**Sorting Algorithms Explained**

**Bubble Sort**

* **How it works**: Repeatedly compares adjacent elements and swaps them if they are in the wrong order.
* **Example**: [5, 3, 8] → [3, 5, 8] after sorting.
* **Best Case**: O(n) (already sorted)
* **Average/Worst Case**: O(n²)

**Insertion Sort**

* **How it works**: Builds the sorted array one element at a time by inserting each item into its correct position.
* **Example**: [4, 3, 2] → [3, 4, 2] → [2, 3, 4]
* **Best Case**: O(n) (already sorted)
* **Average/Worst Case**: O(n²)

**Quick Sort**

* **How it works**: Picks a "pivot", partitions the array into two (less than and greater than pivot), then recursively sorts them.
* **Example**: [9, 3, 7, 1] → pivot = 7 → [3, 1], 7, [9] → sorted.
* **Best/Average Case**: O(n log n)
* **Worst Case**: O(n²) (if pivot is always smallest/largest)

**Merge Sort**

* **How it works**: Divides the array into halves, sorts each half, and merges them back.
* **Example**: [8, 4, 2, 6] → [8, 4], [2, 6] → [4, 8], [2, 6] → [2, 4, 6, 8]
* **Best/Average/Worst Case**: O(n log n)

2. Performance Comparison: Bubble Sort vs Quick Sort

| **Criteria** | **Bubble Sort** | **Quick Sort** |
| --- | --- | --- |
| Best Case | O(n) | O(n log n) |
| Average Case | O(n²) | O(n log n) |
| Worst Case | O(n²) | O(n²) (rare) |
| Extra Space | O(1) | O(log n) (recursion) |
| Stable Sort | Yes | No (default) |
| Use Case | Very small or nearly sorted data | General-purpose fast sorting |

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

* **Much faster on large datasets**: Quick Sort has average-case O(n log n) vs Bubble Sort’s O(n²).
* **Efficient memory usage**: Quick Sort sorts in-place using minimal extra space.
* **Widely optimized**: It's the basis of many real-world sorting libraries (e.g., Java, C++ std::sort).
* **Bubble Sort** is mainly used for teaching purposes and is inefficient in practice.