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

(Step – 1)What is Asymptotic Notation?

Asymptotic Notation is a mathematical way to describe the performance (time or space complexity) of an algorithm as the input size (n) grows.

It helps in comparing algorithms independently of machine or code implementation, focusing on how well they scale.

What is Big O Notation?

Big O notation represents the upper bound of an algorithm's running time. It shows the worst-case time complexity, i.e., how the algorithm performs in the most demanding situation.

- It simplifies complex expressions by focusing on the most dominant term.
- It ignores constants and lower-order terms (e.g., O(2n + 5) becomes O(n)).

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

In algorithm analysis, it's important to consider different runtime behaviors based on input scenarios. These are typically described as best case, average case, and worst case, especially for search operations like linear and binary search.

- Best Case:
  This scenario occurs when the desired item is found immediately.
    - In linear search, this means the element is found at the very first position, resulting in a time complexity of O(1).
    - In binary search, if the item is exactly at the middle of the sorted list on the first comparison, it also takes O(1) time.
- Average Case:
  The average case assumes that the item could be located anywhere in the dataset, and all positions are equally likely.

    - For linear search, this results in a time complexity of approximately O(n).
    - For binary search, it takes around O(log n) time, since the search space is repeatedly halved.
- Worst Case:
  This is the most time-consuming scenario, where the item is either not present in the dataset or is found at the last possible point.
    - In linear search, this means checking every element, resulting in O(n) complexity.
    - In binary search, it still performs efficiently with a worst-case time complexity of O(log n), since it continually divides the dataset.

(step – 2)Defining a product class

```
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 + ")";
    }
}
```

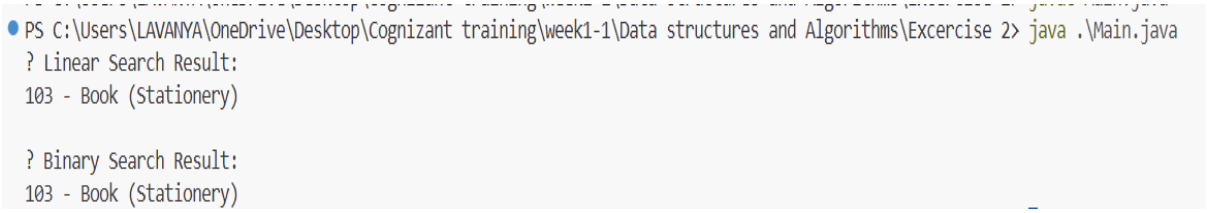
(step - 3) Implementing linear search and binary search to store products in an array

```
import java.util.Arrays;
import java.util.Comparator;

public class Main {
    public static Product linearSearch(Product[] products, String name) {
        for (Product product : products) {
            if (product.productName.equalsIgnoreCase(name)) {
                return product;
            }
        }
        return null;
    }

    public static Product binarySearch(Product[] products, String name) {
        int left = 0, right = products.length - 1;

        while (left <= right) {
            int mid = (left + right) / 2;
            int compare =
name.compareToIgnoreCase(products[mid].productName);

            if (compare == 0) return products[mid];
            else if (compare < 0) right = mid - 1;
            else left = mid + 1;
        }
```

```
        return null;
    }

    public static void main(String[] args) {
        Product[] productList = {
            new Product(101, "Laptop", "Electronics"),
            new Product(102, "Shirt", "Clothing"),
            new Product(103, "Book", "Stationery"),
            new Product(104, "Headphones", "Electronics"),
            new Product(105, "Shoes", "Footwear")
        };

        System.out.println("🔍 Linear Search Result:");
        Product foundLinear = linearSearch(productList, "Book");
        System.out.println(foundLinear != null ? foundLinear : "Product not found");

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

        System.out.println("\n🔍 Binary Search Result:");
        Product foundBinary = binarySearch(productList, "Book");
        System.out.println(foundBinary != null ? foundBinary : "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 productId + " - " + productName + " (" + category + ")";
    }
}
```

Output



(step – 4)Comparing the time complexity of linear search and binary search

➢ Linear Search:

- Best Case (O(1)): Target element is the first in the list.
- Average Case (O(n)): Target element is in the middle or randomly placed.
- Worst Case (O(n)): Element is at the end or not present.
- Works on unsorted data.
- Simple to implement.

➢ Binary Search:

- Best Case (O(1)): Target element is at the middle of the list.
- Average Case (O(log n)): List is halved at each step; quick convergence.
- Worst Case (O(log n)): Max number of divisions needed before concluding.
- Works only on sorted data.
- More efficient for large datasets.

Which Algorithm is Better for an E-Commerce Platform?

✅ Binary Search is more suitable when:

- Your product list is large.
- You can maintain sorted data (e.g., sorted by product name, ID, or price).
- You need fast search performance for real-time applications (e.g., auto-complete, filter-as-you-type).
- Your data is mostly read-only and doesn't change frequently.

❗ Linear Search may be preferred when:

- Your dataset is small.
- The product list is unsorted and sorting is not feasible due to time/memory constraints.