**DATA STRUCTURES AND ALGORITHMS:**

**Exercise 2: E-Commerce platform Search Functionality:**

**➤ What is Big O Notation?**

Big O notation is a way to describe how efficient an algorithm is in terms of time and space as the input size increases. It tells us **how fast or slow** an algorithm will perform, especially when the number of inputs grows very large.

For example, if a search algorithm takes O(n) time, it means it will take time directly proportional to the number of elements. If it's O(log n), it means it’s much faster because it cuts the problem size in half each time.

**➤ Best, Average, and Worst Case Scenarios (for Searching)**

* **Best Case:** The item is found immediately (like at the first position).
* **Average Case:** The item is somewhere in the middle of the list.
* **Worst Case:** The item is not in the list at all, or it is at the very end.

These cases help in understanding how the algorithm performs in different situations, not just in ideal conditions.

**🔹 2. Setup**

I created a simple class called Product with basic attributes that are usually needed in a search:

* productId (int)
* productName (String)
* category (String)

This helps simulate a real e-commerce environment where products are searched by name, category, or ID.

**🔹 3. Implementation**

To explore different search methods, I implemented:

**➤ Linear Search:**

* I stored the products in a normal array.
* Went through each product one by one to check for a match.
* It is simple but becomes slow as the number of products increases.

**➤ Binary Search:**

* Here, I sorted the product array (based on product name).
* Then used binary search logic: check middle element, then left or right half depending on the comparison.
* It is much faster but requires that the array is sorted beforehand.

I used both approaches to compare how they work and how fast they return the result.

**SearchImplementation.java**

import java.util.Arrays;

import java.util.Scanner;

class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

void printDetails() {

System.out.println("Product ID: " + productId);

System.out.println("Product Name: " + productName);

System.out.println("Category: " + category);

System.out.println();

}

}

public class SearchImplementation {

// Linear Search: Checks each product name one by one

static int linearSearch(Product[] products, String searchName) {

for (int i = 0; i < products.length; i++) {

if (products[i].productName.equalsIgnoreCase(searchName)) {

return i;

}

}

return -1; // Not found

}

static int binarySearch(Product[] products, String searchName) {

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int result = products[mid].productName.compareToIgnoreCase(searchName);

if (result == 0) {

return mid;

} else if (result < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1; // Not found

}

public static void main(String[] args) {

// Step 1: Store products in an array

Product[] products = {

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

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

new Product(103, "Book", "Stationery"),

new Product(104, "Mobile", "Electronics"),

new Product(105, "Bottle", "Kitchen")

};

Scanner sc = new Scanner(System.in);

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

String linearInput = sc.nextLine();

int result1 = linearSearch(products, linearInput);

if (result1 != -1) {

System.out.println("Product found using linear search:");

products[result1].printDetails();

} else {

System.out.println("Product not found using linear search.");

}

// Step 3: Sort the array before binary search

Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));

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

String binaryInput = sc.nextLine();

int result2 = binarySearch(products, binaryInput);

if (result2 != -1) {

System.out.println("Product found using binary search:");

products[result2].printDetails();

} else {

System.out.println("Product not found using binary search.");

}

sc.close();

}

}

**Output:**

A black screen with white text

AI-generated content may be incorrect.

**4. Analysis**

| **Search Method** | **Time Complexity** | **Suitable When** |
| --- | --- | --- |
| **Linear Search** | O(n) | Data is unsorted or small |
| **Binary Search** | O(log n) | Data is already sorted and large |

**➤ Which One is Better for an E-commerce Platform?**

For an e-commerce platform, where the number of products can be huge, **binary search is the better option**, but **only if the data is sorted**. It reduces the number of comparisons drastically.

However, if the products are not sorted (like when users search by filters or keywords), we may have to go for **linear search** or even better — use **advanced search techniques** like hash maps or search indexes (but that’s beyond the basics).

For this exercise, binary search is more efficient, but linear search is easier to implement and works in all cases.