2. DataStructures and Algorithms

# 1. Inventory Management System

Product.java

package InventoryManagementSystem;

public class Product{

    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;

    }

    public int getProducID(){

    return productID;

    }

    public String getProductName(){

        return productName;

    }

    public int getQuantity(){

        return quantity;

    }

    public double getProductPrice(){

        return price;

    }

    public void setQuantity(int quantity) { this.quantity = quantity; }

    public void setPrice(double price) { this.price = price; }

    @Override

    public String toString() {

        return "Product[ID=" + productID + ", Name=" + productName + ", Qty=" + quantity + ", Price=RS " + price + "]";

    }

}

InventoryManager.java

package InventoryManagementSystem;

import java.util.HashMap;

public class InventoryManager {

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

    public void addProduct(Product product){

        if(!inventory.containsKey(product.getProducID())){

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

            System.out.println("Product added: "+product);

        }

        else{

            System.out.println("Product ID already exists");

        }

    }

    public void updateProduct(int productID, int quantity,double price ){

        Product p=inventory.get(productID);

        if(p!=null){

            p.setQuantity(quantity);

            p.setPrice(price);

            System.out.println("Product updated: "+p);

        }

        else{

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

        }

    }

    public void deleteProduct(int productId) {

        if (inventory.containsKey(productId)) {

            Product removed = inventory.remove(productId);

            System.out.println("Product removed: " + removed);

        } else {

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

        }

    }

    public void displayAll() {

        if (inventory.isEmpty()) {

            System.out.println("Inventory is empty.");

        } else {

            System.out.println("Inventory:");

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

                System.out.println(p);

            }

        }

    }

}

Main.java

package InventoryManagementSystem;

public class Main {

    public static void main(String[] args) {

        InventoryManager manager = new InventoryManager();

        Product p1 = new Product(101, "Laptop", 10, 55000.00);

        Product p2 = new Product(102, "Mouse", 50, 500.00);

        Product p3 = new Product(103, "Keyboard", 30, 800.00);

        manager.addProduct(p1);

        manager.addProduct(p2);

        manager.addProduct(p3);

        manager.displayAll();

        manager.updateProduct(102, 40, 480.00);

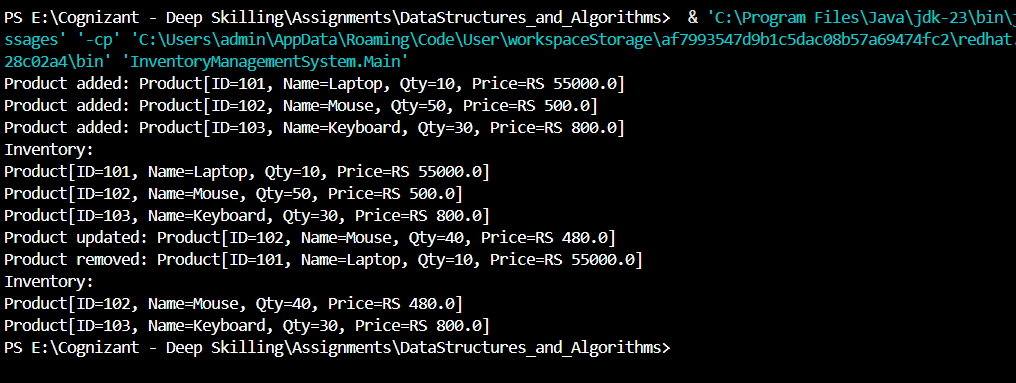
        manager.deleteProduct(101);

        manager.displayAll();

    }

}

Output:



Complexity Analysis

|  |  |  |
| --- | --- | --- |
| **Operation** | **Data Structure** | **Time Complexity** |
| addProduct() | HashMap | O(1) average |
| updateProduct() | HashMap | O(1) average |
| deleteProduct() | HashMap | O(1) average |
| displayAll() | HashMap.values() traversal | O(n) |

# 2: E-commerce Platform Search Function

Product.java

package EcommercePlatformSearch;

public class Product {

    int productID;

