Priority Queues <K, I>

```
void  Insert(int x)     // inserts x
                        // into collection

int   DeleteMin()       // removes & returns
                        // highest priority item

int   FindMin()         // returns highest
                        // priority item w/out removing

bool  IsEmpty()         // true if empty
```

int SearchAndRemove(int x)

void IncreasePriority(int x)

void DecreasePriority(int x)

min-heap
→ low values are more important

winner: 1st place
runner-up: 2nd place

max-heap
high values are more important

winner: most pts
runner-up: 2nd most pts

DeleteMax
FindMax

unsorted list: insert : $O(1)$

FM/DM : $O(n)$ wc & ac

↳ insert & delete of 1 item: $O(n)$ wc, ac

sorted list: insert: $O(n)$ wc ac

FM/DM: $O(1)$

avg $O(\frac{n}{2}) = O(n)$

↳ insert & delete 1 item $O(n)$ wc, ac

BST $O(\lg n)$ bc

insert / search / remove : $O(\lg n)$ ac

$O(n)$ wc

insert: $O(\lg n)$ ac, $O(n)$ wc

deletemin: $O(\lg n)$ ac, $O(n)$ wc

insert & delete of 1 item : $O(n)$ wc

$O(\lg n)$ ac

can we force this? $\rightarrow$ $O(\lg n)$ bc

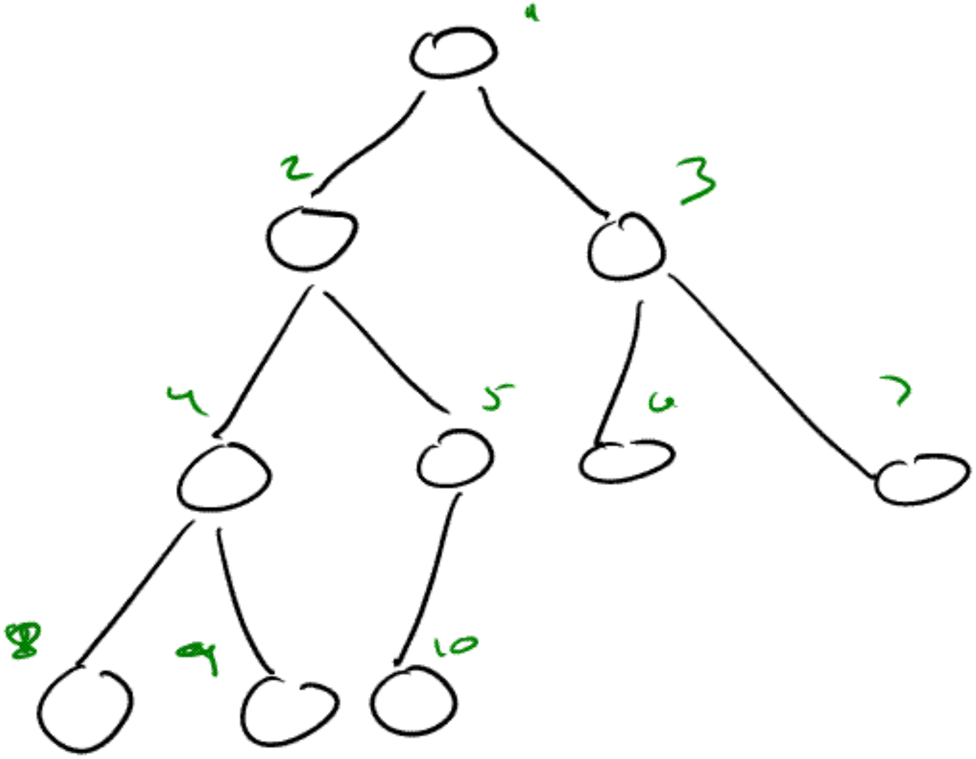
can we force tree to be balanced?

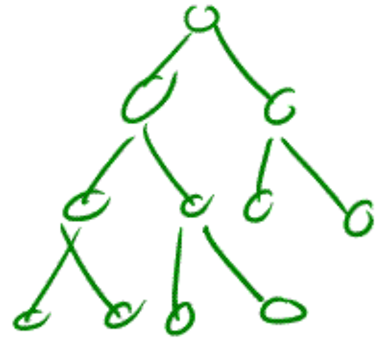
↳ Yes, but we'll give up BST property

Complete tree

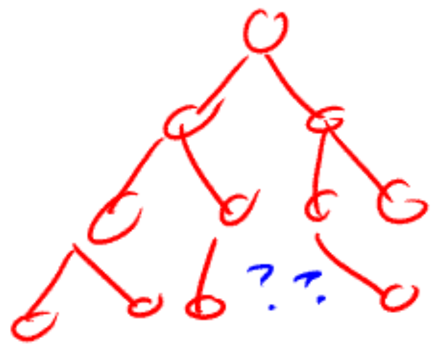
"perfect down to some depth. then one level below that, add children left to right

