Exercise 1: Inventory Management System

Scenario:

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

Steps:

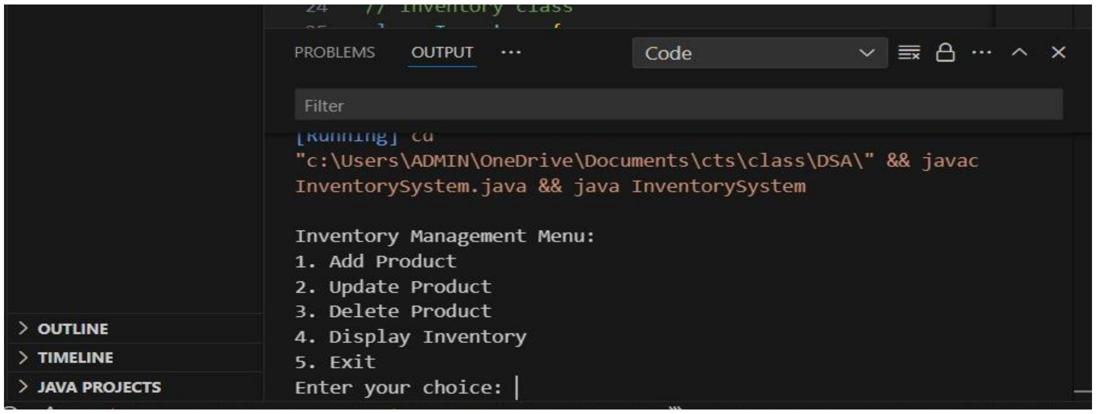
1. **Understand the Problem:**
   - Explain why data structures and algorithms are essential in handling large inventories.
   - Discuss the types of data structures suitable for this problem.
2. **Setup:**
   - Create a new project for the inventory management system.
3. **Implementation:**
   - Define a class Product with attributes like **productId**, **productName**, **quantity**, and **price**.
   - Choose an appropriate data structure to store the products (e.g., ArrayList, HashMap).
   - Implement methods to add, update, and delete products from the inventory.
4. **Analysis:**
   - Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.
   - Discuss how you can optimize these operations.

CODE:

```
import java.util.ArrayList;
import java.util.Scanner;

// Product class
class Product {
  int id;
  String name;
  int quantity;
  double price;

  Product(int id, String name, int quantity, double price) {
    this.id = id;
    this.name = name;
    this.quantity = quantity;
    this.price = price;
  }
```

```
    void print() {
        System.out.println("ID: " + id + ", Name: " + name + ", Quantity: " + quantity + ", Price: " +
price);
    }
}

// Inventory class
class Inventory {
    ArrayList<Product> products = new ArrayList<>();

    void add(Product p) {
        for (Product item : products) {
            if (item.id == p.id) {
                System.out.println("Product ID already exists.");
                return;
            }
        }
        products.add(p);
        System.out.println("Product added.");
    }

    void update(int id, int quantity, double price) {
        for (Product p : products) {
            if (p.id == id) {
                p.quantity = quantity;
                p.price = price;
                System.out.println("Product updated.");
                return;
            }
        }
        System.out.println("Product not found.");
    }

    void delete(int id) {
        for (int i = 0; i < products.size(); i++) {
            if (products.get(i).id == id) {
                products.remove(i);
                System.out.println("Product deleted.");
                return;
            }
        }
        System.out.println("Product not found.");
    }

    void showAll() {
        if (products.isEmpty()) {
            System.out.println("Inventory is empty.");
        } else {
            for (Product p : products) {
                p.print();
            }
        }
    }
}

// Main class
public class InventorySystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Inventory inv = new Inventory();
        int option;

        do {
            System.out.println("\nInventory Management Menu:");
            System.out.println("1. Add Product");
            System.out.println("2. Update Product");
            System.out.println("3. Delete Product");
            System.out.println("4. Display Inventory");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            option = scanner.nextInt();

            switch (option) {
                case 1:
                    System.out.print("Enter Product ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter Product Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter Quantity: ");
                    int qty = scanner.nextInt();
                    System.out.print("Enter Price: ");
                    double price = scanner.nextDouble();
                    inv.add(new Product(id, name, qty, price));
                    break;

                case 2:
                    System.out.print("Enter Product ID to update: ");
                    int uid = scanner.nextInt();
                    System.out.print("Enter New Quantity: ");
                    int newQty = scanner.nextInt();
                    System.out.print("Enter New Price: ");
                    double newPrice = scanner.nextDouble();
                    inv.update(uid, newQty, newPrice);
                    break;

                case 3:
                    System.out.print("Enter Product ID to delete: ");
                    int delId = scanner.nextInt();
                    inv.delete(delId);
                    break;

                case 4:
                    inv.showAll();
                    break;

                case 5:
                    System.out.println("Exiting...");
                    break;

                default:
                    System.out.println("Invalid choice!");
            }
        } while (option != 5);

        scanner.close();
    }
}
```

