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

1. **Understand Asymptotic Notation:**

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

Ans. Big-O, commonly referred to as “Order of”, is a way to express the upper bound of an algorithm’s time complexity, since it analyses the worst-case situation of algorithm.

It provides an upper limit on the time taken by an algorithm in terms of the size of the input. It’s denoted as O(f(n)), where f(n) is a function that represents the number of operations (steps) that an algorithm performs to solve a problem of size n.

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

Ans. For Binary Search-

Best Case : Best case is when the element is at the middle index of the array. It takes only one comparison to find the target element. So the best case complexity is O(1).

Average Case : The average case complexity is **O(logN)**

Worst Case : The worst case will be when the element is present in the first position. As seen in the average case, the comparison required to reach the first element is **logN**. So the time complexity for the worst case is **O(logN)**.

For Linear Search –

Best Case: When the element searched is the first element of the array .So the complexity is O(1).

Average Case : The complexity is O(n).

Worst Case : When the searched element is at the end of the array so the complexity is O(n).

1. **Implementation:**

Code:

import java.util.Arrays;

class Product {

    private int productId;

    private String productName;

    private String category;

    public Product(int productId, String productName, String category) {

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    public int getProductId() {

        return productId;

    }

    public String getProductName() {

        return productName;

    }

    public String getCategory() {

        return category;

    }

}

public class ProductSearch {

    private Product[] products;

    private Product[] sortedProducts;

    public ProductSearch(Product[] products) {

        this.products = products;

        this.sortedProducts = Arrays.copyOf(products, products.length);

        Arrays.sort(sortedProducts, (a, b) -> Integer.compare(a.getProductId(), b.getProductId()));

    }

    // Linear Search

    public Product linearSearch(int productId) {

        for (Product product : products) {

            if (product.getProductId() == productId) {

                return product;

            }

        }

        return null;

    }

    // Binary Search

    public Product binarySearch(int productId) {

        int low = 0;

        int high = sortedProducts.length - 1;

        while (low <= high) {

            int mid = low + (high - low) / 2;

            int midProductId = sortedProducts[mid].getProductId();

            if (midProductId == productId) {

                return sortedProducts[mid];

            } else if (midProductId < productId) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Product A", "Category A"),

            new Product(202, "Product B", "Category B"),

            new Product(303, "Product C", "Category A"),

            new Product(404, "Product D", "Category C"),

            new Product(505, "Product E", "Category B")

        };

        ProductSearch productSearch = new ProductSearch(products);

        // Perform linear search

        Product foundProductLinear = productSearch.linearSearch(303);

        if (foundProductLinear != null) {

            System.out.println("Linear Search Result: " + foundProductLinear.getProductName());

        } else {

            System.out.println("Product not found (Linear Search)");

        }

        // Perform binary search

        Product foundProductBinary = productSearch.binarySearch(303);

        if (foundProductBinary != null) {

            System.out.println("Binary Search Result: " + foundProductBinary.getProductName());

        } else {

            System.out.println("Product not found (Binary Search)");

        }

    }

}

1. **Analysis:**

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

The time complexity of linear search **O (n). The time complexity of a binary search algorithm is O (log n)**, making it a highly efficient and effective method for searching through large data sets.

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

If the products are stored in unsorted manner the linear search is best otherwise if sorted then binary search is more efficient.