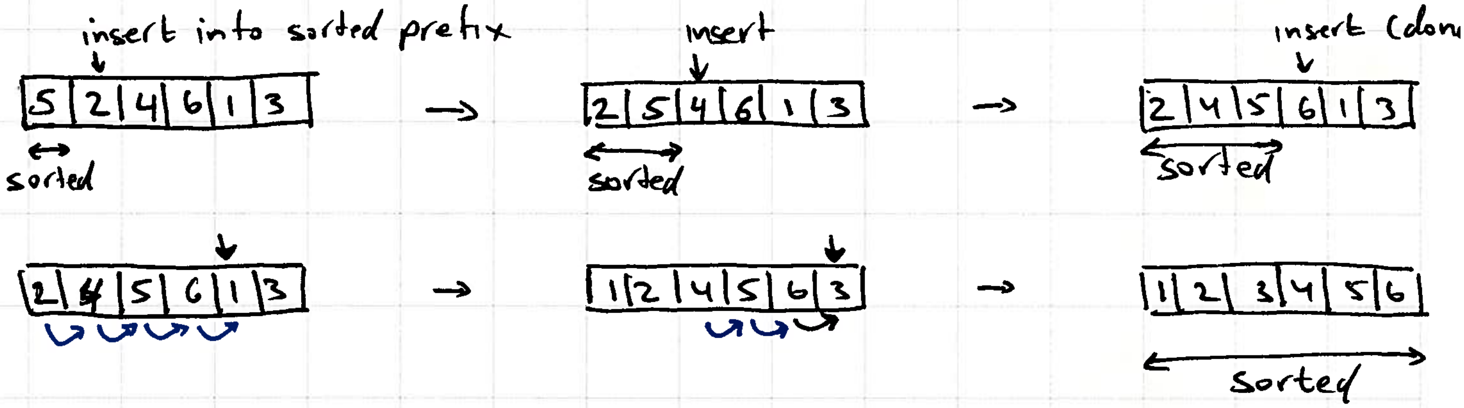
6.006 Lecture 8: Sorting I

- Review: Insertion sort
  Merge Sort
- Sorting in place & Selection sorting
- Heaps
- CLRS 2.1, 2.2, 2.3 Ch. 6 (some on Thu)

Insertion Sort



Array is sorted in place: only extra memory is
2 indices and 1 element; constant extra storage
that does not depend on n (problem size).

Running time is $\Theta(n^2)$

worst case is | n | $n-1$ | $n-2$ | $\cdots$ | 3 | 2 | 1 |

can we do ~~better~~ better in place?

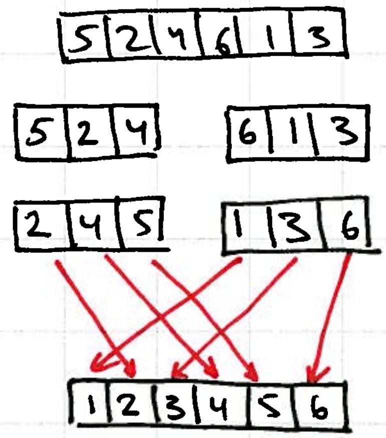
Merge Sort

input

divide

sort recursively

merge



← cannot do this step in place

$\Theta(n)$ work for merging

$\Theta(n \lg n)$ work overall

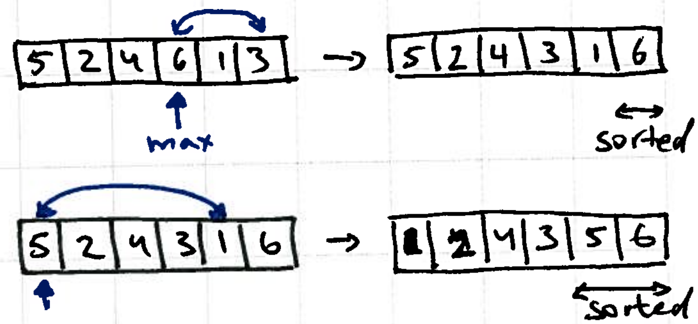
But requires $\Theta(n)$ extra storage!

Selection Sort

Find maximum element, swap with position n-1 (last)

repeat on A[0:n-1]

repeat ...



