**Exercise 3: Sorting Customer Orders**

**Q) Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**

1.Bubble Sort repeatedly compares and swaps adjacent elements if they are in the wrong order.

2.Insertion Sort builds the sorted list one item at a time by inserting each new element into its correct position.

3.Quick Sort uses a pivot to partition the list into smaller and larger elements and recursively sorts them.

4.Merge Sort divides the list into halves, sorts them recursively, and merges the sorted halves.

**Code:**

**Order Class: -**

public class Order {  
 public int orderId;  
 public String customerName;  
 public int totalPrice;  
  
 public Order(int orderId, String customerName, int totalPrice) {  
 this.orderId = orderId;  
 this.customerName = customerName;  
 this.totalPrice = totalPrice;  
 }  
 @Override  
 public String toString() {  
 return "Order ID: " + orderId + ", Name: " + customerName + ", Price: " + totalPrice;  
 }  
  
 public int getOrderId() {  
 return orderId;  
 }  
  
 public void setOrderId(int orderId) {  
 this.orderId = orderId;  
 }  
  
 public String getCustomerName() {  
 return customerName;  
 }  
  
 public void setCustomerName(String customerName) {  
 this.customerName = customerName;  
 }  
  
 public int getTotalPrice() {  
 return totalPrice;  
 }  
  
 public void setTotalPrice(int totalPrice) {  
 this.totalPrice = totalPrice;  
 }  
}

**Sorting Class: -**

public class Sorting {  
 //Bubble Sort:-  
 public void BubbleSort(Order[] orders){  
 int n= orders.length;  
 Order temp=null;  
 boolean swapped;  
 for(int i=0;i<n-1;i++){  
 swapped=false;  
 for(int j=0;j<n-i-1;j++){  
 if(orders[j].getTotalPrice()<orders[j+1].getTotalPrice()){  
 temp=orders[j];  
 orders[j]=orders[j+1];  
 orders[j+1]=temp;  
 swapped=true;  
 }  
 }  
 if(!swapped) break;  
 }  
 }  
  
 //Quick Sort:-  
 public void quickSort(Order[] orders, int low, int high) {  
 if (low < high) {  
 int mid = partition(orders, low, high);  
 quickSort(orders, low, mid - 1);  
 quickSort(orders, mid + 1, high);  
 }  
 }  
  
 private int partition(Order[] orders, int low, int high) {  
 int 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;  
 }  
  
 public void displayOrders(Order[] orders) {  
 for (Order order : orders) {  
 System.*out*.println(order);  
 }  
 }  
}

**Main Class: -**

public class Main {  
 public static void main(String[] args){  
 Order[] orders={  
 new Order(201, "Aarav Sharma", 4599),  
 new Order(202, "Meera Reddy", 1250),  
 new Order(203, "Rahul Verma", 8999),  
 new Order(204, "Sneha Kapoor", 3299),  
 new Order(205, "Vikram Das", 6790)  
 };  
  
 Sorting sorting=new Sorting();  
 System.*out*.println("Original Orders:");  
 sorting.displayOrders(orders);  
  
 sorting.BubbleSort(orders);  
 System.*out*.println("\nOrders sorted by Bubble Sort:");  
 sorting.displayOrders(orders);  
  
 Order[] ordersForQuickSort = {  
 new Order(201, "Aarav Sharma", 4599),  
 new Order(202, "Meera Reddy", 1250),  
 new Order(203, "Rahul Verma", 8999),  
 new Order(204, "Sneha Kapoor", 3299),  
 new Order(205, "Vikram Das", 6790)  
 };  
 Sorting sorting1=new Sorting();  
  
 sorting1.quickSort(ordersForQuickSort, 0, ordersForQuickSort.length - 1);  
 System.*out*.println("\nOrders sorted by Quick Sort:");  
 sorting1.displayOrders(ordersForQuickSort);  
  
 }  
}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Q) Compare the performance (time complexity) of Bubble Sort and Quick Sort.**

Bubble Sort has a time complexity of **O(n²)** in the worst and average cases.  
Quick Sort has an average time complexity of **O(n log n)** and worst-case of **O(n²)**, but with good pivot choices, it's usually much faster than Bubble Sort.

**Q) Discuss why Quick Sort is generally preferred over Bubble Sort.**

Quick Sort is preferred because it is significantly faster on average and handles large datasets efficiently. Bubble Sort is simple but slow and inefficient for anything beyond small inputs.