**Exercise 3: Sorting Customer Orders**

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

**1. Bubble Sort**

Bubble Sort is a basic sorting technique based on repeated comparisons. It scans the array multiple times, comparing adjacent elements and swapping them if they’re out of order.

**Steps:**

* Start from the beginning and compare each adjacent pair.
* Swap if the left element is larger.
* Repeat the process, reducing the range each time, until the list is sorted.

**Efficiency:**

* **Best case:** O(n) – when the array is already sorted.
* **Average and worst cases:** O(n²)

**Pros:**

* Easy to code.
* Doesn't require extra memory.

**Cons:**

* Very inefficient for large datasets.
* Not suitable for performance-critical applications.

**2. Insertion Sort**

Insertion Sort works by gradually growing a sorted section of the list. It repeatedly picks the next element and places it in its proper position within the sorted part.

**Steps:**

* Begin with the second element.
* Compare it to previous items and insert it in the right place.
* Repeat for all elements.

**Efficiency:**

* **Best case:** O(n)
* **Average and worst cases:** O(n²)

**Pros:**

* Simple to implement.
* Performs well on small or mostly sorted arrays.
* Maintains the original order of equal elements (stable).

**Cons:**

* Slows down with larger inputs.

**3. Quick Sort**

Quick Sort is an efficient, divide-and-conquer sorting algorithm. It picks a pivot and rearranges elements around it—those smaller go left, larger go right—and recursively sorts the partitions.

**Steps:**

* Select a pivot element.
* Partition the array into two sublists around the pivot.
* Recursively apply the process to each sublist.

**Efficiency:**

* **Best and average cases:** O(n log n)
* **Worst case:** O(n²), often mitigated with good pivot selection (e.g., randomized)

**Pros:**

* Very fast for large arrays.
* Uses little extra memory (in-place).

**Cons:**

* Performance depends on pivot selection.
* Not stable.

**4. Merge Sort**

Merge Sort also uses divide-and-conquer, but instead of choosing a pivot, it splits the array evenly, sorts both halves, and merges them.

**Steps:**

* Divide the array into halves.
* Sort each half recursively.
* Merge the sorted halves together.

**Efficiency:**

* **Best, average, and worst cases:** O(n log n)

**Pros:**

* Predictable performance.
* Stable sort.

**Cons:**

* Needs extra space equal to the size of the array.
* Slightly more complex to write.

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

1. **Time Complexity and Efficiency:** Quick Sort outperforms Bubble Sort in average cases with a time complexity of *O(n log n)*, while Bubble Sort remains at *O(n²)* in both average and worst scenarios. This makes Quick Sort far more efficient, especially with larger datasets.
2. **Scalability for Large Data:** Quick Sort handles large volumes of data more effectively due to its scalable nature. In contrast, Bubble Sort’s quadratic growth leads to a rapid decline in performance as data size increases.
3. **Divide-and-Conquer Strategy:** Quick Sort’s recursive divide-and-conquer technique partitions the array into manageable parts, enabling faster sorting. This approach yields stronger average-case performance compared to Bubble Sort’s simple adjacent element comparisons.