**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;

**public** Product(**int** id, **String** name, **int** qty, **double** price) {

        this.productId **=** id;

        this.productName **=** name;

        this.quantity **=** qty;

        this.price **=** price;

    }

}

**class** Inventory {

**HashMap**<**Integer**, **Product**> products **=** **new** **HashMap**<>();

**void** addProduct(**Product** p) {

        products.put(p.productId, p);

    }

**void** updateProduct(**int** id, **int** newQty, **double** newPrice) {

**if** (products.containsKey(id)) {

**Product** p **=** products.get(id);

            p.quantity **=** newQty;

            p.price **=** newPrice;

        }

    }

**void** deleteProduct(**int** id) {

        products.remove(id);

    }

**void** displayInventory() {

**for** (**Product** p **:** products.values()) {

            System.out.println("ID: " **+** p.productId **+** ", Name: " **+** p.productName **+**

                    ", Qty: " **+** p.quantity **+** ", Price: ₹" **+** p.price);

        }

    }

}

**public** **class** InventorySystem {

**public** **static** **void** main(**String**[] args) {

**Inventory** inv **=** **new** Inventory();

        inv.addProduct(**new** Product(1, "Mouse", 10, 299.99));

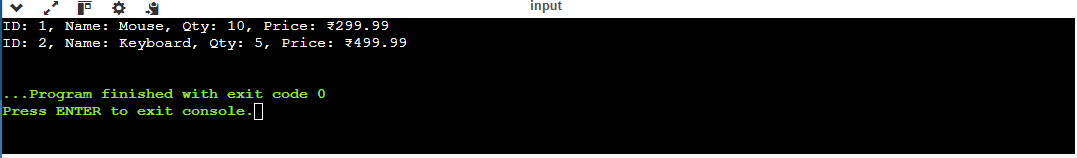
        inv.addProduct(**new** Product(2, "Keyboard", 5, 499.99));

        inv.displayInventory();

    }

}

**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:**

**import** **java.util.\***;

**class** SearchProduct {

**int** productId;

**String** productName;

**String** category;

**public** SearchProduct(**int** id, **String** name, **String** category) {

        this.productId **=** id;

        this.productName **=** name;

        this.category **=** category;

    }

}

**public** **class** SearchFunction {

**static** **SearchProduct** linearSearch(**SearchProduct**[] products, **String** name) {

**for** (**SearchProduct** p **:** products) {

**if** (p.productName.equalsIgnoreCase(name)) **return** p;

        }

**return** **null**;

    }

**static** **SearchProduct** binarySearch(**SearchProduct**[] products, **String** name) {

**int** low **=** 0, high **=** products.length **-** 1;

**while** (low **<=** high) {

**int** mid **=** (low **+** high) **/** 2;

**int** cmp **=** products[mid].productName.compareToIgnoreCase(name);

**if** (cmp **==** 0) **return** products[mid];

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

**else** high **=** mid **-** 1;

        }

**return** **null**;

    }

**public** **static** **void** main(**String**[] args) {

**SearchProduct**[] items **=** {

**new** SearchProduct(1, "Phone", "Electronics"),

**new** SearchProduct(2, "Tablet", "Electronics"),

**new** SearchProduct(3, "Watch", "Accessories")

        };

        Arrays.sort(items, Comparator.comparing(p **->** p.productName));

**SearchProduct** result **=** binarySearch(items, "Watch");

**if** (result **!=** **null**)

            System.out.println("Found: " **+** result.productName);

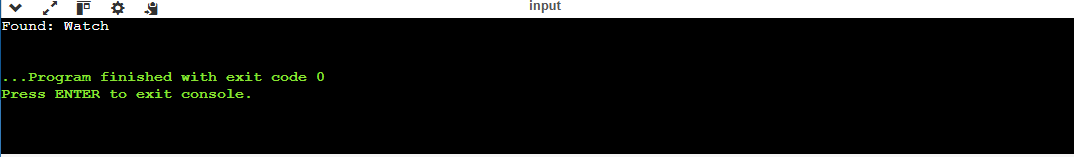
**else**

            System.out.println("Not Found");

    }

}

**OUTPUT:**

****

**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;

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

        this.orderId **=** id;

        this.customerName **=** name;

        this.totalPrice **=** price;

    }

}

**public** **class** OrderSorting {

**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;

                }

            }

        }

    }

**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** printOrders(**Order**[] orders) {

**for** (**Order** o **:** orders) {

            System.out.println("OrderID: " **+** o.orderId **+** ", Name: " **+** o.customerName **+** ", Price: ₹" **+** o.totalPrice);

        }

    }

**public** **static** **void** main(**String**[] args) {

**Order**[] orders **=** {

**new** Order(1, "Alice", 1500),

**new** Order(2, "Bob", 500),

**new** Order(3, "Charlie", 1200)

        };

        System.out.println("Sorted by Bubble Sort:");

        bubbleSort(orders);

        printOrders(orders);

**Order**[] newOrders **=** {

**new** Order(4, "David", 1100),

**new** Order(5, "Eve", 2000),

**new** Order(6, "Frank", 900)

        };

