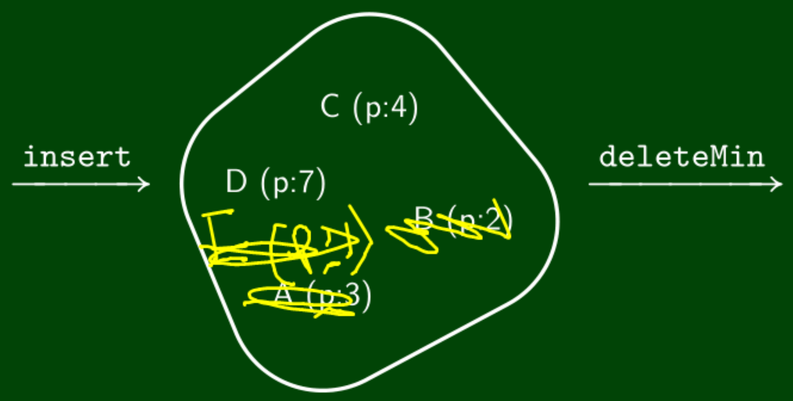
Queue (FIFOQueue) ADT

`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`
- `removeMin`
- `insert(E (p:1))`
- `removeMin`
- `removeMin`

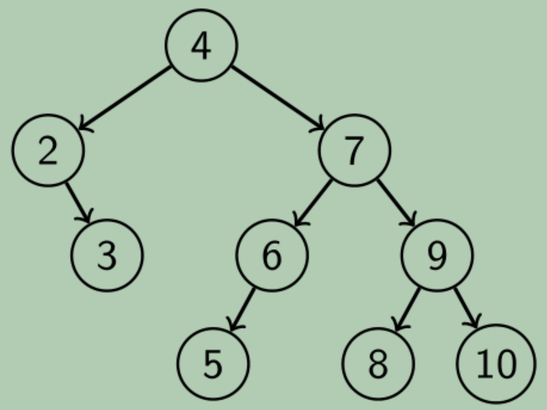
Implementing A Priority Queue

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.

- Unsorted Array
- Unsorted Linked List
- Sorted Circular Linked List
- Sorted Linked List
- Binary Search Tree