    String productName;

    String category;

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

        this.productID=productID;

        this.productName=productName;

        this.category=category;

    }

    public String toString(){

        return "[ID: " + productID + ", Name: " + productName + ", Category: " + category + "]";

    }

}

SearchUtil.java

package EcommercePlatformSearch;

public class SearchUtil {

    public static Product linearSearch(Product[] product, int targetID){

        for(Product p:product){

            if(p.productID==targetID){

                return p;

            }

        }

        return null;

    }

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

        int low=0,high=products.length-1;

        while(low<=high){

            int mid=(low+high)/2;

            if (products[mid].productID==targetId){

                return products[mid];

            }

            else if(products[mid].productID < targetId){

                low=mid+1;

            }

            else{

                high=mid-1;

            }

        }

        return null;

    }

}

Main.java

package EcommercePlatformSearch;

import java.util.Arrays;

public class Main {

    public static void main(String[] args) {

        Product[] products = {

            new Product(102, "Laptop", "Electronics"),

            new Product(101, "T-Shirt", "Clothing"),

            new Product(105, "Shoes", "Footwear"),

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

        };

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

        Product p1 = SearchUtil.linearSearch(products, 105);

        System.out.println(p1);

        Arrays.sort(products, (a, b) -> a.productID - b.productID);

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

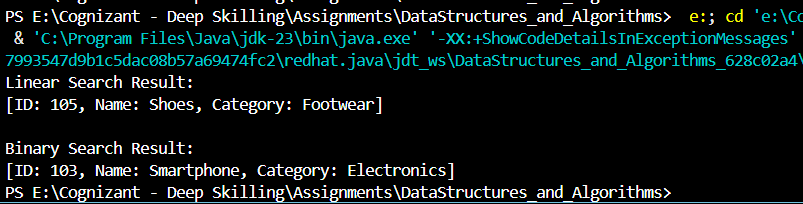
        Product p2 = SearchUtil.binarySearch(products, 103);

        System.out.println(p2);

    }

}

Output:



**Complexity: Best, Average, and Worst-Case Scenarios**

|  |  |  |  |
| --- | --- | --- | --- |
| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| **Linear Search** | O(1) – first item | O(n/2) → O(n) | O(n) |
| **Binary Search** | O(1) – mid item | O(log n) | O(log n) |

# 3: Sorting Customer Orders

Order.java

package SortingCustomerOrders;

public class Order {

    int orderId;

    String customerName;

    double totalPrice;

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

        this.orderId = orderId;

        this.customerName = customerName;

        this.totalPrice = totalPrice;

    }

    public String toString() {

        return "[OrderID: " + orderId + ", Name: " + customerName + ", Total: Rs " + totalPrice + "]";

    }

}

SortUtil.java

package SortingCustomerOrders;

public class SortUtil {

    public static void bubbleSort(Order[] orders) {

        int n = orders.length;

        boolean swapped;

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

            swapped = false;

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

                if (orders[j].totalPrice > orders[j + 1].totalPrice) {

                    Order temp = orders[j];

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

                    orders[j + 1] = temp;

                    swapped = true;

                }

            }

            if (!swapped) break;

        }

    }

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

        if (low < high) {

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

            quickSort(orders, low, pivotIndex - 1);

            quickSort(orders, pivotIndex + 1, high);

        }

    }

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

        double pivot = orders[high].totalPrice;

        int i = low - 1;

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

            if (orders[j].totalPrice < 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

package SortingCustomerOrders;

public class Main {

    public static void main(String[] args) {

        Order[] orders = {

            new Order(201, "Ayshu", 3200.0),

            new Order(202, "Dhiya", 1500.5),

            new Order(203, "Anu", 9800.0),

            new Order(204, "Riya", 6700.75)

        };

        System.out.println("Before Bubble Sort:");

        for (Order o : orders) System.out.println(o);

        SortUtil.bubbleSort(orders);

