Week 2: Data Structures and Algorithms Hands-On

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:

`Product.java`

```java
package InventoryManagementSystem;

public class Product {
    private String productId;
    private String productName;
    private int quantity;
    private double price;

    public Product(String productId, String productName, int quantity, double
price) {
        this.productId = productId;
        this.productName = productName;
        this.quantity = quantity;
        this.price = price;
    }

    public String getProductId() {
        return productId;
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public int getQuantity() {
        return quantity;
    }

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

    public double getPrice() {
        return price;
    }

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

    @Override
    public String toString() {
        return "Product [ID=" + productId + ", Name=" + productName + ",
Quantity=" + quantity + ", Price=" + price + "]";
    }
}
```

InventoryManager.java

```
package InventoryManagementSystem;

import java.util.HashMap;
import java.util.Map;

public class InventoryManager {
    private Map<String, Product> inventory;

    public InventoryManager() {
        inventory = new HashMap<>();
    }

    // Add product
    public void addProduct(Product product) {
        inventory.put(product.getProductId(), product);
        System.out.println("Added: " + product);
    }

    // Update product
    public void updateProduct(String productId, int newQuantity, double
newPrice) {
        if (inventory.containsKey(productId)) {
            Product product = inventory.get(productId);
            product.setQuantity(newQuantity);
            product.setPrice(newPrice);
            System.out.println("Updated: " + product);
        } else {
            System.out.println("Product with ID " + productId + " not found.");
        }
    }

    // Delete product
    public void deleteProduct(String productId) {
        if (inventory.containsKey(productId)) {
            Product removedProduct = inventory.remove(productId);
            System.out.println("Deleted: " + removedProduct);
        } else {
            System.out.println("Product with ID " + productId + " not found.");
        }
    }

    // Get product by ID
    public Product getProduct(String productId) {
        return inventory.get(productId);
    }

    // Display all products
    public void displayAllProducts() {
        System.out.println("\nCurrent Inventory:");
        if (inventory.isEmpty()) {
            System.out.println("Inventory is empty.");
        } else {
            for (Product product : inventory.values()) {
                System.out.println(product);
            }
        }
```

```
    }
}
```

TestInventory.java

```
package InventoryManagementSystem;

public class TestInventory {
    public static void main(String[] args) {
        InventoryManager manager = new InventoryManager();

        // Add products
        manager.addProduct(new Product("P001", "Laptop", 10, 1200.00));
        manager.addProduct(new Product("P002", "Mouse", 50, 25.00));
        manager.addProduct(new Product("P003", "Keyboard", 30, 75.00));

        manager.displayAllProducts();

        // Update a product
        manager.updateProduct("P001", 8, 1150.00);
        manager.displayAllProducts();

        // Delete a product
        manager.deleteProduct("P002");
        manager.displayAllProducts();

        // Try to update a non-existent product
        manager.updateProduct("P004", 5, 100.00);

        // Try to delete a non-existent product
        manager.deleteProduct("P005");
    }
}
```

Output:

```
 Added: Product [ID=P001, Name=Laptop, Quantity=10, Price=1200.0]
Added: Product [ID=P002, Name=Mouse, Quantity=50, Price=25.0]
Added: Product [ID=P003, Name=Keyboard, Quantity=30, Price=75.0]
Current Inventory:
Product [ID=P001, Name=Laptop, Quantity=10, Price=1200.0]
Product [ID=P003, Name=Keyboard, Quantity=30, Price=75.0]
Product [ID=P002, Name=Mouse, Quantity=50, Price=25.0]
Updated: Product [ID=P001, Name=Laptop, Quantity=8, Price=1150.0]
Current Inventory:
Product [ID=P001, Name=Laptop, Quantity=8, Price=1150.0]
Product [ID=P003, Name=Keyboard, Quantity=30, Price=75.0]
Product [ID=P002, Name=Mouse, Quantity=50, Price=25.0]
Deleted: Product [ID=P002, Name=Mouse, Quantity=50, Price=25.0]
Current Inventory:
Product [ID=P001, Name=Laptop, Quantity=8, Price=1150.0]
Product [ID=P003, Name=Keyboard, Quantity=30, Price=75.0]
Product with ID P004 not found.
Product with ID P005 not found.
```

