**WEEK 1**

**DATA STRUCTURES AND ALGORITHMS**

**Exercise 1: Inventory Management System**

1Q. Explain why data structures and algorithms are essential in handling large inventories.

A. Data Structures and Algorithms are very essential for handling large inventories because use of efficient ds algorithms reduces response time for search,insert,delete and update operations.they also help in organizing them logically and maintaining performance.

2Q. Discuss the types of data structures suitable for this problem.

A. ArrayList is suitable for ordered storage but is slow in searching and updating. HashMap is ideal for fast search,insert,delete operations as we use keys.

**CODE:**

//Product.java

public class Product {

    //attributes

    private int productId;

    private String productName;

    private int quantity;

    private double price;

    public Product(int productId, String productName, int quantity, double price) {

        this.productId = productId;

        this.productName = productName;

        this.quantity = quantity;

        this.price = price;

    }

    //methods for operations

    public int getProductId() {

        return productId;

    }

    public void setProductName(String productName) {

        this.productName = productName;

    }

    public void setQuantity(int quantity) {

        this.quantity = quantity;

    }

    public void setPrice(double price) {

        this.price = price;

    }

    //displaying details of products

    public void display() {

        System.out.println("ID: "+ productId + ", Name: " + productName +", Quantity: " + quantity + ", Price: " + price);

    }

}

//InventoryManager.java

import java.util.\*;

//managing the inventory

public class InventoryManager {

    //data structure of managing products

    private HashMap<Integer, Product> inventory = new HashMap<>();

    //adding

    public void addProduct(Product product) {

        inventory.put(product.getProductId(), product);

        System.out.println("Product Added Successfully");

    }

    //updating

    public void updateProduct(int productId, String name, int quantity, double price) {

        Product p = inventory.get(productId);

        if (p != null) {

            p.setProductName(name);

            p.setQuantity(quantity);

            p.setPrice(price);

            System.out.println("Product Updated Sucessfully");

        } else {

            System.out.println("Product Not Found!");

        }

    }

    //deleting

    public void deleteProduct(int productId) {

        if (inventory.remove(productId) != null) {

            System.out.println("Product Removed Successfully");

        } else {

            System.out.println("Product Not Found!");

        }

    }

    //displaying

    public void displayAllProducts() {

        if (inventory.isEmpty()) {

            System.out.println("No products in inventory.");

            return;

        }

        for (Product p : inventory.values()) {

            p.display();

        }

    }

}

//Main.java

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        InventoryManager manager = new InventoryManager();

        Scanner sc = new Scanner(System.in);

        int choice;

        //using dowhile

        do {

            System.out.println("\nINENTORY MANAGEMENT SYSTEM");

            System.out.println("\n1. Add Product\n2. Update Product\n3. Delete Product\n4. Display Products\n5. Exit");

            System.out.print("Enter Your Choice: ");

            choice = sc.nextInt();

            //switchcase for choosing the ops

            switch (choice) {

                case 1:

                    System.out.print("Enter ID, Name, Quantity, Price: ");

                    int id = sc.nextInt();

                    String name = sc.next();

                    int qty = sc.nextInt();

                    double price = sc.nextDouble();

                    manager.addProduct(new Product(id, name, qty, price));

                    break;

                case 2:

                    System.out.print("Enter ID to update: ");

                    int uid = sc.nextInt();

                    System.out.print("Enter new Name, Quantity, Price: ");

                    String uname = sc.next();

                    int uqty = sc.nextInt();

                    double uprice = sc.nextDouble();

                    manager.updateProduct(uid, uname, uqty, uprice);

                    break;

                case 3:

                    System.out.print("Enter ID to delete: ");

                    int did = sc.nextInt();

                    manager.deleteProduct(did);

                    break;

                case 4:

                    manager.displayAllProducts();

                    break;

                case 5:

                    System.out.println("Exiting...");

                    break;

                default:

                    System.out.println("Invalid choice.");

            }

        } while (choice != 5);

        sc.close();

