Sorting Algorithms you should be familiar with

- Insertion Sort
- Selection Sort

- Quick Sort
- Merge Sort

- Radix Sort

- Heap Sort

Sorting Algorithms you should be familiar with

- Insertion Sort
- Selection Sort

} Comparative sorts, one element at a time

- Quick Sort
- Merge Sort

} Comparative sorts, Divide-and-conquer

- Radix Sort — Non-comparative sort

- Heap Sort — Data-structure driven

Insertion Sort vs. Selection Sort

- Both algorithms build up sorted array/list from unsorted array/list one element at a time
  - Can both be implemented in-place
- Both have different implementations for arrays vs. linked lists (swapping vs. remove/insert)
- Differences:
  - How to choose next element from unsorted list?
  - How to add element to sorted list?

Insertion Sort vs. Selection Sort

First difference: How to choose next element from unsorted list?

Insertion Sort:

Always takes next adjacent element

[5, 10, -2, 0, 1]

[5, 10, -2, 0, 1]

O(1) at each step

Selection Sort:

Scans entire array to select next element

[5, 10, -2, 0, 1]

[-2, 5, 10, 0, 1]

O(N) at each step

Insertion Sort vs. Selection Sort

Second difference: How to add element to sorted sequence?

	Array	Linked List
Insertion Sort	Begin at end of sorted sequence, shift elements to right until we find the place for our new element. O(N) time O(1) memory	Remove next element, scan through sorted sequence to find place to insert. O(N) time O(1) memory
Selection Sort	Swap min unsorted element with max sorted element. O(1) time O(1) memory	Remove min unsorted element, insert at end of sorted sequence. O(1) time O(1) memory

Insertion Sort vs. Selection Sort

- Both are $O(n^2)$ in worst case: compare every element to every other element
- Best Case:
  - Insertion: $O(n)$ comparisons and no swaps
  - Selection: $O(n^2)$ comparisons, no swaps
- Average is same as worst case for both (no randomization)

Quick Sort vs. Merge Sort

- Both algorithms divide and conquer
  - Can be done in-place but more common to use extra memory
- Both use merge operation to reconstruct sorted list
- Difference:
  - Choosing "pivot" or location of where to "divide"

Quick Sort vs. Merge Sort

Difference: How to choose where to divide?

Quick Sort:

Always random; allocate elements

[5, 10, -2, 0, 1]

/ \

[-2, 0] [5, 10, 1]

O(N) at each step

O(lg N splits)

Merge Sort:

Always length / 2

[5, 10, -2, 0, 1]

/ \

[5, 10] [-2, 0, 1]

O(1) in-place or O(N) to copy

O(lg N) splits

Quick Sort vs. Merge Sort

Merge Operation

- Always O(lg N) steps
- O(N) at each step (for all sub-arrays together)

Quick Sort vs. Merge Sort

- Merge is O(N lg N) in worst case
- Quick sort is $O(N^2)$ in worst case: inefficient splits
- Best Case? O(N lg N) unless we check for sorted-ness first
- Average is same as best case for both

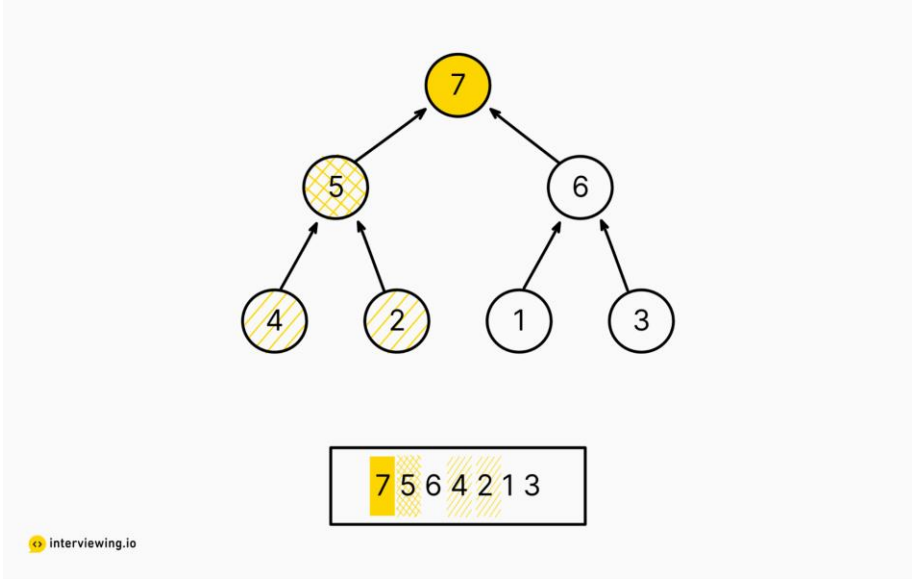
Radix Sort

- NON-comparative sort
- Sort into buckets by digit, starting with least significant figure
- Can pre-allocate space with counting sort
- O(k * N) performance in all cases, where N is number of elements, and k is the length of the elements
- O(k + N) space complexity

https://www.hackerearth.com/practice/algorithms/sorting/radix-sort/visualize/

Heap Sort

- Convert data into heap: can be done in-place in array
- Repeatedly remove largest / smallest element from heap and add to end of array
- O(N lg N) performance in all cases
  - N elements added to heap with lg(N) bubble operations; Nlg(N) ops to remove
  - Except one case: any guesses?



Sorting Algorithm Properties

- Stability
- In-place (vs. auxiliary memory)
- Adaptive
- Online