Priority Queue ADT

- A Priority Queue contains (element, key) pairs, where we normally want to remove and use the element with the smallest key
- [Alternatively, we may want it organized to yield up the max key]
- we may also want to change an element's key.

Priority Queue operations:

init()

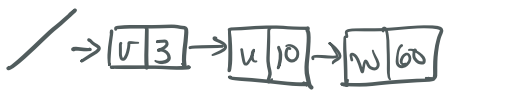
insert(e, k)

element extractMin()

decreaseKey(e, k)

unsorted list

Possible Implementations

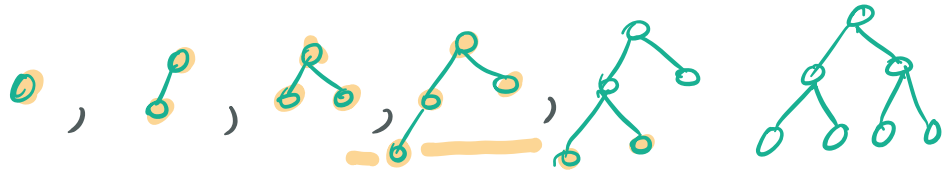


- use n as size of current

Priority Queue Implementations:

	init	insert	extractMin	decreaseKey
Unsorted List	O(1)	O(1)	O(n)	O(1) if given pointer to element
Sorted List	O(1)	O(n)	O(1)	O(n)
"Heap"				

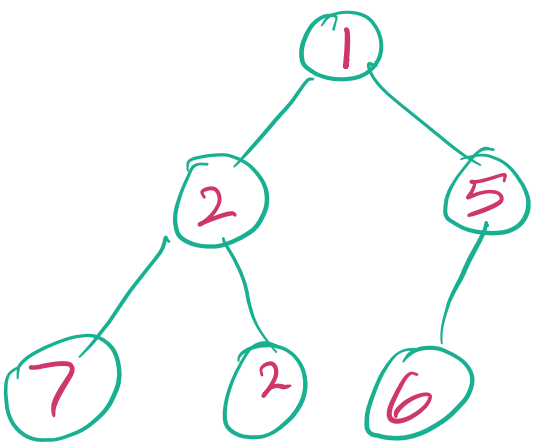
Heap = complete binary tree in heap order (by key value)

"Complete" = 

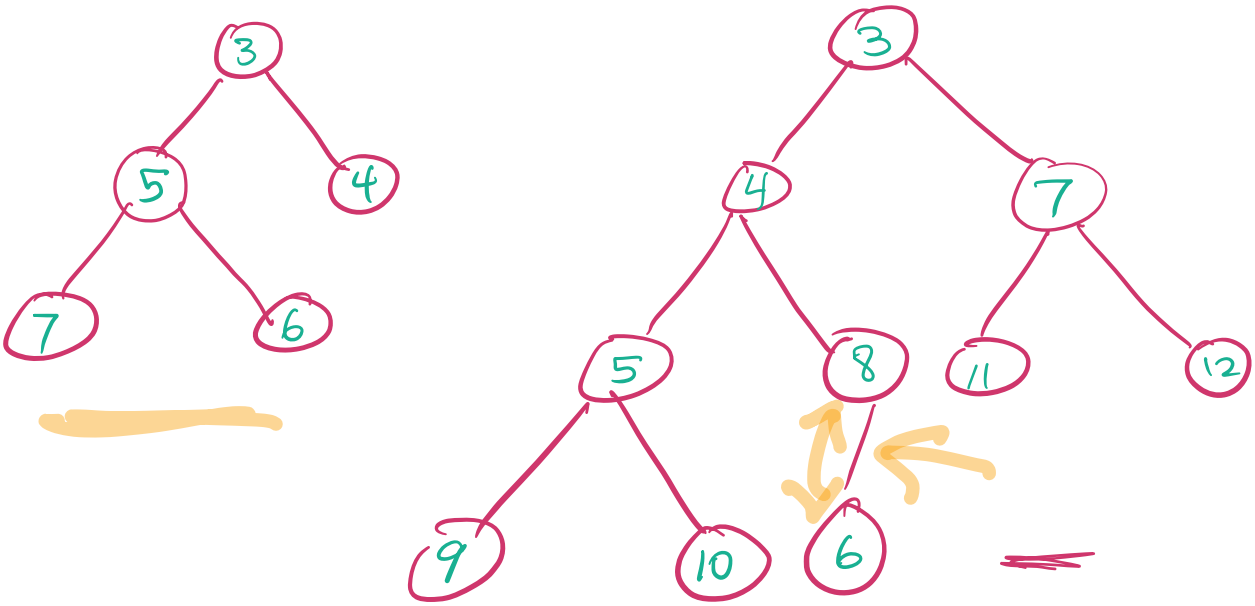
ie each layer is complete except, perhaps, lowest layer, and that one is right-filled.

