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

**1.Understand Asymptotic Notation**

Big O notation describes the upper bound of an algorithm's running time, helping to understand its worst-case efficiency.   
It focuses on how performance scales with input size.  
  
Best, Average, and Worst Cases:  
- Best Case: The item is found in the first position.  
- Average Case: The item is located somewhere in the middle.  
- Worst Case: The item is at the last position or not present at all.  
  
For searching:  
- Linear Search: O(1) best, O(n) average and worst  
- Binary Search: O(log n) best/average/worst (requires sorted input)

**2. Setup:**

Create a `Product` class with attributes `productId`, `productName`, and `category`.

**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 productId + " - " + productName + " [" + category + "]";  
 }  
}

**3. Implementation:**

**SearchUtility.java:**

import java.util.Arrays;  
import java.util.Comparator;  
  
public class SearchUtility {  
  
 // Linear Search  
 public static Product linearSearch(Product[] products, String name) {  
 for (Product p : products) {  
 if (p.productName.equalsIgnoreCase(name)) {  
 return p;  
 }  
 }  
 return null;  
 }  
  
 // Binary Search (sorted by productName)  
 public static Product binarySearch(Product[] products, String name) {  
 Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));  
 int low = 0, high = products.length - 1;  
  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 int cmp = products[mid].productName.compareToIgnoreCase(name);  
  
 if (cmp == 0) return products[mid];  
 else if (cmp < 0) low = mid + 1;  
 else high = mid - 1;  
 }  
 return null;  
 }  
}

**Main.java:**

public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(101, "Shoes", "Footwear"),  
 new Product(102, "T-Shirt", "Clothing"),  
 new Product(103, "Laptop", "Electronics"),  
 new Product(104, "Watch", "Accessories"),  
 new Product(105, "Phone", "Electronics")  
 };  
  
 // Linear Search  
 long start1 = System.nanoTime();  
 Product result1 = SearchUtility.linearSearch(products, "Watch");  
 long end1 = System.nanoTime();  
 System.out.println("Linear Search Result: " + result1);  
 System.out.println("Time: " + (end1 - start1) + " ns");  
  
 // Binary Search  
 long start2 = System.nanoTime();  
 Product result2 = SearchUtility.binarySearch(products, "Watch");  
 long end2 = System.nanoTime();  
 System.out.println("Binary Search Result: " + result2);  
 System.out.println("Time: " + (end2 - start2) + " ns");  
 }  
}

**4. Analysis:**

Time Complexity:  
- Linear Search: O(n) – each element is checked until match is found or end is reached.  
- Binary Search: O(log n) – input must be sorted, uses divide-and-conquer.  
  
Recommendation:  
Binary search is significantly faster for large, sorted datasets and should be used when performance is a concern.  
For unsorted data or small datasets, linear search may still be sufficient and simpler.