Week 1: Data Structures and Algorithms:

Exercise 2: E-commerce Platform Search Function

1: Understanding Asymptotic Notation

Asymptotic notation is a way to describe the efficiency of algorithms, focusing on how their runtime or space usage grows or changes as the input size increases.

There are mainly three asymptotic notations:

- Big-O notation
- Omega notation
- Theta notation

Big-O Notation:

Big-O notation represents the upper bound of the running time of an algorithm. i.e. it gives the worst-case complexity (time/space) of an algorithm.

It helps analyse algorithms by providing a high-level understanding of their efficiency, without complicating it by including machine specific details.

It allows developers to compare different algorithms and determine which one scales better. Understanding an algorithm's worst-case scenario helps anticipate execution time for large datasets.

Best, Average, and Worst-Case Scenarios for Search Operations:

For search algorithms (linear search and binary search in this exercise), we consider 3 different cases:

- Best Case (Ω - Omega Notation): Optimistic scenario where algorithm performs at its best.
- Average Case (Θ - Theta Notation): Average runtime across multiple possible inputs.
- Worst Case (O - Big O Notation): The least efficient (worst-case) scenario where the algorithm takes the maximum possible time.

2: Code

The Implementation consists of 3 java classes(files): Database.java, Product.java, Main.java

The Main.java file is used test the implementation, and also contains a private sub class that has linear and binary search methods

Functions:

- Product.java – Implementation of product object class as specified in the question
- Database.java – Maintains sorted and unsorted array of product objects, for linear & binary search
- Main.java – Has main function for testing, and a private static subclass containing methods for linear and binary search

Product.java:

```
public class Product {
    private int productID; //UNIQUE
    private String productName;
    private String category;

    //CONSTRUCTOR
    public Product(int id, String name, String category) {
        this.productID = id;
        this.productName = name;
        this.category = category;
    }

    //GETTERS
    public int getProductID() {
        return productID;
    }

    public String getProductName() {
        return productName;
    }

    public String getCategory() {
        return category;
    }

    //PRINT METHOD
    @Override
    public String toString() {
        return String.format("Product[ID=%d, Name='%s', Category='%s']", productID,
productName, category);
    }
}
```

Database.java

```
import java.util.*;

//CUSTOM COMPARATOR FOR PRODUCT OBJECT
class ProductComparator implements Comparator<Product> {
    @Override
    public int compare(Product x, Product y) {
        return Integer.compare(x.getProductID(), y.getProductID());
    }
}

public class Database {
    private ArrayList<Product>store;
    private ArrayList<Product>sorted_store;
```

```java
//CREATING EMPTY DATABASE
public Database() {
    store = new ArrayList<>();
    sorted_store = new ArrayList<>();
}

//CREATING DATABASE WITH LIST OF PRODUCTS
public Database(ArrayList<Product>p) {
    store = new ArrayList<>(p);
    sorted_store = new ArrayList<>(p);
    Collections.sort(sorted_store, new ProductComparator());
}

//INSERTING A PRODUCT INTO DATABASE
public void insertProduct(Product p) {
    store.add(p);
    int ind = 0;
    int n = sorted_store.size();
    while(ind < n) {
        if(sorted_store.get(ind).getProductID() < p.getProductID())
            ind++;
        else
            break;
    }
    sorted_store.add(ind, p);
}

//DELETING A PRODUCT FROM DATABASE (UNUSED)
public void deleteProduct(int id) {
    // DELETING FROM UNSORTED
    for (int i = 0; i < store.size(); i++) {
        if (store.get(i).getProductID() == id) {
            store.remove(i);
            break;
        }
    }

    // DELETING FROM SORTED
    for (int i = 0; i < sorted_store.size(); i++) {
        if (sorted_store.get(i).getProductID() == id) {
            sorted_store.remove(i);
            break;
        }
    }
}


//GETTING THE SORTED LIST OF PRODUCT
public ArrayList<Product> getSortedDb() {
    return sorted_store;
}
```

```java
    //GETTING THE UNSORTED LIST OF PRODUCTS
    public ArrayList<Product> getDb() {
        return store;
    }

    //PRINTING DATABASE
    public void printDatabase() {
        System.out.println("-----------------------------------------------");
        System.out.println("|*| PRODUCT DATABASE |*|\n");
        System.out.println("||UNSORTED||");
        for (Product p : store) {
            System.out.println("   " + p);
        }

        System.out.println("\n||SORTED||");
        for (Product p : sorted_store) {
            System.out.println("   " + p);
        }
        System.out.println("-----------------------------------------------");
    }
}
```

