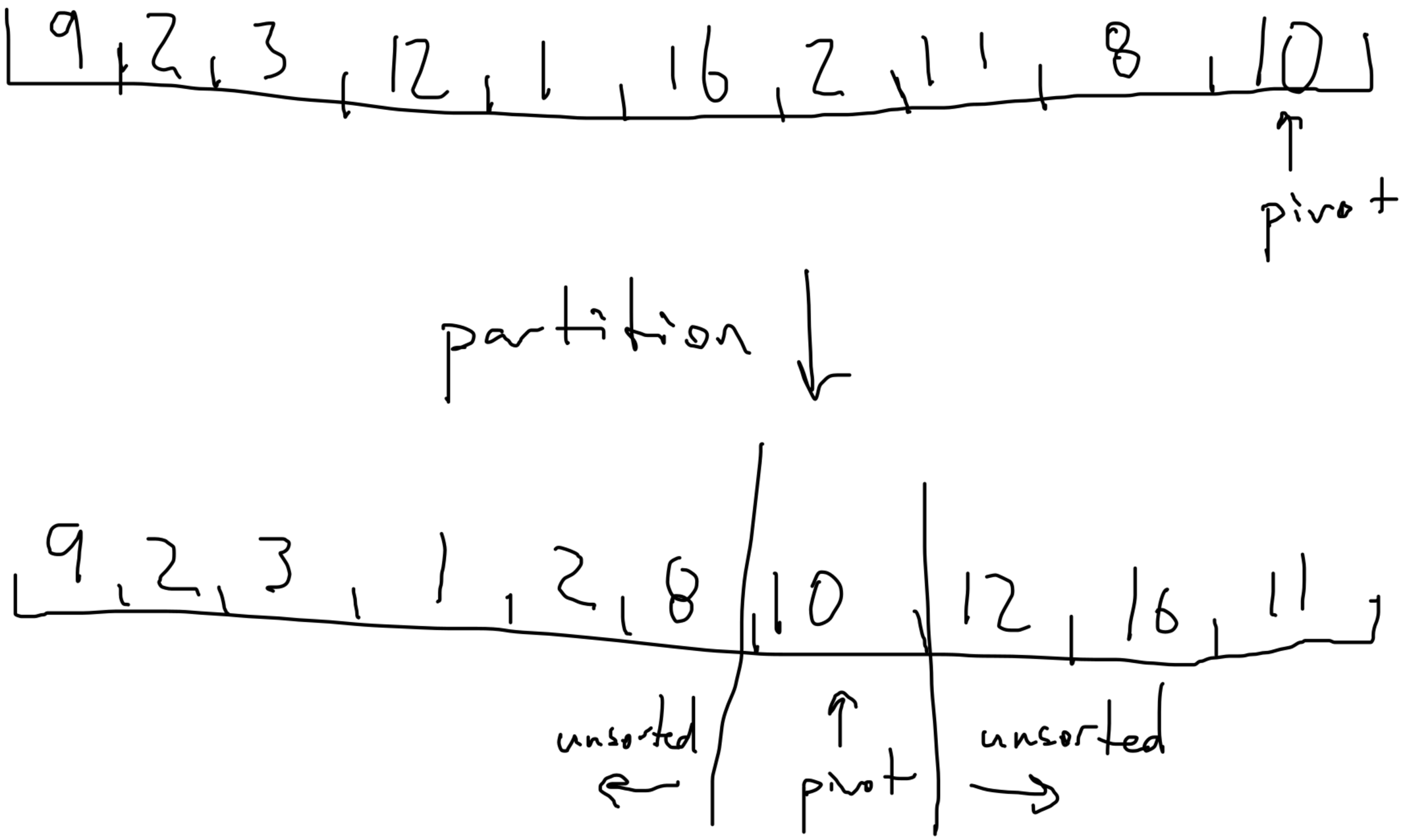
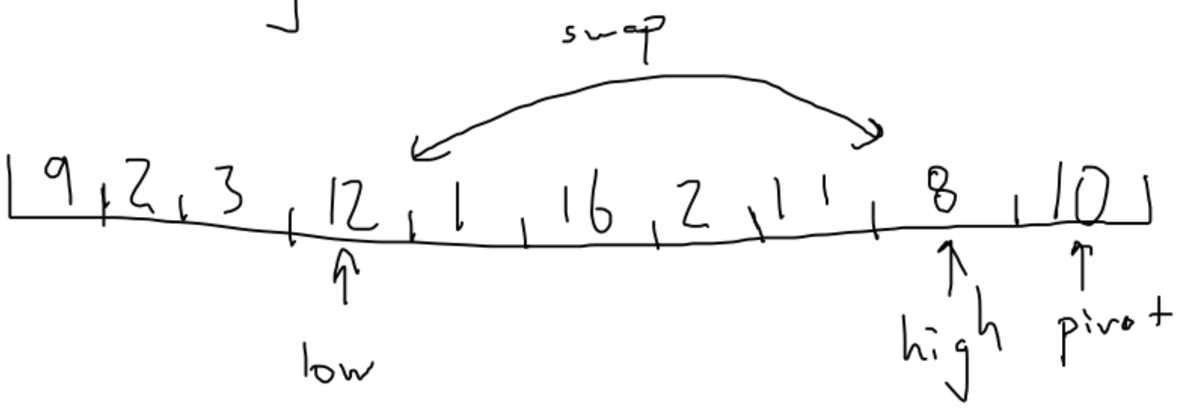
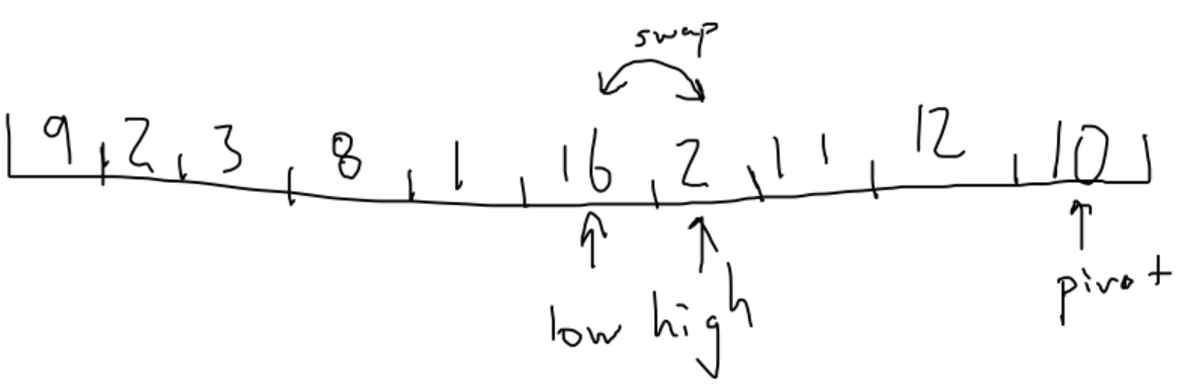
Quicksort

