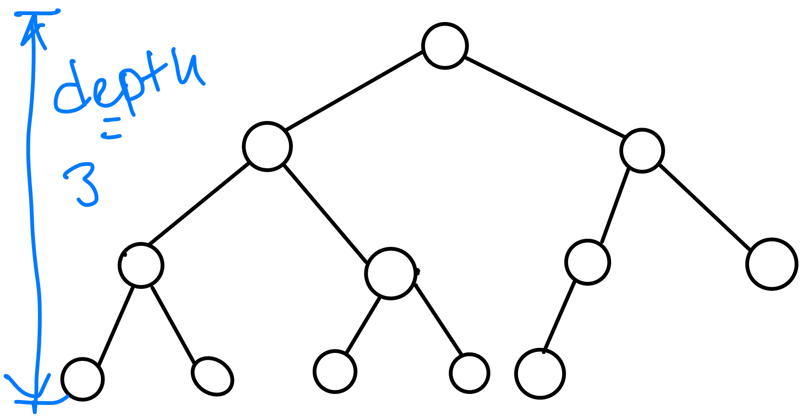
Lecture 14: Heaps Continued

1. Review Heap properties
2. Representing heaps w/ Arrays

Last Time

Complete binary trees (CBTs)



Properties:

1. all nodes at depth $\leq d-2$ have 2 children
2. At most 1 node at depth $d-1$ has 1 child, and it is a left child
3. If v at depth $d-1$ has children and u is to the left (@ depth d), then u has 2 children
4. If v at depth $d-1$ has < 2 children and w is to the right, then w has no children

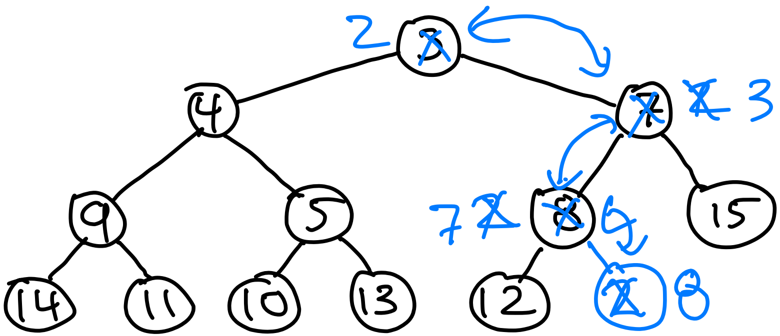
Consequence. If T is a complete binary tree then:

1. There is a unique location that a leaf can be added to result in a CBT
2. There is a unique leaf that can be removed to result in a CBT

Also last time: Heaps

A *heap* is a CBT in which each node stores a comparable element and satisfies:

Heap property the value stored at a node is no larger than the values stored by its children



Adding to a heap:

1. Add element at unique location to append a new leaf
2. "Bubble up":
   - v ← new node
   - while (v's val < v's parent's val)
     - swap values
     - set v = v.parent

Example: add(2)

Removing min from a heap:

1. Store root val (to be returned)

2. Copy value from "last" leaf to root, and set as root value
   - right most leaf @ depth d

3. Remove leaf

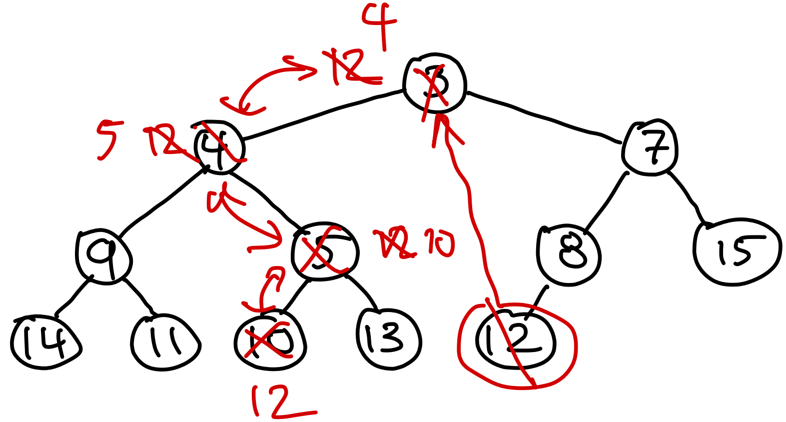
4. "trickle down"