OUTPUT:



```
[Running] cd
"c:\Users\ADMIN\OneDrive\Documents\cts\class\DSA\" && javac
InventorySystem.java && java InventorySystem

Inventory Management Menu:
1. Add Product
2. Update Product
3. Delete Product
4. Display Inventory
5. Exit
Enter your choice: |
```

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:

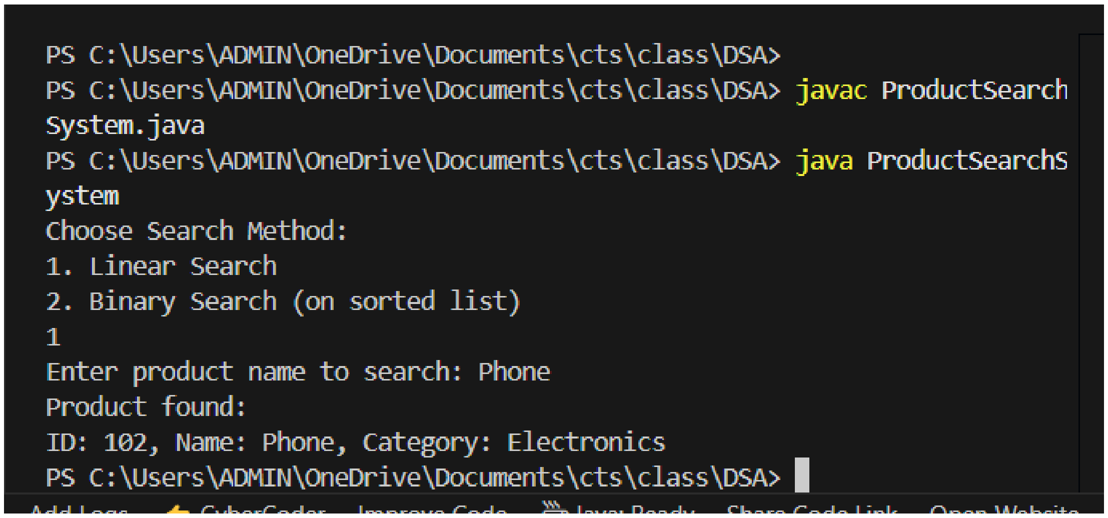
1. **Understand Asymptotic Notation:**
    - Explain Big O notation and how it helps in analyzing algorithms.
    - Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
    - Create a class **Product** with attributes for searching, such as **productId, productName**, 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.

CODE

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;

class Item {
  int id;
  String name;
  String type;

  Item(int id, String name, String type) {
    this.id = id;
    this.name = name;
    this.type = type;
  }

  void showDetails() {
    System.out.println("ID: " + id + ", Name: " + name + ", Category: " + type);
  }
}

public class ItemSearchEngine {

  // Linear Search
  static int performLinearSearch(Item[] items, String targetName) {
    for (int index = 0; index < items.length; index++) {
      if (items[index].name.equalsIgnoreCase(targetName)) {
        return index;
      }
    }
    return -1;
}

// Binary Search (on sorted array)
static int performBinarySearch(Item[] items, String targetName) {
    int start = 0;
    int end = items.length - 1;

    while (start <= end) {
        int middle = (start + end) / 2;
        int comparison = items[middle].name.compareToIgnoreCase(targetName);

        if (comparison == 0) {
            return middle;
        } else if (comparison < 0) {
            start = middle + 1;
        } else {
            end = middle - 1;
        }
    }

    return -1;
}

public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    Item[] inventory = {
        new Item(201, "Laptop", "Electronics"),
        new Item(202, "Phone", "Electronics"),
        new Item(203, "Shirt", "Clothing"),
        new Item(204, "Book", "Stationery"),
        new Item(205, "Shoes", "Footwear")
    };

