**Bubble Sort:**

* **Description:** A simple sorting algorithm that 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:**
  + **Best Case:** O(n) - When the array is already sorted.
  + **Average Case:** O(n^2) - Due to the nested loop structure.
  + **Worst Case:** O(n^2) - When the array is sorted in reverse order.

**Insertion Sort:**

* **Description:** 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) - When the array is already sorted.
  + **Average Case:** O(n^2) - Due to the nested loop structure.
  + **Worst Case:** O(n^2) - When the array is sorted in reverse order.

**Quick Sort:**

* **Description:** A highly efficient sorting algorithm and is based on partitioning the array into smaller arrays. It picks an element as a pivot and partitions the given array around the picked pivot.
* **Time Complexity:**
  + **Best Case:** O(n log n) - When the pivot splits the array in balanced halves.
  + **Average Case:** O(n log n) - Generally occurs for most arrangements of data.
  + **Worst Case:** O(n^2) - When the pivot is the smallest or largest element every time.

**Merge Sort:**

* **Description:** A divide and conquer algorithm that was invented by John von Neumann. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.
* **Time Complexity:**
  + **Best Case, Average Case, and Worst Case:** O(n log n) - Due to the divide-and-conquer approach.