**Bubble Sort** :

Bubble sort is a simple sorting algorithm. Bubble sort is a sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent elements and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted. Bubble sort gets its name because smaller elements bubble toward the top of the list.

Bubble sort is also referred to as sinking sort or comparison sort. It has the worst case time complexity of O(n^2)

Image shows more cleared picture of the same:

A screenshot of a cell phone

Description automatically generated

**Quick Sort:**

Quick Sort is a sorting algorithm, which is commonly used in computer science. Quick Sort is a divide and conquer algorithm. It creates two empty arrays to hold elements less than the pivot value and elements greater than the pivot value, and then recursively sort the sub arrays. There are two basic operations in the algorithm, swapping items in place and partitioning a section of the array.

Steps:

1. Find a “pivot” item in the array. This item is the basis for comparison for a single round.
2. Start a pointer (the left pointer) at the first item in the array.
3. Start a pointer (the right pointer) at the last item in the array.
4. While the value at the left pointer in the array is less than the pivot value, move the left pointer to the right (add 1). Continue until the value at the left pointer is greater than or equal to the pivot value.
5. While the value at the right pointer in the array is greater than the pivot value, move the right pointer to the left (subtract 1). Continue until the value at the right pointer is less than or equal to the pivot value.
6. If the left pointer is less than or equal to the right pointer, then swap the values at these locations in the array.
7. Move the left pointer to the right by one and the right pointer to the left by one.
8. If the left pointer and right pointer don’t meet, go to step 1.

Overall time complexity of Quick Sort is O(nLogn). In the worst case, it makes O(n2) comparisons, though this behavior is rare.

Below image will explain more clearly:

A screenshot of a cell phone

Description automatically generated

**Selection Sort:**

We begin by considering the first element to be sorted and the rest to be unsorted. As the algorithm proceeds, the sorted portion of the list will grow and the unsorted portion will continue to shrink.

Selection Sort is about picking/selecting the smallest element from the list and placing it in the sorted portion of the list. Initially, the first element is considered the minimum and compared with other elements. During these comparisons, if a smaller element is found then that is considered the new minimum. After completion of one full round, the smallest element found is swapped with the first element. This process continues till all the elements are sorted.

Steps:

1. Consider the first element to be sorted and the rest to be unsorted
2. Assume the first element to be the smallest element.
3. Check if the first element is smaller than each of the other elements:
   1. If yes, do nothing
   2. If no, choose the other smaller element as minimum and repeat step 3
4. After completion of one iteration through the list, swap the smallest element with the first element of the list.
5. Now consider the second element in the list to be the smallest and so on till all the elements in the list are covered.

A close up of text on a white background

Description automatically generated

Selection Sort works best with a small number of elements. The worst case runtime complexity of Insertion Sort is o(n^2).

**Merge Sort:**

The idea is to split the unsorted list into smaller groups until there is only one element in a group. Then, group two elements in the sorted order and gradually build the size of the group. Every time the merging happens, the elements in the groups must be compared one by one and combined into a single list in the sorted order. This process continues till all the elements are merged and sorted. Note that when the regrouping happens the sorted order must **always**be maintained.

Steps:

1. Split the unsorted list into groups recursively until there is one element per group
2. Compare each of the elements and then group them
3. Repeat step 2 until the whole list is merged and sorted in the process

The worst case runtime complexity of Merge Sort is*o(nlog(n))*

**A close up of a map

Description automatically generated**