

## 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.HashMap;  
import java.util.Scanner;
```

```
// Product class
```

```
class Product {
```

```
    int productId;
```

```
    String productName;
```

```
    int quantity;
```

```
    double price;
```

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

```
        this.productId = productId;
```

```
        this.productName = productName;
```

```

        this.quantity = quantity;
        this.price = price;
    }

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

// Inventory class
class Inventory {
    HashMap<Integer, Product> inventory = new HashMap<>();

    void addProduct(Product p) {
        inventory.put(p.productId, p);
        System.out.println("Product added.");
    }

    void updateProduct(int productId, int quantity, double price) {
        if (inventory.containsKey(productId)) {
            Product p = inventory.get(productId);
            p.quantity = quantity;
            p.price = price;
            System.out.println("Product updated.");
        } else {
            System.out.println("Product not found.");
        }
    }

    void deleteProduct(int productId) {
        if (inventory.remove(productId) != null) {
            System.out.println("Product deleted.");
        } else {
            System.out.println("Product not found.");
        }
    }
}

```

```
void displayInventory() {
    if (inventory.isEmpty()) {
        System.out.println("Inventory is empty.");
    } else {
        for (Product p : inventory.values()) {
            p.display();
        }
    }
}
```

```
// Main class to demonstrate
public class InventorySystem {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Inventory inv = new Inventory();
        int choice;

        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: ");
            choice = sc.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter Product ID: ");
                    int id = sc.nextInt();
                    sc.nextLine(); // clear buffer
                    System.out.print("Enter Product Name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter Quantity: ");
                    int qty = sc.nextInt();
                    System.out.print("Enter Price: ");
```

```
double price = sc.nextDouble();  
Product p = new Product(id, name, qty, price);  
inv.addProduct(p);  
break;
```

case 2:

```
System.out.print("Enter Product ID to update: ");  
int uid = sc.nextInt();  
System.out.print("Enter New Quantity: ");  
int uq = sc.nextInt();  
System.out.print("Enter New Price: ");  
double up = sc.nextDouble();  
inv.updateProduct(uid, uq, up);  
break;
```

case 3:

```
System.out.print("Enter Product ID to delete: ");  
int did = sc.nextInt();  
inv.deleteProduct(did);  
break;
```

case 4:

```
inv.displayInventory();  
break;
```

case 5:

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

default:

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

```
}
```

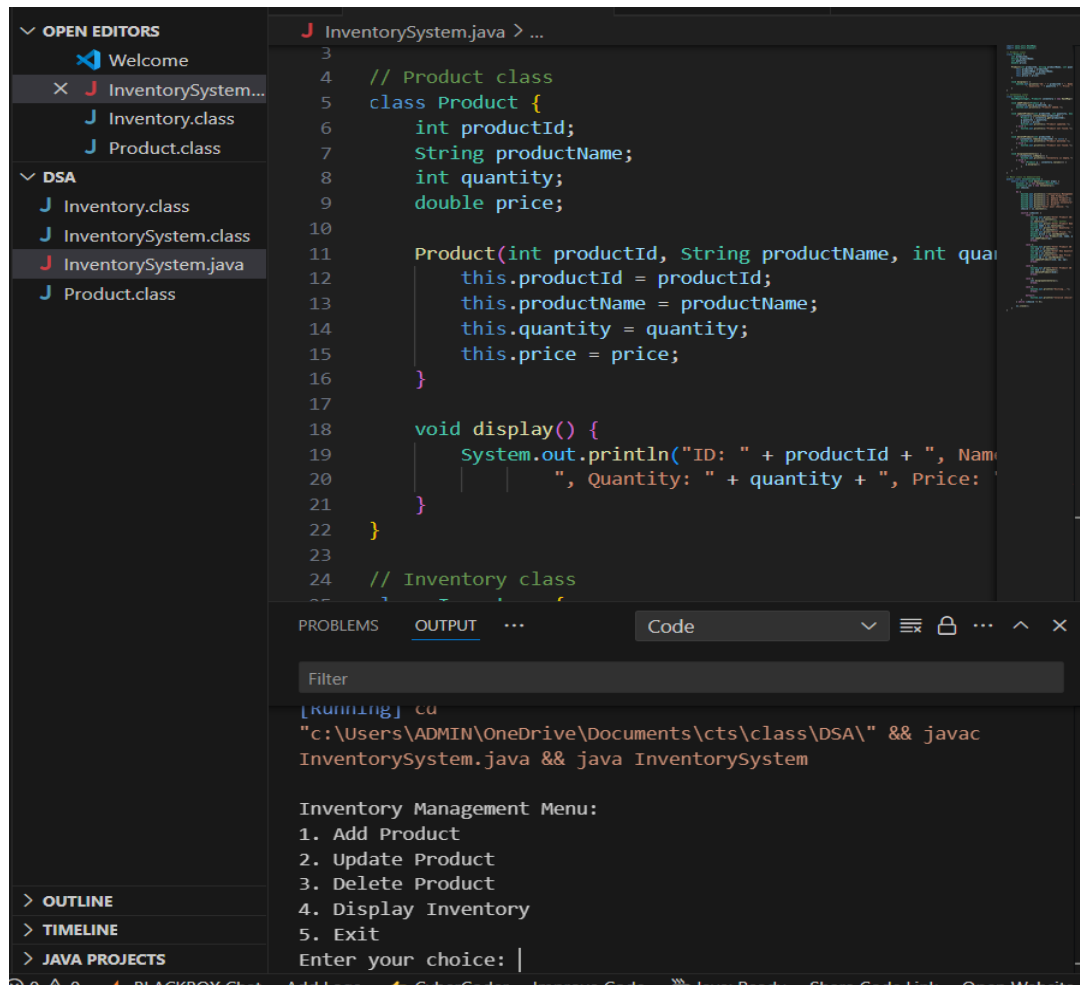
```
} while (choice != 5);
```

