**Data Structures and Algorithms – Exercise 1**

**Inventory Management System**

**Usage of Data Structures in handling Large Inventories:**

**Data structures and algorithms are essential in handling large inventories** because they help manage, search, and update data efficiently, which is critical for performance, especially in large-scale systems like Ecommerce platforms.

**Efficient Data Access:**   
Proper data structures (like arrays, hash tables, trees) allow quick retrieval of product information, prices, stock levels, etc.

**Faster Searching and Sorting:**   
Algorithms like binary search or quicksort help find or organize inventory items much faster than naive approaches.

**Optimized Memory Usage:**   
Good data structures reduce memory waste, ensuring the platform can handle large volumes of products smoothly.

**Scalability:**   
Efficient algorithms ensure the system performs well even as the number of products grows.

**Improved User Experience:**   
Faster search results, real-time filtering, and quick inventory updates lead to better customer satisfaction.

**Usage of ArrayList in our Problem:**

An ArrayList is useful for this problem because it allows dynamic storage of products, making it easy to add, update, and delete items from the inventory. It provides built-in methods and maintains insertion order, which simplifies inventory management for small to medium-sized Ecommerce platforms.

**Inventory.java**

import java.util.\*;

public class Inventory{

    static Scanner sc = new Scanner(System.in);

    static List<Product> products = new ArrayList<>();

    public static void main(String[] args){

        boolean exit = false;

        System.out.println("\nChoose an option:");

        System.out.println("1. Add Product");

        System.out.println("2. Perform Linear Search");

        System.out.println("3. Perform Binary Search");

        System.out.println("4. Delete");

        System.out.println("5. Exit\n");

        while(!exit){

            System.out.println("Enter Choice: ");

            int choice = sc.nextInt();

            switch(choice){

                case 1:

                    addProduct();

                    break;

                case 2:

                    linearSearch();

                    break;

                case 3:

                    binarySearch();

                    break;

                case 4:

                    deleteProduct();

                    break;

                case 5:

                    System.out.println("\nExiting... Thank You :)\n");

                    exit = true;

                    break;

                default:

                System.out.println("\nInvalid Choice, Please try again :(\n");

            }

        }

    }

    static void deleteProduct(){

        if(products.isEmpty()){

            System.out.println("\nNo Products Available\n");

            return;

        }

        System.out.println("Enter the Product ID to Delete");

        int id = sc.nextInt();

        int idx = -1;

        boolean found = false;

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

            Product curr = products.get(i);

            if(curr.productID==id){

                idx = i;

                found = true;

                break;

            }

        }

        if(found){

            products.remove(idx);

            System.out.println("\nProduct "+id+" is being deleted.\n");

        }

        else

        System.out.println("\nProduct "+id+" is not found in the Inventory.\n");

    }

    static void addProduct(){

        System.out.println("Enter Product ID: ");

        int id = sc.nextInt();

        sc.nextLine();

        System.out.println("Enter Product Name: ");

        String name = sc.nextLine();

        System.out.println("Enter Product Quantity");

        String quantity = sc.nextLine();

        System.out.println("Enter Product Price");

        int price = sc.nextInt();

        products.add(new Product(id, name, quantity, price));

        System.out.println("\nProduct Added Successfully :)\n");

    }

    static void linearSearch(){

        if(products.isEmpty()){

            System.out.println("No Products Available\n");

            return;

        }

        System.out.println("Enter Product ID to Search");

        int id = sc.nextInt();

        for(Product p: products){

            if(p.productID == id){

                System.out.println("\nProduct is Found in Linear Search :)\n");

                return;

            }

        }

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

    }

    static void binarySearch(){

        if(products.isEmpty()){

            System.out.println("\nNo Products Available\n");

            return;

        }

        System.out.println("Enter Product ID to Search");

        int id = sc.nextInt();

        products.sort(Comparator.comparingInt(p -> p.productID));

        int low = 0, high = products.size()-1;

        while(low<=high){

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

            Product midProduct = products.get(mid);

            if(midProduct.productID == id){

                System.out.println("\nProduct is Found in Binary Search :)\n");

                return;