    }}

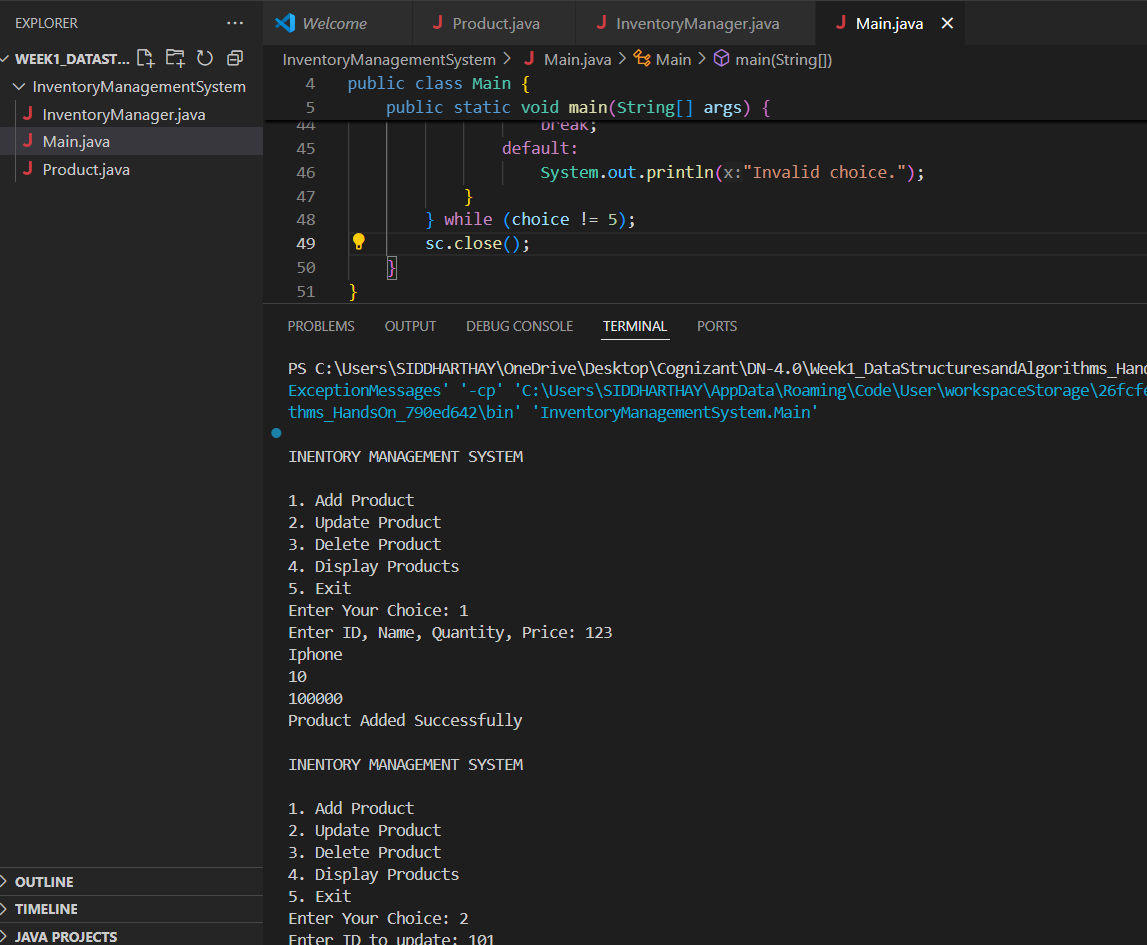
3Q. Analyze the time complexity of each operation (add, update, delete) in your chosen data structure

A. Using HashMap the tc for ADD : O(1), UPADTE : O(1), DELETE : O(1) as we do each operation using the keys.

4Q. Discuss how you can optimize these operations.

A. we can use TreeMap for getting sorted product id’s, databases like mysql for real word applications.

**OUTPUT:**

****

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

1Q. Explain Big O notation and how it helps in analyzing algorithms.

A. Big O notation describes the complexity of an algorithms in terms of time and space,and shows how time or space complexities increase with input size

2Q. Describe the best, average, and worst-case scenarios for search operations.

A. Search Algorithm best average worst

Linear Search O(1) O(n) O(n)

Binary Search O(1) O(logn) O(logn)

**CODE :**

//Product.java

public class Product {

    //atributes

    private int productId;

    private String productName;

    private String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    //ops

    public int getProductId() {

        return productId;

    }

    public String getProductName() {

        return productName;

    }

    public String getCategory() {

        return category;

    }

    public void display() {

        System.out.println("ID: " + productId + ", Name: " + productName + ", Category: " + category);

    }

}

//Search.java

import java.util.\*;

public class Search{

    //linear search by name

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

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

            if (products[i].getProductName().equalsIgnoreCase(name)) {

                return i;

            }

        }

        return -1;

    }

    //binary search by name (on sorted array)

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

        int low = 0;

        int high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int compare = products[mid].getProductName().compareToIgnoreCase(name);

            if (compare == 0) return mid;

            else if (compare < 0) low = mid + 1;

            else high = mid - 1;

        }

        return -1;

    }

    //sorting array for bs

    public static void sortProductsByName(Product[] products) {

        Arrays.sort(products, Comparator.comparing(Product::getProductName, String.CASE\_INSENSITIVE\_ORDER));

    }

}

//Main.java

public class Main {

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(102, "Shoes", "Fashion"),

            new Product(103, "Smartphone", "Electronics"),

            new Product(104, "CricketBat", "Sports")

        };

        String key = "CricketBat";

        //linear search

        int lidx = Search.linearSearch(products, key);

        System.out.println("Linear Search:");

        if (lidx != -1) products[lidx].display();

        else System.out.println("Product not found.");

        // Binary Search (requires sorting first)

        Search.sortProductsByName(products);

        int bidx = Search.binarySearch(products, key);

        System.out.println("Binary Search:");

        if (bidx != -1) products[bidx].display();

        else System.out.println("Product not found.");

    }

}

3Q. Compare the time complexity of linear and binary search algorithms.

