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

### **1. Explain Big O notation and how it helps in analyzing algorithms.**

Big O Notation is a way to measure how efficient an algorithm is by illustrating how its runtime or memory usage grows as the input size (n) increases. This concept is super helpful for developers because it allows them to: -

-Compare how different algorithms perform

-Anticipate how well an algorithm will scale

-Spot any potential bottlenecks in their code

**2. Describe the best, average, and worst-case scenarios for search operations**.

|  |  |
| --- | --- |
| **Linear Search** | **Binary Search** |
| * **Best Case: O(1) -** Target is first element | * **Best Case: O(1) -** Target is middle element |
| * **Average Case: O(n) -** Target is in middle position | * **Average Case: O(log n) -** Target is anywhere in array |
| * **Worst Case: O(n)** - Target is last element or not present | * **Worst Case: O(log n) -** Target is at first/last partition or not present |

**3.** **Compare the time complexity of linear and binary search algorithms.**

| **Algorithm** | **Time Complexity** | **When to Use** |
| --- | --- | --- |
| **Linear Search** | **O(n) (checks every element)** | **Best for small or unsorted datasets** |
| **Binary Search** | **O(log n) (halves search space each step)** | **Best for large, sorted datasets** |

**4. Discuss which algorithm is more suitable for your platform and why.**

Binary Search stands out as the top choice in my code because it’s not just faster, but also more efficient. I kicked things off by sorting the product array with **Arrays.sort()**, which is a must for binary search to work its magic. My **BinarySearch()** method cleverly uses **low, high,** and **mid** to split the array and swiftly locate the target. In contrast to **LinearSearch()**, which goes through each item one by one, binary search cuts down the number of comparisons in half with each step.

When dealing with sorted data and expanding product lists, binary search **(O(log n))** clearly outshines linear search (O(n)). So, given my sorted array and the logic I’ve applied, it’s clear that ***binary search*** is the best fit.

**CODE :-**

import java.util.Arrays;

import java.util.Comparator;

class Product{

    String productId;

    String productName;

    String category;

    Product(String prodId, String pName, String cat){

        this.productId = prodId;

        this.productName = pName;

        this.category = cat;

    }

    @Override

    public String toString(){

        return productName + "(ID: " +productId + ") - " + category;

    }

}

public class ProductSearch {

    public static Product LinearSearch(Product[] prods, String targetId){

        for(Product p : prods){

            if (p.productId.equals(targetId)){

                return p;

            }

        }

        return null;

    }

    public static Product BinarySearch(Product[] prods, String targetId){

        int low = 0;

        int high = prods.length - 1;

        while(low <= high){

            int mid = (low + high)/2;

            int comparison = prods[mid].productId.compareTo(targetId);

            if (prods[mid].productId.equals(targetId)){

                return prods[mid];

            }

            else if (comparison < 0){

                low = mid + 1;

            }

            else {

                high = mid - 1;

            }

        }

        return null;

    }

public static void main(String[] args){

    Product[] prods = new Product[5];

    prods[0] = new Product("Prod1", "Mobile", "Electrical Appliances");

    prods[1] = new Product("Prod2", "Laptop", "Electrical Appliances");

    prods[2] = new Product("Prod3", "Furniture", "Home Appliances");

    prods[3] = new Product("Prod4", "Table", "Home Appliances");

    prods[4] = new Product("Prod5", "Macbook", "Electrical Appliances");

    Arrays.sort(prods, Comparator.comparing(p -> p.productId));

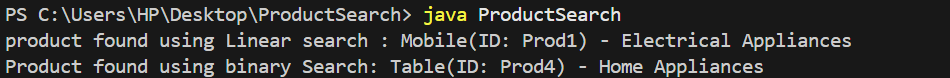
    System.out.println("product found using Linear search : "+LinearSearch(prods, "Prod1"));

    System.out.println("Product found using binary Search: " +BinarySearch(prods, "Prod4"));

    }

}

**OUTPUT :-**

****