A New Data Structure: Heap

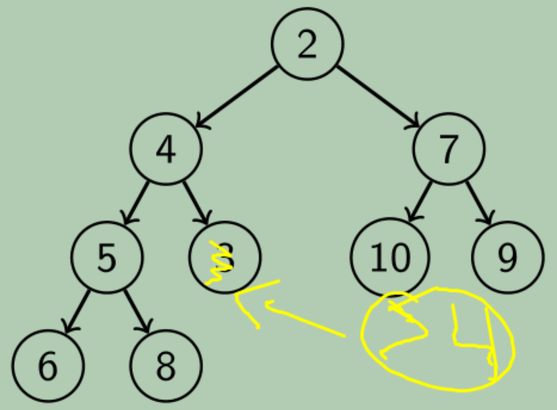
Recall BSTs



BST Property:
Left Children are smaller
Right Children are larger

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

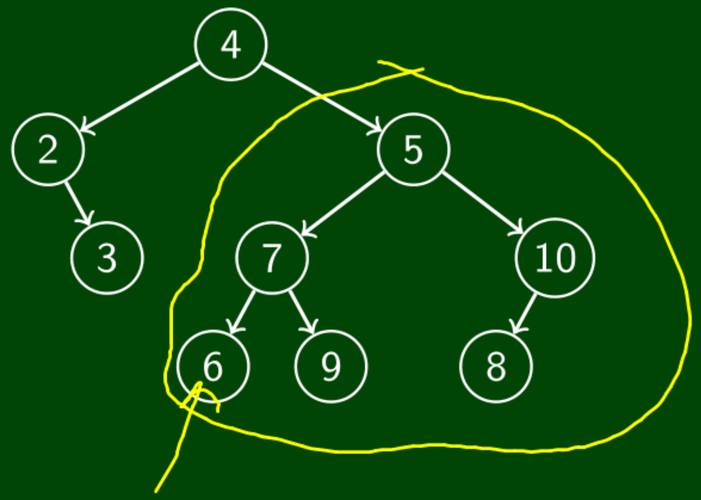
And Now, Heaps

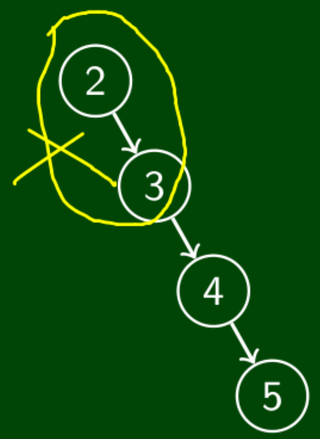


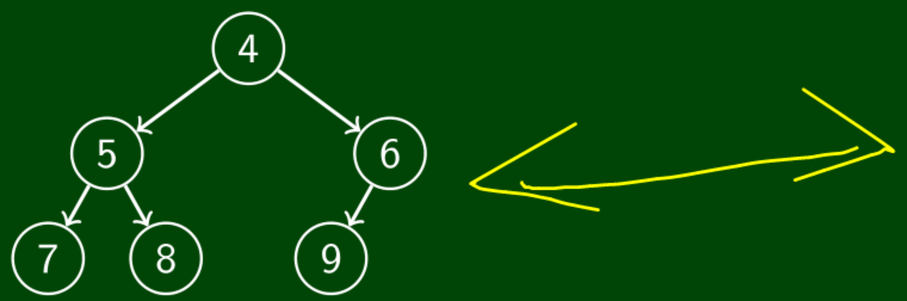
Heap Property:
All Children are larger

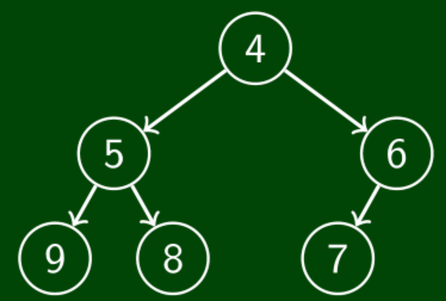
Is It A Heap?

For each of the following, is it a heap?

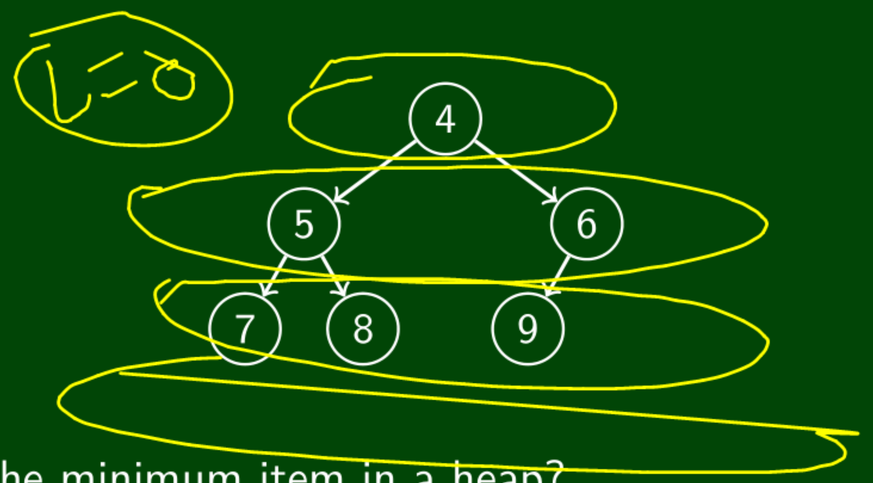








Heap Properties?



- Where is the minimum item in a heap?

  It's at the top!

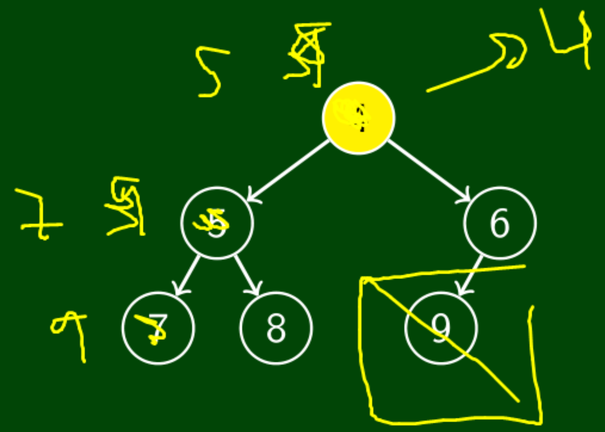
- What is the height of a heap with n items?