            }

            else if(midProduct.productID < id){

                low = mid+1;

            }

            else{

                high = mid-1;

            }

        }

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

    }

}

class Product{

    int productID;

    String productName;

    String quantity;

    int price;

    public Product(int productID, String productName, String quantity, int price){

        this.productID = productID;

        this.productName = productName;

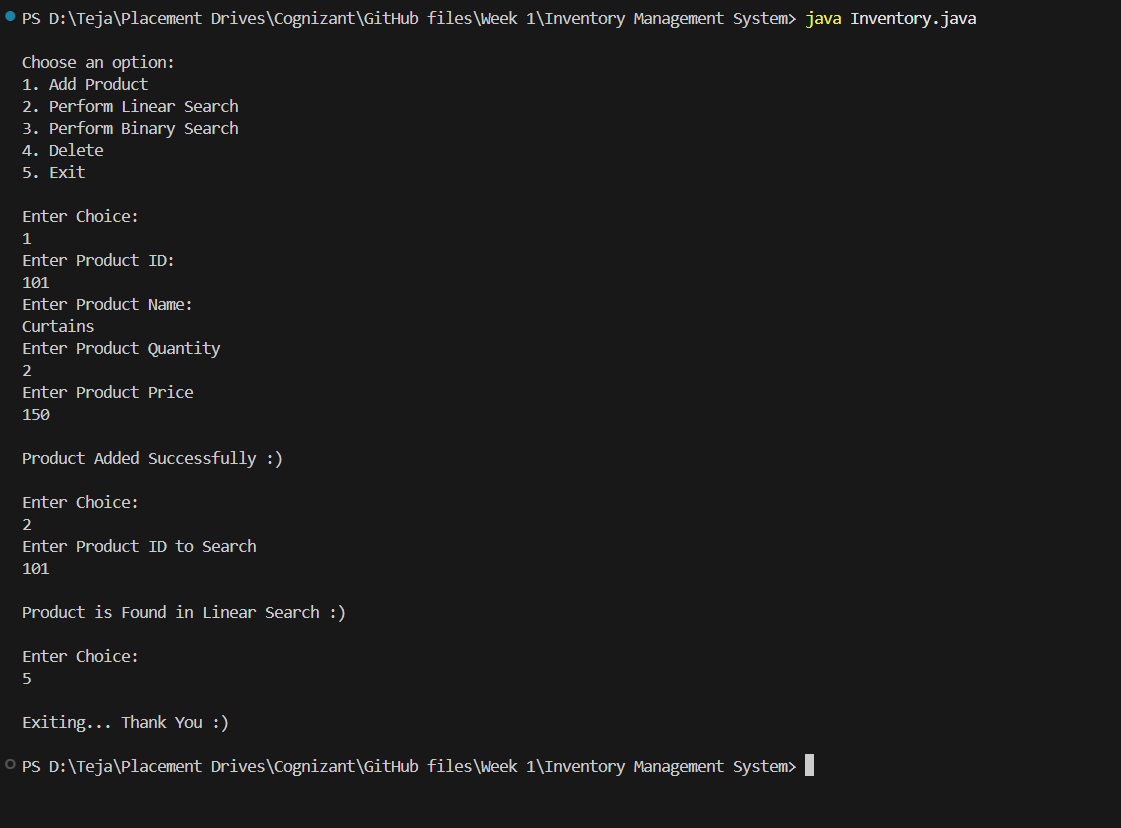
        this.quantity = quantity;

        this.price = price;

    }

}

**Output**

****

**Time Complexity for Add, Update and Delete Operations:**

Add Operation – O (1)

Update Operation – O (n)

Delete Operation – O (n)

**Improvement of Time Complexity:**

Yes, we can improve the time complexities of these the add, update and delete operations with the help of the HashMap Data Structure.

We have to use the ProductId as the Key for the HashMap and all other details as the Value.

Now using this ProductId, the searching, updating, deleting etc. operations become efficient.

**Time Complexities using HashMap:**

Add Operation – O (1)

Update Operation – O (1)

Delete Operation – O (1)