        System.out.println("\nSorted by Quick Sort:");

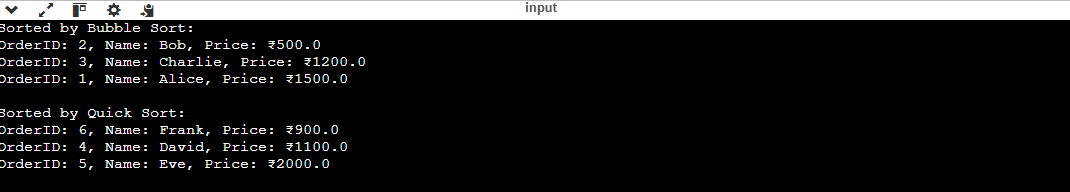
        quickSort(newOrders, 0, newOrders.length **-** 1);

        printOrders(newOrders);

    }

}.

**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;

**public** Employee(**int** id, **String** name, **String** pos, **double** sal) {

        this.employeeId **=** id;

        this.name **=** name;

        this.position **=** pos;

        this.salary **=** sal;

    }

}

**public** **class** EmployeeManager {

**Employee**[] employees **=** **new** **Employee**[100];

**int** count **=** 0;

**void** addEmployee(**Employee** e) {

**if** (count **<** employees.length) {

            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** 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**;

**break**;

            }

        }

    }

**void** displayEmployees() {

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

            System.out.println("ID: " **+** employees[i].employeeId **+** ", Name: " **+** employees[i].name **+**

                    ", Position: " **+** employees[i].position **+** ", Salary: ₹" **+** employees[i].salary);

        }

    }

**public** **static** **void** main(**String**[] args) {

**EmployeeManager** manager **=** **new** EmployeeManager();

        manager.addEmployee(**new** Employee(101, "Asha", "Developer", 50000));

        manager.addEmployee(**new** Employee(102, "Bala", "Tester", 40000));

        manager.displayEmployees();

        System.out.println("\nAfter deleting employee with ID 101:");

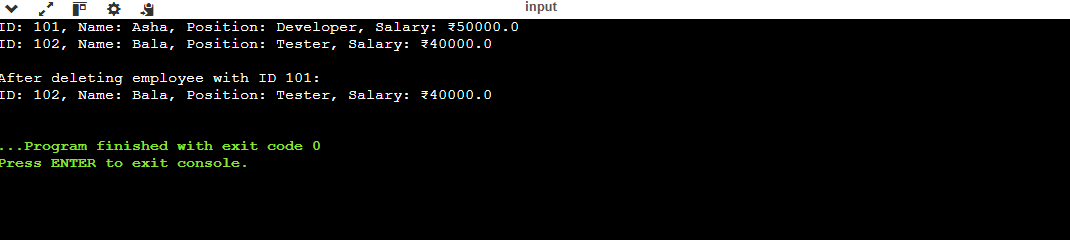
        manager.deleteEmployee(101);

        manager.displayEmployees();

    }

}

**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;

**public** Task(**int** id, **String** name, **String** status) {

        this.taskId **=** id;

        this.taskName **=** name;

        this.status **=** status;

        this.next **=** **null**;

    }

}

**public** **class** TaskManager {

**Task** head;

**void** addTask(**Task** newTask) {

**if** (head **==** **null**) {

            head **=** newTask;

        } **else** {

**Task** current **=** head;

**while** (current.next **!=** **null**) {

                current **=** current.next;

            }

            current.next **=** newTask;

        }

    }

**void** deleteTask(**int** id) {

**if** (head **==** **null**) **return**;

**if** (head.taskId **==** id) {

            head **=** head.next;

**return**;

        }

**Task** current **=** head;

**while** (current.next **!=** **null** **&&** current.next.taskId **!=** id) {

            current **=** current.next;

        }

**if** (current.next **!=** **null**) {

            current.next **=** current.next.next;

        }

    }

**Task** searchTask(**int** id) {

**Task** current **=** head;

**while** (current **!=** **null**) {

**if** (current.taskId **==** id) **return** current;

            current **=** current.next;

        }

**return** **null**;

    }

**void** displayTasks() {

**Task** current **=** head;

**while** (current **!=** **null**) {

            System.out.println("ID: " **+** current.taskId **+** ", Task: " **+** current.taskName **+** ", Status: " **+** current.status);

            current **=** current.next;

        }

    }

**public** **static** **void** main(**String**[] args) {

**TaskManager** manager **=** **new** TaskManager();

        manager.addTask(**new** Task(1, "Design UI", "Pending"));

        manager.addTask(**new** Task(2, "Write Backend", "In Progress"));

        manager.displayTasks();

