**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.

**Steps:**

1. **Understand Sorting Algorithms:**
   * **Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**

**Bubble Sort**

* **Description**: Bubble Sort is a simple comparison-based sorting algorithm. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.
* **Time Complexity**:
  + Best: O(n)
  + Average: O(n^2)
  + Worst: O(n^2)
* **Space Complexity**: O(1) (in-place)

**Insertion Sort**

* **Description**: Insertion Sort builds the final sorted array one item at a time. It is much like sorting playing cards in your hands. The algorithm removes one element from the input data, finds the location it belongs to within the sorted list, and inserts it there.
* **Time Complexity**:
  + Best: O(n)
  + Average: O(n^2)
  + Worst: O(n^2)
* **Space Complexity**: O(1) (in-place)

**Quick Sort**

* **Description**: Quick Sort is a highly efficient sorting algorithm. It uses a divide-and-conquer approach to partition the array into sub-arrays and then sorts the sub-arrays independently. The key operation is partitioning the array around a pivot element.
* **Time Complexity**:
  + Best: O(n log n)
  + Average: O(n log n)
  + Worst: O(n^2)
* **Space Complexity**: O(log n) (in-place)

**Merge Sort**

* **Description**: Merge Sort is a stable, comparison-based, and divide-and-conquer sorting algorithm. It divides the array into two halves, recursively sorts them, and finally merges the sorted halves.
* **Time Complexity**:
  + Best: O(n log n)
  + Average: O(n log n)
  + Worst: O(n log n)
* **Space Complexity**: O(n) (not in-place)

1. **Setup:**
   * **Create a class Order with attributes like orderId, customerName, and totalPrice.**
2. **Implementation:**
   * **Implement Bubble Sort to sort orders by totalPrice.**

import java.util.List;

public class BubbleSort {

public static void sort(List<Order> orders) {

int n = orders.size();

boolean swapped;

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

swapped = false;

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

if (orders.get(j).getTotalPrice() > orders.get(j + 1).getTotalPrice()) {

// swap orders[j] and orders[j+1]

Order temp = orders.get(j);

orders.set(j, orders.get(j + 1));

orders.set(j + 1, temp);

swapped = true;

}

}

// If no two elements were swapped by inner loop, then break

if (!swapped) break;

}

}

}

* + **Implement Quick Sort to sort orders by totalPrice.**

import java.util.List;

public class QuickSort {

public static void sort(List<Order> orders) {

quickSort(orders, 0, orders.size() - 1);

}

private static void quickSort(List<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(List<Order> orders, int low, int high) {

double pivot = orders.get(high).getTotalPrice();

int i = (low - 1); // index of smaller element

for (int j = low; j < high; j++) {

if (orders.get(j).getTotalPrice() <= pivot) {

i++;

// swap orders[i] and orders[j]

Order temp = orders.get(i);

orders.set(i, orders.get(j));

orders.set(j, temp);

}

}

// swap orders[i+1] and orders[high] (or pivot)

Order temp = orders.get(i + 1);

orders.set(i + 1, orders.get(high));

orders.set(high, temp);

return i + 1;

}

}

1. **Analysis:**
   * **Compare the performance (time complexity) of Bubble Sort and Quick Sort.**

 **Bubble Sort:** Time complexity is O(n^2). It is inefficient for large datasets because it performs a large number of swaps and comparisons.

 **Quick Sort:** Time complexity is O(n log n) on average. It is more efficient for larger datasets compared to Bubble Sort. It performs fewer comparisons and swaps due to the divide-and-conquer approach.

* + **Discuss why Quick Sort is generally preferred over Bubble Sort.**

 Quick Sort generally outperforms Bubble Sort because it reduces the number of comparisons and swaps needed to sort the array.

 Quick Sort's divide-and-conquer strategy makes it much more efficient on average, achieving O(n log n) time complexity.

 Bubble Sort can be useful for small datasets or nearly sorted arrays, but Quick Sort is better suited for larger and more complex datasets due to its better average-case performance.