**Exercise 3: Sorting Customer Orders**

**1. Bubble Sort**

* **Description**: Bubble Sort repeatedly traverses the list, comparing and swapping adjacent elements until no more swaps are needed, which means the list is sorted. This simple method is easy to understand but inefficient for large datasets.
* **Time Complexity**: Best case is O(n) if the list is already sorted; average and worst cases are O(n^2). This makes it slow for large datasets.
* **Space Complexity**: O(1), as it sorts in place without additional memory.

**2. Insertion Sort**

* **Description**: Insertion Sort builds the sorted list incrementally by taking each item from the unsorted portion and inserting it into its correct position in the sorted portion. It’s efficient for small or nearly sorted lists.
* **Time Complexity**: Best case is O(n) for an already sorted list; average and worst cases are O(n^2). While efficient for small lists, it’s less suitable for larger ones.
* **Space Complexity**: O(1), as it sorts in place and doesn’t require extra storage.

**3. Quick Sort**

* **Description**: Quick Sort is a divide-and-conquer algorithm that selects a pivot, partitions the list into elements less than and greater than the pivot, and recursively sorts the partitions. It’s efficient and commonly used.
* **Time Complexity**: Best and average cases are O(n log n); worst case is O(n^2) with poor pivot selection. Optimizations like choosing a good pivot mitigate this.
* **Space Complexity**: O(log n) due to the recursion stack, making it more space-efficient compared to some other sorting algorithms.

**4. Merge Sort**

* **Description**: Merge Sort divides the list into two halves, recursively sorts each half, and merges them back together. It guarantees consistent performance and is useful for large datasets.
* **Time Complexity**: Best, average, and worst cases are all O(n log n), ensuring reliable and predictable performance regardless of the initial order.
* **Space Complexity**: O(n) because it requires additional space for the merging process, creating temporary arrays.

**Comparison**

* Quick Sort is preferred over Bubble Sort due to its O(n log n) average time complexity compared to Bubble Sort’s O(n^2). Quick Sort's efficiency and in-place sorting make it more suitable for larger datasets, while Bubble Sort is simpler but less efficient.