**Bubble Sort**

* **Algorithm**: Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The process is repeated until the list is sorted.
* **Time Complexity**: O(n2)O(n^2)O(n2)
* **Space Complexity**: O(1)O(1)O(1)
* **Best Case**: O(n)O(n)O(n) (when the array is already sorted)
* **Worst Case**: O(n2)O(n^2)O(n2) (when the array is sorted in reverse order)
* **Characteristics**: Simple to understand and implement but inefficient on large lists.

**Insertion Sort**

* **Algorithm**: Builds the sorted array one item at a time, by repeatedly picking the next item and inserting it into its correct position.
* **Time Complexity**: O(n2)O(n^2)O(n2)
* **Space Complexity**: O(1)O(1)O(1)
* **Best Case**: O(n)O(n)O(n) (when the array is already sorted)
* **Worst Case**: O(n2)O(n^2)O(n2) (when the array is sorted in reverse order)
* **Characteristics**: Efficient for small data sets and mostly sorted arrays.

**Quick Sort**

* **Algorithm**: Selects a 'pivot' element from the array and partitions the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.
* **Time Complexity**: O(nlog⁡n)O(n \log n)O(nlogn) on average
* **Space Complexity**: O(log⁡n)O(\log n)O(logn)
* **Best Case**: O(nlog⁡n)O(n \log n)O(nlogn)
* **Worst Case**: O(n2)O(n^2)O(n2) (when the smallest or largest element is always chosen as the pivot)
* **Characteristics**: Fast and efficient for large datasets. The choice of pivot and partitioning scheme greatly affect its performance.

**Merge Sort**

* **Algorithm**: Divides the array into two halves, recursively sorts them, and then merges the sorted halves.
* **Time Complexity**: O(nlog⁡n)O(n \log n)O(nlogn)
* **Space Complexity**: O(n)O(n)O(n)
* **Best Case**: O(nlog⁡n)O(n \log n)O(nlogn)
* **Worst Case**: O(nlog⁡n)O(n \log n)O(nlogn)
* **Characteristics**: Stable and efficient for large datasets. Requires additional space proportional to the array size.

**Performance Comparison (Time Complexity)**

* **Bubble Sort**:
  + Best Case: O(n)O(n)O(n)
  + Average Case: O(n2)O(n^2)O(n2)
  + Worst Case: O(n2)O(n^2)O(n2)
* **Quick Sort**:
  + Best Case: O(nlog⁡n)O(n \log n)O(nlogn)
  + Average Case: O(nlog⁡n)O(n \log n)O(nlogn)
  + Worst Case: O(n2)O(n^2)O(n2) (though with good pivot selection strategies, this can be avoided)

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

1. **Efficiency**: Quick Sort has an average-case time complexity of O(nlog⁡n)O(n \log n)O(nlogn), making it much faster than Bubble Sort's O(n2)O(n^2)O(n2) for large datasets.
2. **Adaptability**: Quick Sort can be optimized with different pivot selection strategies (e.g., choosing a random pivot, median-of-three pivot) to avoid the worst-case scenario.
3. **Memory Usage**: Both Quick Sort and Bubble Sort have similar space complexities for in-place versions, but Quick Sort's divide-and-conquer approach allows it to be more efficient in practice.