**Sorting Customer Orders**

**Overview of Sorting Algorithms**

Sorting algorithms are essential for organizing customer orders efficiently. Different algorithms have various strengths and weaknesses depending on the dataset size and nature.

1. **Bubble Sort**: Bubble sort is a simple algorithm that repeatedly traverses the list, comparing and swapping adjacent elements if they are out of order. This process continues until no swaps are needed, signifying that the list is sorted. Despite its simplicity, bubble sort is inefficient for large datasets, with both average and worst-case time complexities of O(n^2).
2. **Insertion Sort**: Insertion sort builds the sorted list one element at a time by inserting each new element into its correct position within the already sorted portion. It performs well with small or nearly sorted datasets, boasting a best-case time complexity of O(n) when the list is already sorted. However, its average and worst-case time complexities remain O(n^2), making it less suitable for large or unsorted datasets.
3. **Quick Sort**: Quick sort employs a divide-and-conquer strategy. It selects a pivot element, partitions the list into elements less than and greater than the pivot, and recursively sorts the sublists. It is efficient for large datasets, with an average-case time complexity of O(n log n). However, its worst-case time complexity is O(n^2) if poor pivot choices occur. Quick sort is generally fast and cache-efficient but can be unstable if not implemented with good pivot selection strategies.
4. **Merge Sort**: Merge sort divides the list into smaller sublists, sorts each sublist recursively, and then merges the sorted sublists into a final sorted list. It is known for its stability and consistent performance with a time complexity of O(n log n) in all cases. Merge sort is ideal for large datasets and can be parallelized, though it requires additional memory for merging operations.

**Performance Analysis**

**Comparing Performance:**

* **Bubble Sort**: The algorithm has a time complexity of O(n^2) in both the average and worst cases. Its performance degrades significantly with larger datasets due to its quadratic time complexity.
* **Quick Sort**: On average, Quick Sort performs with a time complexity of O(n log n). While its worst-case time complexity is O(n^2), it can be minimized with effective pivot selection methods, such as choosing the median of the list.

**Why Quick Sort is Preferred**

Quick Sort is generally favored over Bubble Sort due to its superior performance and scalability. With an average time complexity of O(n log n), it efficiently handles large datasets compared to Bubble Sort’s O(n^2). Quick Sort also benefits from cache efficiency and can be optimized to avoid worst-case scenarios, making it a more robust and practical choice for sorting operations, particularly in e-commerce platforms.