```
sc.close();
```

```
}
```

```
}
```

## OUTPUT:



The screenshot shows an IDE with the following components:

- OPEN EDITORS:** A list of open files including `Welcome`, `InventorySystem...`, `Inventory.class`, `Product.class`, and a section for `DSA` containing `Inventory.class`, `InventorySystem.class`, `InventorySystem.java`, and `Product.class`.
- Code Editor:** Displays the `InventorySystem.java` file. The code includes a `Product` class with attributes `productId`, `productName`, `quantity`, and `price`. It also shows a constructor and a `display()` method. The `Inventory` class is partially visible at the bottom.
- OUTPUT:** A terminal window showing the execution of the program. It displays the command used to compile and run the code, followed by the output of the `Inventory` class, which shows a menu with five options: 1. Add Product, 2. Update Product, 3. Delete Product, 4. Display Inventory, and 5. Exit. The prompt "Enter your choice:|" is shown at the end.

## 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 Product {
    int productId;
    String productName;
    String category;

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

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

public class ProductSearchSystem {

    // Linear Search
    static int linearSearch(Product[] arr, String target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i].productName.equalsIgnoreCase(target)) {
                return i;
            }
        }
        return -1;
    }
}
```

```

// Binary Search (Assumes sorted by productName)
static int binarySearch(Product[] arr, String target) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        int cmp = arr[mid].productName.compareToIgnoreCase(target);
        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 sc = new Scanner(System.in);

    Product[] products = {
        new Product(101, "Laptop", "Electronics"),
        new Product(102, "Phone", "Electronics"),
        new Product(103, "Shirt", "Clothing"),
        new Product(104, "Book", "Stationery"),
        new Product(105, "Shoes", "Footwear")
    };

    System.out.println("Choose Search Method:");
    System.out.println("1. Linear Search");
    System.out.println("2. Binary Search (on sorted list)");
    int choice = sc.nextInt();
    sc.nextLine(); // clear buffer
    System.out.print("Enter product name to search: ");
    String searchName = sc.nextLine();

    int result = -1;
    if (choice == 1) {
        result = linearSearch(products, searchName);
    } else if (choice == 2) {
        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase())); // sort
        before binary search
        result = binarySearch(products, searchName);
    } else {

```

```

        System.out.println("Invalid choice!");
        return;
    }

    if (result != -1) {
        System.out.println("Product found:");
        products[result].display();
    } else {
        System.out.println("Product not found.");
    }

    sc.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:**



- 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:

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

```
class Order {
```

```
    int orderId;
```

```
    String customerName;
```

```
    double totalPrice;
```

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

```
        this.orderId = orderId;
```

```
        this.customerName = customerName;
```

```
        this.totalPrice = totalPrice;
```

```
    }
```

```
    void display() {
```

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

```
    }
```

```
}
```

```
public class OrderSortingSystem {
```

```
    static void bubbleSort(Order[] orders) {
```

```

for (int i = 0; i < orders.length - 1; i++) {
    for (int j = 0; j < orders.length - i - 1; j++) {
        if (orders[j].totalPrice > orders[j + 1].totalPrice) {
            Order temp = orders[j];
            orders[j] = orders[j + 1];
            orders[j + 1] = temp;
        }
    }
}

```