    System.out.println("Select Search Option:");
    System.out.println("1. Linear Search");
    System.out.println("2. Binary Search (sorted list)");
    System.out.print("Your Choice: ");
    int option = input.nextInt();
    input.nextLine(); // Consume newline

    System.out.print("Enter item name to find: ");
    String itemToFind = input.nextLine();

    int foundIndex = -1;
```

```
        if (option == 1) {
            foundIndex = performLinearSearch(inventory, itemToFind);
        } else if (option == 2) {
            Arrays.sort(inventory, Comparator.comparing(i -> i.name.toLowerCase()));
            foundIndex = performBinarySearch(inventory, itemToFind);
        } else {
            System.out.println("Invalid option selected!");
            input.close();
            return;
        }

        if (foundIndex != -1) {
            System.out.println("Item found:");
            inventory[foundIndex].showDetails();
        } else {
            System.out.println("Item not found.");
        }

        input.close();
    }
}
```

OUTPUT:



```
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA>
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> javac ProductSearch
System.java
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java ProductSearchS
ystem
Choose Search Method:
1. Linear Search
2. Binary Search (on sorted list)
1
Enter product name to search: Phone
Product found:
ID: 102, Name: Phone, Category: Electronics
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA>
```

Exercise 3: Sorting Customer Orders

Scenario:

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

Steps:

1. **Understand Sorting Algorithms:**
    - Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).

2. **Setup:**
    - Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.
3. **Implementation:**
    - mplement **Bubble Sort** to sort orders by **totalPrice**.
    - Implement **Quick Sort** to sort orders by **totalPrice**.
4. **Analysis:**
    - Compare the performance (time complexity) of Bubble Sort and Quick Sort.
    - Discuss why Quick Sort is generally preferred over Bubble Sort.

CODE:

Main.java

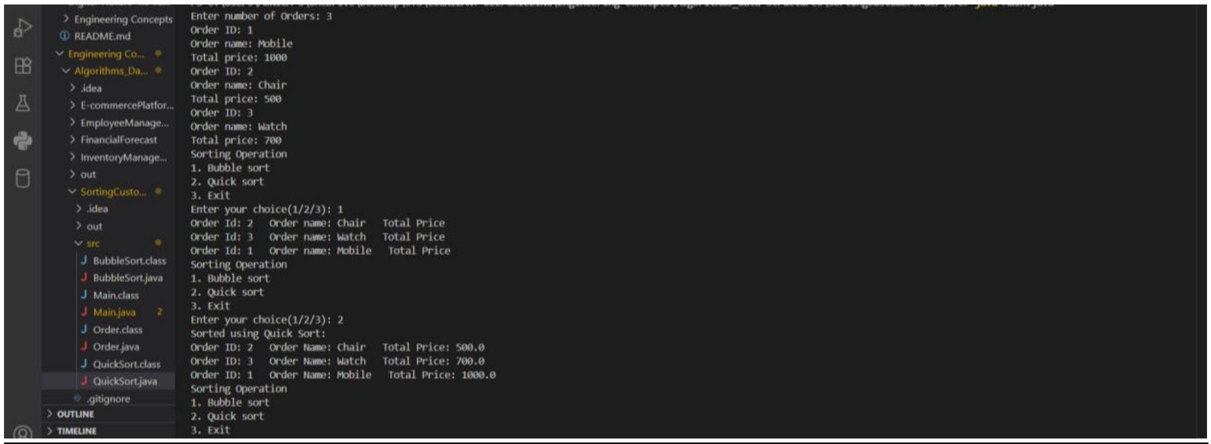
```
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {

    Scanner io=new Scanner(System.in);
    int OrderId;
    String OrderName;
    double totalPrice;
    System.out.print("Enter number of Orders: ");
    int n=io.nextInt();
    Order[] order=new Order[n];
    for(int i=0;i<n;i++) {
      System.out.print("Order ID: ");
      OrderId=io.nextInt();
      System.out.print("Order name: ");
      OrderName=io.next();
      System.out.print("Total price: ");
      totalPrice=io.nextDouble();
      Order o=new Order(OrderId, OrderName, totalPrice);
      order[i]=o;
    }
    while (true) {
      System.out.println("Sorting Operation");
      System.out.println("1. Bubble sort\n2. Quick sort\n3. Exit");
      System.out.print("Enter your choice(1/2/3): ");
      int ch=io.nextInt();
      switch (ch) {
        case 1:
          BubbleSort.sort(order);
          break;
        case 2:
          QuickSort.sort(order, 0, n - 1);
              System.out.println("Sorted using Quick Sort:");
              QuickSort.printOrders(order);
```