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:

`Product.java`

```java
package EcommerceSearchFunction;

public class Product {
    private String productId;
    private String productName;
    private String category;

    public Product(String productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }

    public String getProductId() {
        return productId;
    }

    public String getProductName() {
        return productName;
    }

    public String getCategory() {
        return category;
    }

    @Override
    public String toString() {
        return "Product [ID=" + productId + ", Name=" + productName + ",
Category=" + category + "]";
    }
}
```

`SearchAlgorithms.java`

```
 package EcommerceSearchFunction;

import java.util.Arrays;
import java.util.Comparator;

public class SearchAlgorithms {

    // Linear Search
    public static Product linearSearch(Product[] products, String searchName) {
        for (Product product : products) {
            if (product.getProductName().equalsIgnoreCase(searchName)) {
                return product;
            }
        }
        return null;
    }

    // Binary Search (requires sorted array)
    public static Product binarySearch(Product[] products, String searchName) {
        int low = 0;
        int high = products.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            int cmp =
products[mid].getProductName().compareToIgnoreCase(searchName);

            if (cmp == 0) {
                return products[mid];
            } else if (cmp < 0) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return null;
    }
}
```

TestSearch.java

```
 package EcommerceSearchFunction;

import java.util.Arrays;

public class TestSearch {
    public static void main(String[] args) {
        Product[] products = {
            new Product("P001", "Laptop", "Electronics"),
            new Product("P002", "Mouse", "Electronics"),
            new Product("P003", "Keyboard", "Electronics"),
            new Product("P004", "Monitor", "Electronics"),
            new Product("P005", "Desk Chair", "Furniture")
        };

        // Test Linear Search
        System.out.println("\n--- Linear Search ---");
        Product foundProduct = SearchAlgorithms.linearSearch(products,
"Mouse");
        if (foundProduct != null) {
            System.out.println("Found: " + foundProduct);
        } else {
            System.out.println("Product not found.");
        }

        foundProduct = SearchAlgorithms.linearSearch(products, "Table");
        if (foundProduct != null) {
            System.out.println("Found: " + foundProduct);
        } else {
            System.out.println("Product not found.");
        }

        // Test Binary Search (requires sorted array)
        System.out.println("\n--- Binary Search ---");
        // Sort products by productName for binary search
        Arrays.sort(products, (p1, p2) ->
p1.getProductName().compareToIgnoreCase(p2.getProductName()));

        foundProduct = SearchAlgorithms.binarySearch(products, "Keyboard");
        if (foundProduct != null) {
            System.out.println("Found: " + foundProduct);
        } else {
            System.out.println("Product not found.");
        }

        foundProduct = SearchAlgorithms.binarySearch(products, "Speaker");
        if (foundProduct != null) {
            System.out.println("Found: " + foundProduct);
        } else {
            System.out.println("Product not found.");
        }
    }
}
```

Output:

```
 --- Linear Search ---
Found: Product [ID=P002, Name=Mouse, Category=Electronics]
Product not found.
--- Binary Search ---
Found: Product [ID=P003, Name=Keyboard, Category=Electronics]
Product not found.
```

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: Implement 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:

`Order.java`

```
package SortingCustomerOrders;

public class Order {
    private String orderId;
    private String customerName;
    private double totalPrice;

    public Order(String orderId, String customerName, double totalPrice) {
        this.orderId = orderId;
        this.customerName = customerName;
        this.totalPrice = totalPrice;
    }

    public String getOrderId() {
        return orderId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public double getTotalPrice() {
        return totalPrice;
    }

    @Override
    public String toString() {
        return "Order [ID=" + orderId + ", Customer=" + customerName + ",
TotalPrice=" + totalPrice + "]";
    }
}
```