```

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);
    }
}

```

```

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;  
}
```

```
static void displayOrders(Order[] orders) {  
    for (Order o : orders) {  
        o.display();  
    }  
}
```

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

```
    Order[] orders = {  
        new Order(201, "Arun", 1450.00),  
        new Order(202, "Meena", 3999.99),  
        new Order(203, "John", 875.25),  
        new Order(204, "Riya", 2340.50),  
        new Order(205, "Sam", 1200.00)  
    };
```

```
    System.out.println("Choose sorting algorithm:");  
    System.out.println("1. Bubble Sort");  
    System.out.println("2. Quick Sort");  
    int choice = sc.nextInt();
```

```

        if (choice == 1) {
            bubbleSort(orders);

            System.out.println("\nOrders sorted using Bubble Sort (by Total Price):");
        } else if (choice == 2) {
            quickSort(orders, 0, orders.length - 1);

            System.out.println("\nOrders sorted using Quick Sort (by Total Price):");
        } else {
            System.out.println("Invalid choice!");

            sc.close();

            return;
        }

        displayOrders(orders);

        sc.close();
    }
}

```

## OUTPUT:

```

PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> javac OrderSortingSystem.java
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java OrderSortingSystem
Choose sorting algorithm:
1. Bubble Sort
2. Quick Sort
2

Orders sorted using Quick Sort (by Total Price):
Order ID: 203, Customer: John, Total: ₹875.25
Order ID: 205, Customer: Sam, Total: ₹1200.0
Order ID: 201, Customer: Arun, Total: ₹1450.0
Order ID: 204, Customer: Riya, Total: ₹2340.5
Order ID: 202, Customer: Meena, Total: ₹3999.99
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA>

```

## 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.Scanner;
```

```
class Employee {  
    int employeeId;  
    String name;  
    String position;  
    double salary;
```

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

```

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

```

```

public class EmployeeManagementSystem {
    static Employee[] employees = new Employee[100];
    static int count = 0;

    static void addEmployee(Employee e) {
        if (count < employees.length) {
            employees[count++] = e;
            System.out.println("Employee added successfully.");
        } else {
            System.out.println("Employee list is full.");
        }
    }
}

```

```

static void searchEmployee(int id) {
    for (int i = 0; i < count; i++) {
        if (employees[i].employeeId == id) {
            employees[i].display();
            return;
        }
    }
    System.out.println("Employee not found.");
}

```

```

static void deleteEmployee(int id) {
    for (int i = 0; i < count; i++) {
        if (employees[i].employeeId == id) {
            for (int j = i; j < count - 1; j++) {
                employees[j] = employees[j + 1];
            }
        }
    }
}

```

```

        employees[--count] = null;
        System.out.println("Employee deleted.");
        return;
    }
}
System.out.println("Employee not found.");
}

```

```

static void traverseEmployees() {
    if (count == 0) {
        System.out.println("No employees to display.");
    } else {
        for (int i = 0; i < count; i++) {
            employees[i].display();
        }
    }
}
}

```

```

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

    do {
        System.out.println("\nEmployee Management Menu:");
        System.out.println("1. Add Employee");
        System.out.println("2. Search Employee");
        System.out.println("3. Delete Employee");
        System.out.println("4. Display All Employees");
        System.out.println("5. Exit");
        System.out.print("Enter choice: ");
        choice = sc.nextInt();

        switch (choice) {
            case 1:
                System.out.print("Enter ID: ");

```

```
int id = sc.nextInt();
sc.nextLine();
System.out.print("Enter Name: ");
String name = sc.nextLine();
System.out.print("Enter Position: ");
String pos = sc.nextLine();
System.out.print("Enter Salary: ");
double salary = sc.nextDouble();
addEmployee(new Employee(id, name, pos, salary));
break;
```

case 2:

```
System.out.print("Enter Employee ID to search: ");
int sid = sc.nextInt();
searchEmployee(sid);
break;
```

case 3:

```
System.out.print("Enter Employee ID to delete: ");
int did = sc.nextInt();
deleteEmployee(did);
break;
```

case 4:

```
traverseEmployees();
break;
```

case 5:

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

default:

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

```
}
```

```
} while (choice != 5);
```



```
        sc.close();
    }
}
```