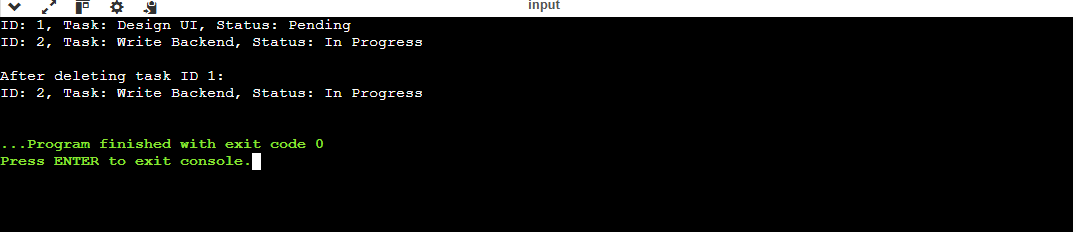
        System.out.println("\nAfter deleting task ID 1:");

        manager.deleteTask(1);

        manager.displayTasks();

    }

}

**Output: **

**Exercise 6: Library Management System**

**Scenario:**

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

**Code:**

**import** **java.util.\***;

**class** Book {

**int** bookId;

**String** title;

**String** author;

**public** Book(**int** id, **String** title, **String** author) {

        this.bookId **=** id;

        this.title **=** title;

        this.author **=** author;

    }

}

**public** **class** LibrarySearch {

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

**for** (**Book** b **:** books) {

**if** (b.title.equalsIgnoreCase(title)) **return** b;

        }

**return** **null**;

    }

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

**int** low **=** 0, high **=** books.length **-** 1;

**while** (low **<=** high) {

**int** mid **=** (low **+** high) **/** 2;

**int** cmp **=** books[mid].title.compareToIgnoreCase(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(101, "Data Structures", "Balagurusamy"),

**new** Book(102, "Operating Systems", "Galvin"),

**new** Book(103, "Computer Networks", "Tanenbaum")

        };

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

        System.out.println("Binary Search for 'Operating Systems':");

**Book** found **=** binarySearch(books, "Operating Systems");

**if** (found **!=** **null**)

            System.out.println("Found: " **+** found.title **+** " by " **+** found.author);

**else**

            System.out.println("Book not found.");

        System.out.println("\nLinear Search for 'Data Structures':");

        found **=** linearSearch(books, "Data Structures");

**if** (found **!=** **null**)

            System.out.println("Found: " **+** found.title **+** " by " **+** found.author);

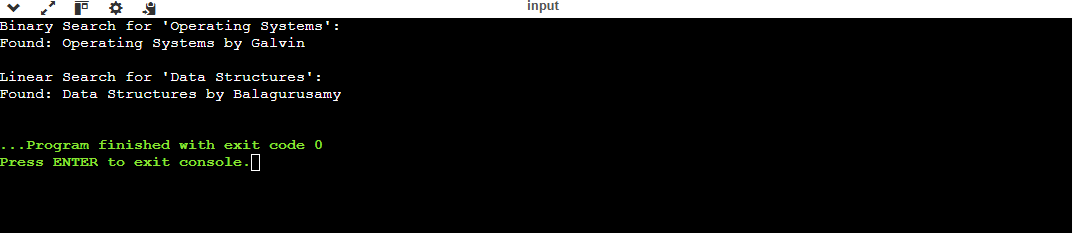
**else**

            System.out.println("Book not found.");

    }

}

**Output:**

****

**Exercise 7: Financial Forecasting**

**Scenario:**

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

**Code:**

**public** **class** ForeCastTool {

**static** **double** predictRecursive(**double** currentValue, **double** rate, **int** years) {

**if** (years **==** 0) **return** currentValue;

**return** predictRecursive(currentValue **\*** (1 **+** rate), rate, years **-** 1);

    }

**static** **double** predictIterative(**double** currentValue, **double** rate, **int** years) {

**for** (**int** i **=** 0; i **<** years; i**++**) {

            currentValue **\*=** (1 **+** rate);

        }

**return** currentValue;

    }

**public** **static** **void** main(**String**[] args) {

**double** baseAmount **=** 10000;

**double** growthRate **=** 0.1

**int** numYears **=** 5;

**double** recursiveResult **=** predictRecursive(baseAmount, growthRate, numYears);

**double** iterativeResult **=** predictIterative(baseAmount, growthRate, numYears);

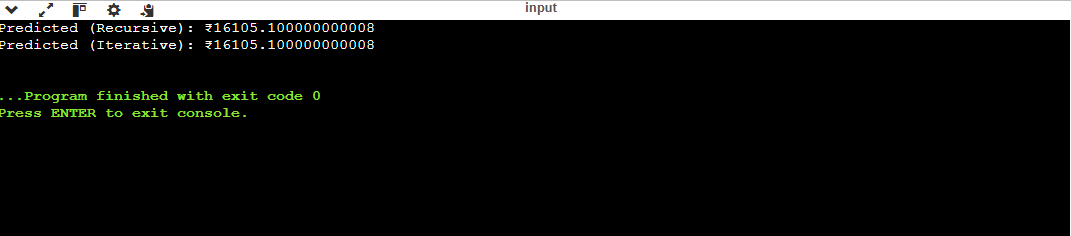
        System.out.println("Predicted (Recursive): ₹" **+** recursiveResult);

        System.out.println("Predicted (Iterative): ₹" **+** iterativeResult);

    }

}

**Output:**

****