SortingAlgorithms.java

```java
package SortingCustomerOrders;

public class SortingAlgorithms {

    // Bubble Sort
    public static void bubbleSort(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].getTotalPrice() > orders[j + 1].getTotalPrice())
{
                    // Swap orders[j] and orders[j+1]
                    Order temp = orders[j];
                    orders[j] = orders[j + 1];
                    orders[j + 1] = temp;
                }
            }
        }
    }

    // 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); // index of smaller element
        for (int j = low; j < high; j++) {
            // If current element is smaller than or equal to pivot
            if (orders[j].getTotalPrice() <= pivot) {
                i++;

                // swap orders[i] and orders[j]
                Order temp = orders[i];
                orders[i] = orders[j];
                orders[j] = temp;
            }
        }

        // swap orders[i+1] and orders[high] (or pivot)
        Order temp = orders[i + 1];
        orders[i + 1] = orders[high];
        orders[high] = temp;

        return i + 1;
    }
}
```

TestSorting.java

```java
package SortingCustomerOrders;

import java.util.Arrays;

public class TestSorting {
    public static void main(String[] args) {
        Order[] orders1 = {
            new Order("O001", "Alice", 250.50),
            new Order("O002", "Bob", 100.00),
            new Order("O003", "Charlie", 500.75),
            new Order("O004", "David", 120.25)
        };

        Order[] orders2 = {
            new Order("O005", "Eve", 300.00),
            new Order("O006", "Frank", 150.50),
            new Order("O007", "Grace", 400.25),
            new Order("O008", "Heidi", 90.00)
        };

        System.out.println("--- Bubble Sort ---");
        System.out.println("Original Orders: " + Arrays.toString(orders1));
        SortingAlgorithms.bubbleSort(orders1);
        System.out.println("Sorted Orders: " + Arrays.toString(orders1));

        System.out.println("\n--- Quick Sort ---");
        System.out.println("Original Orders: " + Arrays.toString(orders2));
        SortingAlgorithms.quickSort(orders2, 0, orders2.length - 1);
        System.out.println("Sorted Orders: " + Arrays.toString(orders2));
    }
}
```

Output:

```
--- Bubble Sort ---
Original Orders: [Order [ID=O001, Customer=Alice, TotalPrice=250.5], Order
[ID=O002, Customer=Bob, TotalPrice=100.0], Order [ID=O003, Customer=Charlie,
TotalPrice=500.75], Order [ID=O004, Customer=David, TotalPrice=120.25]]
Sorted Orders: [Order [ID=O002, Customer=Bob, TotalPrice=100.0], Order
[ID=O004, Customer=David, TotalPrice=120.25], Order [ID=O001, Customer=Alice,
TotalPrice=250.5], Order [ID=O003, Customer=Charlie, TotalPrice=500.75]]

--- Quick Sort ---
Original Orders: [Order [ID=O005, Customer=Eve, TotalPrice=300.0], Order
[ID=O006, Customer=Frank, TotalPrice=150.5], Order [ID=O007, Customer=Grace,
TotalPrice=400.25], Order [ID=O008, Customer=Heidi, TotalPrice=90.0]]
Sorted Orders: [Order [ID=O008, Customer=Heidi, TotalPrice=90.0], Order
[ID=O006, Customer=Frank, TotalPrice=150.5], Order [ID=O005, Customer=Eve,
TotalPrice=300.0], Order [ID=O007, Customer=Grace, TotalPrice=400.25]]
```

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:

`Employee.java`

```java
package EmployeeManagementSystem;

public class Employee {
    private String employeeId;
    private String name;
    private String position;
    private double salary;

    public Employee(String employeeId, String name, String position, double
salary) {
        this.employeeId = employeeId;
        this.name = name;
        this.position = position;
        this.salary = salary;
    }

    public String getEmployeeId() {
        return employeeId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee [ID=" + employeeId + ", Name=" + name + ", Position="
+ position + ", Salary=" + salary + "]";
    }
}
```

