Definition:
A sorting algorithm is **adaptive** if it runs faster when the data is already partially or fully sorted.

Definition:
A sorting algorithm is **stable** if elements with equal value are in the same relative order before and after the sort.

Stable sorts are useful because you may need to re-sort by separate criteria. For instance, if you begin with a list of students sorted alphabetically, then you sort this list by year using a stable sort, then you end up with a list of all 1st years (still sorted alphabetically), then all 2nd years (still sorted alphabetically), etc. You can use stable sorts to preserve prior ordering.

Definition:
A sorting algorithm is **in-place** if you do not need additional an data structure to perform the sort.

23.1 Sort Category #1: Iterative Sorts

With an iterative sort, you can only fix the location of one item at a time. These are typically non-recursive.

23.1.1 Bubble Sort

Iteration 1
0 2 5 4 1 3
0 2 5 4 1 3
0 2 5 4 1 3
0 2 4 5 1 3
0 2 4 1 5 3
0 2 4 1 3 5

After one iteration of bubble sort, the largest item is guaranteed to be at the end of the list, where it belongs. So, on the second iteration, you do not have to check the final index.

Iteration 2
0 2 4 1 3 5
0 2 4 1 3 5
0 2 4 1 3 5
0 2 1 4 3 5
0 2 1 3 4 5

After two iterations, both 4 and 5 are guaranteed to be in their correct spaces. Notice how the second iteration only took four comparisons, while the first iteration took five.

Iteration 3
0 2 1 3 4 5
0 2 1 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5

As a human, we can see that the data has been fully sorted. However, it is not immediately clear how your *algorithm* would "know" this. Ideally, we do not want to waste time sorting items that have already been sorted. One way to optimize this is to check if at least one swap has occurred during that iteration. The more-advanced way to do this is to check the location of the last swap– after which you can conclude all items are sorted.

23.2 Sort Category #2: Divide and Conquer sorts

With a divide-and-conquer sort, you typically break the problem into smaller problems, solve each sub-problem, then combine them back together into a solution for the original problem. These are typically recursive solutions.