A. Search Algorithm best average worst

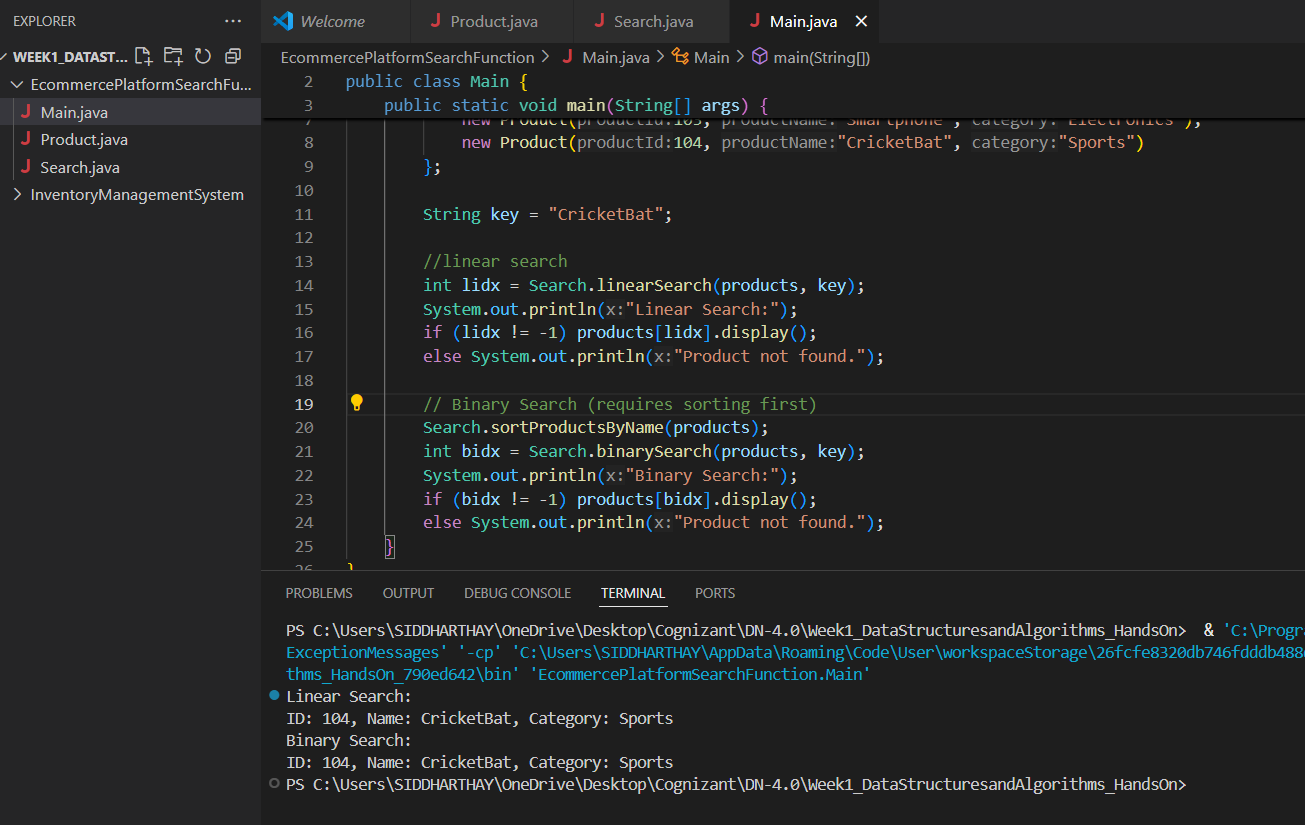
Linear Search O(1) O(n) O(n)

Binary Search O(1) O(logn) O(logn)

4Q. Discuss which algorithm is more suitable for your platform and why.

A. Binary Search is more suitable for platforms like e commerce due to its requirements for fast accessing and managing products nature as we can sort the products in advance which is needed for binary search where linear search is only suitable for small or unsorted data.

**OUTPUT :**

****

**Exercise 3: Sorting Customer Orders**

**CODE :**

//Order.java

public class Order {

    //attributes

    private int orderId;

    private String customerName;

    private double totalPrice;

    public Order(int orderId, String customerName, double totalPrice) {

        this.orderId = orderId;

        this.customerName = customerName;

        this.totalPrice = totalPrice;

    }

    //ops

    public double getTotalPrice() {

        return totalPrice;

    }

    public void display() {

        System.out.println("Order ID: " + orderId + ", Customer: " + customerName + ", Total: " + totalPrice);

    }

}

//Sort.java

//sorting algos

public class Sort {

    //bubble sort

    public static void bubbleSort(Order[] orders) {

        int n = orders.length;

        for (int i=0; i<n-1; i++) {

            boolean swapped = false;

            for (int j=0; j<n-i-1; j++) {

                if (orders[j].getTotalPrice() < orders[j+1].getTotalPrice()) {

                    Order temp = orders[j];

                    orders[j] = orders[j+1];

                    orders[j+1] = temp;

                    swapped = true;

                }

            }

            if (!swapped) break;

        }

    }

    //quick sort

    public static void quickSort(Order[] orders, int low, int high) {

        if (low < high) {

            int pi = partition(orders, low, high);

            quickSort(orders, low, pi-1);

            quickSort(orders, pi+1, high);

        }

    }

    private static int partition(Order[] orders, int low, int high) {

        double pivot = orders[high].getTotalPrice();

        int i = low-1;

        for (int j=low; j<high; j++) {

            if (orders[j].getTotalPrice() > pivot) {

                i++;

                Order temp = orders[i];

                orders[i] = orders[j];

                orders[j] = temp;

            }

        }

        Order temp = orders[i+1];

        orders[i+1] = orders[high];

        orders[high] = temp;

        return i+1;

    }

}

//Main.java

public class Main {

    public static void main(String[] args) {

        // for bubble sort

        Order[] orders = {

            new Order(1, "Siddhu", 7070),

            new Order(2, "Ram", 800),

            new Order(3, "Ashs", 2400),

            new Order(4, "Chai", 1500)

        };

        System.out.println("Before Sorting Orders:");

        for (Order o : orders) o.display();

        //bubble sort

        Sort.bubbleSort(orders);

        System.out.println("\nBubble Sort:");

        for (Order o : orders) o.display();

        //for quick sort

        orders = new Order[]{

            new Order(1, "Siddhu", 7070),

            new Order(2, "Ram", 800),

            new Order(3, "Ashs", 2400),

            new Order(4, "Chai", 1500)

        };

        //quick sort

        Sort.quickSort(orders, 0, orders.length-1);

