Sorting Algorithms (in Plain-ish English)

Bubble Sort

- To sort a list of *n* elements:
  - Work through the list from left to right, swapping elements *i* and *i* + 1 if element *i* is larger than element *i* + 1.
  - Repeat until the list is sorted (it will take at most *n* passes through the list to achieve this).
- Each pass “bubbles up” the largest element to the end of the list.

Selection Sort

- To sort a list of *n* elements:
  - **Select** the minimum element from the entire list. Swap it with the element at index 0.
  - **Select** the minimum element from the list, excluding index 0. Swap it with the element at index 1.
  - **Select** the minimum element from the list, excluding indices 0 and 1. Swap it with the element at index 2.
  - Repeat for the entire list.

Insertion Sort

- To sort a list of *n* elements:
  - Assume the element at index 0 is already sorted.
  - **Insert** the element at index 1 in its correct place, relative to the element at index 0.
  - **Insert** the element at index 2 in its correct place, relative to the elements at indices 0 and 1.
  - **Insert** the element at index 3 in its correct place, relative to the elements at indices 0, 1, and 2.
  - Repeat for the entire list.

Shell Sort

- "Divide and conquer" variant of insertion sort
- Basic idea is to divide the array into smaller parts and run insertion sort on each part
  - Takes advantage of the fact that insertion sort does less work if the array is already almost sorted
- Named after its creator Donald Shell, who came up with the first version of the algorithm in 1959

Shell Sort

- To sort a list of *n* elements:
  - Determine a **gap size**
    - We will use half the length of the list
  - Insertion sort the elements at indices {0, *gapSize*, 2**gapSize*, …}, {1, 1 + *gapSize*, 1 + 2**gapSize*, … }, {2, 2 + *gapSize*, 2 + 2**gapSize*, … }, and so on
  - Reduce the gap size and repeat the insertion sort step until the gap size becomes 0
    - We will use the following algorithm: If the gap size is 2, make the gap size 1. Otherwise divide the gap size by 2.2.

Note that a gap size of 1 corresponds to a "regular" insertion sort

Merge Sort

- To sort a list of *n* elements:
  - Divide the list into left and right halves
  - Merge sort the left half
  - Merge sort the right half
  - **Merge** the two sorted halves back together

Heapsort

- To sort a list of *n* elements:
  - First convert the list into a max-heap. We can do this as follows:
    - Make element 0 the root of the heap.
    - Add elements 1, 2, 3, … , *n* to the heap, using the add algorithm for heaps that we discussed earlier.
  - Then repeatedly remove the top (maximum) element from the heap, placing it at the end of the list.

Quicksort

- To sort a list of *n* elements:
  - **Partition** the list around a **pivot** element. After the partition, the elements less than or equal to the pivot should be on the left of the pivot, and the elements greater than the pivot should be on the right. The pivot will be in its final sorted position.
  - Quicksort the part of the list left of the pivot.
  - Quicksort the part of the list right of the pivot.