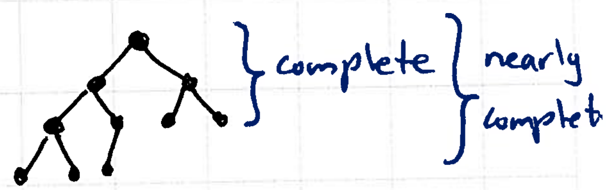
In place, but finding maximum at every step costs $\Theta(n)$, so total is $\Theta(n^2)$ work.

Can we find the max in $\Theta(\lg n)$ time?

Heaps

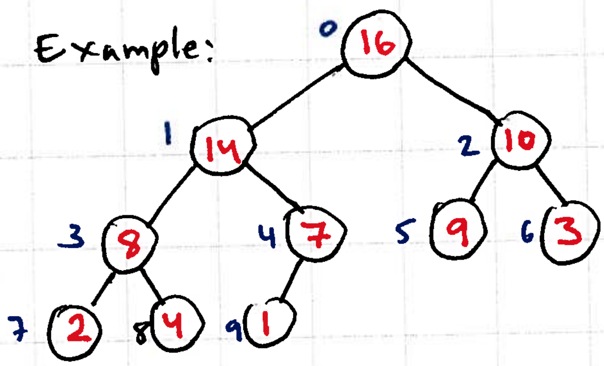
(term is also used for an area used for dynamic memory allocation; not related to this lecture)

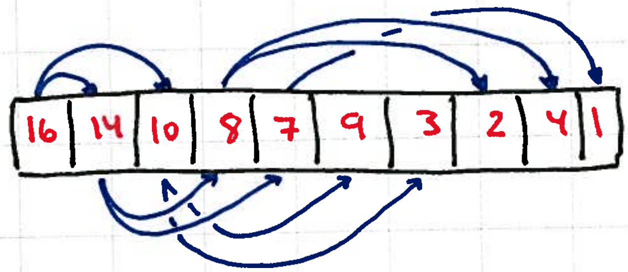
- Abstractly: a binary tree, values at nodes
- node value larger than values at children (compare to a different invariant in BSTs)
- Tree is nearly complete



- Because it is nearly complete, we can store it in an array (Python list) with no explicit pointers.

Example:





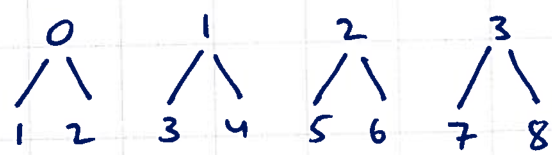
Not sorted (up or down)!

Root at A[0]

$\text{left}(i) = 2i+1$

$\text{right}(i) = 2i+2$

$\text{parent}(j) = \left\lfloor \frac{j-1}{2} \right\rfloor$

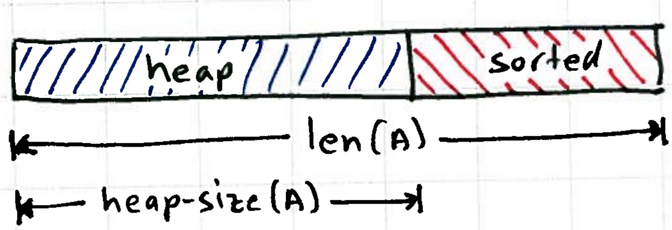


$i = \left\lfloor \frac{j-1}{2} \right\rfloor$

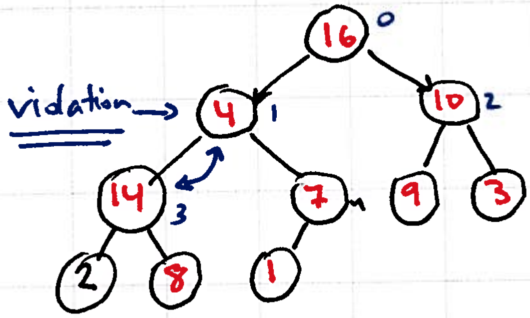
$j = \begin{matrix} 2i+1 \\ 2i+2 \end{matrix}$

The array can be larger than the heap

In heapsort, the end of the array contains sorted elements (largest), ~~beggin~~ beginning contains a heap.

