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

**Steps: Understand Asymptotic Notation**

**Big O Notation**

Big O notation is a mathematical notation used to describe the upper bound of an algorithm's time complexity as a function of the input size nnn. It helps in analyzing and comparing the performance of algorithms by providing an estimate of the worst-case scenario.

**Best, Average, and Worst-Case Scenarios for Search Operations**

* **Best Case:** The minimum time required to complete an operation, typically when the desired element is found immediately.
* **Average Case:** The expected time required to complete an operation, averaged over all possible inputs.
* **Worst Case:** The maximum time required to complete an operation, typically when the desired element is at the end of the array or not present at all.

**Setup**

Create a Product class with attributes for searching.

public class Product {

private String productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category; }

// Getters

public String getProductId() {

return productId;}

public String getProductName() {

return productName;}

public String getCategory() {

return category;

}

}

**Implementation**

**Linear Search**

public class Search {

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

for (Product product : products) {

if (product.getProductId().equals(productId)) {

return product; }}

return null;}}

**Binary Search**

import java.util.Arrays;

import java.util.Comparator;

public class Search {

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

for (Product product : products) {

if (product.getProductId().equals(productId)) {

return product; }}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

Product midProduct = products[mid];

int comparison = midProduct.getProductId().compareTo(productId);

if (comparison == 0) {

return midProduct;

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid – 1;}}

return null; }

public static void main(String[] args) {

Product[] products = {

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

new Product("102", "Smartphone", "Electronics"),

new Product("103", "Desk", "Furniture"),

new Product("104", "Chair", "Furniture"),

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

};

// Sort products by productId for binary search

Arrays.sort(products, Comparator.comparing(Product::getProductId));

// Linear Search

Product result1 = linearSearch(products, "103");

System.out.println(result1 != null ? result1.getProductName() : "Product not found");

// Binary Search

Product result2 = binarySearch(products, "103");

System.out.println(result2 != null ? result2.getProductName() : "Product not found");

}

}

**Analysis**

**Time Complexity Comparison**

* **Linear Search:**
  + **Best Case:** O(1)O(1)O(1) - The desired product is the first element.
  + **Average Case:** O(n)O(n)O(n) - The desired product is in the middle or the average position.
  + **Worst Case:** O(n)O(n)O(n) - The desired product is the last element or not present.
* **Binary Search:**
  + **Best Case:** O(1)O(1)O(1) - The desired product is the middle element.
  + **Average Case:** O(log⁡n)O(\log n)O(logn) - The search space is halved each time.
  + **Worst Case:** O(log⁡n)O(\log n)O(logn) - The desired product is in the last possible position or not present.

**Suitability for the Platform**

**Binary Search** is generally more suitable for the e-commerce platform due to its significantly lower time complexity, especially for large datasets. However, it requires the data to be sorted. In cases where the dataset is not sorted or frequently updated, the cost of sorting may offset the benefits of binary search. In such cases, linear search might be simpler and more practical.