"heap-order": each node has key value no greater than that of its children.

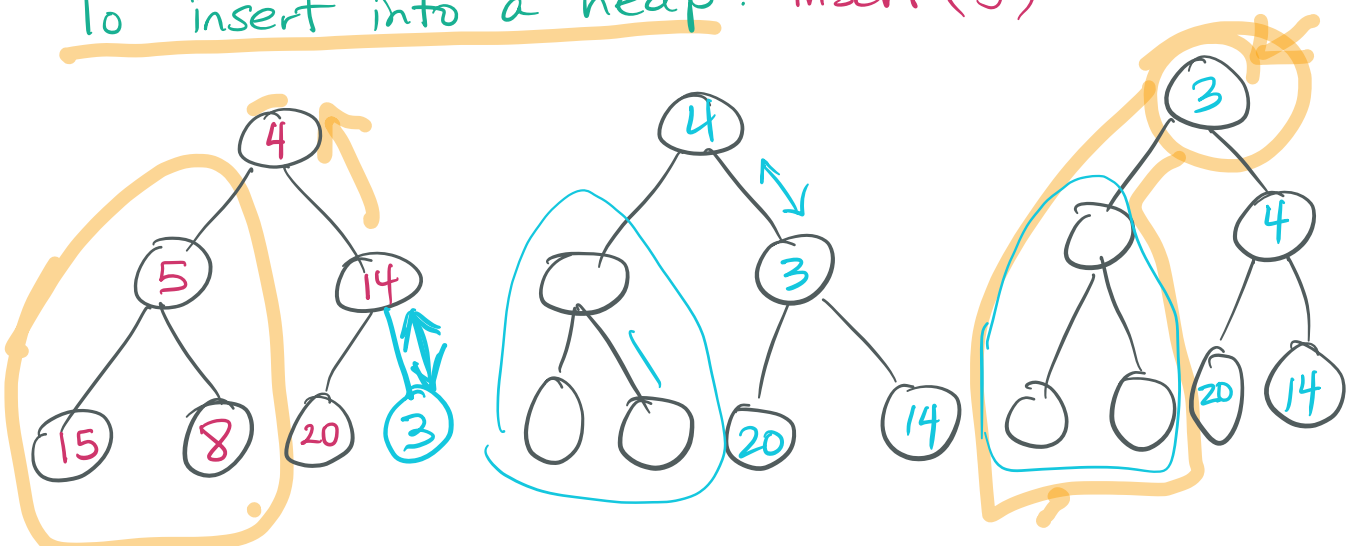
Eg: Keys are $\{1, 2, 2, 5, 6, 7\}$



Is this a heap?



