**Bubble Sort [Best: O(n), Worst:O(N^2)]**

Starting on the left, compare adjacent items and keep “bubbling” the larger one to the right (it’s in its final place). Bubble sort the remaining N -1 items.

* Though “simple” I found bubble sort nontrivial. In general, sorts where you iterate backwards (decreasing some index) were counter-intuitive for me. With bubble-sort, either you bubble items “forward” (left-to-right) and move the endpoint backwards (decreasing), or bubble items “backward” (right-to-left) and increase the left endpoint. Either way, some index is decreasing.
* You also need to keep track of the next-to-last endpoint, so you don’t swap with a non-existent item.

**Selection Sort [Best/Worst: O(N^2)]**

Scan all items and find the smallest. Swap it into position as the first item. Repeat the selection sort on the remaining N-1 items.

* I found this the most intuitive and easiest to implement — you always iterate forward (i from 0 to N-1), and swap with the smallest element (always i).

**Insertion Sort [Best: O(N), Worst:O(N^2)]**

Start with a sorted list of 1 element on the left, and N-1 unsorted items on the right. Take the first unsorted item (element #2) and insert it into the sorted list, moving elements as necessary. We now have a sorted list of size 2, and N -2 unsorted elements. Repeat for all elements.

* Like bubble sort, I found this counter-intuitive because you step “backwards”
* This is a little like bubble sort for moving items, except when you encounter an item smaller than you, you stop. If the data is reverse-sorted, each item must travel to the head of the list, and this becomes bubble-sort.
* There are various ways to move the item leftwards — you can do a swap on each iteration, or copy each item over its neighbour

MERGE SORT

Although [heapsort](https://en.wikipedia.org/wiki/Heapsort) has the same time bounds as merge sort, it requires only Θ(1) auxiliary space instead of merge sort's Θ(*n*). On typical modern architectures, efficient [quicksort](https://en.wikipedia.org/wiki/Quicksort) implementations generally outperform mergesort for sorting RAM-based arrays.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] On the other hand, merge sort is a stable sort and is more efficient at handling slow-to-access sequential media. Merge sort is often the best choice for sorting a [linked list](https://en.wikipedia.org/wiki/Linked_list): in this situation it is relatively easy to implement a merge sort in such a way that it requires only Θ(1) extra space, and the slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

|  |  |
| --- | --- |
| [**Worst-case performance**](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) | O(*n* log *n*) |
| [**Best-case performance**](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) | O(*n* log *n*) typical, |