```
                        break;

                case 3:
                    System.out.println("Bye...");
                    return;
                default:
            }
        }
    }
}
```
Order.java
```
public class Order {

    int OrderId;
    String OrderName;
    double totalPrice;

    public Order(int OrderId, String OrderName, double totalPrice){
        this.OrderId=OrderId;
        this.OrderName=OrderName;
        this.totalPrice=totalPrice;
    }

}
```
BubbleSort.java
```
public class BubbleSort {
    public static void sort(Order[] orders) {
        int n=orders.length;
        for (int i=0;i<n-1;i++) {
            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;
                }
            }
        }
        for(int i=0;i<n;i++) {
            System.out.print("Order Id: "+orders[i].OrderId+"   Order name: "+orders[i].OrderName+"   Total
Price\n");
        }
    }
}
```

QuickSort.java

```
public class QuickSort {

    public static void sort(Order[] orders, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(orders, low, high);
            sort(orders, low, pivotIndex - 1);
            sort(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;
    }

    public static void printOrders(Order[] orders) {
        for (Order o : orders) {
            System.out.println("Order ID: " + o.OrderId + "   Order Name: " + o.OrderName + "   Total Price: "
+ o.totalPrice);
        }
    }
}
```

OUTPUT:



```
Enter number of Orders: 3
Order ID: 1
Order name: Mobile
Total price: 1000
Order ID: 2
Order name: Chair
Total price: 500
Order ID: 3
Order name: Watch
Total price: 700
Sorting Operation
1. Bubble sort
2. Quick sort
3. Exit
Enter your choice(1/2/3): 1
Order Id: 2   Order name: Chair   Total Price
Order Id: 3   Order name: Watch   Total Price
Order Id: 1   Order name: Mobile   Total Price
Sorting Operation
1. Bubble sort
2. Quick sort
3. Exit
Enter your choice(1/2/3): 2
Sorted using Quick Sort:
Order ID: 2   Order Name: Chair   Total Price: 500.0
Order ID: 3   Order Name: Watch   Total Price: 700.0
Order ID: 1   Order Name: Mobile   Total Price: 1000.0
Sorting Operation
1. Bubble sort
2. Quick sort
3. Exit
```

Exercise 4: Employee Management System

Scenario:

You are developing an employee management system for a company. Efficiently managing employee records is crucial.

Steps:

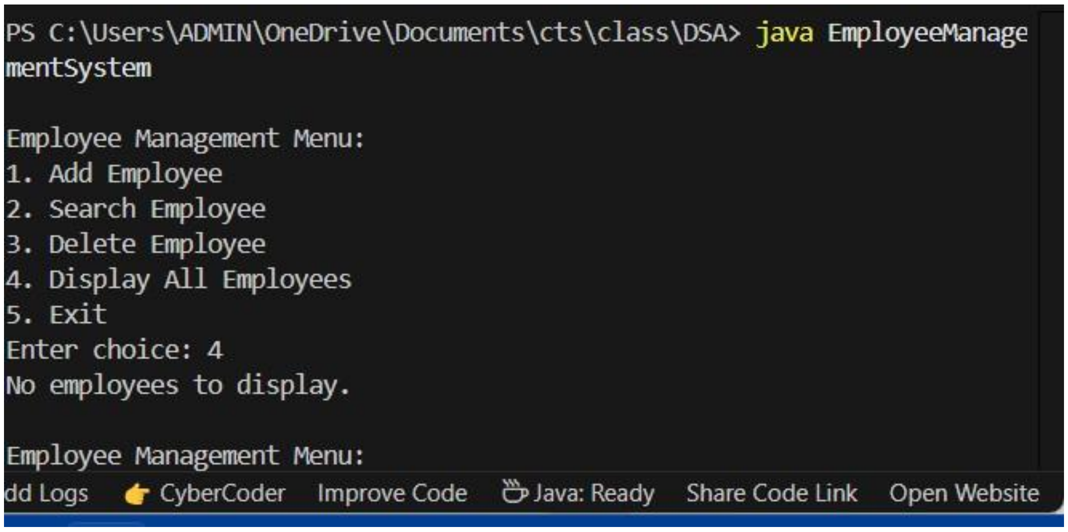
1. **Understand Array Representation:**
   - Explain how arrays are represented in memory and their advantages.
2. **Setup:**
   - Create a class Employee with attributes like **employeeId**, **name**, **position**, and **salary**.
3. **Implementation:**
   - Use an array to store employee records.
   - Implement methods to **add**, **search**, **traverse**, and **delete** employees in the array.
4. **Analysis:**
   - Analyze the time complexity of each operation (add, search, traverse, delete).
   - Discuss the limitations of arrays and when to use them.

CODE:

```
import java.util.ArrayList;
import java.util.Scanner;

class Staff {
   int id;
   String name;
   String position;
   double salary;

   Staff(int id, String name, String position, double salary) {
      this.id = id;
      this.name = name;
      this.position = position;
      this.salary = salary;
   }

   void display() {
      System.out.println("ID: " + id + ", Name: " + name + ", Position: " + position +
", Salary: ₹" + salary);
   }
```

```
}

public class StaffManager {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<Staff> staffRecords = new ArrayList<>();

        while (true) {
            System.out.println("\nStaff Management Menu:");
            System.out.println("1. Add Staff");
            System.out.println("2. Search Staff");
            System.out.println("3. Remove Staff");
            System.out.println("4. Display All Staff");
            System.out.println("5. Exit");
            System.out.print("Enter choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter Position: ");
                    String role = scanner.nextLine();
                    System.out.print("Enter Salary: ");
                    double salary = scanner.nextDouble();
                    staffRecords.add(new Staff(id, name, role, salary));
                    System.out.println("Staff added successfully.");
                    break;

                case 2:
                    System.out.print("Enter Staff ID to search: ");
                    int searchId = scanner.nextInt();
                    boolean found = false;
                    for (Staff s : staffRecords) {
                        if (s.id == searchId) {
                            s.display();
                            found = true;
                            break;
                        }
                    }
                    if (!found) {
                        System.out.println("Staff not found.");
                    }
                    break;

                case 3:
                    System.out.print("Enter Staff ID to remove: ");
                    int removeId = scanner.nextInt();
                    boolean removed = staffRecords.removeIf(s -> s.id == removeId);
                    if (removed) {
                        System.out.println("Staff removed.");
                    } else {
                        System.out.println("Staff not found.");
                    }
                    break;

                case 4:
                    if (staffRecords.isEmpty()) {
                        System.out.println("No staff to display.");
                    } else {
                        for (Staff s : staffRecords) {
                            s.display();
                        }
                    }
                    break;

                case 5:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;

                default:
                    System.out.println("Invalid choice.");
            }
        }
    }
```

}OUTPUT:



```
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java EmployeeManage
mentSystem

Employee Management Menu:
1. Add Employee
2. Search Employee
3. Delete Employee
4. Display All Employees
5. Exit
Enter choice: 4
No employees to display.

Employee Management Menu:
```

Exercise 5: Task Management System

Scenario:

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

Steps:

1. **Understand Linked Lists:**
   - Explain the different types of linked lists (Singly Linked List, Doubly Linked List).
2. **Setup:**
   - Create a class **Task** with attributes like **taskId**, **taskName**, and **status**.
3. **Implementation:**
   - Implement a singly linked list to manage tasks.
   - Implement methods to **add**, **search**, **traverse**, and **delete** tasks in the linked list.
4. **Analysis:**
   - Analyze the time complexity of each operation.
   - Discuss the advantages of linked lists over arrays for dynamic data.

CODE:

Main.java

```
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner io=new Scanner(System.in);
    int taskId;
    String taskName;
    String status;
    while (true) {
      System.out.println("Task Management System" +
          "\n1. Add" +
          "\n2. Delete" +
          "\n3. Traverse" +
          "\n4. Search");
      System.out.print("Enter your choice: ");
      int ch=io.nextInt();
      switch (ch) {
        case 1:
          System.out.print("Task ID: ");
          taskId=io.nextInt();
          io.nextLine();
          System.out.print("Task Name: ");
          taskName=io.nextLine();
          System.out.print("Status: ");
          status=io.nextLine();
          Node.add(taskId,taskName,status);
          break;
        case 2:
          System.out.print("Task ID to delete: ");
          taskId=io.nextInt();
          Node.delete(taskId);
          break;
        case 3:
          Node.display();
          break;
        case 4:
          System.out.print("Task ID: ");
          taskId=io.nextInt();
          Node.search(taskId);
          break;
        default:
          System.out.println("Enter valid choice");
      }
    }
  }
}
```

Node.java

