1. Bubble Sort

- **Concept**: Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The process repeats until the list is sorted.
- **Time Complexity**:
    - Best Case: O(n) (when the list is already sorted)
    - Average Case: O(n^2)
    - Worst Case: O(n^2)
- **Characteristics**: Simple to implement but inefficient for large datasets due to its quadratic time complexity. Often used for educational purposes rather than practical applications.

2. Insertion Sort

- **Concept**: Insertion Sort builds the final sorted array one item at a time. It picks an element and inserts it into its correct position among the previously sorted elements.
- **Time Complexity**:
    - Best Case: O(n) (when the list is already sorted)
    - Average Case: O(n^2)
    - Worst Case: O(n^2)
- **Characteristics**: Efficient for small or nearly sorted datasets. It is adaptive, meaning it works well when sorting a list that is already partially sorted.

3. Quick Sort

- **Concept**: Quick Sort is a divide-and-conquer algorithm that selects a "pivot" element and partitions the array into two sub-arrays: elements less than the pivot and elements greater than the pivot. The sub-arrays are then sorted recursively.
- **Time Complexity**:
    - Best and Average Case: O(n log n)
    - Worst Case: O(n^2) (when the pivot selection consistently results in unbalanced partitions, such as when the smallest or largest element is chosen)
- **Characteristics**: Generally fast and efficient, Quick Sort is widely used in practice. The worst-case performance can be mitigated by using good pivot selection strategies (e.g., random pivot, median-of-three).

4. Merge Sort

- **Concept**: Merge Sort is a divide-and-conquer algorithm that divides the array into halves, sorts each half, and then merges the sorted halves into a single sorted list.
- **Time Complexity**: O(n log n) for all cases (best, average, worst)
- **Characteristics**: Merge Sort is stable (it preserves the relative order of equal elements) and efficient with large datasets. However, it requires additional space for merging, making it less suitable for environments with limited memory.

Time Complexity Comparison

- **Bubble Sort**:
    - Best Case: O(n) (if the array is already sorted).
    - Average and Worst Case: O(n^2).
- **Quick Sort**:
    - Best and Average Case: O(n log n).
    - Worst Case: O(n^2) (when the pivot is the smallest or largest element, causing unbalanced partitions).

Why Quick Sort is Preferred Over Bubble Sort

- **Efficiency**: Quick Sort generally performs faster than Bubble Sort, especially for large datasets, due to its O(n log n) average time complexity.
- **Practical Use**: Quick Sort is more commonly used in practice because it efficiently handles large datasets. However, it requires careful pivot selection to avoid worst-case scenarios, which can be mitigated by techniques like random pivoting or median-of-three.
- **Space Complexity**: Both Bubble Sort and Quick Sort are in-place sorting algorithms, but Quick Sort may use additional stack space due to recursion.