**Exercise 3: Sorting Customer Orders – Discussion**

**Scenario**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This allows the system to prioritize high-value orders for quicker processing or better customer service.

**Understanding Sorting Algorithms**

**1. Bubble Sort**

Bubble Sort 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 Case: O(n) *(already sorted)*
  + Average Case: O(n²)
  + Worst Case: O(n²)
* **Space Complexity**: O(1) *(in-place)*

**2. Insertion Sort *(not implemented)***

Insertion Sort builds the sorted list one element at a time by comparing and inserting it into the correct position.

* **Time Complexity**: Best: O(n), Worst: O(n²)
* **Use Case**: Efficient for small datasets or nearly sorted data.

**3. Quick Sort**

Quick Sort is a divide-and-conquer algorithm that partitions the array around a pivot. It recursively sorts elements on the left and right of the pivot.

* **Time Complexity**:
  + Best Case: O(n log n)
  + Average Case: O(n log n)
  + Worst Case: O(n²) *(rare, when pivot is always the smallest/largest element)*
* **Space Complexity**: O(log n) *(recursive stack)*

**4. Merge Sort *(not implemented)***

Merge Sort divides the list into halves, recursively sorts each half, and then merges the sorted halves.

* **Time Complexity**: O(n log n) in all cases.
* **Space Complexity**: O(n) due to additional array for merging.

**Implementation Details**

**Classes Created:**

* **OrderProduct** – Represents a product in the order with fields: productId, productName, price.
* **Order** – Represents an order with orderId, customerName, and a list of OrderProduct. The totalPrice is calculated based on the products.

**Sorting Implementations:**

1. **Bubble Sort**:
   * Implemented with an optimization to stop early if the array is already sorted.
   * Stable but inefficient on large datasets.
2. **Quick Sort**:
   * Efficient and recursive implementation based on totalPrice as the pivot comparator.
   * Used cloned array to maintain original data integrity.

**Testing:**

* Original orders were cloned before sorting.
* Results were printed before and after sorting using each algorithm for comparison.

**Performance Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time Complexity (Avg)** | **Space Complexity** | **Notes** |
| Bubble Sort | O(n²) | O(1) | Simple but inefficient |
| Quick Sort | O(n log n) | O(log n) | Preferred for larger datasets |

**Conclusion**

* **Quick Sort** is generally **preferred** over **Bubble Sort** because of its superior time efficiency on larger datasets.
* Bubble Sort is educational and suitable for very small lists, but not practical in real-world applications.
* In an industrial setting like an e-commerce platform, using **efficient algorithms like Quick Sort** helps scale better as the number of orders increases.