**QUESTION:**

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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**UNDERSTANDING:**

Asymptotic notations allow to measure how fast an algorithm will grow as the size of the input increases. Many notations can be use to measure this and the most common one is the Big O notation.

The Big O notation provides use with the worst case measure of an algorithm. Gives us the time and memory it takes as the size of the input increases.

Time Complexity for a search function:

**Best case**: O(1)

**Worst case:** O(n)

**Average case:** O(n/2), O(n)

**CODE:**

import java.util.\*;

class Product{

    int productId;

    String productName;

    String category;

    Product(){};

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    String linearsearch(Product[] array, String searchName){

        for(Product p: array){

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

                return "Product found: "+p.productId+" - "+p.productName+" - "+p.category;

            }

        }

        return "Product not found";

    }

    String binarysearch(Product[] array, String searchName){

        int low = 0;

        int high = array.length-1;

        while(low<=high){

            int mid = (low+high)/2;

            if(array[mid].productName.equalsIgnoreCase(searchName)){

                return "Product found: "+array[mid].productId+" - "+array[mid].productName+" - "+array[mid].category;

            }else if(array[mid].productName.compareToIgnoreCase(searchName) > 0){

                high = mid - 1;

            }else{

                low = mid + 1;

            }

        }

        return "Product not found";

    }

}

public class EcommSearch {

    public static void main(String[] args) {

        Product[] array = {

            new Product(1, "shirt", "clothes"),

            new Product(2, "pant", "clothes"),

            new Product(3, "shoes", "footwear"),

            new Product(4, "watch", "accessories"),

            new Product(5, "hat", "accessories")

        };

        Product p = new Product();

        Arrays.sort(array, Comparator.comparing(pr->pr.productName));

        System.out.println("Products sorted by name!");

        System.out.println("Searching using linear search:");

        System.out.println(p.linearsearch(array, "shoes"));

        System.out.println("Searching using binary search:");

        System.out.println(p.binarysearch(array, "shoes"));

        System.out.println("Searching using linear search:");

        System.out.println(p.linearsearch(array, "chain"));

        System.out.println("Searching using binary search:");

        System.out.println(p.binarysearch(array, "chain"));

    }

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

Binary search is more suitable for this application because it can easily handle large sets of data, assuming that the data is sorted.

**Time Complexity of Linear and Binary Search**

|  |  |  |
| --- | --- | --- |
|  | Binary Search | Linear Search |
| Best Case | O(1) | O(1) |
| Worse Case | O(log n) | O(n) |
| Average Case | O(log n) | O(n) |