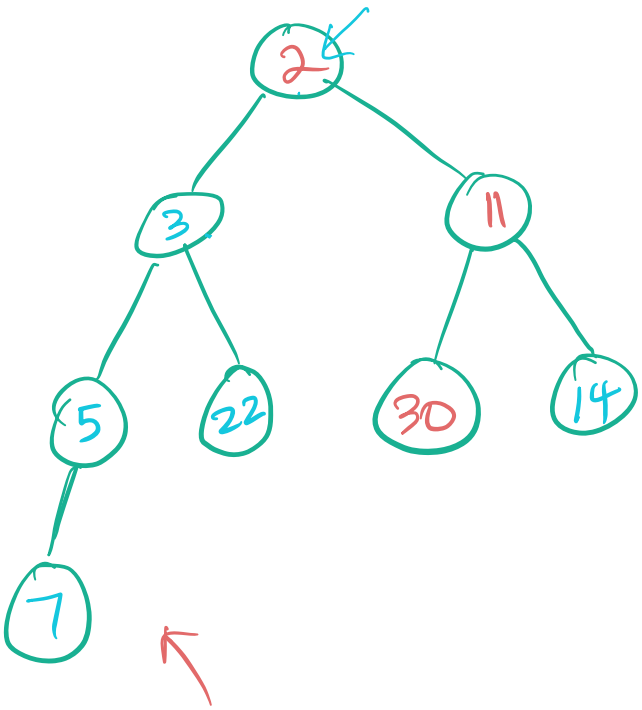
To insert into a heap: insert(3)



Is each of these swaps "safe"...

after a swap of the "new" value into node v, is v's subtree in heap order?

To remove a min (from a MinHeap)