$$n = \sum_{i=0}^{k-1} 2^i = 2^k - 1$$

$\log(n-1) = k$ $\quad$ $\log(n+1) = k$

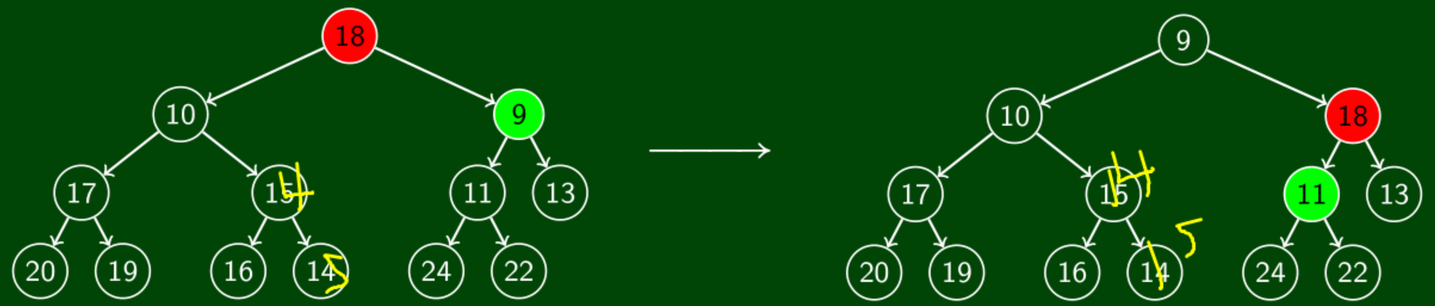
Implementing `deleteMin` For a Heap

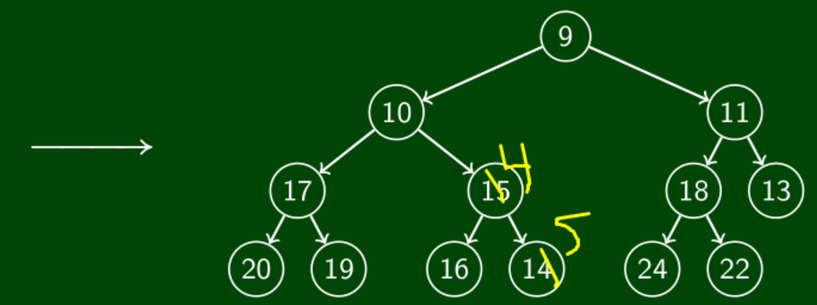
- Find the `min`:



"Percolate Down" (Another Example)


```
1  percolateDown(node) {
2     while (node.data is greater than either child) {
3        swap data with smaller child
4     }
5  }
```

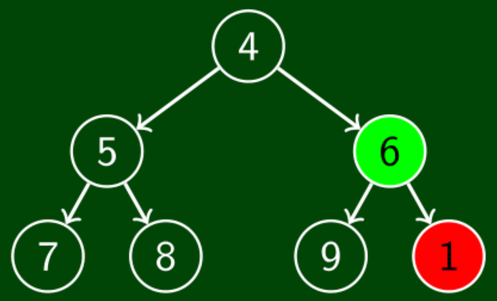




Runtime Analysis?

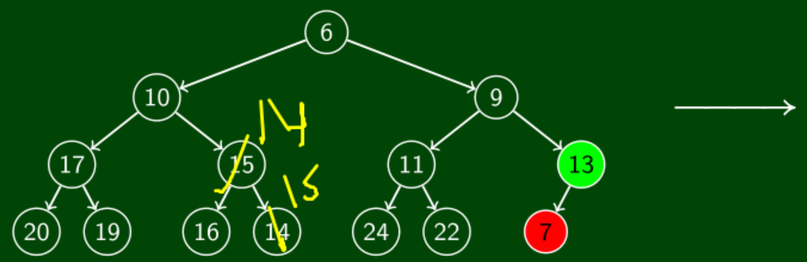
“Percolate Up”?


```
1  percolateUp(node) {
2     while (node.data is greater than parent) {
3        swap data with parent
4     }
5  }
```



“Percolate Up” (Another Example)


```
1 percolateUp(node) {
2    while (node.data is greater than parent) {
3       swap data with parent
4    }
5 }
```



“Percolate Up” (Another Example)


```
1  percolateUp(node) {
2     while (node.data is greater than parent) {
3        swap data with parent
4     }
5  }
```