        System.out.println("\nAfter Bubble Sort:");

        for (Order o : orders) System.out.println(o);

        orders = new Order[] {

            new Order(201, "Ayshu", 3200.0),

            new Order(202, "Dhiya", 1500.5),

            new Order(203, "Anu", 9800.0),

            new Order(204, "Riya", 6700.75)

        };

        System.out.println("\nBefore Quick Sort:");

        for (Order o : orders) System.out.println(o);

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

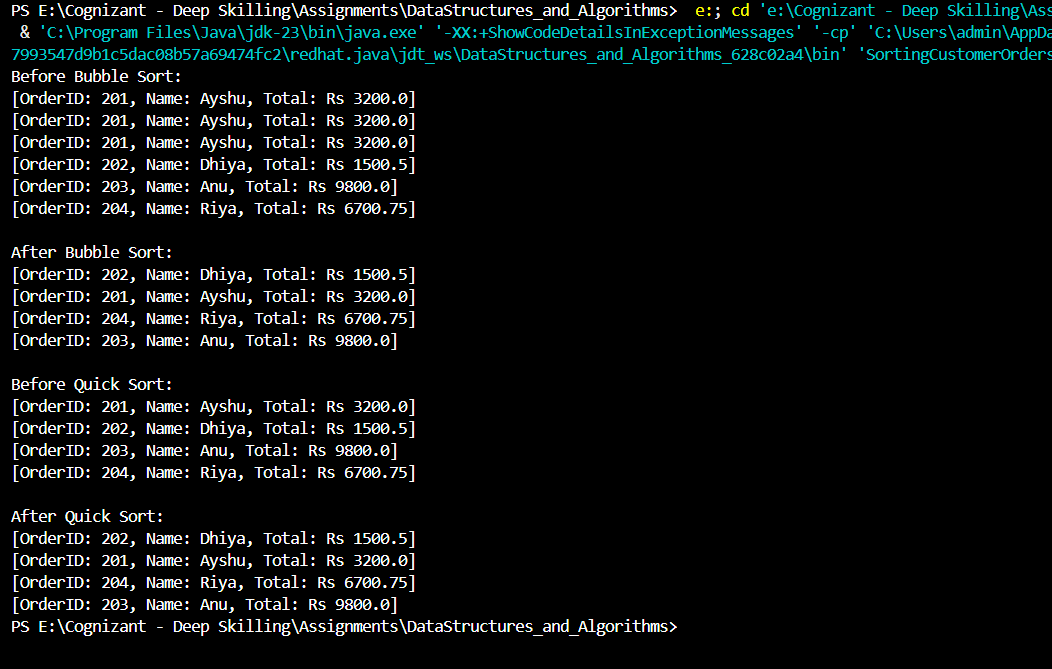
        System.out.println("\nAfter Quick Sort:");

        for (Order o : orders) System.out.println(o);

    }

}

Output:



Complexity Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | **Space Complexity** | **Preferred When** |
| Bubble Sort | O(n²) | O(1) | Small data / educational purpose |
| Quick Sort | O(n log n) avg case | O(log n) | General-purpose fast sorting |

# 4. Employee Management System

Employee.java

package EmployeeManagementSystem;

public class Employee {

    int employeeId;

    String name;

    String position;

    double salary;

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

        this.employeeId = employeeId;

        this.name = name;

        this.position = position;

        this.salary = salary;

    }

    public String toString() {

        return "[ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: Rs " + salary + "]";

    }

}

EmployeeManager.java

package EmployeeManagementSystem;

public class EmployeeManager {

    private Employee[] employees;

    private int size;

    public EmployeeManager(int capacity) {

        employees = new Employee[capacity];

        size = 0;

    }

    public void addEmployee(Employee e) {

        if (size < employees.length) {

            employees[size++] = e;

            System.out.println("Added: " + e);

        } else {

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

        }

    }

    public Employee searchEmployee(int id) {

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

            if (employees[i].employeeId == id)

                return employees[i];

        }

        return null;