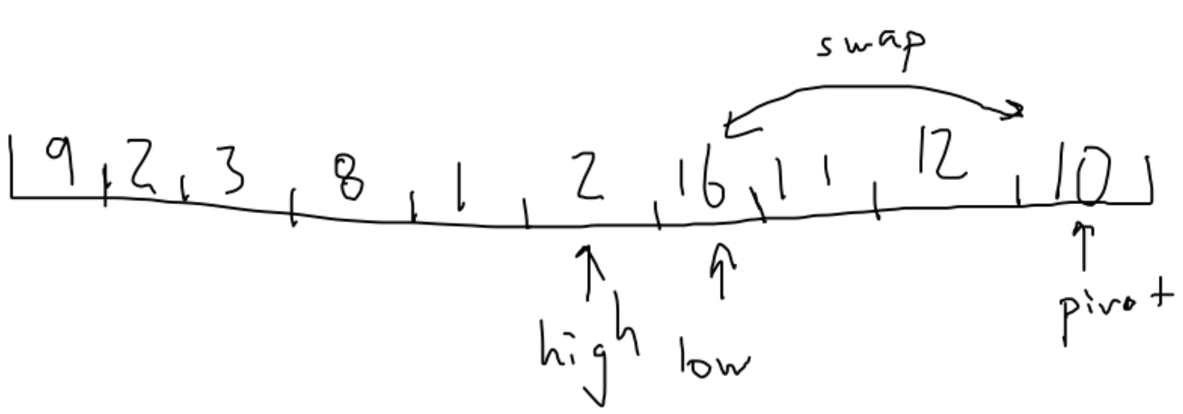

9, 2, 3, 12, 1, 16, 2, 11, 8, 10
pivot
partition
9, 2, 3, 1, 2, 8, 10, 12, 16, 11
unsorted
pivot
unsorted

