**Sort Customer Order**

**Bubble Sort**

Bubble sort works by repeatedly iterating through a list, comparing adjacent elements and swapping them if they're in the wrong order. This process is repeated until no more swaps are needed, indicating the list is sorted. It's simple to implement but inefficient for large datasets. Average and worst-case time complexity is O(n^2).

**Insertion Sort**

Insertion sort iterates through a list, inserting each element into its proper position within the previously sorted portion of the list. It's efficient for small datasets or nearly sorted lists. Average and worst-case time complexity is O(n^2), but best-case is O(n) for already sorted lists.

**Quick Sort**

Quick sort selects a pivot element, partitions the list around it, and recursively sorts the sublists. It's efficient for large datasets, with average-case time complexity of O(n log n). However, worst-case is O(n^2) if poor pivot choices are made. It's a divide-and-conquer approach that's generally fast but can be unstable.

**Merge Sort**

Merge sort divides a list into smaller chunks, sorts each chunk recursively, and then merges the sorted chunks back together. It's a stable, efficient algorithm with a time complexity of O(n log n) in all cases. It's useful for large datasets and can be easily parallelized. However, it requires extra memory for the merged chunks.

Quick Sort is preferred over Bubble Sort due to its faster performance, scalability, and efficient memory use. It has an average time complexity of O(n log n), making it suitable for large datasets, whereas Bubble Sort's O(n^2) complexity makes it impractically slow. Quick Sort's simplicity and cache efficiency also contribute to its widespread adoption.