    }

    public void displayAll() {

        if (size == 0) {

            System.out.println("No employees to display.");

            return;

        }

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

            System.out.println(employees[i]);

        }

    }

    public void deleteEmployee(int id) {

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

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

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

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

                }

                employees[--size] = null;

                System.out.println("Deleted employee with ID: " + id);

                return;

            }

        }

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

    }

}

Main.java

package EmployeeManagementSystem;

public class Main {

    public static void main(String[] args) {

        EmployeeManager manager = new EmployeeManager(5);

        Employee e1 = new Employee(1, "Ayshu", "Developer", 50000);

        Employee e2 = new Employee(2, "Ravi", "Tester", 40000);

        Employee e3 = new Employee(3, "Meena", "Manager", 75000);

        manager.addEmployee(e1);

        manager.addEmployee(e2);

        manager.addEmployee(e3);

        System.out.println("\nAll Employees:");

        manager.displayAll();

        System.out.println("\nSearching for employee with ID 2:");

        System.out.println(manager.searchEmployee(2));

        System.out.println("\nDeleting employee with ID 1:");

        manager.deleteEmployee(1);

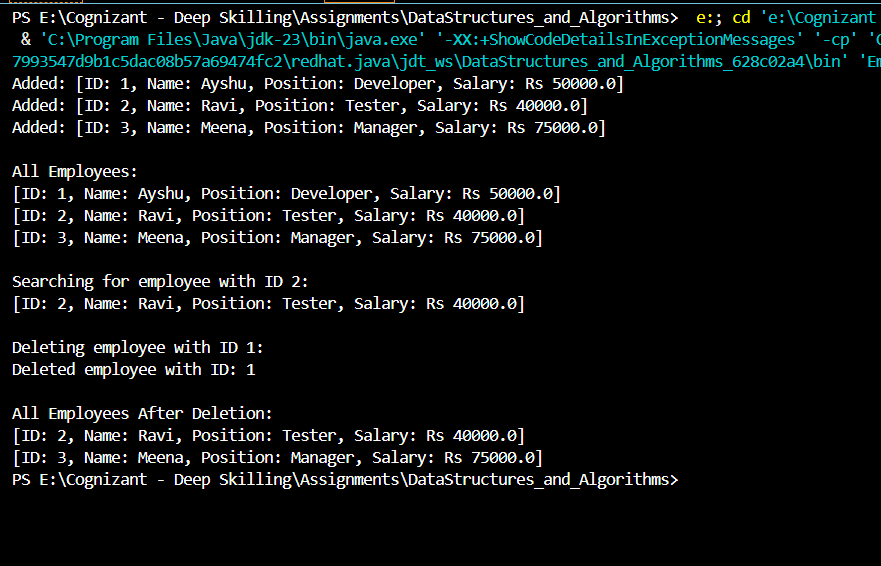
        System.out.println("\nAll Employees After Deletion:");

        manager.displayAll();

    }

}

Output:



Complexity Analysis

|  |  |  |
| --- | --- | --- |
| **Operation** | **Time Complexity** | **Explanation** |
| Add | O(1) | Append at end |
| Search | O(n) | Linear scan by ID |
| Traverse | O(n) | Visit each element |
| Delete | O(n) | Search + shift elements |

# 5. Task Management System

Task.java

package TaskManagementSystem;

public class Task {

    int taskId;

    String taskName;

    String status;

    Task next;

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

        this.taskId = taskId;

        this.taskName = taskName;

        this.status = status;

        this.next = null;

    }

    public String toString() {

        return "[ID: " + taskId + ", Name: " + taskName + ", Status: " + status + "]";

    }

}

TaskManager.java

package TaskManagementSystem;

public class TaskManager {

    private Task head;

    public void addTask(int id, String name, String status) {

        Task newTask = new Task(id, name, status);

        if (head == null) {

            head = newTask;

        } else {

            Task current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newTask;

        }

        System.out.println("Added: " + newTask);

    }

    public void displayTasks() {

        if (head == null) {

            System.out.println("No tasks available.");

            return;

