**WEEK -1  
DATA STRUCTURES AND ALORITHM**

**Exercise 2**: E-commerce Platform Search Function  
**UNDERSTAND ASYMPTOTIC NOTATION**

1. What is Big O Notation?

* Big O notation describes the upper bound of an algorithm's running time as the input size increases.
* It helps to analyze performance independently of hardware and exact input size.
* It focuses on scalability and efficiency.

2. Why is it useful?

* It lets developers compare algorithms objectively.
* Helps in identifying potential bottlenecks in large-scale applications.
* Guides developers to select the most efficient algorithm for a given task.
  1. Search Scenarios (Linear and Binary Search)

| Search Type | Best Case | Average Case | Worst Case |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n/2) ≈ O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |

**CODE:**

**PROJECT NAME:** ECommerceSearchExample

**CLASS NAME:** Product  
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;

}

public String toString() {

return "[" + productId + ", " + productName + ", " + category + "]";

}

}

**CLASS NAME:** ProductSearch  
ProductSearch.java:  
import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

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

for (Product p : products) {

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

return p;

}

}

return null;

}

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

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

while (left <= right) {

int mid = (left + right) / 2;

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

if (cmp == 0)

return products[mid];

else if (cmp < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Shampoo", "Personal Care"),

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

new Product(103, "Book", "Stationery"),

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

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

};

System.out.println("🔍 Linear Search for 'Pen'");

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

System.out.println(result1 != null ? "Found: " + result1 : "Not Found");

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

System.out.println("\n🔍 Binary Search for 'Pen'");

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

System.out.println(result2 != null ? "Found: " + result2 : "Not Found");

}

}

**ANALYSIS**

* 1. **Time Complexity Comparison:**
* **Linear Search**
  + Best Case: O(1) - First element matches
  + Average/Worst Case: O(n) - Element is in the middle or not present
* **Binary Search**
  + Best Case: O(1) - Middle element matches
  + Average/Worst Case: O(log n) - Repeatedly divide the array

**2. Suitability for E-commerce Platform:**

* **Linear Search** is:
  + Simple and works on **unsorted data**
  + Inefficient for large datasets
* **Binary Search** is:
  + Much **faster** and **scalable**
  + Requires the array to be **sorted**
  + Best suited for platforms with large product databases where search speed is critical

For an e-commerce platform handling thousands of products, Binary Search is more suitable due to its superior performance and scalability, despite the need for sorted data.