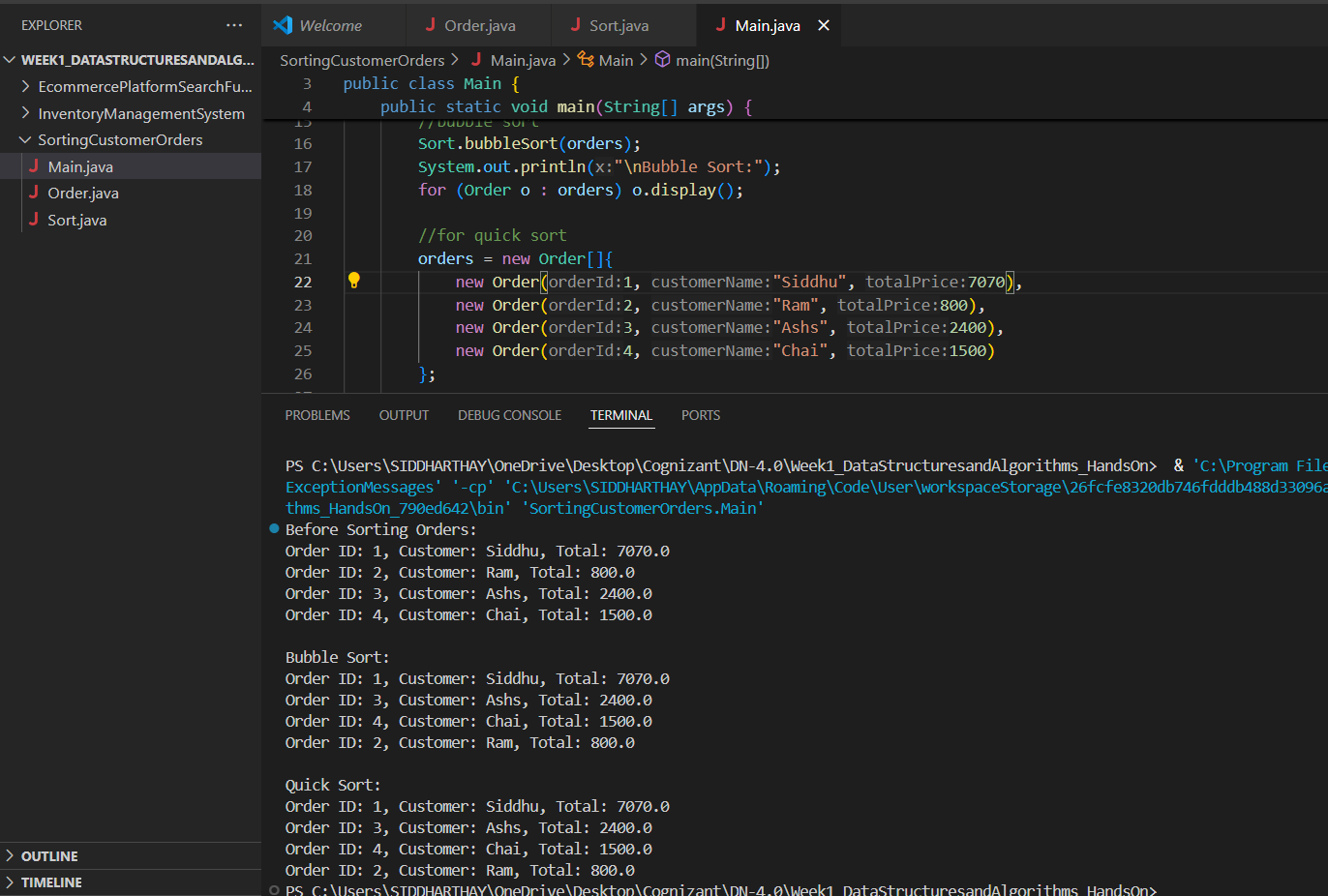
        System.out.println("\nQuick Sort:");

        for (Order o : orders) o.display();

    }

}

**OUTPUT :**

****

**Exercise 4: Employee Management System**

**CODE :**

//Employee.java

public class Employee {

    //attributes

    private int employeeId;

    private String name;

    private String position;

    private double salary;

    public Employee(int employeeId, String name, String position, double salary) {

        this.employeeId = employeeId;

        this.name = name;

        this.position = position;

        this.salary = salary;

    }

    //ops

    public int getEmployeeId() {

        return employeeId;

    }

    public void display() {

        System.out.println("ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: " + salary);

    }

}

//Manager.java

public class Manager {

    private Employee[] employees;

    private int size;

    public Manager(int capacity) {

        employees = new Employee[capacity];

        size = 0;

    }

    //adding

    public void addEmployee(Employee emp) {

        if (size<employees.length) {

            employees[size++] = emp;

            System.out.println("Employee Added");

        } else {

            System.out.println("Employee Capacity is full");

        }

    }

    //searching

    public Employee searchEmployee(int id) {

        for (int i=0; i<size; i++) {

            if (employees[i].getEmployeeId()==id) {

                return employees[i];

            }

        }

        return null;

    }

    //deleting

    public void deleteEmployee(int id) {

        for (int i=0; i<size; i++) {

            if (employees[i].getEmployeeId() == id) {

                // Shift elements to the left

                for (int j=i; j<size-1; j++) {

                    employees[j] = employees[j+1];

                }

                employees[size-1] = null;

                size--;

                System.out.println("Employee removed");

                return;

            }

        }

        System.out.println("Employee not found");

    }

    //displaying

    public void displayAllEmployees() {

        if (size==0) {

            System.out.println("No employee records found.");

        } else {

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

                employees[i].display();

            }

        }

    }

}

//Main.java

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Manager manager = new Manager(10);

        int choice;