Partitioning

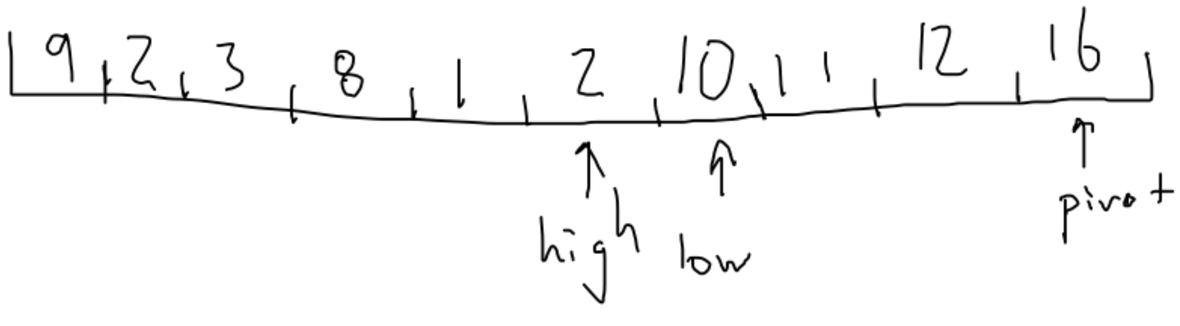


- repeat until low > high
  - find a low > pivot
  - find a high < pivot
  - if low < high, swap items at low & high

