Basic Sorts

Insertion Sort

> **Theorem 0.1**
>
> At iteration j, everything before index j is sorted. "Insert" the item at index j into the sorted array from 0 to j - 1 by swapping leftwards.

For insertion sort, we basically create a sub array starting from the front of the main array that is always sorted. Then for each element, we place it into its correct location in the sorted subarray.

> **Note 0.1**
>
> For **stability**, do not swap items with the same value!
>
> Example:
>
> (2, 4, 6, 8, 6)
>
> When at the second 6, we will swap it iwth 8, instead of 6 in order to main stability.
>
> Another note: In insertion sort, before the last iteration, it is possible that none of the items are in their final position.

> **Theorem 0.2**
>
> **Time Complexity**
>
> **Best**: Already sorted: O(n)
> - This is because we still need to look at every data item to make sure everything is in order.
>
> **Worst**: Reverse sorted order: O(n^2)
> - Each element needs to be swapped all the way to the front (aka moving it the maximum distance).

Selection Sort

> **Theorem 0.3**
>
> For each index in the array, find the largest/smallest item in the unsorted part of the array, and then place it into that position repeatedly until sorted.