        //using dowhile

        do {

            System.out.println("\nWELCOME TO EMPLOYEE MANAGEMENT SYSTEM");

            System.out.println("\n1. Add Employee\n2. Search Employee\n3. Delete Employee\n4. Display All\n5. Exit");

            System.out.print("Enter choice: ");

            choice = sc.nextInt();

            //switch case for choosing opts

            switch (choice) {

                case 1:

                    System.out.print("Enter ID, Name, Position, Salary: ");

                    int id = sc.nextInt();

                    String name = sc.next();

                    String pos = sc.next();

                    double salary = sc.nextDouble();

                    manager.addEmployee(new Employee(id, name, pos, salary));

                    break;

                case 2:

                    System.out.print("Enter ID to search: ");

                    int sid = sc.nextInt();

                    Employee emp = manager.searchEmployee(sid);

                    if (emp != null) emp.display();

                    else System.out.println("Employee not found.");

                    break;

                case 3:

                    System.out.print("Enter ID to remove: ");

                    int did = sc.nextInt();

                    manager.deleteEmployee(did);

                    break;

                case 4:

                    manager.displayAllEmployees();

                    break;

                case 5:

                    System.out.println("Exited");

                    break;

                default:

                    System.out.println("Invalid choice");

            }

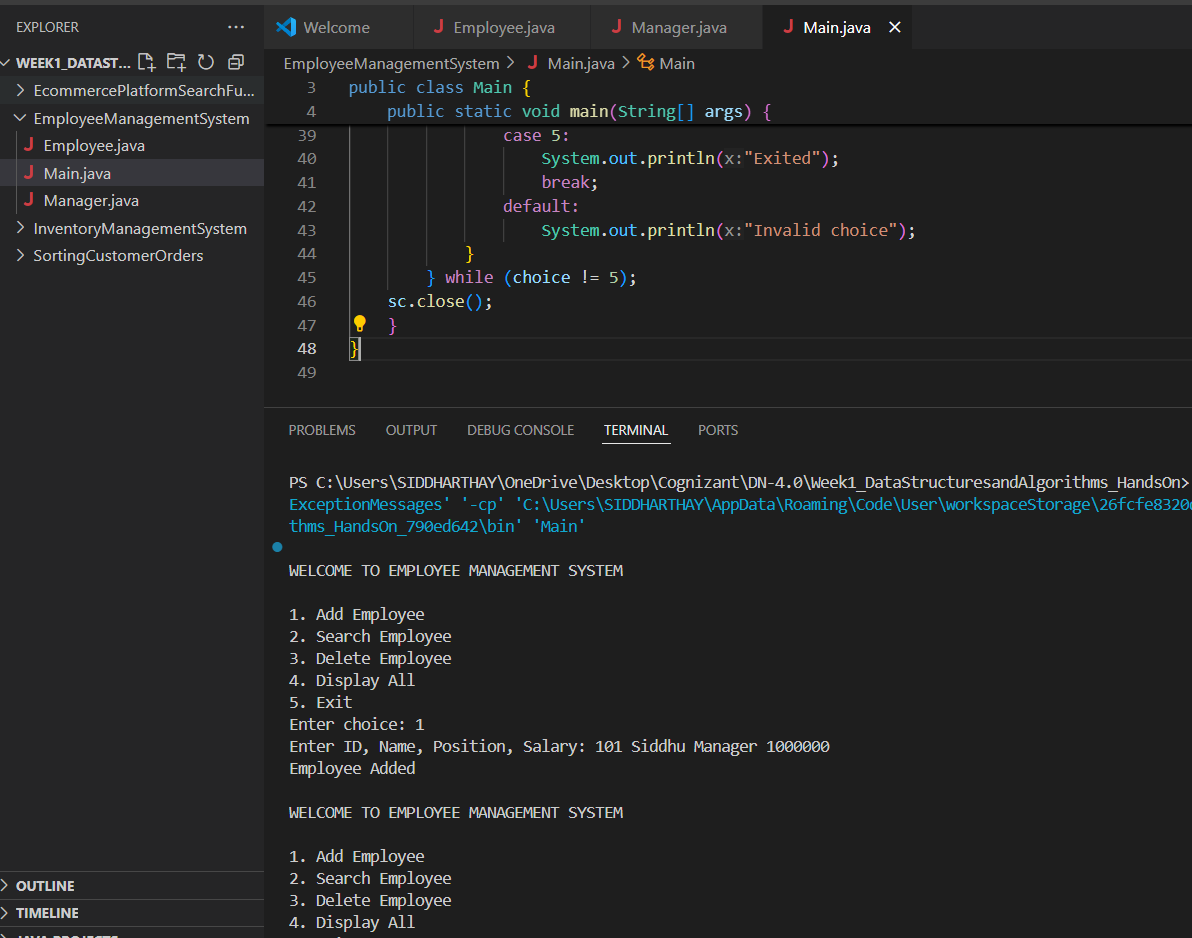
        } while (choice != 5);

    sc.close();

    }

}

**OUTPUT :**

****

**Exercise 5: Task Management System**

**CODE :**

//Task.java

public class Task {

    //atributes

    private int taskId;

    private String taskName;

    private String status;

    public Task(int taskId, String taskName, String status) {

        this.taskId = taskId;

        this.taskName = taskName;

        this.status = status;

    }

    //ops

    public int getTaskId() {

        return taskId;

    }

    public void display() {

        System.out.println("Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

    }

}

//Node.java

public class Node {

    Task task;

    Node next;

    public Node(Task task){

        this.task = task;

        this.next = null;

    }}

//TaskManager.java

public class TaskManager {

    private Node head;

