**Data Structures and Algorithms – Exercise 2**

**E-Commerce Platform Search Function**

**Big O Notation:**

**Big O notation** is a way to describe the **efficiency** of an algorithm in terms of **time or space** as the input size grows.

It helps in analysing:

* **Time complexity** – how fast an algorithm runs
* **Space complexity** – how much memory it uses

It shows the **worst-case performance**, helping compare algorithms regardless of hardware.

**Search Operations Time Complexity Comparison:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search Method** | **Best Case** | **Average Case** | **Worst Case** | **Note!** |
| Linear Search | O(1) | O(n) | O(n) | No need for Sorting |
| Binary Search | O(1) | O(nlogn) | O(nlogn) | Requires Sorted Array |

**Ecommerce.java**

import java.util.\*;

public class ECommerce{

    public static void main(String[] args){

        Product[] productsLinear = {

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

            new Product(205, "Shoes", "Footwear"),

            new Product(150, "Phone", "Electronics"),

            new Product(350, "Shirt", "Apparel"),

            new Product(120, "Book", "Stationery")

        };

        Product[] productsBinary = {

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

            new Product(120, "Book", "Stationery"),

            new Product(150, "Phone", "Electronics"),

            new Product(205, "Shoes", "Footwear"),

            new Product(350, "Shirt", "Apparel")

        };

        int targetId = 150;

        // Linear Search

        System.out.println("\nLinear Search: \n");

        Product linear = linearSearch(productsLinear, targetId);

        if(linear!=null)

        System.out.println("Product found in Linear Search: " + linear);

        else

        System.out.println("\nProduct not found in Linear Search\n");

        //Binary Search

        System.out.println("\nBinary Search: \n");

        Product binary = binarySearch(productsBinary, targetId);

        if (binary!=null)

        System.out.println("Product found in Binary Search: " + binary);

        else

        System.out.println("\nProduct not found in Binary Search\n");

    }

    public static Product linearSearch(Product[] products, int targetId){

        for(Product p : products){

            if(p.productId == targetId){

                return p;

            }

        }

        return null;

    }

    public static Product binarySearch(Product[] products, int targetId){

        int low = 0, high = products.length - 1;

        while(low <= high){

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

            if(products[mid].productId == targetId)

                return products[mid];

            else if(products[mid].productId < targetId)

                low = mid + 1;

            else

                high = mid - 1;

        }

        return null;

    }

}

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 "ID: " + productId + ", Name: " + productName + ", Category: " + category;

    }

}

**Output**

****

**Choosing Better Algorithm for the Platform:**

Binary search is better suited for an Ecommerce platform because it significantly reduces search time by repeatedly dividing the sorted product list in half, making it much more efficient than linear search, especially when dealing with large catalogues. To implement binary search, the product data must be sorted and then the algorithm can be used to compare the target value with the middle element of the array, narrowing the search range until the product is found or confirmed absent.