EmployeeManager.java

```java
package EmployeeManagementSystem;

import java.util.Arrays;

public class EmployeeManager {
    private Employee[] employees;
    private int count;

    public EmployeeManager(int capacity) {
        employees = new Employee[capacity];
        count = 0;
    }

    // Add employee
    public void addEmployee(Employee employee) {
        if (count < employees.length) {
            employees[count++] = employee;
            System.out.println("Added: " + employee);
        } else {
            System.out.println("Array is full. Cannot add more employees.");
        }
    }

    // Search employee by ID
    public Employee searchEmployee(String employeeId) {
        for (int i = 0; i < count; i++) {
            if (employees[i].getEmployeeId().equals(employeeId)) {
                return employees[i];
            }
        }
        return null;
    }

    // Traverse and display all employees
    public void traverseEmployees() {
        System.out.println("\n--- Employee List ---");
        if (count == 0) {
            System.out.println("No employees in the system.");
        } else {
            for (int i = 0; i < count; i++) {
                System.out.println(employees[i]);
            }
        }
    }

    // Delete employee by ID
    public void deleteEmployee(String employeeId) {
        for (int i = 0; i < count; i++) {
            if (employees[i].getEmployeeId().equals(employeeId)) {
                // Shift elements to the left to fill the gap
                for (int j = i; j < count - 1; j++) {
                    employees[j] = employees[j + 1];
                }
                employees[count - 1] = null; // Clear the last element
                count--;
                System.out.println("Deleted employee with ID: " + employeeId);
                return;
            }
```

```
        }
        System.out.println("Employee with ID " + employeeId + " not found.");
    }
}
```

TestEmployeeManagement.java

```
package EmployeeManagementSystem;

public class TestEmployeeManagement {
    public static void main(String[] args) {
        EmployeeManager manager = new EmployeeManager(5); // Max 5 employees

        // Add employees
        manager.addEmployee(new Employee("E001", "Alice", "Software Engineer",
70000.00));
        manager.addEmployee(new Employee("E002", "Bob", "Project Manager",
90000.00));
        manager.addEmployee(new Employee("E003", "Charlie", "HR Specialist",
60000.00));

        manager.traverseEmployees();

        // Search employee
        Employee found = manager.searchEmployee("E002");
        if (found != null) {
            System.out.println("\nFound Employee: " + found);
        } else {
            System.out.println("\nEmployee not found.");
        }

        // Delete employee
        manager.deleteEmployee("E001");
        manager.traverseEmployees();

        // Try to delete non-existent employee
        manager.deleteEmployee("E005");
    }
}
```

Output:

```
 Added: Employee [ID=E001, Name=Alice, Position=Software Engineer,
Salary=70000.0]
Added: Employee [ID=E002, Name=Bob, Position=Project Manager, Salary=90000.0]
Added: Employee [ID=E003, Name=Charlie, Position=HR Specialist, Salary=60000.0]

--- Employee List ---
Employee [ID=E001, Name=Alice, Position=Software Engineer, Salary=70000.0]
Employee [ID=E002, Name=Bob, Position=Project Manager, Salary=90000.0]
Employee [ID=E003, Name=Charlie, Position=HR Specialist, Salary=60000.0]

Found Employee: Employee [ID=E002, Name=Bob, Position=Project Manager,
Salary=90000.0]
Deleted employee with ID: E001
--- Employee List ---
Employee [ID=E002, Name=Bob, Position=Project Manager, Salary=90000.0]
Employee [ID=E003, Name=Charlie, Position=HR Specialist, Salary=60000.0]
Employee with ID E005 not found.
```

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:

`Task.java`

```java
package TaskManagementSystem;

public class Task {
    private String taskId;
    private String taskName;
    private String status;

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

    public String getTaskId() {
        return taskId;
    }

    public String getTaskName() {
        return taskName;
    }

    public void setTaskName(String taskName) {
        this.taskName = taskName;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    @Override
    public String toString() {
        return "Task [ID=" + taskId + ", Name=" + taskName + ", Status=" +
status + "]";
    }
}
```