    //adding task

    public void addTask(Task task) {

        Node newNode = new Node(task);

        if (head==null) {

            head = newNode;

        } else {

            Node temp = head;

            while (temp.next!=null) {

                temp = temp.next;

            }

            temp.next = newNode;

        }

        System.out.println("Task added");

    }

    //searching

    public Task searchTask(int taskId) {

        Node temp = head;

        while (temp!=null) {

            if (temp.task.getTaskId()==taskId) {

                return temp.task;

            }

            temp = temp.next;

        }

        return null;

    }

    //deleting

    public void deleteTask(int taskId) {

        if (head==null) {

            System.out.println("Task list is empty");

            return;

        }

        if (head.task.getTaskId()==taskId) {

            head = head.next;

            System.out.println("Task deleted");

            return;

        }

        Node current = head;

        Node prev = null;

        while (current!=null && current.task.getTaskId()!=taskId) {

            prev = current;

            current = current.next;

        }

        if (current!=null) {

            prev.next = current.next;

            System.out.println("Task deleted");

        } else {

            System.out.println("Task not found");

        }

    }

    //displaying

    public void displayTasks() {

        Node temp = head;

        if (temp==null) {

            System.out.println("No tasks in the list");

            return;

        }

        while (temp!=null) {

            temp.task.display();

            temp = temp.next;

        }}}

//Main.java

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        TaskManager manager = new TaskManager();

        int choice;

        //using dowhile

        do {

            System.out.println("WELCOME TO TASK MANAGEMENT SYSTEM");

            System.out.println("\n1. Add Task\n2. Search Task\n3. Delete Task\n4. Display All\n5. Exit");

            System.out.print("Enter choice: ");

            choice = sc.nextInt();

            // switch for opts

            switch (choice) {

                case 1:

                    System.out.print("Enter Task ID, Name, Status: ");

                    int id = sc.nextInt();

                    String name = sc.next();

                    String status = sc.next();

                    manager.addTask(new Task(id, name, status));

                    break;

                case 2:

                    System.out.print("Enter Task ID to search: ");

                    int sid = sc.nextInt();

                    Task t = manager.searchTask(sid);

                    if (t!=null) t.display();

                    else System.out.println("Task not found.");

                    break;

                case 3:

                    System.out.print("Enter Task ID to delete: ");

                    int did = sc.nextInt();

                    manager.deleteTask(did);

                    break;

                case 4:

                    manager.displayTasks();

                    break;

                case 5:

                    System.out.println("Exited.");

                    break;

                default:

                    System.out.println("Invalid choice");

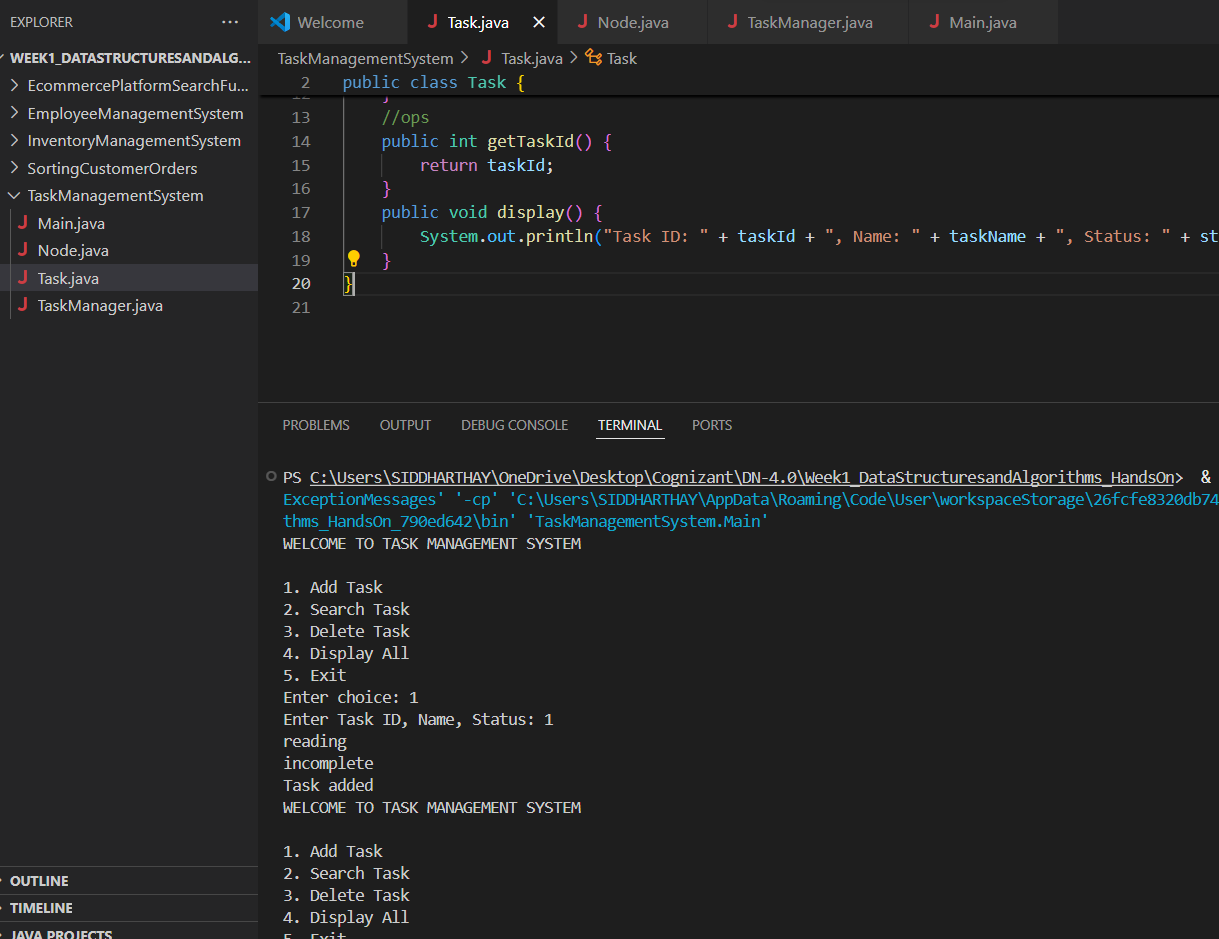
            }

        } while (choice!=5);

    sc.close();