Element Keys are

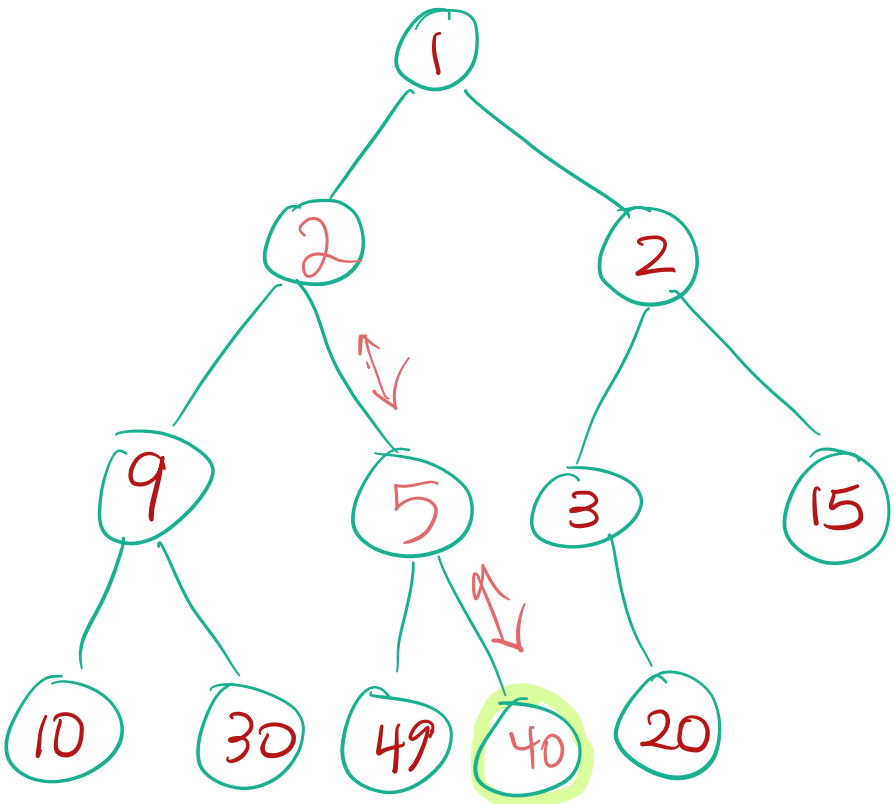
1 2 3 5 7 11 14 22 30

r = 1

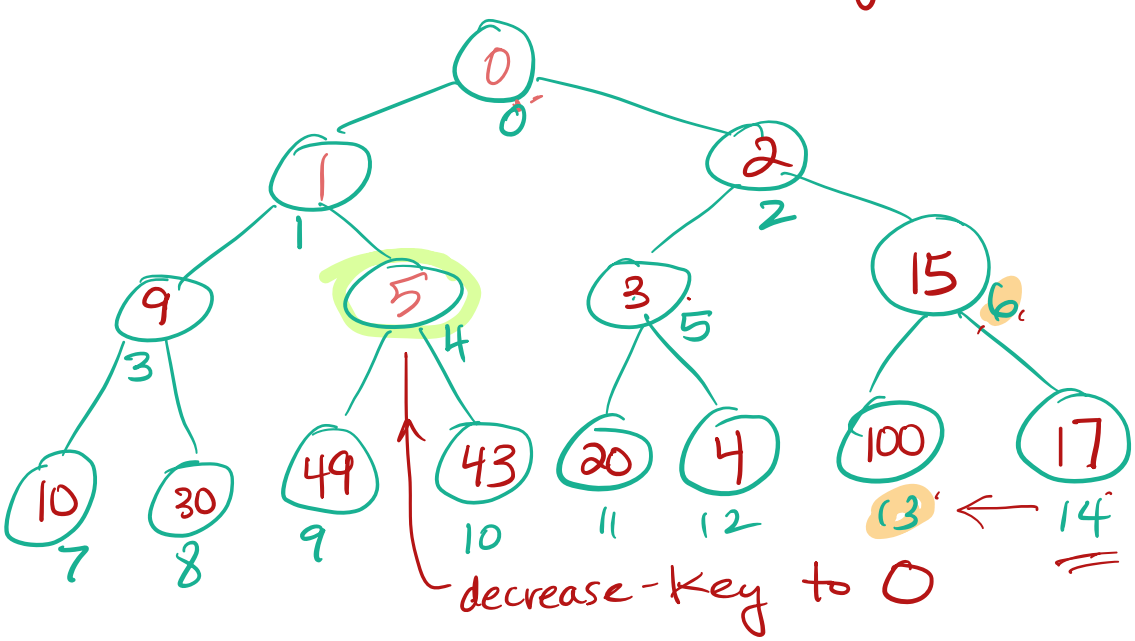
return element that we saved (old root)

- remove root element r, ready to return it
- remove X = Rightmost element of lowest layer
  - place X in root
- percolate X to it proper position
  - if it is bigger than both children, be sure to swap with the smaller one.

To decrease a key

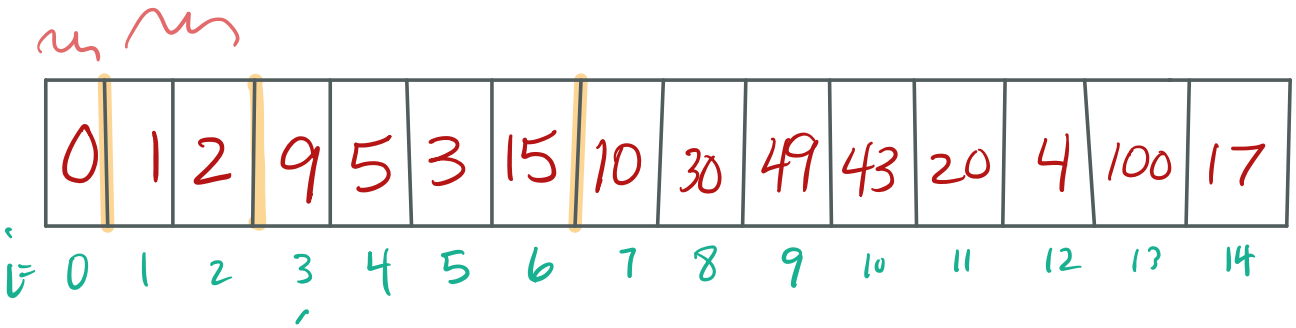


decrease-key to 2



decrease-key to 0

How to implement a heap



if i is even and >0, $p(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$

if i is odd, $p(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$

```
int p(int i)                    // integer division
    if i==0 return 0            // is used to
    return (i-1)/2              // return ⌊(i-1)/2⌋
```

```
int left(i)
    return 2i+1
```

```
int right(i)
```

return 2i+2