## OUTPUT:

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

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:
dd Logs  🖱️ CyberCoder  Improve Code  ☕ Java: Ready  Share Code Link  Open Website
```

## 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:**

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

```
class Task {
```

```
    int taskId;
```

```
    String taskName;
```

```
    String status;
```

```
    Task next;
```

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

```
        this.taskId = taskId;
```

```
        this.taskName = taskName;
```

```
        this.status = status;
```

```
        this.next = null;
```

```
    }
```

```
    void display() {
```

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

```
    }
```

```
}
```

```
public class TaskManagementSystem {
```

```
    static Task head = null;
```

```
    static void addTask(Task newTask) {
```

```
        if (head == null) {
```

```
            head = newTask;
```

```
        } else {
```

```
            Task temp = head;
```

```
            while (temp.next != null) {
```

```
                temp = temp.next;
```

```
            }
```

```
        temp.next = newTask;
    }
    System.out.println("Task added.");
}
```

```
static void searchTask(int taskId) {
    Task temp = head;
    while (temp != null) {
        if (temp.taskId == taskId) {
            temp.display();
            return;
        }
        temp = temp.next;
    }
    System.out.println("Task not found.");
}
```

```
static void deleteTask(int taskId) {
    if (head == null) {
        System.out.println("List is empty.");
        return;
    }
```

```
    if (head.taskId == taskId) {
        head = head.next;
        System.out.println("Task deleted.");
        return;
    }
```

```
    Task temp = head;
    while (temp.next != null && temp.next.taskId != taskId) {
        temp = temp.next;
    }
```

```
    if (temp.next != null) {  
        temp.next = temp.next.next;  
        System.out.println("Task deleted.");  
    } else {  
        System.out.println("Task not found.");  
    }  
}
```

```
static void traverseTasks() {  
    if (head == null) {  
        System.out.println("No tasks available.");  
        return;  
    }  
}
```

```
Task temp = head;  
System.out.println("Task List:");  
while (temp != null) {  
    temp.display();  
    temp = temp.next;  
}  
}
```

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int choice;  
  
    do {  
        System.out.println("\nTask Management Menu:");  
        System.out.println("1. Add Task");  
        System.out.println("2. Search Task");  
        System.out.println("3. Delete Task");  
        System.out.println("4. Display All Tasks");  
    }  
}
```

```
System.out.println("5. Exit");  
System.out.print("Enter your choice: ");  
choice = sc.nextInt();
```

```
switch (choice) {
```

```
    case 1:
```

```
        System.out.print("Enter Task ID: ");  
        int id = sc.nextInt();  
        sc.nextLine();  
        System.out.print("Enter Task Name: ");  
        String name = sc.nextLine();  
        System.out.print("Enter Task Status: ");  
        String status = sc.nextLine();  
        addTask(new Task(id, name, status));  
        break;
```

```
    case 2:
```

```
        System.out.print("Enter Task ID to search: ");  
        int sid = sc.nextInt();  
        searchTask(sid);  
        break;
```

```
    case 3:
```

```
        System.out.print("Enter Task ID to delete: ");  
        int did = sc.nextInt();  
        deleteTask(did);  
        break;
```

```
    case 4:
```

```
        traverseTasks();  
        break;
```

```
    case 5:
```

```

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

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

sc.close();
}
}

```

### OUTPUT:

```

PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java
System

Task Management Menu:
1. Add Task
2. Search Task
3. Delete Task
4. Display All Tasks
5. Exit
Enter your choice: 4
No tasks available.

Task Management Menu:
Add Logs  CyberCoder  Improve Code  Java Ready  Share Code

```

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

class Book {
    int bookId;
    String title;
    String author;

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

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

    // Linear Search
    static int linearSearch(Book[] books, String target) {
        for (int i = 0; i < books.length; i++) {
            if (books[i].title.equalsIgnoreCase(target)) {
                return i;
            }
        }
        return -1;
    }

    // Binary Search (Assumes sorted by title)
    static int binarySearch(Book[] books, String target) {
```

```

int left = 0, right = books.length - 1;
while (left <= right) {
    int mid = (left + right) / 2;
    int comp = books[mid].title.compareToIgnoreCase(target);
    if (comp == 0)
        return mid;
    else if (comp < 0)
        left = mid + 1;
    else
        right = mid - 1;
}
return -1;
}
}

```

