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

**Scenario:**

I were working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Step 1: Understand Asymptotic Notation**

**➤ What is Big O Notation?**

Big O notation is used to describe the **time complexity** or **space complexity** of an algorithm. It helps to measure algorithm performance as input size grows.

**➤ Best, Average, and Worst-Case Scenarios:**

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |

**🛠️ Step 2: Setup**

* Create a Product class with the following attributes:
  + productId
  + productName
  + category

**Step 3: Implementation**

* Implement **Linear Search**:
  + Works on any array
  + Searches each element one by one
* Implement **Binary Search**:
  + Requires a **sorted** array
  + Divides the search range by half each time

**Step 4: Analysis**

**➤ Time Complexity:**

| **Algorithm** | **Time Complexity** | **Requirements** |
| --- | --- | --- |
| Linear Search | O(n) | Works on unsorted array |
| Binary Search | O(log n) | Requires sorted array |

**➤ Recommendation:**

* For **small or unsorted** datasets, Linear Search is acceptable.
* For **large and sorted** product datasets (like in real e-commerce platforms), Binary Search is more efficient and scalable.

**Solution:**

**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;

}

}

**SearchUtil.java**

import java.util.Arrays;

import java.util.Comparator;

public class SearchUtil {

// Linear search by product name

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

for (Product product : products) {

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

return product;

}

}

return null;

}

// Binary search by product name (requires sorted array)

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

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

while (left <= right) {

int mid = (left + right) / 2;

int comparison = products[mid].productName.compareToIgnoreCase(name);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

// Utility to sort by productName for binary search

public static void sortProductsByName(Product[] products) {

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

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

Product[] products = {

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

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

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

new Product(104, "Book", "Stationery"),

new Product(105, "Watch", "Accessories")

};

// Linear Search

Product foundLinear = SearchUtil.linearSearch(products, "Mobile");

System.out.println("Linear Search Result: " + (foundLinear != null ? foundLinear : "Product not found"));

// Binary Search (Sort first)

SearchUtil.sortProductsByName(products);

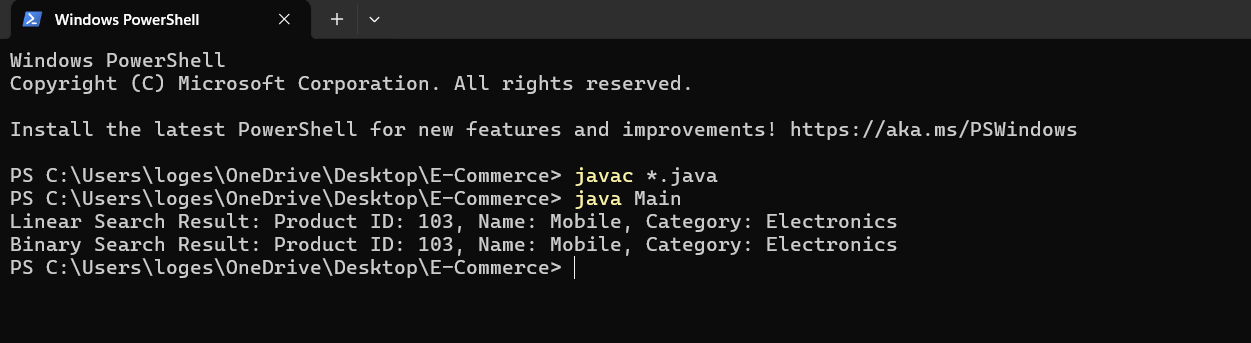
Product foundBinary = SearchUtil.binarySearch(products, "Mobile");

System.out.println("Binary Search Result: " + (foundBinary != null ? foundBinary : "Product not found"));

}

}

**Output:**

****