**Bubble Sort**

**Description:**  
• Simple comparison-based algorithm.  
• Repeatedly compares adjacent elements and swaps them if they are in the wrong order.  
• Continues until no swaps are needed, meaning the list is sorted.

**Steps:**

1. Start from the beginning of the array.
2. Compare adjacent elements.
3. Swap if they are in the wrong order.
4. Repeat the process for all elements until no swaps occur.

**Insertion Sort**

**Description:**  
• Builds the sorted list one element at a time.  
• Takes one item and inserts it into its correct position in the already sorted part of the array.

**Steps:**

1. Assume the first element is sorted.
2. Pick the next element and compare it with elements in the sorted part.
3. Shift elements if necessary and insert the current element at the correct position.
4. Repeat until all elements are sorted.

**Quick Sort**

**Description:**  
• A divide-and-conquer algorithm.  
• Selects a pivot, partitions the array into elements less than and greater than the pivot, and recursively sorts the partitions.  
• Highly efficient for large datasets.

**Steps:**

1. Choose a pivot element.
2. Partition the array such that elements less than pivot go left, greater go right.
3. Recursively apply the above steps to both partitions.

**Merge Sort**

**Description:**  
• A divide-and-conquer algorithm.  
• Splits the array into halves, sorts each half, and then merges them.  
• Guarantees O(n log n) time even in worst case.

**Steps:**

1. Divide the array into two halves.
2. Recursively sort each half.
3. Merge the sorted halves to get the final sorted array.

**4. Analysis**

**Comparison: Time Complexity**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Bubble Sort** | O(n) | O(n²) | O(n²) |
| **Quick Sort** | O(n log n) | O(n log n) | O(n²)\* |

\*Worst case occurs when pivot selection is poor (e.g., always picking first element on sorted data).

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

| **Feature** | **Bubble Sort** | **Quick Sort** |
| --- | --- | --- |
| Efficiency | Inefficient for large datasets | Very efficient for large datasets |
| Time Complexity (average) | O(n²) | O(n log n) |
| Practical Use | Educational / very small lists | Widely used in real-world systems |
| Memory Usage | In-place (no extra memory needed) | In-place (except recursive stack) |