    }}

**OUTPUT :**

****

**Exercise 6: Library Management System**

**CODE :**

//Book.java

public class Book {

    //attributes

    private int bookId;

    private String title;

    private String author;

    public Book(int bookId, String title, String author) {

        this.bookId = bookId;

        this.title = title;

        this.author = author;

    }

    //ops

    public String getTitle() {

        return title;

    }

    public void display() {

        System.out.println("Book ID: " + bookId + ", Title: " + title + ", Author: " + author);

    }

}

//Library.java

import java.util.\*;

public class Library {

    //linear search by title

    public static int linearSearch(Book[] books, String title) {

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

            if (books[i].getTitle().equalsIgnoreCase(title)) {

                return i;

            }

        }

        return -1;

    }

    //binary search by title

    public static int binarySearch(Book[] books, String title) {

        int low = 0;

        int high = books.length-1;

        while (low<=high) {

            int mid = (low+high) / 2;

            int cmp = books[mid].getTitle().compareToIgnoreCase(title);

            if (cmp==0) return mid;

            else if (cmp<0) low = mid+1;

            else high = mid-1;

        }

        return -1;

    }

    //sorting books by title for bs

    public static void sortBooksByTitle(Book[] books) {

        Arrays.sort(books, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));

    }

}

//Main.java

public class Main {

    public static void main(String[] args) {

        Book[] books = {

            new Book(101, "abc", "Siddhu"),

            new Book(102, "def", "Ram"),

            new Book(103, "ghi", "Ashs"),

            new Book(104, "jkl", "Chai")

        };

        String searchTitle = "jkl";

        //linear search

        int lidx = Library.linearSearch(books, searchTitle);

        System.out.println("Linear Search:");

        if (lidx != -1) books[lidx].display();

        else System.out.println("Book not found");

        //sorting for bs

        Library.sortBooksByTitle(books);

        //binary search

        int bidx = Library.binarySearch(books, searchTitle);

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

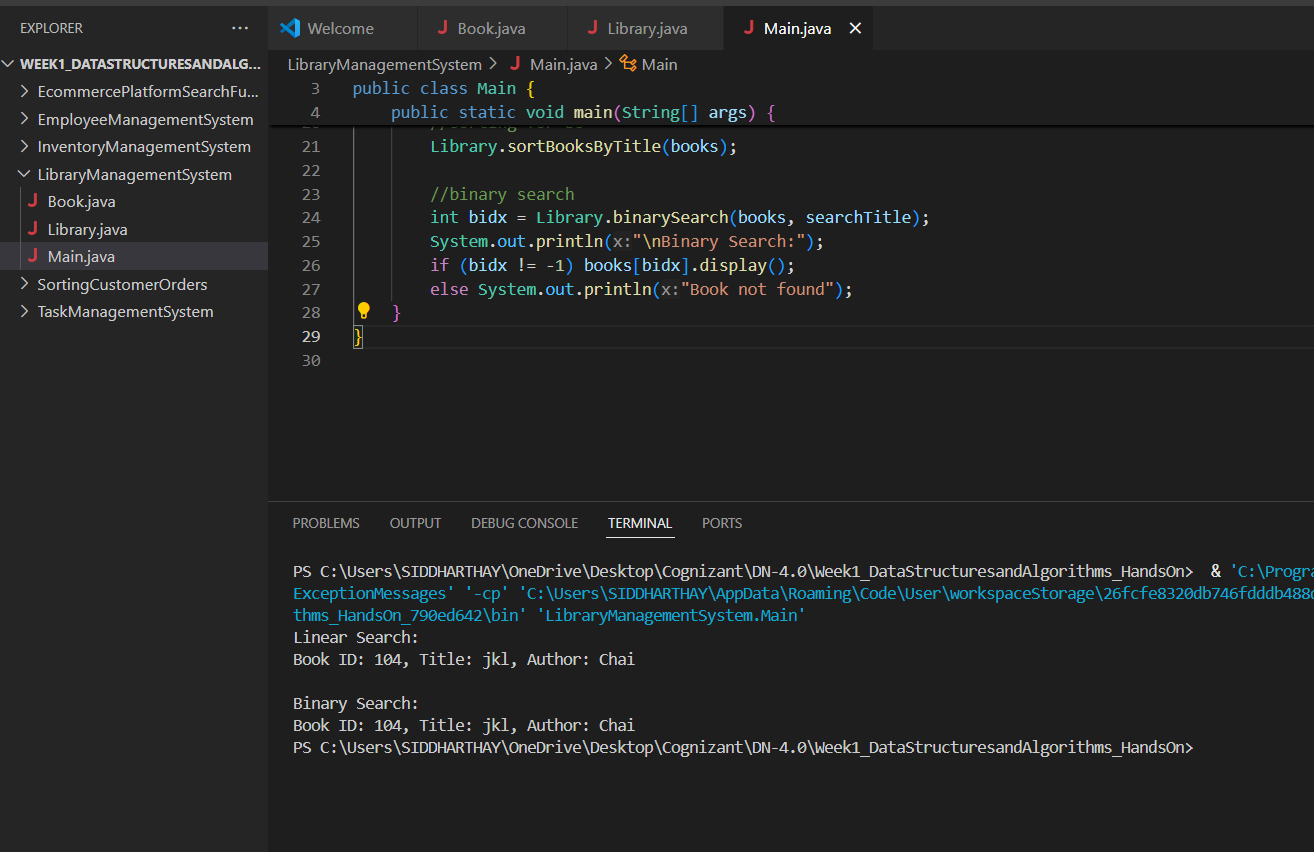
        if (bidx != -1) books[bidx].display();

        else System.out.println("Book not found");

