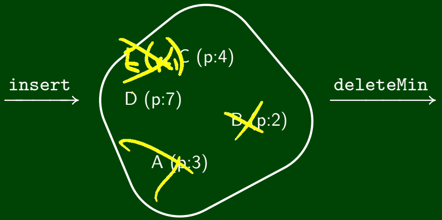
PriorityQueue ADT

`insert(`**val**`)`	Adds **val** to the queue.
`deleteMin()`	Returns the **highest priority** item not already returned by a `deleteMin`. (Errors if empty.)
`findMin()`	Returns the **highest priority** item not already returned by a `deleteMin`. (Errors if empty.)
`isEmpty()`	Returns true if all inserted elements have been returned by a `deleteMin`.

- Data in `PriorityQueues` **must be comparable** (by priority)!
- Highest Priority = Lowest Priority Value
- The ADT **does not specify how to deal with ties**!



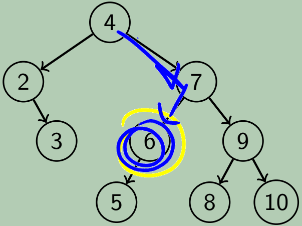
- `findMin` → B
- `deleteMin` → B
- `insert(E (p:1))` ✓
- `deleteMin` → E
- `deleteMin` → A

For each of the following potential implementations, what is the worst case runtime for `insert` and `deleteMin`? Assume all arrays do not need to resize.

	insert	delete min
Unsorted Array	$O(1)$	$O(n)$
Unsorted Linked List	$O(1)$	$O(n)$
Sorted Circular Array List	$O(\lg n)$ $O(n)$	$O(1)$
Sorted Linked List	$O(n)$	$O(1)$
Binary Search Tree	~~$O(\lg n)$ (?)~~ $O(n)$?	$O(n)$

A New Data Structure: `Heap`

Recall BSTs

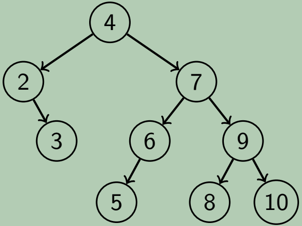


BST Property:
Left Children are smaller
Right Children are larger

For a `PriorityQueue`, how could we store the items in a tree?

A New Data Structure: `Heap`

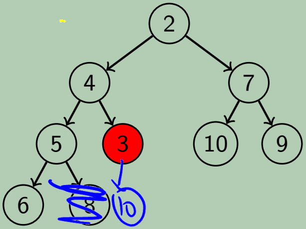
Recall BSTs



BST Property:
Left Children are smaller
Right Children are larger

For a `PriorityQueue`, how could we store the items in a tree?

And Now, `Heaps`

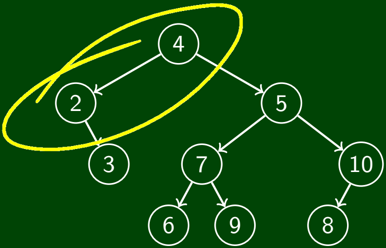


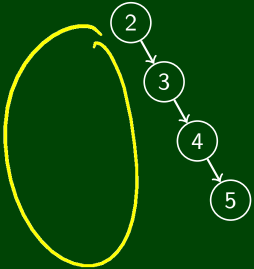
Heap Property:
All Children are larger

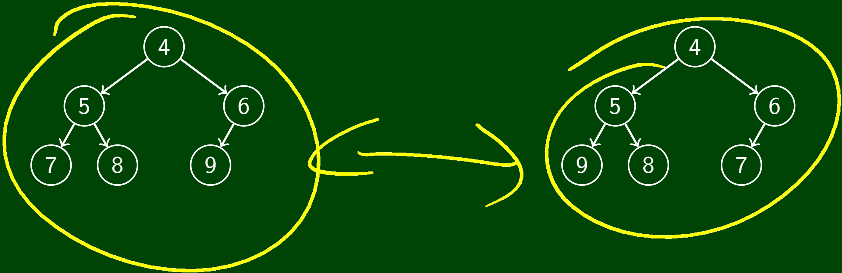
Structure Property:
Insist the tree has no “gaps”

Is It A Heap?