        }

        Task current = head;

        while (current != null) {

            System.out.println(current);

            current = current.next;

        }

    }

    public Task searchTask(int id) {

        Task current = head;

        while (current != null) {

            if (current.taskId == id)

                return current;

            current = current.next;

        }

        return null;

    }

    public void deleteTask(int id) {

        if (head == null) {

            System.out.println("No tasks to delete.");

            return;

        }

        if (head.taskId == id) {

            head = head.next;

            System.out.println("Deleted task with ID: " + id);

            return;

        }

        Task current = head;

        while (current.next != null && current.next.taskId != id) {

            current = current.next;

        }

        if (current.next != null) {

            current.next = current.next.next;

            System.out.println("Deleted task with ID: " + id);

        } else {

            System.out.println("Task ID " + id + " not found.");

        }

    }

}

Main.java

package TaskManagementSystem;

public class Main {

    public static void main(String[] args) {

        TaskManager manager = new TaskManager();

        manager.addTask(1, "Design UI", "Pending");

        manager.addTask(2, "Implement Backend", "In Progress");

        manager.addTask(3, "Testing", "Pending");

        System.out.println("\nAll Tasks:");

        manager.displayTasks();

        System.out.println("\nSearch Task with ID 2:");

        System.out.println(manager.searchTask(2));

        System.out.println("\nDelete Task with ID 1:");

        manager.deleteTask(1);

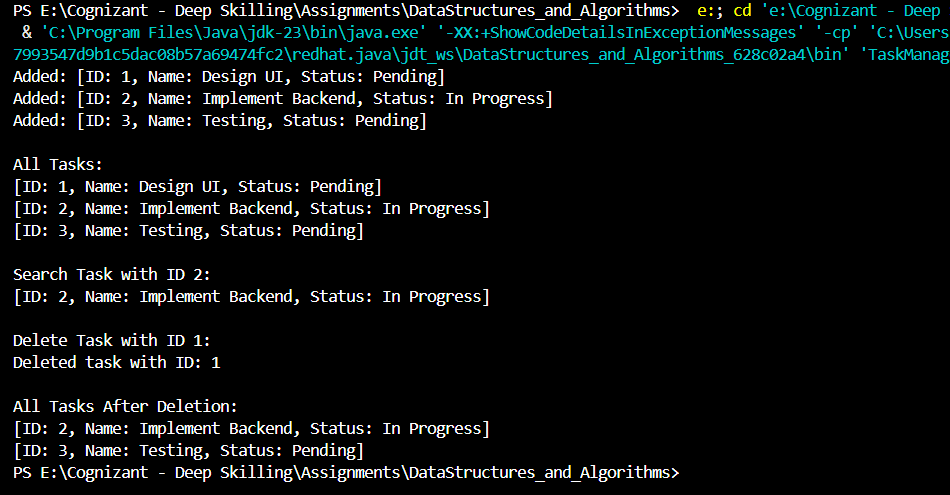
        System.out.println("\nAll Tasks After Deletion:");

        manager.displayTasks();

    }

}

Output



# 6. Library Management System

Book.java

package LibraryManagementSystem;

public class Book {

    int bookId;

    String title;

    String author;

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

        this.bookId = bookId;

        this.title = title;

        this.author = author;

    }

    public String toString() {

        return "[ID: " + bookId + ", Title: " + title + ", Author: " + author + "]";

    }

}

LibraryManager.java

package LibraryManagementSystem;

import java.util.Arrays;

import java.util.Comparator;

public class LibraryManager {

    // Linear search

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

        for (Book book : books) {

            if (book.title.equalsIgnoreCase(title)) {

                return book;

            }

        }

        return null;

    }

    // Binary search

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

        Arrays.sort(books, Comparator.comparing(b -> b.title.toLowerCase())); // Sort by title

        int low = 0;

        int high = books.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

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

            if (result == 0) {

                return books[mid];

            } else if (result < 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

    // Display books

    public static void displayBooks(Book[] books) {

        for (Book book : books) {

            System.out.println(book);