    }

}

**OUTPUT :**

****

**Exercise 7: Financial Forecasting**

1Q.Explain the concept of recursion and how it can simplify certain problems.

A. Recursion is the way in which a function calls itself to solve smaller instances of the same problem which simplifies the problems which have repeating pattern example factorial,fibonnaci

**CODE :**

//FinancialForecast.java

public class FinancialForecast {

    //recursion method to calculate future values

    public static double predictFutureValue(double currValue, double growthRate, int years) {

        if (years==0) {

            return currValue;

        }

        return (1+growthRate)\*predictFutureValue(currValue,growthRate,years-1);

    }

}

//Main.java

public class Main {

    public static void main(String[] args) {

        double initialValue = 10000.0;   //init investment

        double growthRate = 0.10;        //assuming 10% as growthrate

        int years = 5;                   //for 5y

        double futureValue = FinancialForecast.predictFutureValue(initialValue, growthRate, years);

        System.out.printf("Predicted Future Value : %.2f\n", futureValue);

    }

}

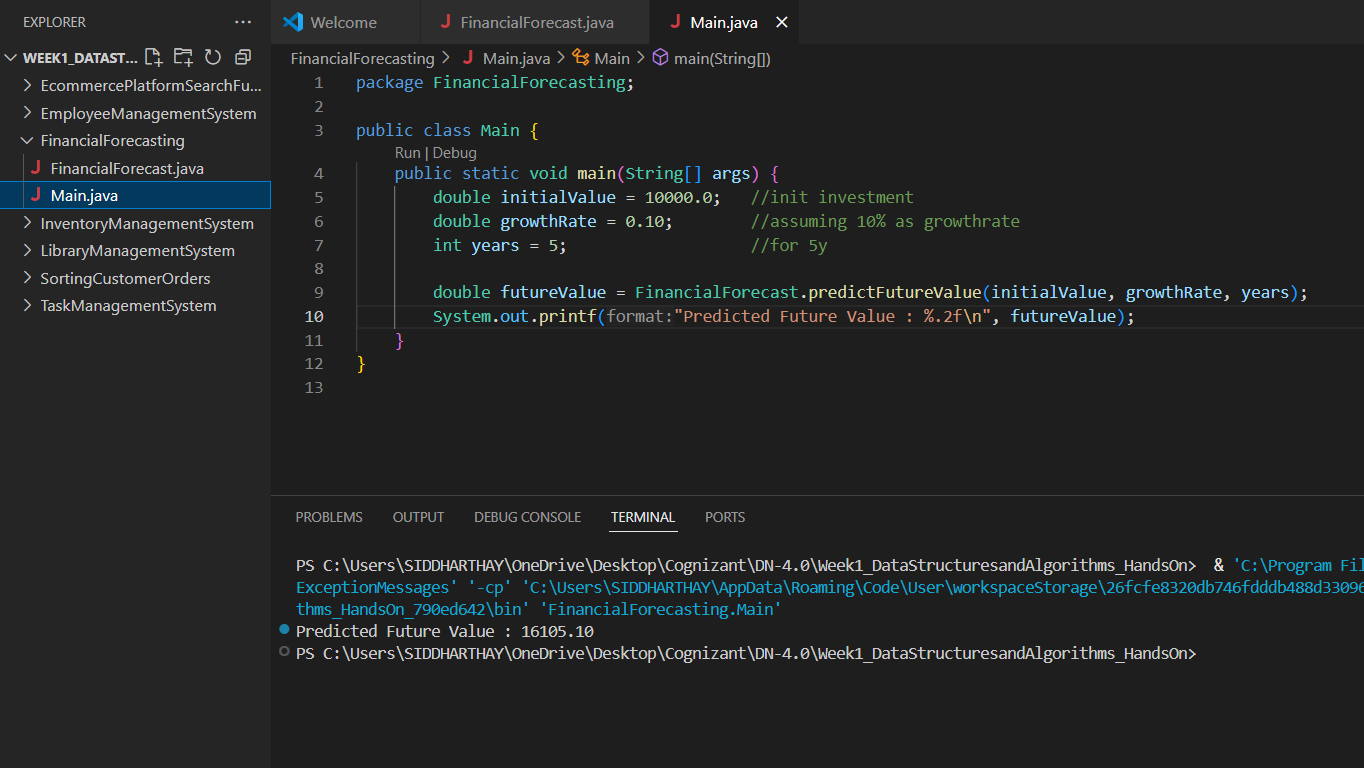
2Q. Discuss the time complexity of your recursive algorithm.

A. TC : O(n) since we call function once per year and for each call as year decreases its O(n)

3Q. Explain how to optimize the recursive solution to avoid excessive computation.

A. To Optimize the recursive solution to avoid excessive computation we can use memoization as it avoids duplicate computations

**OUTPUT :**

****