ALGORITHMS AND DATA STRUCTURE

Exercise 1: Inventory Management System

```
// Exercise 1: Inventory Management System
import java.util.*;

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

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

    public String toString() {
        return productId + ": " + productName + " - Qty: " + quantity + ",
Price: " + price;
    }
}

public class InventoryManagementSystem {
    Map<Integer, Product> inventory = new HashMap<>();

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

    public void updateProduct(int id, int qty, double price) {
        if (inventory.containsKey(id)) {
            Product p = inventory.get(id);
            p.quantity = qty;
```

```
            p.price = price;
            System.out.println("Updated product: " + p);
        }
    }

    public void deleteProduct(int id) {
        if (inventory.containsKey(id)) {
            System.out.println("Deleted product: " + inventory.get(id));
            inventory.remove(id);
        }
    }

    public static void main(String[] args) {
        InventoryManagementSystem ims = new InventoryManagementSystem();
        ims.addProduct(new Product(101, "Mouse", 50, 499.99));
        ims.updateProduct(101, 40, 459.99);
        ims.deleteProduct(101);
    }
}
```

OUTPUT:

```
Added product: 101: Mouse - Qty: 50, Price: 499.99
Updated product: 101: Mouse - Qty: 40, Price: 459.99
Deleted product: 101: Mouse - Qty: 40, Price: 459.99
```

Exercise 2: E-commerce Platform Search Function

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

class ECommerceSearch {
    static class Product {
        int productId;
        String productName;
        String category;
```

```
        Product(int id, String name, String cat) {
            productId = id;
            productName = name;
            category = cat;
        }
    }

    public static int linearSearch(Product[] products, String name) {
        for (int i = 0; i < products.length; i++) {
            if (products[i].productName.equals(name))
                return i;
        }
        return -1;
    }

    public static int binarySearch(Product[] products, String name) {
        int left = 0, right = products.length - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            int cmp = products[mid].productName.compareTo(name);
            if (cmp == 0) return mid;
            else if (cmp < 0) left = mid + 1;
            else right = mid - 1;
        }
        return -1;
    }

    public static void main(String[] args) {
        Product[] products = {
            new Product(1, "Laptop", "Electronics"),
            new Product(2, "Phone", "Electronics"),
            new Product(3, "Tablet", "Electronics")
        };
        Arrays.sort(products,
Comparator.comparing((ECommerceSearch.Product p) -> p.productName));

        System.out.println("Linear Search Index: " +
linearSearch(products, "Phone"));
        System.out.println("Binary Search Index: " +
binarySearch(products, "Phone"));
```

```
    }
}
```

OUTPUT:

```
Linear Search Index: 1
Binary Search Index: 1
```

Exercise 3: Sorting Customer Orders

```
class CustomerOrderSorting {
    static class Order {
        int orderId;
        String customerName;
        double totalPrice;

        Order(int id, String name, double price) {
            orderId = id;
            customerName = name;
            totalPrice = price;
        }
    }

    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].totalPrice > orders[j + 1].totalPrice) {
                    Order temp = orders[j];
                    orders[j] = orders[j + 1];
                    orders[j + 1] = temp;
                }
            }
        }
    }

    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[] arr, int low, int high) {
        double pivot = arr[high].totalPrice;
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j].totalPrice <= pivot) {
                i++;
                Order temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        Order temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void main(String[] args) {
        Order[] orders = {
            new Order(1, "Alice", 500.0),
            new Order(2, "Bob", 1500.0),
            new Order(3, "Charlie", 800.0)
        };
        quickSort(orders, 0, orders.length - 1);
        System.out.println("Sorted Orders by Total Price:");
        for (Order o : orders)
            System.out.println(o.customerName + ": " + o.totalPrice);
    }
}
```

OUTPUT:

```
Sorted Orders by Total Price:
Alice: 500.0
Charlie: 800.0
Bob: 1500.0
```

Exercise 4: Employee Management System

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

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

    Employee[] employees = new Employee[100];
    int size = 0;

    public void addEmployee(Employee emp) {
        employees[size++] = emp;
        System.out.println("Added employee: " + emp.name);
    }

    public Employee searchEmployee(int id) {
        for (int i = 0; i < size; i++) {
            if (employees[i].employeeId == id) return employees[i];
        }
        return null;
    }

    public void deleteEmployee(int id) {
        for (int i = 0; i < size; i++) {
```

```
            if (employees[i].employeeId == id) {
                for (int j = i; j < size - 1; j++) {
                    employees[j] = employees[j + 1];
                }
                size--;
                System.out.println("Deleted employee with ID: " + id);
                break;
            }
        }
    }

    public void traverseEmployees() {
        System.out.println("All Employees:");
        for (int i = 0; i < size; i++) {
            System.out.println(employees[i].name);
        }
    }

    public static void main(String[] args) {
        EmployeeManagement em = new EmployeeManagement();
        em.addEmployee(new Employee(101, "Neha", "Engineer", 60000));
        em.traverseEmployees();
    }
}
```