Main.java

```java
import java.util.*;

public class Main {

    //SUB CLASS CONTAINING METHODS FOR PRODUCT SEARCH
    private static class productSearch {

        //LINEAR SEARCH USING PRODUCT ID
        private static Product linearSearch(ArrayList<Product> p, int ID) {
            for(Product i : p) {
                if(i.getProductID() == ID) return i;
            }
            return null;
        }

        //BINARY SEARCH USING PRODUCT ID
        private static Product binarySearch(ArrayList<Product> p, int ID) {
            int b = 0, e = p.size() - 1;
            while(b <= e) {
                int mid = b + (e - b) / 2;
                if(p.get(mid).getProductID() < ID) b = mid + 1;
                else if(p.get(mid).getProductID() > ID) e = mid - 1;
                else return p.get(mid);
            }
            return null;
        }
    }
```

```java
    // TESTING
    public static void main(String[] args) {

        //DEFINING SOME PRODUCTS
        Product p1 = new Product(3, "Keyboard", "Electronics");
        Product p2 = new Product(1, "Notebook", "Stationery");
        Product p3 = new Product(5, "Chair", "Furniture");
        Product p4 = new Product(2, "Pen", "Stationery");
        Product p5 = new Product(4, "Monitor", "Electronics");

        //CREATING A DATABASE WITH FIRST 3 PRODUCTS
        ArrayList<Product> list = new ArrayList<>(Arrays.asList(p1, p2, p3));
        Database db = new Database(list);

        //INSERTING TWO PRODUCTS INTO DATABASE
        db.insertProduct(p4);
        db.insertProduct(p5);

        //DISPLAYING DATABASE
        db.printDatabase();

        int prodId = 4; //SEARCHING FOR PRODUCT WITH productID = 4

        //LINEAR SEARCH DEMO
        ArrayList<Product>arr_unsorted = db.getDb();
        Product target1 = productSearch.linearSearch(arr_unsorted, prodId);
        System.out.println("\n ||LINEAR SEARCH||");
        if(target1 != null)
            System.out.println(target1);
        else
            System.out.println("Product Not Found");

        //BINARY SEARCH DEMO
        ArrayList<Product>arr_sorted = db.getSortedDb();
        Product target2 = productSearch.binarySearch(arr_sorted, prodId);
        System.out.println("\n ||BINARY SEARCH||");
        if(target2 != null)
            System.out.println(target2);
        else
            System.out.println("Product Not Found");
    }
}
```

4. Comparison of Linear and Binary Search Algorithms

Complexity Type	Linear Search	Binary Search
Best Case	O(1)	O(1)
Average Case	O(n)	O(log n)
Worst Case	O(n)	O(log n)
Requirement	Unsorted array	Sorted array

For an e-commerce search system, binary search is generally more efficient because:

- E-commerce platforms have large product catalogs.
- Users expect fast search results. Binary search scales better with increasing dataset sizes.
- But one must maintain a sorted dataset to use binary search, linear search is useful when dataset is unordered (eg – when subgroup of data is unsorted, or other temporary purpose).

OUTPUT

```
ws\src_c29a37d\bin Main "
----------------------------------------------
|*| PRODUCT DATABASE |*|

||UNSORTED||
  Product[ID=3, Name='Keyboard', Category='Electronics']
  Product[ID=1, Name='Notebook', Category='Stationery']
  Product[ID=5, Name='Chair', Category='Furniture']
  Product[ID=2, Name='Pen', Category='Stationery']
  Product[ID=4, Name='Monitor', Category='Electronics']

||SORTED||
  Product[ID=1, Name='Notebook', Category='Stationery']
  Product[ID=2, Name='Pen', Category='Stationery']
  Product[ID=3, Name='Keyboard', Category='Electronics']
  Product[ID=4, Name='Monitor', Category='Electronics']
  Product[ID=5, Name='Chair', Category='Furniture']
----------------------------------------------

 ||LINEAR SEARCH||
Product[ID=4, Name='Monitor', Category='Electronics']

 ||BINARY SEARCH||
Product[ID=4, Name='Monitor', Category='Electronics']
```