**ALGORITHMS AND DATA STRUCTURES**

**SOLUTIONS**

# **Exercise 1: Inventory Management System**

## **Scenario:**

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

## **Code:**

import java.util.\*;

class Product {

    int productId;

    String productName;

    int quantity;

    double price;

    Product(int id, String name, int qty, double pr) {

        productId = id;

        productName = name;

        quantity = qty;

        price = pr;

    }

}

public class InventorySystem {

    Product[] products = new Product[100];

    int count = 0;

    void addProduct(Product p) {

        products[count++] = p;

    }

    void updateProduct(int productId, int quantity, double price) {

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

            if (products[i].productId == productId) {

                products[i].quantity = quantity;

                products[i].price = price;

                return;

            }

        }

    }

    void deleteProduct(int productId) {

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

            if (products[i].productId == productId) {

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

                    products[j] = products[j + 1];

                }

                count--;

                return;

            }

        }

    }

    void printInventory() {

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

            System.out.println(products[i].productId + " " + products[i].productName + " " + products[i].quantity + " " + products[i].price);

        }

    }

    public static void main(String[] args) {

        InventorySystem inv = new InventorySystem();

        inv.addProduct(new Product(1, "Pen", 100, 1.5));

        inv.addProduct(new Product(2, "Notebook", 50, 2.0));

        inv.printInventory();

        inv.updateProduct(1, 120, 1.6);

        inv.deleteProduct(2);

        System.out.println("After Update and Delete:");

        inv.printInventory();

    }

}

## **Output:**

# **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.

## **Code:**

class ProductSearch {

    int productId;

    String productName;

    String category;

    ProductSearch(int id, String name, String cat) {

        productId = id;

        productName = name;

        category = cat;

    }

}

public class Search {

    ProductSearchlinearSearch(ProductSearch[] arr, String name) {

        for (int i = 0; i<arr.length; i++) {

            if (arr[i].productName.equals(name)) return arr[i];

        }

        return null;

    }

    ProductSearchbinarySearch(ProductSearch[] arr, String name) {

        int low = 0;

        int high = arr.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int cmp = arr[mid].productName.compareTo(name);

            if (cmp == 0) return arr[mid];

            else if (cmp< 0) low = mid + 1;

            else high = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

        ProductSearch[] products = {

            new ProductSearch(1, "Apple", "Fruit"),

            new ProductSearch(2, "Banana", "Fruit"),

            new ProductSearch(3, "Carrot", "Vegetable")

        };

        Search s = new Search();

        ProductSearch result = s.linearSearch(products, "Banana");

        if (result != null) System.out.println("Linear Search Found: " + result.productName);

        ProductSearch[] sorted = {

            new ProductSearch(1, "Apple", "Fruit"),

            new ProductSearch(2, "Banana", "Fruit"),

            new ProductSearch(3, "Carrot", "Vegetable")

        };

        ProductSearch result2 = s.binarySearch(sorted, "Carrot");

        if (result2 != null) System.out.println("Binary Search Found: " + result2.productName);

    }

}

## **Output:**

A black screen with white text

AI-generated content may be incorrect.

# **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.

## **Code:**

class Order {

    int orderId;

    String customerName;

    double totalPrice;

    Order(int id, String name, double price) {

        orderId = id;

        customerName = name;

        totalPrice = price;

    }

}

public class OrderSorting {

    void bubbleSort(Order[] orders) {

        int n = orders.length;

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

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

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

                    Order temp = orders[j];

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

                    orders[j + 1] = temp;

                }

            }

        }

    }

    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);

        }

    }

    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;

    }

    void printOrders(Order[] orders) {

        for (Order o : orders) {

            System.out.println(o.orderId + " " + o.customerName + " $" + o.totalPrice);

        }

    }

    public static void main(String[] args) {

        Order[] orders = {

            new Order(1, "Alice", 250.0),

            new Order(2, "Bob", 150.0),

            new Order(3, "Charlie", 200.0)

        };

        OrderSorting sorter = new OrderSorting();

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

        sorter.printOrders(orders);

        sorter.bubbleSort(orders);

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

        sorter.printOrders(orders);

        Order[] orders2 = {

            new Order(1, "Alice", 250.0),

            new Order(2, "Bob", 150.0),

            new Order(3, "Charlie", 200.0)

        };

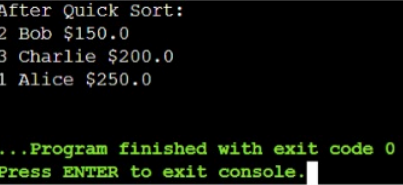
        System.out.println("After Quick Sort:");

        sorter.quickSort(orders2, 0, orders2.length - 1);

        sorter.printOrders(orders2);

