**Sorting Customer Orders**

**Bubble Sort:**

* **Description:** Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.
* **Time Complexity:**
  + **Best Case:** O (n) – When the list is already sorted.
  + **Average Case:** O (n^2) – Due to the nested loops comparing each pair of elements.
  + **Worst Case:** O (n^2) – When the list is in reverse order.

**Insertion Sort:**

* **Description:** Insertion Sort builds the sorted list one item at a time by repeatedly taking the next item and inserting it into its correct position among the previously sorted items.
* **Time Complexity:**

**Best Case:** O (n) – When the list is already sorted.

**Average Case:** O (n^2) – Due to the nested loops.

**Worst Case:** O (n2) – When the list is in reverse order.

**Quick Sort:**

* **Description:** Quick Sort is a divide-and-conquer algorithm. It selects a 'pivot' element, partitions the list into elements less than the pivot and elements greater than the pivot, and recursively sorts the partitions.
* **Time Complexity:**

**Best Case:** O (n \log n) – When the pivot divides the list into nearly equal halves.

**Average Case:** O (n \log n) – On average, Quick Sort performs efficiently.

**Worst Case:** O (n^2) – When the pivot selections lead to unbalanced partitions (e.g., always picking the smallest or largest element as the pivot).

**Performance Comparison:**

* **Bubble Sort:**

**Best Case:** O (n)

**Average Case:**  O (n2)

**Worst Case:** O (n^2)

**Usage:** Generally used for educational purposes or when simplicity is desired for very small datasets.

* **Quick Sort:**

**Best Case:** O (n log n)

**Average Case:** O (n log n)

**Worst Case:** O (n^2) – Can be mitigated by using techniques like choosing a better pivot or using randomized Quick Sort.

**Usage:** Preferred for its efficient average-case performance and good practical performance in most scenarios.

**Quick sort is preferred**

**Efficiency:** Quick Sort typically outperforms Bubble Sort due to its O(n log n) average-case time complexity versus Bubble Sort's O(n^2). This makes Quick Sort more scalable for large datasets.

**Divide-and-Conquer:** Quick Sort's divide-and-conquer approach is generally more efficient in practice as it minimizes the number of comparisons needed to sort the data.

**Memory Usage:** Quick Sort is an in-place sorting algorithm, meaning it requires less additional memory compared to Merge Sort, which needs extra space for merging.