**EXERCISE 3:**

**SORTING CUSTOMER ORDERS:**

Bubble Sort:

* Simple comparison-based algorithm where each pair of adjacent elements is compared, and the elements are swapped if they are in the wrong order.
* Time Complexity: O(n^2) in the worst and average case, O(n) in the best case (already sorted array).

Insertion Sort:

* Builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
* Time Complexity: O(n^2) in the worst and average case, O(n) in the best case .

Quick Sort:

* Divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.
* Time Complexity: O(n log n) on average, O(n^2) in the worst case, which can be mitigated with good pivot selection strategies.

Merge Sort:

* Divide-and-conquer algorithm. It works by recursively dividing the array into two halves, sorting each half, and then merging the two sorted halves.
* Time Complexity: O(n log n) in all cases (best, average, and worst).

|  |  |
| --- | --- |
| **BUBBLE SORT** | **QUICK SORT** |
| Bubble Sort is simpler to implement but is inefficient for large datasets due to its O(n^2) time complexity in the average and worst cases | Quick Sort is more complex but generally preferred over Bubble Sort for large datasets because it has an average-case time complexity of O(n log n) |
| Best Case: O(n) (when the array is already sorted)  Average Case: O(n^2)  Worst Case: O(n^2) | Best Case: O(n log n)  Average Case: O(n log n)  Worst Case: O(n^2) (when the pivot selection is poor) |

// ECommerceOrderSortingExample.java  
  
class Order {  
 private String orderId;  
 private String customerName;  
 private double totalPrice;  
  
 public Order(String orderId, String customerName, double totalPrice) {  
 this.orderId = orderId;  
 this.customerName = customerName;  
 this.totalPrice = totalPrice;  
 }  
  
 public String getOrderId() { return orderId; }  
 public String getCustomerName() { return customerName; }  
 public double getTotalPrice() { return totalPrice; }  
  
 @Override  
 public String toString() {  
 return "Order{" +  
 "orderId='" + orderId + '\'' +  
 ", customerName='" + customerName + '\'' +  
 ", totalPrice=" + totalPrice +  
 '}';  
 }  
}  
  
class Sorting {  
  
 // Bubble Sort  
 public static void bubbleSort(Order[] orders) {  
 int n = orders.length;  
 boolean swapped;  
 for (int i = 0; i < n - 1; i++) {  
 swapped = false;  
 for (int j = 0; j < n - 1 - i; j++) {  
 if (orders[j].getTotalPrice() > orders[j + 1].getTotalPrice()) {  
 // Swap orders[j] and orders[j + 1]  
 Order temp = orders[j];  
 orders[j] = orders[j + 1];  
 orders[j + 1] = temp;  
 swapped = true;  
 }  
 }  
 // If no two elements were swapped in the inner loop, then the array is already sorted  
 if (!swapped) break;  
 }  
 }  
  
 // Quick Sort  
 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[] orders, int low, int high) {  
 double pivot = orders[high].getTotalPrice();  
 int i = (low - 1); // index of smaller element  
 for (int j = low; j < high; j++) {  
 if (orders[j].getTotalPrice() <= pivot) {  
 i++;  
 // Swap orders[i] and orders[j]  
 Order temp = orders[i];  
 orders[i] = orders[j];  
 orders[j] = temp;  
 }  
 }  
 // Swap orders[i + 1] and orders[high] (or pivot)  
 Order temp = orders[i + 1];  
 orders[i + 1] = orders[high];  
 orders[high] = temp;  
 return i + 1;  
 }  
}  
  
public class SortingCustomerOrders {  
 public static void main(String[] args) {  
 Order[] orders = {  
 new Order("1", "Manoj", 250.0),  
 new Order("2", "Kumar", 150.0),  
 new Order("3", "Microsoft", 300.0),  
 new Order("4", "Google", 200.0)  
 };  
  
 // Bubble Sort  
 Sorting.*bubbleSort*(orders);  
 System.*out*.println("Orders sorted by totalPrice using Bubble Sort:");  
 for (Order order : orders) {  
 System.*out*.println(order);  
 }  
  
 // Reset the array to original order for Quick Sort demonstration  
 orders = new Order[]{  
 new Order("1", "Manoj", 250.0),  
 new Order("2", "Kumar", 150.0),  
 new Order("3", "Microsoft", 300.0),  
 new Order("4", "Google", 200.0)  
 };  
  
 // Quick Sort  
 Sorting.*quickSort*(orders, 0, orders.length - 1);  
 System.*out*.println("\nOrders sorted by totalPrice using Quick Sort:");  
 for (Order order : orders) {  
 System.*out*.println(order);  
 }  
 }  
}