**SORTING CUSTOMER ORDERS**

**Explanation Of Different Sorting Algorithms**

**1. Bubble Sort**

Bubble Sort is a basic sorting algorithm that repeatedly compares adjacent elements in an array and swaps them if they are in the wrong order. This process continues until the array is completely sorted. With each pass, the largest unsorted element "bubbles" up to its correct position at the end of the array. While it is easy to understand and implement, it is not efficient for large datasets.

In terms of performance, Bubble Sort has a best-case time complexity of **O(n)** when the array is already sorted. However, in average and worst-case scenarios, it performs at **O(n²)**, making it unsuitable for large input sizes. Despite its simplicity and no requirement for extra memory, its inefficiency makes it a poor choice for real-world applications involving significant data.

**2. Insertion Sort**

Insertion Sort works by building the sorted portion of the array one element at a time. Starting from the second element, each item is compared with the elements before it and inserted into its correct position in the sorted part of the array. It is very effective for small datasets or nearly sorted lists due to its low overhead and simplicity.

Its best-case time complexity is **O(n)**, particularly when the array is already nearly sorted. However, the average and worst-case complexities are **O(n²)**. Insertion Sort is a **stable** algorithm, meaning it preserves the relative order of equal elements, and it requires no additional memory. Still, like Bubble Sort, it does not scale well for large datasets.

**3. Quick Sort**

Quick Sort is a highly efficient divide-and-conquer algorithm. It works by selecting a pivot element and partitioning the array into two parts: elements smaller than the pivot and elements larger than the pivot. This process is then applied recursively to the sub-arrays. Quick Sort is known for its speed and is commonly used in practice.

Its average and best-case time complexity is **O(n log n)**, which makes it much more efficient than simple sorting algorithms like Bubble or Insertion Sort. However, if a poor pivot is chosen consistently (like the smallest or largest element in a sorted array), the time complexity can degrade to **O(n²)**. Despite this, it performs well in most cases and is often preferred for large datasets. It is **not a stable algorithm**, but it sorts in place with minimal additional memory usage.

**4. Merge Sort**

Merge Sort is another divide-and-conquer algorithm that divides the array into two halves, sorts them recursively, and then merges the sorted halves. It guarantees consistent performance regardless of the input order.

All of its time complexities—best, average, and worst—are **O(n log n)**, making it a reliable choice even when input data is unsorted or repetitive. Merge Sort is a **stable** sorting algorithm and ensures accurate ordering of equal elements. However, it requires additional space for the merging process, which can be a disadvantage when working with limited memory.

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

**Bubble Sort:**

* **Best Case:** O(n)
  + When the array is already sorted.
* **Average Case:** O(n²)
  + Requires many swaps and comparisons.
* **Worst Case:** O(n²)
  + When the array is sorted in reverse order.
* **Space Complexity:** O(1) (in-place sorting)
* **Stable:** Yes (preserves order of equal elements)

**Quick Sort:**

* **Best Case:** O(n log n)
  + When the pivot divides the array into equal halves.
* **Average Case:** O(n log n)
  + Efficient due to divide-and-conquer strategy.
* **Worst Case:** O(n²)
  + Occurs when pivot is smallest or largest element every time (can be reduced using random/median pivot).
* **Space Complexity:** O(log n) (due to recursive stack)
* **Stable:** No (unless implemented with extra logic)

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

Quick Sort is generally considered far superior to Bubble Sort for several reasons. Firstly, its **average-case time complexity of O(n log n)** makes it vastly faster than Bubble Sort's **O(n²)** on most datasets. This efficiency becomes more noticeable as the size of the dataset increases.

Secondly, **Quick Sort scales better** with large amounts of data due to its divide-and-conquer nature, which effectively breaks down the problem into smaller chunks, leading to faster sorting. On the other hand, Bubble Sort's simple pairwise swapping becomes increasingly inefficient as the data grows.

Finally, Quick Sort's design allows for **faster average performance**, and although it can degrade to O(n²) in the worst case, techniques such as random pivot selection or choosing the median of three elements help reduce the likelihood of that happening. For these reasons, Quick Sort is commonly used in practical applications, whereas Bubble Sort is typically used only for teaching purposes or very small datasets.