   - $v \leftarrow$ root

   - while (v's val > some child's val)

     - u = smaller child of v
     - swap u and v's vals
     - update $v \leftarrow u$

Value = 3



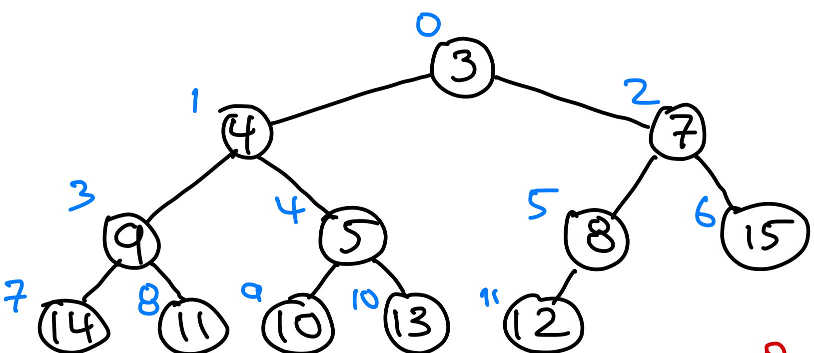
Example : removeMin()

<u>Results</u>: add and removeMin maintain heap property / CBT and can be performed with $O(\log n)$ compare/swap operations

Representing CBTs as Arrays

Since CBTs have predictable structure, we can represent them neatly as arrays:

- root at index 0
- left/right children @ indices 1, 2
- grandchildren @ 3,4,5,6 (left to right)

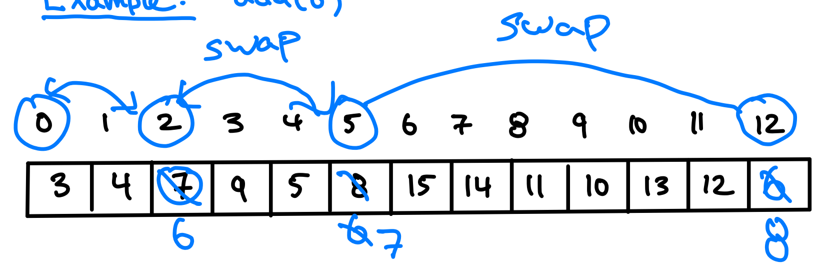


0	1	2	3	4	5	6	7	8	9	10	11	12
3	4	7	9	5	8	15	14	11	10	13	12	

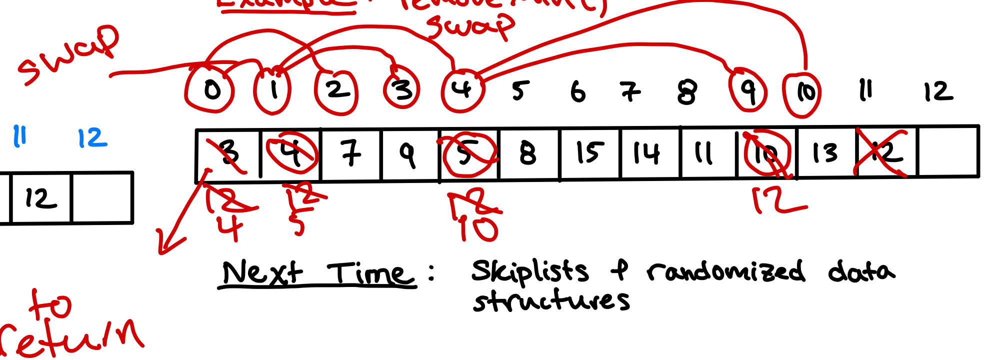
We can access children/parents directly from the array:

- left child of index i is $2*i+1$
- right child of index i is $2*i+2$
- parent of index i is $(i-1)/2$

Example: add(6)



Example: removeMin()



Next Time: Skiplists & randomized data structures