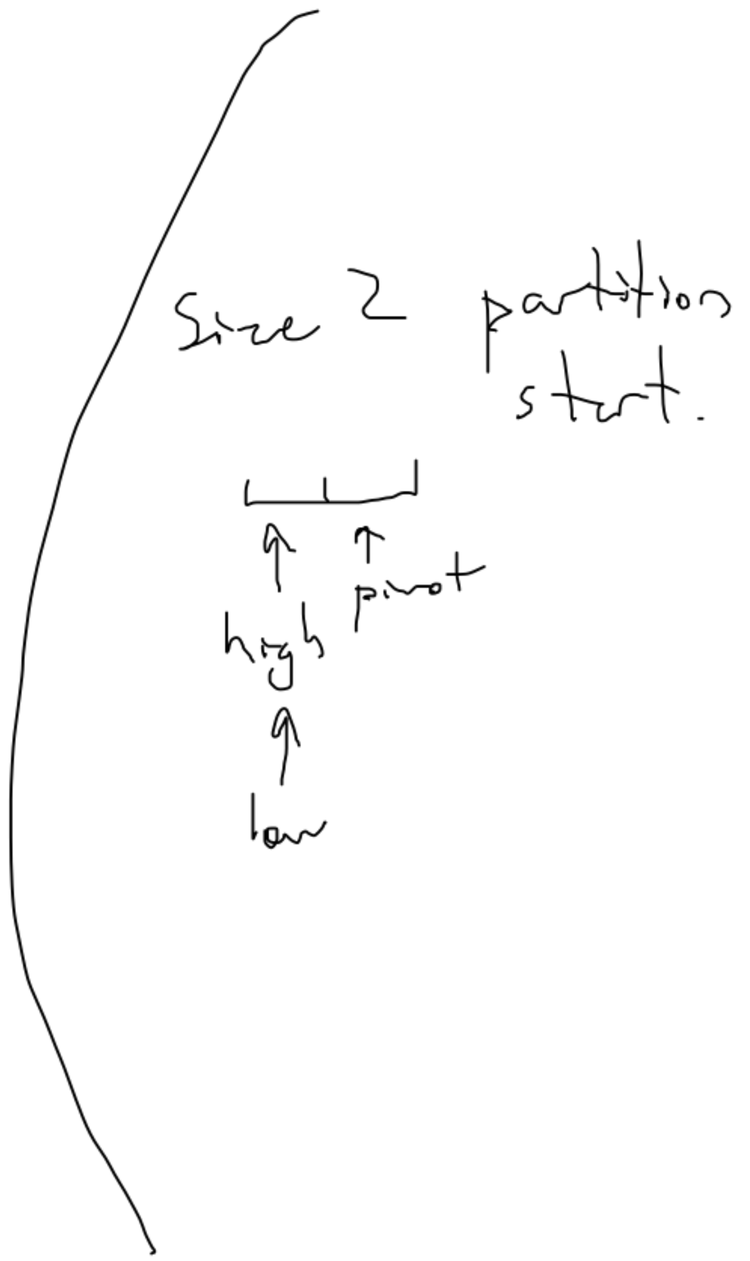




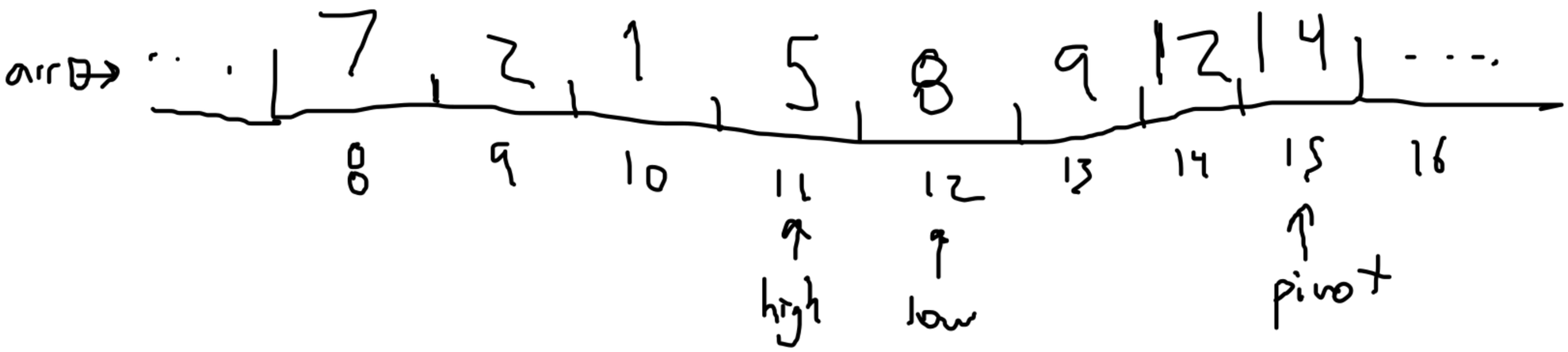
- once low & high cross, swap pivot & low



- recursively sort < low & > low



Partition Tracing



partition (arr, 8, 16)

pivot | 15

low | ~~8 9 10 11~~ 12

high | ~~14 13 12~~ 11

mid becomes 12

Partition Analysis

- let n be the size of array
- moves of low & high total ~n — O(n)
- up to $\frac{n}{2}$ swaps — O(n)
- final pivot swap — O(1)

O(n)

Quicksort Analysis

- with the best case, each partition splits 50/50

$$T(n) = O(n) + 2T(n/2)$$

$$\rightarrow O(n \lg n)$$

- in the worst case, pivot is largest or smallest

$$T(n) = O(n) + T(n-1)$$

$$\rightarrow O(n^2)$$

- on average with random data:

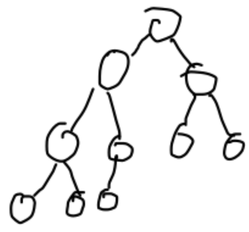
$$O(n \lg n)$$

- due to the big swaps, it is unstable

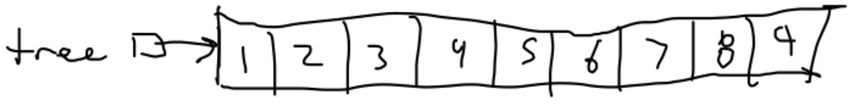
- unlike mergesort, doesn't need second array $\rightarrow$ in place sort

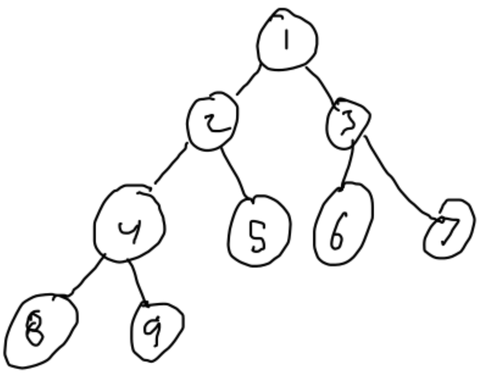
Trees in Arrays

- if you have a left-complete tree, you can store it in an array without wasting space



- the root is index 0
- root's left child is index 1, right child is index 2,
- each level is added left to right





- if a node is at index i, what is the index of its parent and what are the indices of its children