```
public class Node {
  Task data;
  Node next;
  static Node head = null;
```

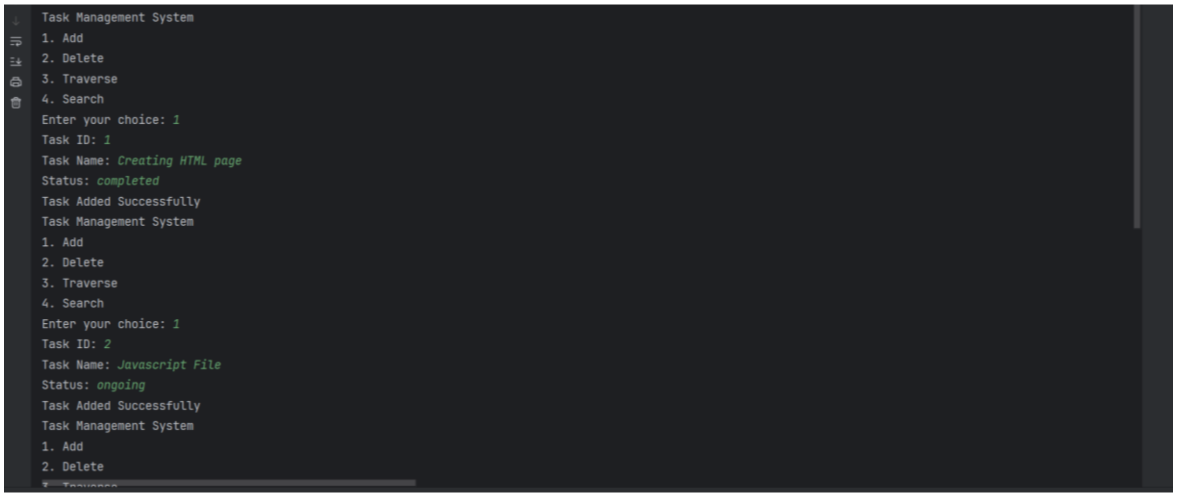
```
public Node(Task data) {
    this.data = data;
    this.next = null;
}

public static void add(int taskId, String taskName, String status) {
    Task t = new Task(taskId, taskName, status);
    Node newNode = new Node(t);

    if (head == null) {
        head = newNode;
    } else {
        Node curr = head;
        while (curr.next != null) {
            curr = curr.next;
        }
        curr.next = newNode;
    }

    System.out.println("Task Added Successfully");
}

public static void delete(int taskId) {
    if (head == null) {
        System.out.println("Empty Task List");
        return;
    }
    if (head.data.taskId == taskId) {
        head = head.next;
        System.out.println("Task deleted successfully.");
        return;
    }
    Node curr = head;
    Node prev = null;

    while (curr != null && curr.data.taskId != taskId) {
        prev = curr;
        curr = curr.next;
    }

    if (curr == null) {
        System.out.println("Task not found.");
    } else {
        prev.next = curr.next;
        System.out.println("Task deleted successfully.");
    }
}

public static void display() {
    if (head==null) {
        System.out.println("Empty Task list");
```

```
            return;
        }
        Node curr=head;
        while (curr!=null) {
            System.out.println(curr.data.taskId+"->"+curr.data.taskName+"->"+curr.data.status);
            curr=curr.next;
        }
    }
    public static void search(int taskId) {
        if(head==null) {
            System.out.println("Empty Task List");
            return;
        }
        Node curr=head;
        while (curr!=null) {
            if(curr.data.taskId==taskId) {
                System.out.println(head.data.taskId+"->"+head.data.taskName+"->"+head.data.status);
                return;
            }
            curr=curr.next;
        }
    }
}
```

Task.java

```
public class Task {
    int taskId;
    String taskName;
    String status;
    public Task(int taskId, String taskName, String status) {
        this.taskId=taskId;
        this.taskName=taskName;
        this.status=status;
    }
}
```

OUTPUT:



```
Task Management System
1. Add
2. Delete
3. Traverse
4. Search
Enter your choice: 1
Task ID: 1
Task Name: Creating HTML page
Status: completed
Task Added Successfully
Task Management System
1. Add
2. Delete
3. Traverse
4. Search
Enter your choice: 1
Task ID: 2
Task Name: Javascript File
Status: ongoing
Task Added Successfully
Task Management System
1. Add
2. Delete
```

Exercise 6: Library Management System

Scenario:

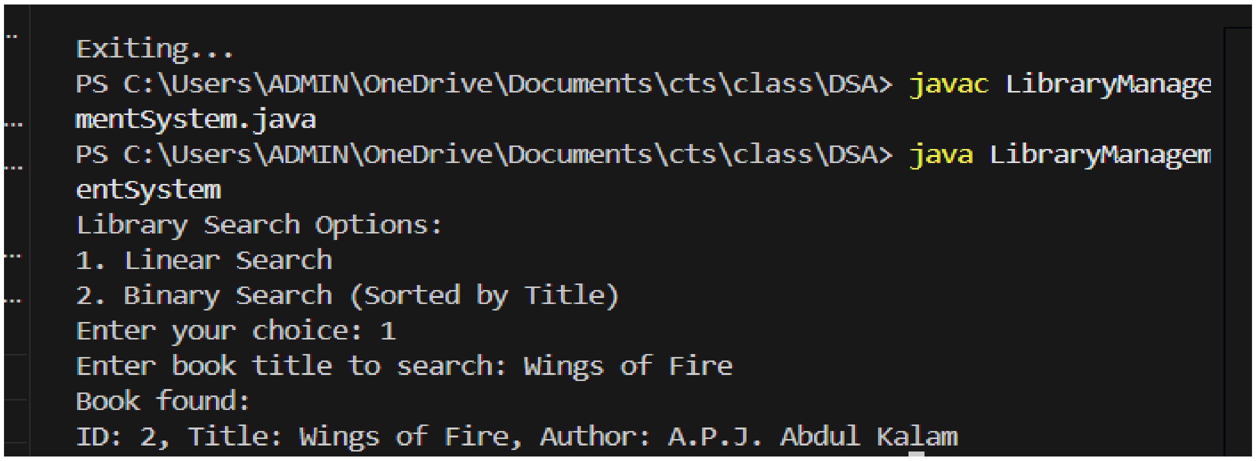
You are developing a library management system where users can search for books by title or author.

Steps:

1. **Understand Search Algorithms:**
   - Explain linear search and binary search algorithms.
2. **Setup:**
   - Create a class **Book** with attributes like **bookId**, **title**, and **author**.
3. **Implementation:**
   - Implement linear search to find books by title.
   - Implement binary search to find books by title (assuming the list is sorted).
4. **Analysis:**
   - Compare the time complexity of linear and binary search.
   - Discuss when to use each algorithm based on the data set size and order.

CODE:

```
import java.util.*;
import java.util.stream.IntStream;

class Publication {
  int id;
  String title;
  String author;

  Publication(int id, String title, String author) {
    this.id = id;
    this.title = title;
    this.author = author;
  }

  void display() {
    System.out.println("ID: " + id + ", Title: " + title + ", Author: " + author);
  }
}

public class LibrarySystem {

  // Binary Search (separate utility method)
  static int performBinarySearch(Publication[] list, String searchTitle) {
    int left = 0, right = list.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        int cmp = list[mid].title.compareToIgnoreCase(searchTitle);
        if (cmp == 0)
            return mid;
        else if (cmp < 0)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}

public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    Publication[] library = {
        new Publication(1, "The Alchemist", "Paulo Coelho"),
        new Publication(2, "Wings of Fire", "A.P.J. Abdul Kalam"),
        new Publication(3, "Rich Dad Poor Dad", "Robert Kiyosaki"),
        new Publication(4, "Harry Potter", "J.K. Rowling"),
        new Publication(5, "Atomic Habits", "James Clear")
    };

    System.out.println("Library Search Options:");
    System.out.println("1. Linear Search using Stream");
    System.out.println("2. Binary Search (after sorting)");
    System.out.print("Enter your choice: ");
    int option = input.nextInt();
    input.nextLine(); // Clear buffer