OUTPUT:

```
Added employee: Neha
All Employees:
Neha
```

Exercise 5: Task Management System

```
class TaskManagementSystem {
    static class Task {
        int taskId;
        String taskName;
        String status;
        Task next;
```

```
        Task(int id, String name, String status) {
            taskId = id;
            taskName = name;
            this.status = status;
            next = null;
        }
    }

    Task head = null;

    public void addTask(Task task) {
        task.next = head;
        head = task;
        System.out.println("Added task: " + task.taskName);
    }

    public Task searchTask(int id) {
        Task current = head;
        while (current != null) {
            if (current.taskId == id) return current;
            current = current.next;
        }
        return null;
    }

    public void deleteTask(int id) {
        Task current = head, prev = null;
        while (current != null) {
            if (current.taskId == id) {
                if (prev == null) head = current.next;
                else prev.next = current.next;
                System.out.println("Deleted task with ID: " + id);
                return;
            }
            prev = current;
            current = current.next;
        }
    }

    public void traverse() {
```

```
        System.out.println("All Tasks:");
        Task current = head;
        while (current != null) {
            System.out.println(current.taskName);
            current = current.next;
        }
    }

    public static void main(String[] args) {
        TaskManagementSystem tms = new TaskManagementSystem();
        tms.addTask(new Task(1, "Complete Assignment", "Pending"));
        tms.traverse();
    }
}
```

OUTPUT:

```
Added task: Complete Assignment
All Tasks:
Complete Assignment
```

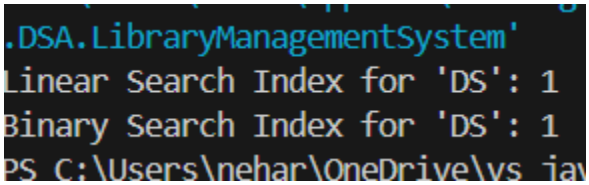
Exercise 6: Library Management System

```
import java.util.Arrays;

import java.util.Comparator;



class LibraryManagementSystem {

    static class Book {

        int bookId;

        String title;
```

```
        String author;

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

    public static int linearSearch(Book[] books, String title) {
        for (int i = 0; i < books.length; i++) {
            if (books[i].title.equals(title)) return i;
        }
        return -1;
    }

    public static int binarySearch(Book[] books, String title) {
        int left = 0, right = books.length - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            int cmp = books[mid].title.compareTo(title);
            if (cmp == 0) return mid;
            else if (cmp < 0) left = mid + 1;
```

```
            else right = mid - 1;

        }

        return -1;

    }



    public static void main(String[] args) {

        Book[] books = {

            new Book(1, "AI", "Russell"),

            new Book(2, "DS", "Tanenbaum"),

            new Book(3, "OS", "Silberschatz")

        };



        Arrays.sort(books,
Comparator.comparing((LibraryManagementSystem.Book b) -> b.title));

        System.out.println("Linear Search Index for 'DS': " +
linearSearch(books, "DS"));

        System.out.println("Binary Search Index for 'DS': " +
binarySearch(books, "DS"));

    }

}
```

OUTPUT:

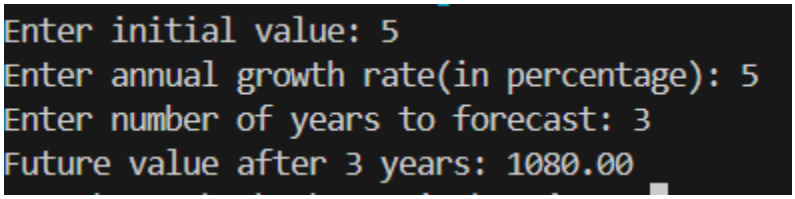


Exercise 7: Financial Forecasting

```
import java.util.Scanner;



class FinancialForecasting {

   public static double calculateFutureValue(double baseValue, double
growthRate, int years) {



    if (years == 0) {



    return baseValue;



}



return calculateFutureValue(baseValue, growthRate, years - 1) * (1 +
growthRate);



}




public static void main(String[] args) {



Scanner sc = new Scanner(System.in);
```

```
System.out.print("Enter initial value: ");


double baseValue = sc.nextDouble();




System.out.print("Enter annual growth rate(in percentage): ");


double growthRate = sc.nextDouble();




System.out.print("Enter number of years to forecast: ");


int years = sc.nextInt();




double futureValue = calculateFutureValue(baseValue, growthRate, years);




System.out.printf("Future value after %d years: %.2f\n", years,
futureValue);


}


}
```

OUTPUT:


Enter initial value: 5
Enter annual growth rate(in percentage): 5
Enter number of years to forecast: 3
Future value after 3 years: 1080.00