For each of the following, is it a heap?

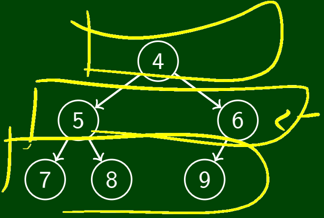






Heap Properties?

$h = k-1$

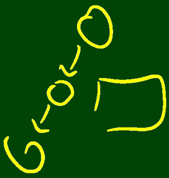


2^0

2^1

2^2

$\vdots$



- Where is the minimum item in a heap?

  It's at the top!

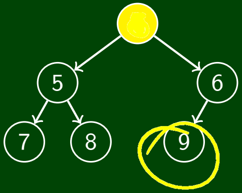
- What is the height of a heap with n items?

  k-levels

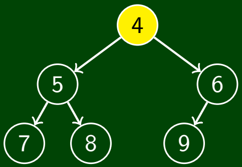
$$n = 2^0 + 2^1 + 2^2 + \cdots + 2^{k-1} = 2^k - 1$$

Implementing `deleteMin` For a `Heap`

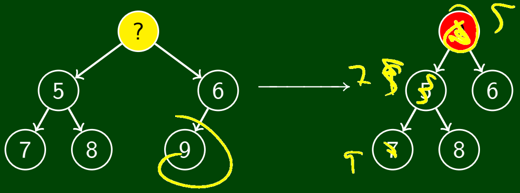
- Find the `min`:



- Find the `min`:

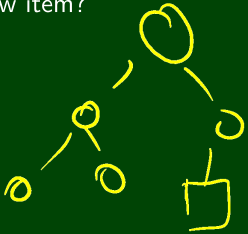


- Remove the `min` and fill the hole with the last child

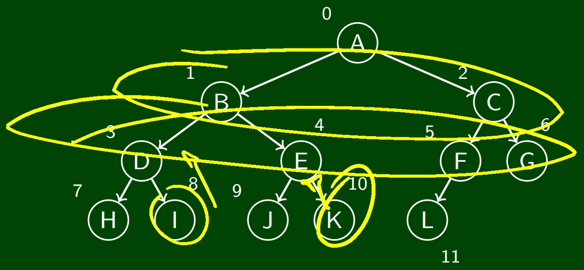


Let's try `insert(1)`:

- Where do we put a new item?



We've insisted that the tree be complete to be a valid Heap. Why?



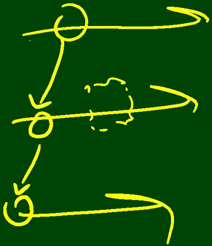
Fill in an array in **level-order** of the tree:

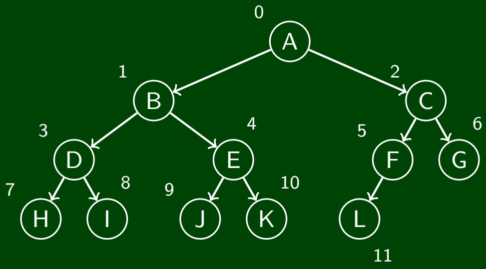
heap:	A	B	C	D	E	F	G	H	I	J	K	L	0	0	0
	h[0]	h[1]	h[2]	h[3]	h[4]	h[5]	h[6]	h[7]	h[8]	h[9]	h[10]	h[11]	h[12]	h[13]	h[14]

If I have the node at index i, how do I get its:

And. . . how do we implement Heap?

We've insisted that the tree be complete to be a valid Heap. Why?





Fill in an array in **level-order** of the tree:

heap:

A	B	C	D	E	F	G	H	I	J	K	L	0	0	0
h[0]	h[1]	h[2]	h[3]	h[4]	h[5]	h[6]	h[7]	h[8]	h[9]	h[10]	h[11]	h[12]	h[13]	h[14]

If I have the node at index i, how do I get its:

- Parent? $3 \to 1,\ 4 \to 1,\ 10 \to 4,\ 9 \to 4,\ 1 \to 0$

  This indicates that it's approximately $n/2$. In fact, it's $\dfrac{n-1}{2}$.
- Left Child? $2(n+1)-1$
- Right Child? $2(n+1)$