Node.java

```java
package TaskManagementSystem;

public class Node {
    Task task;
    Node next;

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

SinglyLinkedList.java

```java
package TaskManagementSystem;

public class SinglyLinkedList {
    private Node head;

    public SinglyLinkedList() {
        this.head = null;
    }

    // Add task to the end of the list
    public void addTask(Task task) {
        Node newNode = new Node(task);
        if (head == null) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
        System.out.println("Added: " + task);
    }

    // Search task by ID
    public Task searchTask(String taskId) {
        Node current = head;
        while (current != null) {
            if (current.task.getTaskId().equals(taskId)) {
                return current.task;
            }
            current = current.next;
        }
        return null;
    }

    // Traverse and display all tasks
    public void traverseTasks() {
        System.out.println("\n--- Task List ---");
        if (head == null) {
            System.out.println("No tasks in the list.");
        } else {
            Node current = head;
            while (current != null) {
                System.out.println(current.task);
                current = current.next;
            }
        }
    }

    // Delete task by ID
    public void deleteTask(String taskId) {
        if (head == null) {
            System.out.println("List is empty. Cannot delete.");
            return;
        }

        if (head.task.getTaskId().equals(taskId)) {
```

```
            System.out.println("Deleted: " + head.task);
            head = head.next;
            return;
        }

        Node current = head;
        while (current.next != null &&
!current.next.task.getTaskId().equals(taskId)) {
            current = current.next;
        }

        if (current.next != null) {
            System.out.println("Deleted: " + current.next.task);
            current.next = current.next.next;
        } else {
            System.out.println("Task with ID " + taskId + " not found.");
        }
    }
}
```

TestTaskManagement.java

```
package TaskManagementSystem;

public class TestTaskManagement {
    public static void main(String[] args) {
        SinglyLinkedList taskList = new SinglyLinkedList();

        // Add tasks
        taskList.addTask(new Task("T001", "Prepare presentation", "Pending"));
        taskList.addTask(new Task("T002", "Send meeting minutes",
"Completed"));
        taskList.addTask(new Task("T003", "Follow up with client", "In
Progress"));

        taskList.traverseTasks();

        // Search task
        Task foundTask = taskList.searchTask("T002");
        if (foundTask != null) {
            System.out.println("\nFound Task: " + foundTask);
        } else {
            System.out.println("\nTask not found.");
        }

        // Delete task
        taskList.deleteTask("T001");
        taskList.traverseTasks();

        // Try to delete non-existent task
        taskList.deleteTask("T004");
    }
}
```

Output:

```
 Added: Task [ID=T001, Name=Prepare presentation, Status=Pending]
Added: Task [ID=T002, Name=Send meeting minutes, Status=Completed]
Added: Task [ID=T003, Name=Follow up with client, Status=In Progress]

--- Task List ---
Task [ID=T001, Name=Prepare presentation, Status=Pending]
Task [ID=T002, Name=Send meeting minutes, Status=Completed]
Task [ID=T003, Name=Follow up with client, Status=In Progress]

Found Task: Task [ID=T002, Name=Send meeting minutes, Status=Completed]
Deleted: Task [ID=T001, Name=Prepare presentation, Status=Pending]
--- Task List ---
Task [ID=T002, Name=Send meeting minutes, Status=Completed]
Task [ID=T003, Name=Follow up with client, Status=In Progress]
Task with ID T004 not found.
```

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:

Book.java

```
 package LibraryManagementSystem;

public class Book {
    private String bookId;
    private String title;
    private String author;

    public Book(String bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
    }

    public String getBookId() {
        return bookId;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    @Override
    public String toString() {
        return "Book [ID=" + bookId + ", Title=" + title + ", Author=" + author
+ "]";
    }
}
```

`LibrarySearch.java`

```java
package LibraryManagementSystem;

import java.util.Arrays;
import java.util.Comparator;

public class LibrarySearch {

    // Linear Search by Title
    public static Book linearSearchByTitle(Book[] books, String searchTitle) {
        for (Book book : books) {
            if (book.getTitle().equalsIgnoreCase(searchTitle)) {
                return book;
            }
        }
        return null;
    }

