### **Sorting Algorithms**

#### **Bubble Sort**

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

* **Time Complexity:**
  + Best-case: O(n)
  + Average-case: O(n^2)
  + Worst-case: O(n^2)

#### **Insertion Sort**

Insertion Sort builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.

* **Time Complexity:**
  + Best-case: O(n)
  + Average-case: O(n^2)
  + Worst-case: O(n^2)

#### **Quick Sort**

Quick Sort is an efficient, divide-and-conquer, and comparison-based sorting algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.

* **Time Complexity:**
  + Best-case: O(n log n)
  + Average-case: O(n log n)
  + Worst-case: O(n^2) (rare-When array is already sorted in ascending or descending order)

#### **Merge Sort**

Merge Sort is an efficient, stable, comparison-based, divide and conquer sorting algorithm. Most implementations produce a stable sort, meaning that the implementation preserves the input order of equal elements in the sorted output.

* **Time Complexity:**
  + Best-case: O(n log n)
  + Average-case: O(n log n)
  + Worst-case: O(n log n)

#### **Performance Comparison**

* **Bubble Sort**:
  + Best-case: O(n) - When the array is already sorted.
  + Average-case: O(n^2)
  + Worst-case: O(n^2) - When the array is sorted in reverse order or ascending order.
* **Quick Sort**:
  + Best-case: O(n log n)
  + Average-case: O(n log n)
  + Worst-case: O(n^2) - When the pivot selection is poor (e.g., always picking the smallest or largest element as the pivot).

#### **Why Quick Sort is Preferred over Bubble Sort**

1. **Efficiency**: Quick Sort is generally more efficient than Bubble Sort for large datasets due to its average-case time complexity of O(n log n), compared to Bubble Sort's O(n^2).
2. **Divide-and-Conquer**: Quick Sort uses the divide-and-conquer approach, which makes it more suitable for large datasets and parallel processing.
3. **Cache Performance**: Quick Sort tends to have better cache performance due to its sequential access patterns.
4. **Flexibility**: Quick Sort is a comparison sort and can be adapted to different types of data and specific conditions, making it versatile.