3\_Sorting Customer Orders

**a. Understanding Sorting Algorithms**

**Bubble Sort**

This is one of the simplest sorting techniques. It works by **repeatedly comparing** each pair of adjacent elements and **swapping them** if they’re in the wrong order. It “bubbles” the largest (or smallest) elements to the end with each pass.

* **Easy to understand**
* **Very slow for large data**, not used in real-world applications

**Performance:**

* Best case (already sorted): O(n)
* Average case: O(n²)
* Worst case: O(n²)

**Insertion Sort**

This algorithm builds the sorted array one item at a time by **picking an element and inserting it** in the right spot among the previously sorted items. Think of how you sort cards in your hand while playing.

* Fast for **small or nearly sorted** datasets
* Not efficient for large or random data

**Performance:**

* Best case (already sorted): O(n)
* Average case: O(n²)
* Worst case: O(n²)

**Quick Sort**

Quick Sort is a popular and efficient sorting algorithm. It uses a **divide-and-conquer** approach:

1. Picks a **pivot** value.
2. Splits the array into two parts — items less than the pivot and items greater than the pivot.
3. Recursively sorts both parts.

* **Very fast in practice**, even with large datasets
* Commonly used in built-in sorting functions (e.g., Java, Python)

**Performance:**

* Best case: O(n log n)
* Average case: O(n log n)
* Worst case (rare, bad pivot choice): O(n²)

**b. Performance Comparison: Bubble Sort vs Quick Sort**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Bubble Sort** | **Quick Sort** |
| **Best Time** | O(n) | O(n log n) |
| **Average Time** | O(n²) | O(n log n) |
| **Worst Time** | O(n²) | O(n²) (rare) |
| **Space** | O(1) (in-place) | O(log n) (recursive stack) |
| **Speed** | Slow for large data | Fast even for 100K+ elements |
| **Real-world use** | Rarely used | Widely used in libraries |

**Why Quick Sort is Preferred?**

* Bubble Sort is **too slow** for large datasets (O(n²)).
* Quick Sort is **much faster** in most real-world scenarios.
* It’s used in many real applications, like Java’s own Arrays.sort() (for primitives).
* With proper pivot strategy, Quick Sort performs close to optimal.