    // Binary Search by Title (requires sorted array)
    public static Book binarySearchByTitle(Book[] books, String searchTitle) {
        int low = 0;
        int high = books.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            int cmp = books[mid].getTitle().compareToIgnoreCase(searchTitle);

            if (cmp == 0) {
                return books[mid];
            } else if (cmp < 0) {
                low = mid + 1;
            } else {
                high = mid - 1;
            };
        }
        return null;
    }
}
```

TestLibrarySearch.java

```
 package LibraryManagementSystem;

import java.util.Arrays;

public class TestLibrarySearch {
    public static void main(String[] args) {
        Book[] books = {
            new Book("B001", "The Great Gatsby", "F. Scott Fitzgerald"),
            new Book("B002", "1984", "George Orwell"),
            new Book("B003", "To Kill a Mockingbird", "Harper Lee"),
            new Book("B004", "Pride and Prejudice", "Jane Austen"),
            new Book("B005", "The Catcher in the Rye", "J.D. Salinger")
        };

        // Test Linear Search
        System.out.println("\n--- Linear Search by Title ---");
        Book foundBook = LibrarySearch.linearSearchByTitle(books, "1984");
        if (foundBook != null) {
            System.out.println("Found: " + foundBook);
        } else {
            System.out.println("Book not found.");
        }

        foundBook = LibrarySearch.linearSearchByTitle(books, "Moby Dick");
        if (foundBook != null) {
            System.out.println("Found: " + foundBook);
        } else {
            System.out.println("Book not found.");
        }

        // Test Binary Search (requires sorted array)
        System.out.println("\n--- Binary Search by Title ---");
        // Sort books by title for binary search
        Arrays.sort(books, (b1, b2) ->
b1.getTitle().compareToIgnoreCase(b2.getTitle()));

        foundBook = LibrarySearch.binarySearchByTitle(books, "Pride and
Prejudice");
        if (foundBook != null) {
            System.out.println("Found: " + foundBook);
        } else {
            System.out.println("Book not found.");
        }

        foundBook = LibrarySearch.binarySearchByTitle(books, "The Hobbit");
        if (foundBook != null) {
            System.out.println("Found: " + foundBook);
        } else {
            System.out.println("Book not found.");
        }
    }
}
```

Output:

```
 --- Linear Search by Title ---
Found: Book [ID=B002, Title=1984, Author=George Orwell]
Book not found.

--- Binary Search by Title ---
Found: Book [ID=B004, Title=Pride and Prejudice, Author=Jane Austen]
Book not found.
```

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:

`FinancialForecasting.java`

```java
package FinancialForecasting;

public class FinancialForecasting {

    // Recursive method to calculate future value
    public static double calculateFutureValue(double presentValue, double
growthRate, int years) {
        // Base case: if no more years, return present value
        if (years == 0) {
            return presentValue;
        }
        // Recursive step: calculate future value for one less year and apply
growth
        return calculateFutureValue(presentValue * (1 + growthRate),
growthRate, years - 1);
    }
}
```

`TestFinancialForecasting.java`

```java
package FinancialForecasting;

public class TestFinancialForecasting {
    public static void main(String[] args) {
        double presentValue = 1000.0;
        double growthRate = 0.05; // 5%
        int years = 10;

        double futureValue =
FinancialForecasting.calculateFutureValue(presentValue, growthRate, years);
        System.out.printf("Future Value after %d years: %.2f%n", years,
futureValue);

        presentValue = 500.0;
        growthRate = 0.10; // 10%
        years = 5;
        futureValue = FinancialForecasting.calculateFutureValue(presentValue,
growthRate, years);
        System.out.printf("Future Value after %d years: %.2f%n", years,
futureValue);
    }
}
```

Output:

```
 Future Value after 10 years: 1628.89
Future Value after 5 years: 805.26
```