```

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

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

        System.out.println("Library Search Options:");
        System.out.println("1. Linear Search");
        System.out.println("2. Binary Search (Sorted by Title)");
        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();
        sc.nextLine(); // consume newline
        System.out.print("Enter book title to search: ");
        String targetTitle = sc.nextLine();

        int result = -1;
        if (choice == 1) {

```



```

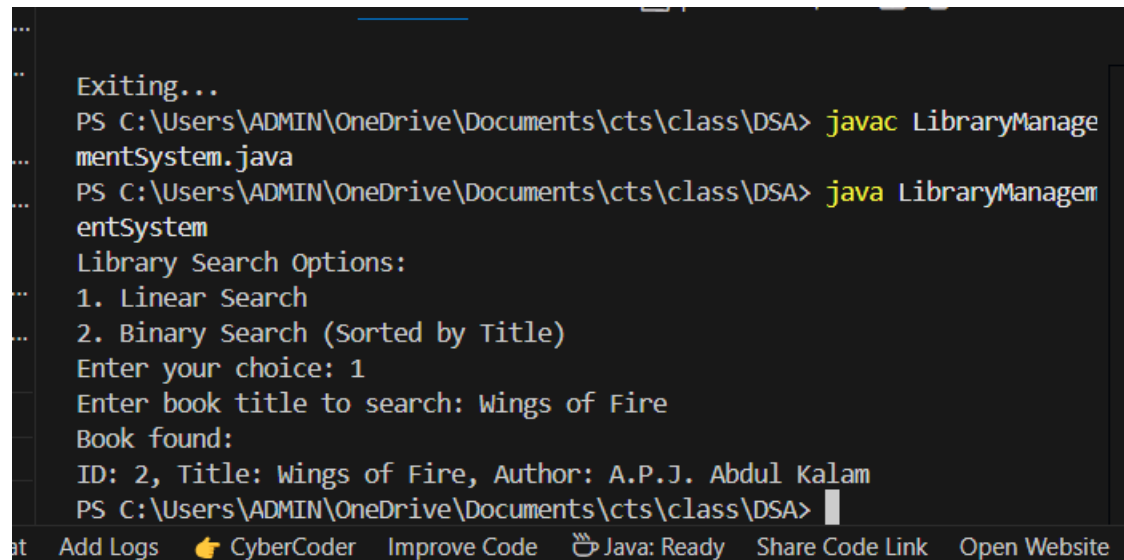
        result = Book.linearSearch(books, targetTitle);
    } else if (choice == 2) {
        Arrays.sort(books, Comparator.comparing(b -> b.title.toLowerCase()));
        result = Book.binarySearch(books, targetTitle);
    } else {
        System.out.println("Invalid choice.");
        sc.close();
        return;
    }

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

    sc.close();
}
}

```

## OUTPUT:



```

...
... Exiting...
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> javac LibraryManagementSystem.java
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java LibraryManagementSystem
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
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA>

```

at Add Logs 🖱️ CyberCoder Improve Code ☕ Java: Ready Share Code Link Open Website

## 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 FinancialForecasting {
```

```
    // Recursive approach
```

```
    static double predictValueRecursive(double value, double growthRate, int years)
```

```
{
```

```
    if (years == 0)
```

```
        return value;
```

```
    return predictValueRecursive(value * (1 + growthRate), growthRate, years -
```

```
1);
```

```
}
```

```
    // Optimized approach using formula
```

```

    static double predictValueOptimized(double value, double growthRate, int
years) {
        return value * Math.pow(1 + growthRate, years);
    }

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

        System.out.print("Enter current value: ");
        double currentValue = sc.nextDouble();

        System.out.print("Enter annual growth rate (as a decimal, e.g., 0.08 for 8%):
");
        double growthRate = sc.nextDouble();

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

        double futureValueRec = predictValueRecursive(currentValue, growthRate,
years);
        double futureValueOpt = predictValueOptimized(currentValue, growthRate,
years);

        System.out.printf("\nFuture Value using Recursion: ₹%.2f\n",
futureValueRec);
        System.out.printf("Future Value using Optimized Formula: ₹%.2f\n",
futureValueOpt);

        sc.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 FinancialForecasting.java
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> java FinancialForecasting
Enter current value: 5
Enter annual growth rate (as a decimal, e.g., 0.08 for 8%): 0.25
Enter number of years: 3

Future Value using Recursion: ?9.77
Future Value using Optimized Formula: ?9.77
PS C:\Users\ADMIN\OneDrive\Documents\cts\class\DSA> 
```

Add Logs  CyberCoder Improve Code  Java: Ready Share Code Link Open Website