**1. Understand Sorting Algorithms**

**Bubble Sort**:

* **Description**: Compares adjacent elements and swaps them if they are in the wrong order. This process repeats until the array is sorted.
* **Time Complexity**:
  + **Best Case**: O(n) (when the array is already sorted).
  + **Average Case**: O(n²) (for a random order).
  + **Worst Case**: O(n²) (when the array is sorted in reverse).

**Insertion Sort**:

* **Description**: Builds the final sorted array one item at a time by repeatedly taking an element from the unsorted portion and inserting it into the correct position in the sorted portion.
* **Time Complexity**:
  + **Best Case**: O(n) (when the array is already sorted).
  + **Average Case**: O(n²) (for a random order).
  + **Worst Case**: O(n²) (when the array is sorted in reverse).

**Quick Sort**:

* **Description**: Uses a divide-and-conquer approach by selecting a "pivot" element and partitioning the array into elements less than the pivot and elements greater than the pivot. It then recursively sorts the partitions.
* **Time Complexity**:
  + **Best Case**: O(n log n) (when the pivot divides the array into roughly equal parts).
  + **Average Case**: O(n log n) (with good pivot choices).
  + **Worst Case**: O(n²) (with poor pivot choices, e.g., when the smallest or largest element is chosen as the pivot).

**Merge Sort**:

* **Description**: Uses a divide-and-conquer approach by dividing the array into halves, sorting each half, and then merging the sorted halves.
* **Time Complexity**:
  + **Best Case**: O(n log n).
  + **Average Case**: O(n log n).
  + **Worst Case**: O(n log n).