comparingInt*(p -> p.productId));

Product result = *binarySearchByIdAndName*(products, 0, products.length - 1, searchId, searchName);

System.out.println("\n Binary Search Result: " + result);

} **else** {

System.out.println("Invalid option.");

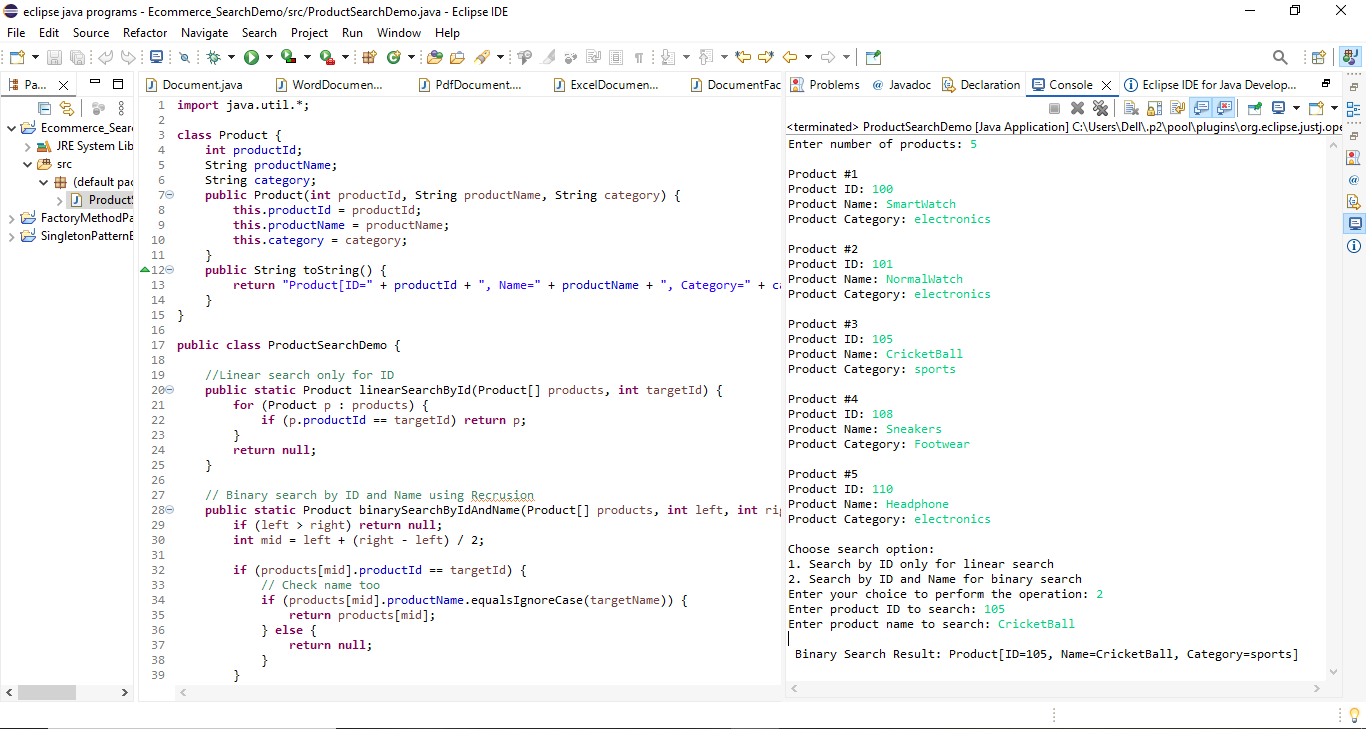
}

sc.close();

}

}

**OUTPUT:**

****

**ANALYSIS:**

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

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

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

If your product list is **small** (e.g. a few hundred items) and is **not sorted**, then **linear search** is straightforward and efficient enough.

If your platform:

* Stores **thousands or millions** of products.
* Has data **already sorted** or can be sorted once at the beginning.
* Needs **fast and repeated lookups** by product ID and ProductName.

**Binary search** **is** **much** **more** **scalable and faster** **(O(log n)).**