**Sorting Customer Orders**

# **Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**

* **Bubble Sort:** Bubble Sort is a comparison-based sorting algorithm. It repeatedly steps through the array, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until no swaps are needed, indicating that the list is sorted.
  + Worst-case time complexity of O(n²)
  + Not suitable for large datasets.
  + O(1) space complexity.
* **Insertion Sort:** Insertion Sort picks each element from the input data and inserts it into the correct position in an already sorted part of the array.
  + Efficient for small datasets and partially sorted arrays.
  + Worst-case time complexity of O(n²).
  + O(1) space complexity.
* **Quick Sort:** Quick Sort is a **divide-and-conquer** sorting algorithm. It works by selecting a 'pivot' element from the array, partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot. Then, it recursively does sorting the sub-arrays.
  + Average-case time complexity of O(n log n).
  + Worst-case is O(n²), when the partitioning algorithm picks the largest or smallest element as the pivot element every time.
  + Auxiliary space: O(log(n))
* **Merge Sort:** Merge Sort is another divide-and-conquer algorithm that divides the input array into two halves, recursively sorts them, and then merges the sorted halves. It guarantees a stable sort.
  + Time complexity of O(n log n) in all cases.
  + Efficient for large datasets.
  + O(n) additional space is required for the temporary arrays used during the merge process.

# **Compare the performance (time complexity) of Bubble Sort and Quick Sort.**

* **Bubble Sort:**
  + **Best case:** O(n)
  + **Average case:** O(n2)
  + **Worst case:** O(n2)
* **Quick Sort:**
  + **Best case:** O(n log(n))
  + **Average case:** O(nlog(n))
  + **Worst case:** O(n log(n))

# **Discuss why Quick Sort is generally preferred over Bubble Sort.**

* Bubble Sort is a comparison-based sorting algorithm. It repeatedly steps through the array, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until no swaps are needed, indicating that the list is sorted.
  + This makes it worst performing sorting algorithm for a large dataset with worst time complexity **O(n^2)**.
* Quick Sort is a **divide-and-conquer** sorting algorithm. It works by selecting a 'pivot' element from the array, partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot. Then, it recursively does sorting the sub-arrays.
  + Though, it is not totally a stable sorting algorithm, but in average cases, it does sorting at a time complexity of **O(n log(n)).**
  + Worst-case happens (time complexity O(n²)), when the partitioning algorithm picks the largest or smallest element as the pivot element every time.