    }  }

## **Output:**



# **Exercise 4: Employee Management System**

## **Scenario:**

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

## **Code:**

class Employee {

    int employeeId;

    String name;

    String position;

    double salary;

    Employee(int id, String nm, String pos, double sal) {

        employeeId = id;

        name = nm;

        position = pos;

        salary = sal;

    }

}

public class EmployeeSystem {

    Employee[] employees = new Employee[100];

    int count = 0;

    void addEmployee(Employee e) {

        employees[count++] = e;

    }

    Employee searchEmployee(int id) {

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

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

        }

        return null;

    }

    void traverseEmployees() {

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

            System.out.println(employees[i].employeeId + " " + employees[i].name);

        }

    }

    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];

                }

                count--;

                return;

            }

        }

    }

    public static void main(String[] args) {

        EmployeeSystem sys = new EmployeeSystem();

        sys.addEmployee(new Employee(101, "John", "Manager", 80000));

        sys.addEmployee(new Employee(102, "Sara", "Engineer", 60000));

        sys.traverseEmployees();

        sys.deleteEmployee(101);

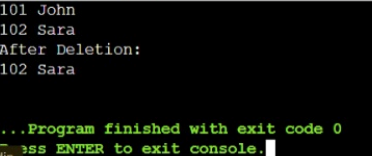
        System.out.println("After Deletion:");

        sys.traverseEmployees();

    }

}

## **Output:**



# **Exercise 5: Task Management System**

## **Scenario:**

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

## **Code:**

class Task {

    int taskId;

    String taskName;

    String status;

    Task next;

    Task(int id, String name, String stat) {

        taskId = id;

        taskName = name;

        status = stat;

        next = null;

    }

}

public class TaskList {

    Task head = null;

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

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

        if (head == null) {

            head = newTask;

        } else {

            Task temp = head;

            while (temp.next != null) temp = temp.next;

            temp.next = newTask;

        }

    }

    Task searchTask(int id) {

        Task temp = head;

        while (temp != null) {

            if (temp.taskId == id) return temp;

            temp = temp.next;

        }

        return null;

    }

    void traverseTasks() {

        Task temp = head;

        while (temp != null) {

            System.out.println(temp.taskId + " " + temp.taskName + " " + temp.status);

            temp = temp.next;

        }

    }

    void deleteTask(int id) {

        if (head == null) return;

        if (head.taskId == id) {

            head = head.next;

            return;

        }

        Task prev = head;

        Task curr = head.next;

        while (curr != null) {

            if (curr.taskId == id) {

                prev.next = curr.next;

                return;

            }

            prev = curr;

            curr = curr.next;

        }

    }

    public static void main(String[] args) {

        TaskList list = new TaskList();

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

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

        list.traverseTasks();

        list.deleteTask(1);

        System.out.println("After Deletion:");

        list.traverseTasks();

    }

}

## **Output:**A screen shot of a computer program

# **Exercise 6: Library Management System**

## **Scenario:**

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

## **Code:**

class Book {

    int bookId;

    String title;

    String author;

    Book(int id, String t, String a) {

        bookId = id;

        title = t;

        author = a;

    }

}

public class LibrarySearch {

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

        for (int i = 0; i<books.length; i++) {

            if (books[i].title.equals(title)) return books[i];

        }

        return null;

    }

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

        int low = 0;

        int high = books.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int cmp = books[mid].title.compareTo(title);

            if (cmp == 0) return books[mid];

            else if (cmp< 0) low = mid + 1;

            else high = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

        Book[] books = {

            new Book(1, "Algorithms", "Sedgewick"),

            new Book(2, "Data Structures", "Goodrich"),

            new Book(3, "Java Basics", "James Gosling")

        };

        LibrarySearch search = new LibrarySearch();

        Book found = search.linearSearch(books, "Java Basics");

        if (found != null) System.out.println("Linear Search: " + found.title);

        Book[] sortedBooks = {

            new Book(1, "Algorithms", "Sedgewick"),

            new Book(2, "Data Structures", "Goodrich"),

            new Book(3, "Java Basics", "James Gosling")

        };

        Book found2 = search.binarySearch(sortedBooks, "Data Structures");

        if (found2 != null) System.out.println("Binary Search: " + found2.title);

    }

}

## **Output:**

A screen shot of a computer

AI-generated content may be incorrect.

# **Exercise 7: Financial Forecasting**

## **Scenario:**

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

## **Code:**

public class Forecast {

    double futureValue(double currentValue, double growthRate, int years) {

        if (years == 0) return currentValue;

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

    }

    public static void main(String[] args) {

        Forecast f = new Forecast();

        double result = f.futureValue(1000, 0.05, 3);

        System.out.println("Future Value after 3 years: $" + result);

    }

}

## **Output:**

