Sorting: Comparisons

	Run-time	**Stable?**	**In-Place?**
Insertion Sort	Best Case: $\mathcal{O}(n)$ Worst Case: $\mathcal{O}(n^2)$ Average Case: $\mathcal{O}(n^2)$	Stable	In-place
Selection Sort	$\mathcal{O}(n^2)$	Not Stable	In-place
Heap Sort	$\mathcal{O}(n \log n)$	Not Stable	In-place
Merge Sort	$\mathcal{O}(n \log n)$	Stable	Not In-place
Quick Sort ("Hoare" Partition)	Best Case: $\mathcal{O}(n \log n)$ Worst Case: $\mathcal{O}(n^2)$ Average Case: $\mathcal{O}(n \log n)$	Not Stable	In-place
Bucket Sort	$\mathcal{O}(n + k)$	Stable	Not In-place
Radix Sort	$\mathcal{O}\big(d \cdot (n + k)\big)$	Stable	Not In-place

Sorting: Summary

- Simple $\mathcal{O}(n^2)$ Sorting
  - Selection Sort, Insertion Sort, etc.
  - Good for "below a cut-off" (e.g., for small input size n) to help divide-and-conquer sorts
- $\mathcal{O}(n \log n)$ Sorting
  - Heap Sort, Not stable but in-place, not parallelizable (later)
  - Merge Sort, Stable but not in-place and works as external sort
  - Quick Sort, Not stable but in-place and $\mathcal{O}(n^2)$ in worst-case
    - often fastest, but depends on costs of comparisons/copies (Java uses this)
- $\Omega(n \log n)$ lower-bound for comparison sorting
- Non-Comparison Sorting
  - Bucket Sort for small number of `k-v`
  - Radix Sort for digits
- Best way to sort? lol depends