        }

    }

}

Main.java

package LibraryManagementSystem;

public class Main {

    public static void main(String[] args) {

        Book[] books = {

            new Book(101, "The Alchemist", "Paulo Coelho"),

            new Book(102, "Wings of Fire", "A.P.J. Abdul Kalam"),

            new Book(103, "1984", "George Orwell"),

            new Book(104, "Sapiens", "Yuval Noah Harari"),

            new Book(105, "To Kill a Mockingbird", "Harper Lee")

        };

        System.out.println("Library Collection:");

        LibraryManager.displayBooks(books);

        System.out.println("\nSearching for 'Sapiens' using Linear Search:");

        Book linearResult = LibraryManager.linearSearch(books, "Sapiens");

        System.out.println(linearResult != null ? linearResult : "Book not found");

        System.out.println("\nSearching for 'Sapiens' using Binary Search:");

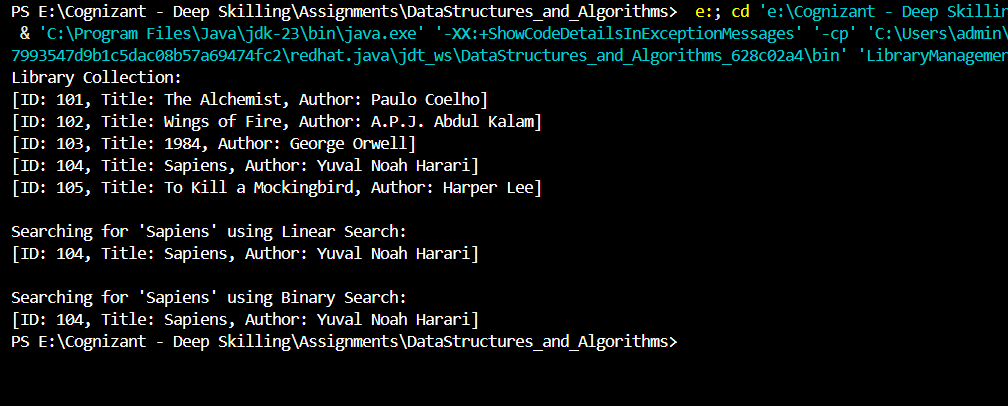
        Book binaryResult = LibraryManager.binarySearch(books, "Sapiens");

        System.out.println(binaryResult != null ? binaryResult : "Book not found");

    }

}

Output:



Complexity Analysis

|  |  |  |
| --- | --- | --- |
| **Operation** | **Linear Search** | **Binary Search** |
| Time Complexity | O(n) | O(log n) |
| Data Requirement | Unsorted | Sorted |

# 7. Financial Forecast.java

FinancialForecast.java

package FinancialForecasting;

public class FinancialForecast {

    public static double futureValue(double presentValue, double growthRate, int years) {

        if (years == 0) {

            return presentValue;

        }

        return futureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

    }

    public static void main(String[] args) {

        double presentValue = 10000;

        double annualGrowthRate = 0.08;

        int forecastYears = 5;

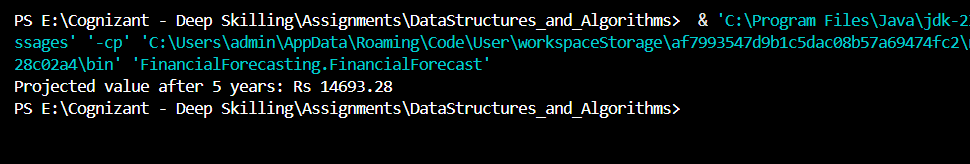
        double result = futureValue(presentValue, annualGrowthRate, forecastYears);

        System.out.printf("Projected value after %d years: Rs %.2f\n", forecastYears, result);

    }

}

Output:



**Complexity**

* Each call depends on futureValue(...) with n - 1, so:

Time Complexity=O(n)

* Due to recursive call stack:

Space Complexity=O(n)