**3. Sorting Customer Orders**

**1. Bubble Sort**

* Bubble Sort is a basic, comparison-based sorting algorithm that works by repeatedly traversing the list and swapping adjacent elements if they are out of order.
* This process is repeated until no more swaps are needed, meaning the list is sorted.

**Algorithm:**

1. Begin at the first index of the array.
2. Compare the current element with the next one.
3. If the current element is greater, swap them.
4. Repeat steps 2–3 for the entire array.
5. Repeat the entire process for all elements, excluding the last sorted ones each time.

**Time Complexity:**

* **Best:** O(n) — when the array is already sorted.
* **Average:** O(n²)
* **Worst:** O(n²)

**Advantages:**

* Very easy to understand and implement.
* Does not require extra memory.

**Disadvantages:**

* Poor performance on large datasets.
* Inefficient due to the high number of unnecessary comparisons and swaps.

**2. Insertion Sort**

* Insertion Sort works by building a sorted portion of the array one element at a time.
* It takes one element and inserts it into its correct position among the already sorted elements.

**Algorithm:**

1. Start with the second element as the "key".
2. Compare the key with previous elements and shift larger elements to the right.
3. Insert the key in the appropriate position.
4. Repeat this process for each remaining element.

**Time Complexity:**

* **Best:** O(n) — if the input is already sorted.
* **Average:** O(n²)
* **Worst:** O(n²)

**Advantages:**

* Simple to code and understand.
* Performs well on small or nearly sorted datasets.
* Stable sorting algorithm.

**Disadvantages:**

* Not efficient for large datasets due to quadratic complexity.
* More comparisons and shifts are required as data size increases.

**3. Quick Sort**

* Quick Sort is a fast, divide-and-conquer sorting technique.
* It selects a pivot element and partitions the array into two groups: elements less than the pivot and elements greater than the pivot.
* It then recursively sorts both subgroups.

**Algorithm:**

1. Choose a pivot element (e.g., first, last, or random).
2. Reorder elements so that all smaller elements are before the pivot and all greater elements are after.
3. Recursively apply the same logic to the left and right sub-arrays.

**Time Complexity:**

* **Best:** O(n log n)
* **Average:** O(n log n)
* **Worst:** O(n²) — occurs when the pivot is always the smallest or largest element.

**Advantages:**

* Excellent average-case performance for large datasets.
* In-place sorting (minimal memory usage).
* Faster than simple algorithms like Bubble Sort and Insertion Sort.

**Disadvantages:**

* Not stable by default.
* Worst-case performance can be poor if pivot selection is not optimized.

**4. Merge Sort**

* Merge Sort is another divide-and-conquer algorithm that ensures consistent performance by recursively splitting the array and merging it in sorted order.
* It divides the array into halves, sorts each half, and then merges them together.

**Algorithm:**

1. Divide the array into two equal halves.
2. Recursively sort each half.
3. Merge the two sorted halves into a single sorted array.

**Time Complexity:**

* **Best:** O(n log n)
* **Average:** O(n log n)
* **Worst:** O(n log n)

**Advantages:**

* Predictable performance across all cases.
* Stable sorting method.
* Very effective for linked lists and external sorting.

**Disadvantages:**

* Requires additional memory for merging arrays.
* Slightly more complex to implement than Quick Sort or Insertion Sort.

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

**1. Time Complexity Efficiency:**  
Quick Sort typically operates in O(n log n) time in average cases, while Bubble Sort consistently performs at O(n²) for all but the best-case scenario. This makes Quick Sort far more efficient for sorting customer orders, especially as the number of orders increases.

**2. Better Scalability:**  
Quick Sort handles large data volumes effectively due to its divide-and-conquer method. It reduces the problem size with each recursive call, whereas Bubble Sort inefficiently traverses the entire array multiple times, making it impractical for high-volume operations.

**3. In-Place Sorting with Low Overhead:**  
Quick Sort does not require additional memory space like Merge Sort, making it a memory-efficient solution. Although not stable by default, its performance makes it a better trade-off than Bubble Sort, which offers stability but poor speed.

**4. Real-World Usage:**  
In production systems and frameworks, Quick Sort or its variants (e.g., introsort) are widely used for sorting due to their robust and consistent performance. Bubble Sort is often used only for educational purposes due to its simplicity.