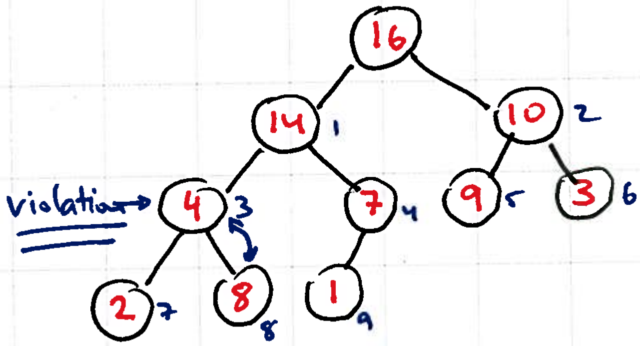


~~Heap~~ Heapify: a procedure for fixing a heap with one invariant violation



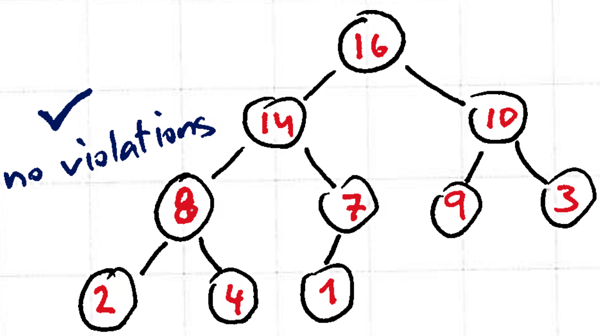
heapify (A, 1)

exchange A[1] with larger of A[left(1)] ←
A[right(1)]
[or stop if A[1] larger]



heapify (A, 3)

exchange A[3] with larger of A[left(3)]
A[right(3)]
if necessary



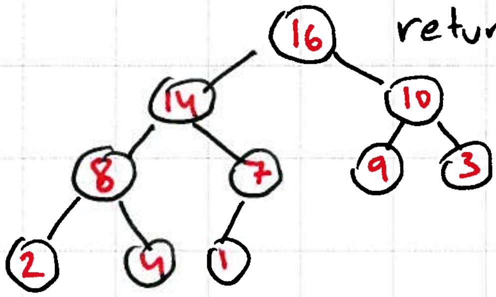
heapify (A, 8)

no children → nothing to do.

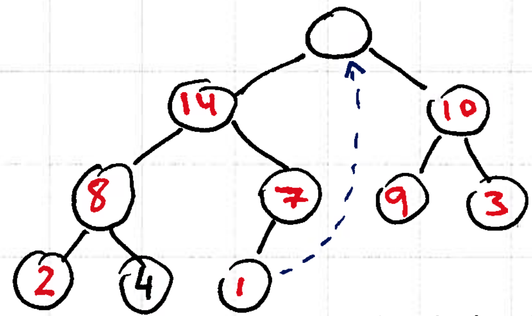
constant # operation at each level of the tree (possibly not every level; we may stop early)

$\Rightarrow \Theta(\lg n)$ work

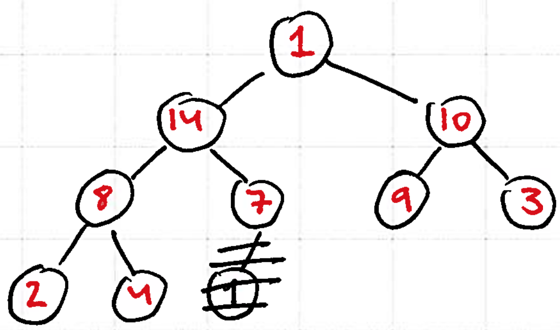
Extract-Max: return max element & delete it from the heap



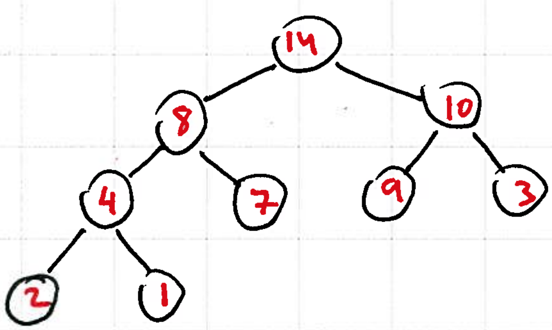
return 16



put last leaf at A[0],
heap-size = heap-size -



heapify (A,0)



DONE!

constant # operations + one call to heapify $\Rightarrow \Theta(\lg n)$ work.