    System.out.print("Enter book title to search: ");
    String query = input.nextLine();

    int foundIndex = -1;

    if (option == 1) {
        // Linear search using IntStream
        foundIndex = IntStream.range(0, library.length)
                .filter(i -> library[i].title.equalsIgnoreCase(query))
                .findFirst()
                .orElse(-1);
    } else if (option == 2) {
        Arrays.sort(library, Comparator.comparing(p -> p.title.toLowerCase()));
        foundIndex = performBinarySearch(library, query);
    } else {
        System.out.println("Invalid choice.");
        input.close();
        return;
    }

    if (foundIndex != -1) {
        System.out.println("Book found:");
        library[foundIndex].display();
    } else {
        System.out.println("Book not found.");
    }

    input.close();
}
}
```
OUTPUT:



```
Exiting...
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> javac LibraryManage
mentSystem.java
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java LibraryManagem
entSystem
Library Search Options:
1. Linear Search
2. Binary Search (Sorted by Title)
Enter your choice: 1
Enter book title to search: Wings of Fire
Book found:
ID: 2, Title: Wings of Fire, Author: A.P.J. Abdul Kalam
```

Exercise 7: Financial Forecasting

Scenario:

You are developing a financial forecasting tool that predicts future values based on past data.

Steps:

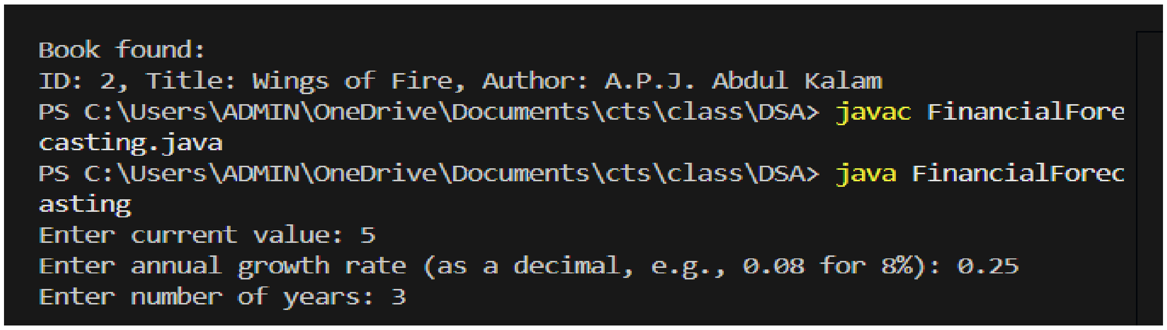
1. **Understand Recursive Algorithms:**
    - Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
    - Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
    - Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
    - Discuss the time complexity of your recursive algorithm.
    - Explain how to optimize the recursive solution to avoid excessive computation.

CODE:

```
import java.util.Scanner;

public class InvestmentCalculator {

  // Loop-based approach (replacing recursion)
  static double computeFutureUsingLoop(double principal, double rate, int period) {
    double futureValue = principal;
    for (int i = 1; i <= period; i++) {
      futureValue *= (1 + rate);
    }
    return futureValue;
  }

  // Using a custom power function instead of Math.pow (replacing optimized method)
  static double computeFutureWithCustomPower(double principal, double rate, int period) {
    double multiplier = 1.0;
    for (int i = 0; i < period; i++) {
      multiplier *= (1 + rate);
    }
    return principal * multiplier;
  }

  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.print("Enter present amount: ");
```

```
        double base = input.nextDouble();

        System.out.print("Enter annual growth rate (e.g., 0.08 for 8%): ");
        double annualRate = input.nextDouble();

        System.out.print("Enter number of years: ");
        int time = input.nextInt();

        double resultFromLoop = computeFutureUsingLoop(base, annualRate, time);
        double resultFromPower = computeFutureWithCustomPower(base, annualRate, time);

        System.out.printf("\nFuture Amount using Loop: ₹%.2f\n", resultFromLoop);
        System.out.printf("Future Amount using Custom Power Logic: ₹%.2f\n", resultFromPower);

        input.close();
    }
}
```

OUTPUT:



```
Book found:
ID: 2, Title: Wings of Fire, Author: A.P.J. Abdul Kalam
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> javac FinancialFore
casting.java
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java FinancialForec
asting
Enter current value: 5
Enter annual growth rate (as a decimal, e.g., 0.08 for 8%): 0.25
Enter number of years: 3
```