**Sorting Customer Orders**

### Bubble Sort

Bubble Sort is a simple comparison-based sorting algorithm. It works by repeatedly stepping through the list, comparing adjacent elements and swapping them if they are in the wrong order. This process is repeated until the list is sorted.

**Algorithm:**

1. Starting from the beginning of the list, compare each pair of adjacent elements.
2. If the first element is greater than the second, swap them.
3. Repeat steps 1 and 2 for all elements in the list.
4. Repeat the entire process for the entire list until no swaps are needed.

**Time Complexity:**

* Best Case: O(n) (when the list is already sorted)
* Average Case: O(n^2)
* Worst Case: O(n^2)

### Insertion Sort

Insertion Sort builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.

**Algorithm:**

1. Start with the second element. Compare it to the elements before it and insert it into the correct position.
2. Move to the next element and repeat step 1.
3. Continue until the entire list is sorted.

**Time Complexity:**

* Best Case: O(n) (when the list is already sorted)
* Average Case: O(n^2)
* Worst Case: O(n^2)

### Quick Sort

Quick Sort is a highly efficient sorting algorithm and is based on the divide-and-conquer approach. It works by selecting a 'pivot' element and partitioning the array into two sub-arrays, according to whether they are less than or greater than the pivot.

**Algorithm:**

1. Choose a pivot element from the array.
2. Partition the array into two sub-arrays: elements less than the pivot and elements greater than the pivot.
3. Recursively apply the above steps to the sub-arrays.
4. Combine the results to get the final sorted array.

**Time Complexity:**

* Best Case: O(nlogn)
* Average Case: O(nlogn)
* Worst Case: O(n^2) (when the pivot selection is poor, e.g., always picking the smallest or largest element)

### Merge Sort

Merge Sort is another efficient, comparison-based, divide-and-conquer sorting algorithm. It works by dividing the array into two halves, sorting each half, and then merging the sorted halves.

**Algorithm:**

1. Divide the unsorted list into n sub-lists, each containing one element.
2. Repeatedly merge sub-lists to produce new sorted sub-lists until there is only one sub-list remaining.

**Time Complexity:**

* Best Case: O(nlogn)
* Average Case: O(nlogn)
* Worst Case: O(nlogn)

### Comparison of Performance (Bubble Sort vs. Quick Sort)

**Best Case Time Complexity:**

* + Bubble Sort: O(n)
  + Quick Sort: O(nlogn)

**Average Case Time Complexity:**

* + Bubble Sort: O(n^2)
  + Quick Sort: O(nlogn)

**Worst Case Time Complexity:**

* + Bubble Sort: O(n^2)
  + Quick Sort: O(n^2) (though typically can be improved with better pivot selection)

### Why Quick Sort is Generally Preferred Over Bubble Sort

1. **Efficiency:** Quick Sort is generally much faster than Bubble Sort due to its average case time complexity of O(nlogn) compared to Bubble Sort's O(n2).
2. **Divide-and-Conquer Approach:** Quick Sort uses a divide-and-conquer strategy which is efficient and makes good use of the memory hierarchy. Bubble Sort, on the other hand, repeatedly traverses the list which can be inefficient.
3. **Practical Performance:** In practice, Quick Sort performs very well on average, and with good pivot selection techniques (e.g., using median-of-three or random pivot), its worst-case performance can be mitigated.
4. **In-place Sorting:** Quick Sort is an in-place sorting algorithm (though not stable without modifications), meaning it requires only a small, constant amount of extra storage space. Bubble Sort also is in-place, but this advantage doesn't outweigh its inefficiency.

Because of these reasons, Quick Sort is generally preferred over Bubble Sort in most practical applications.