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

**Describe Big O notation and its application to algorithm analysis.**An algorithm's efficiency in terms of time or space as the input size (n) increases can be expressed mathematically using asymptotic notation. It enables us to compare algorithms without regard to implementation specifics or hardware.  
**Big O Notation:** An algorithm's worst-case time complexity is expressed using Big O notation. It helps us understand how an algorithm scales by providing an upper bound on how long it can take.   
  
For instance, a search algorithm is said to operate in O(n) time (linear time) if it requires a maximum of n steps to locate an element in a list of size n.

**Describe the best, average, and worst-case scenarios for search operations.**  
1. **Best Case:** The best case scenario is when the search is finished in the shortest amount of time, like when the desired item is found at the top of a list. The time complexity in this situation may be as low as O(1), or constant time.   
**2. Average Case:** The average case shows the expected performance for a variety of typical inputs, such as whether the desired item is likely to be in the middle of a list. Despite being O(n/2), which simplifies to O(n), the average case for linear search is still O(n).

**3. Worst Case:** The worst case occurs when the algorithm takes the maximum time to complete, such as when the item is not in the list at all or is located at the end. In linear search, this would also result in a time complexity of O(n)

**Code:  
  
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;

}

@Override

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

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

import java.util.Comparator;

public class ProductSearch {

// ---------- LINEAR SEARCH METHODS ----------

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

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

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

for (Product product : products) {

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

return product;

}

}

return null;

}

public static Product linearSearchByCategory(Product[] products, String targetCategory) {

for (Product product : products) {

if (product.category.equalsIgnoreCase(targetCategory)) {

return product;

}

}

return null;

}

// ---------- BINARY SEARCH METHODS ----------

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

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

return products[mid];

} else if (products[mid].productId < targetId) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

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

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

String midName = products[mid].productName.toLowerCase();

if (midName.equals(targetName.toLowerCase())) {

return products[mid];

} else if (midName.compareTo(targetName.toLowerCase()) < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static Product binarySearchByCategory(Product[] products, String targetCategory) {

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

String midCategory = products[mid].category.toLowerCase();

if (midCategory.equals(targetCategory.toLowerCase())) {

return products[mid];

} else if (midCategory.compareTo(targetCategory.toLowerCase()) < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

}

**ProductSearchTest.java**public class ProductSearchTest {

public static void main(String[] args) {

Product[] products = {

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

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

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

new Product(104, "Watch", "Accessories"),

new Product(105, "Tablet", "Electronics")

};

// Linear Search Tests

System.out.println("Linear Search by ID:");

System.out.println(ProductSearch.linearSearchById(products, 103));

System.out.println("\nLinear Search by Name:");

System.out.println(ProductSearch.linearSearchByName(products, "Watch"));

System.out.println("\nLinear Search by Category:");

System.out.println(ProductSearch.linearSearchByCategory(products, "Footwear"));

// Binary Search Tests

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

System.out.println(ProductSearch.binarySearchById(products, 104));

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

System.out.println(ProductSearch.binarySearchByName(products, "Phone"));

