**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.

**Solution:**

**Big O Notation:**

The Big O notation is used to describe the rate at which the time taken increases,for a program to run, with respect to increase in the input size of the program.

We can represent this using O(time function).

Some common example are:

O(1) : Constant time. Eg. Accessing an array element.

O(n): Linear time. Eg. Linearly searching for an element in an array of size n. (OR) a single for loop.

There are a few key rules to compute the time complexity of a given program.They are as follows:

* Always calculate and note the worst case scenario for time complexity.
* Always try to avoid constants as they are pretty insignificant while computing the larger time complexity.
* Always avoid lower values.

For an example: There is a time complexity function O(4n^3 +3n^2 +8). And the input size is n=10^5.

As we can see the addition of 8 is pretty much insignificant when compared to the other larger values. Thus we avoid constants and lower values.

There are generally 3 notations used to describe time complexities,namely:

1. **Big O**: O(n),used to represent the worst case scenario.
2. **Theta**: Used to represent the average case scenario.
3. **Omega**: Used to represent the best case scenario.

There are mainly two popular search operations, Linear Search and Binary Search.

**Linear Search** works by iterating through the data one-by-one until the desired value is found.

**Binary search** is much more efficient ,but it works only on sorted arrays. It takes a sorted collection of data , halves it by finding the midpoint and then chooses the half which contains the desired value, it repeats this until the value is found.

**Linear Search:**

Worst case scenario: O(n).

Average case scenario: O(n).

Best case scenario: O(1).

**Binary Search:**

Worst case scenario: O(log n).

Average case scenario: O(log n).

Best case scenario: O(1).

**Code:**

package Coding.SearchOperations;

public class Product {

  int productId;

  String productName;

  String category;

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

    this.productId=productId;

    this.productName=productName;

    this.category=category;

  }

  public static Product linearsearch(Product[] products,String Search){

    for(int i=0;i<products.length;i++){

        if(products[i].productName.equalsIgnoreCase(Search)){

          return products[i];

        }

    }

    return null;

  }

public static Product binarysearch(Product[] products,String Search){

  int left=0;

  int right=products.length-1;

  while(left<=right){

    int mid=(left+right)/2;

    int result=products[mid].productName.compareToIgnoreCase(Search);

    if(result==0){

      return products[mid];

    }

    else if(result<0){

      left=mid+1;

    }

    else{

      right=mid-1;

    }

  }

  return null;

}

}

package Coding.SearchOperations;

import java.util.\*;

public class SearchOperationsExample {

  public static void main(String[] args) {

    Product[] products={

      new Product(1,"Sunrise Coffee","Kitchen"),

      new Product(2,"Lenovo Laptop","Electronics"),

      new Product(3,"FaceWash","Beauty"),

      new Product(4,"Watch","Accessories"),

      new Product(5,"Shoes","Footwear")

    };

    Product result1=Product.linearsearch(products,"Shoes");

    System.out.println("Linear Search: "+result1.productName+" at Id: "+result1.productId);

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

    Product result2=Product.binarysearch(products, "Watch");

    System.out.println("Binary Search: "+result2.productName+" at Id: "+result2.productId);

  }

}

A screen shot of a computer program

AI-generated content may be incorrect.**Output:**