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

**Asymptotic Notation**

**❖ Big O Notation:**

Big O notation describes how the runtime or space requirements of an algorithm scale with input size n. In simple inputsize directly proportional to timetaken.

It helps in:

* Comparing algorithms' performance.
* Predicting scalability as data grows.
* Identifying inefficiencies.

**Examples:**

* **O(1)**: Constant time (fastest).
* **O(n)**: Linear time.
* **O(log n)**: Logarithmic time (common in binary search).
* **O(n²)**: Quadratic time (inefficient for large inputs).

**❖ Best, Average, and Worst Cases for Searching:**

| Search Type | Best Case | Average Case | Worst Case |
| --- | --- | --- | --- |
| Linear Search | O(1) — first item match | O(n/2) ≈ O(n) | O(n) — last item or not found |
| Binary Search | O(1) — middle match | O(log n) | O(log n) |

**Step 2: Code**

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;

}

}

**Implementation**

**Linear Search:**

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

for (Product product : products) {

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

return product;

}

}

return null;

}

**Binary Search:**

Binary search requires the array to be **sorted** (e.g., by productName).

import java.util.\*;

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

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

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

while (left <= right) {

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

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

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

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

else right = mid - 1;

}

return null;

}

**Analysis**

| **Criteria** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Time Complexity** | O(n) | O(log n) |
| **Space Complexity** | O(1) | O(1) |
| **Sorted Input Needed** | No | Yes |
| **Scalability** | Poor for large data | Efficient for large data |

**Suitable Algorithm:**

* **Binary search** is more suitable **for large datasets with frequent lookups**, assuming data can be sorted.
* **Linear search** is better **for unsorted or very small datasets**, or **when inserting new products often** (because sorting becomes costly).

**Final Suggestion:**

For an **e-commerce platform** with **large product data**, **binary search** (or even more advanced techniques like **hash maps** is ideal for fast and scalable performance. As hashmap takes O(1) time for insert,remove etc… and also if the data contains any redundance data then hashmap is useful.

**Total code**

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

// Sort the array based on product name (case-insensitive)

Arrays.sort(products, new Comparator<Product>() {

@Override

public int compare(Product p1, Product p2) {

return p1.productName.toLowerCase().compareTo(p2.productName.toLowerCase());

}

});

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

while (left <= right) {

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

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

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

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

else right = mid - 1;

}

return null;

}

public static void main(String[] args) {

// Sample products

Product[] products = {

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

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

new Product(103, "Watch", "Accessories"),

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

new Product(105, "Bag", "Travel")

};

String target = "Phone";

Product linearResult = linearSearch(products, target);

System.out.println("Linear Search Result:");

if (linearResult != null) {

System.out.println("Product found: " + linearResult);

} else {

System.out.println("Product not found.");

}

Product binaryResult = binarySearch(products, target);

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

if (binaryResult != null) {

System.out.println("Product found: " + binaryResult);

} else {

System.out.println("Product not found.");

}

}

}

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;

}

}