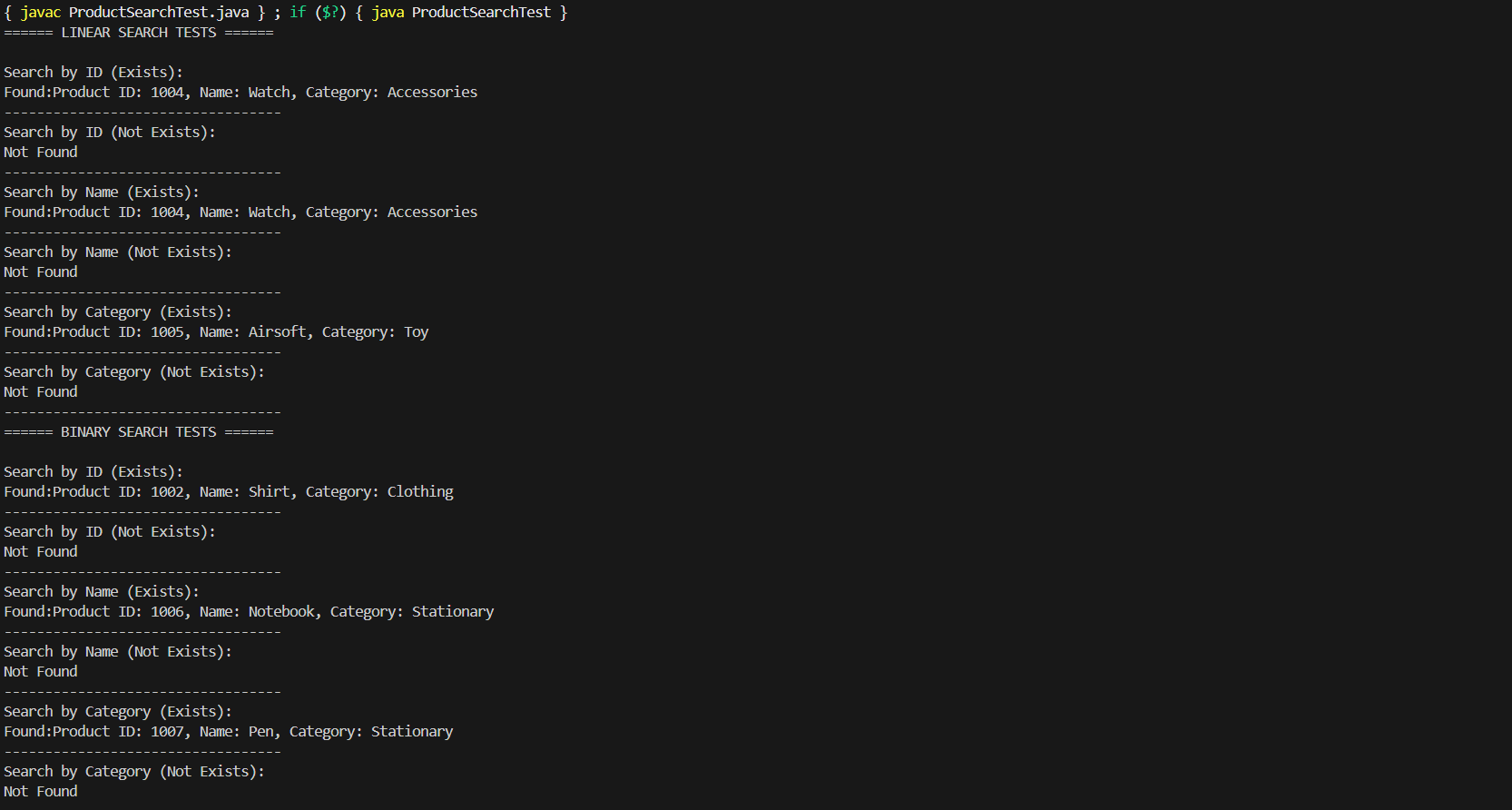
System.out.println("\nBinary Search by Category:");

System.out.println(ProductSearch.binarySearchByCategory(products, "Accessories"));

}

}

**Output:**

****

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

Linear search has a time complexity of O(n), i.e., it could be checking every item singly until the target or the end of the list is reached. On the other hand, binary search has a time complexity of O(log n) since it reduces the space to be searched by half with each move. Binary search is much quicker for large lists but needs the list to be sorted. For instance, for a list of one million, binary search can take as little as 20 comparisons, whereas linear search can take as much as a million. Therefore, binary search is more efficient if sorting is feasible.

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

For an e-commerce site with big data, binary search is better because it has fast O(log n) performance, particularly when performing searches on sorted data like product IDs or names. It provides fast response times and improved scalability. Linear search can still be good for unsorted or infrequently searched data, though, like flexible category names or tags. Practically, websites tend to integrate methods, applying binary search or database indexing to structured fields while implementing linear or full-text search for unstructured fields. As a whole, binary search is more performant in high-traffic, data-focused websites.