"insertion order is Level order on infinitely deep full binary tree"

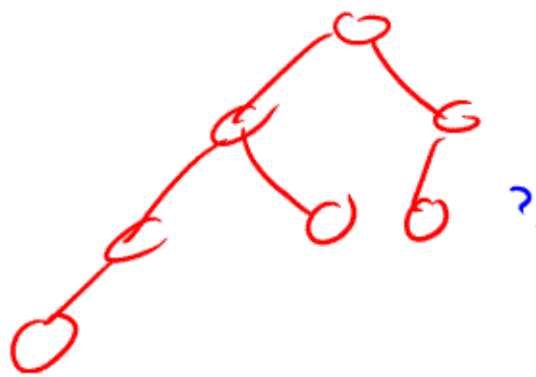




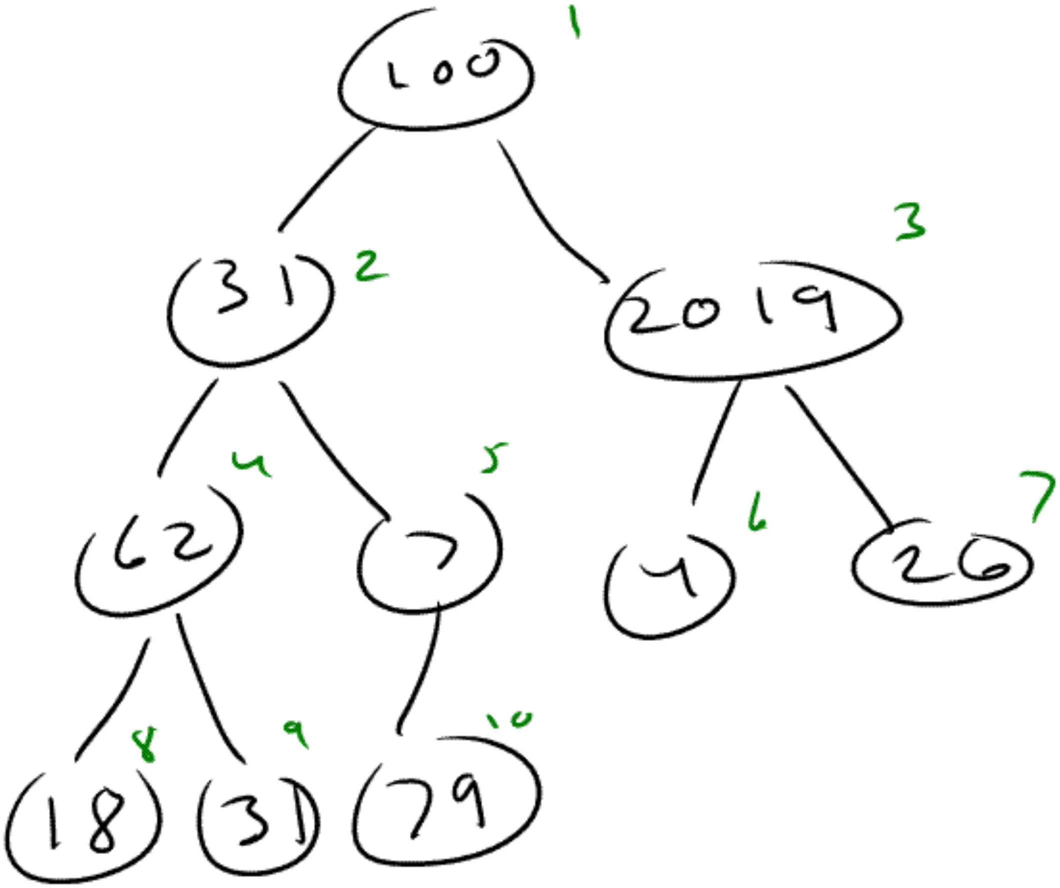
OK



NO



NO



i

$$\text{LeftChild}(i) = 2i$$

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

$$\text{Parent}(i) = \lfloor i/2 \rfloor$$

100	31	2019	62	7	4	26	18	31	79
1	2	3	4	5	6	7	8	9	10

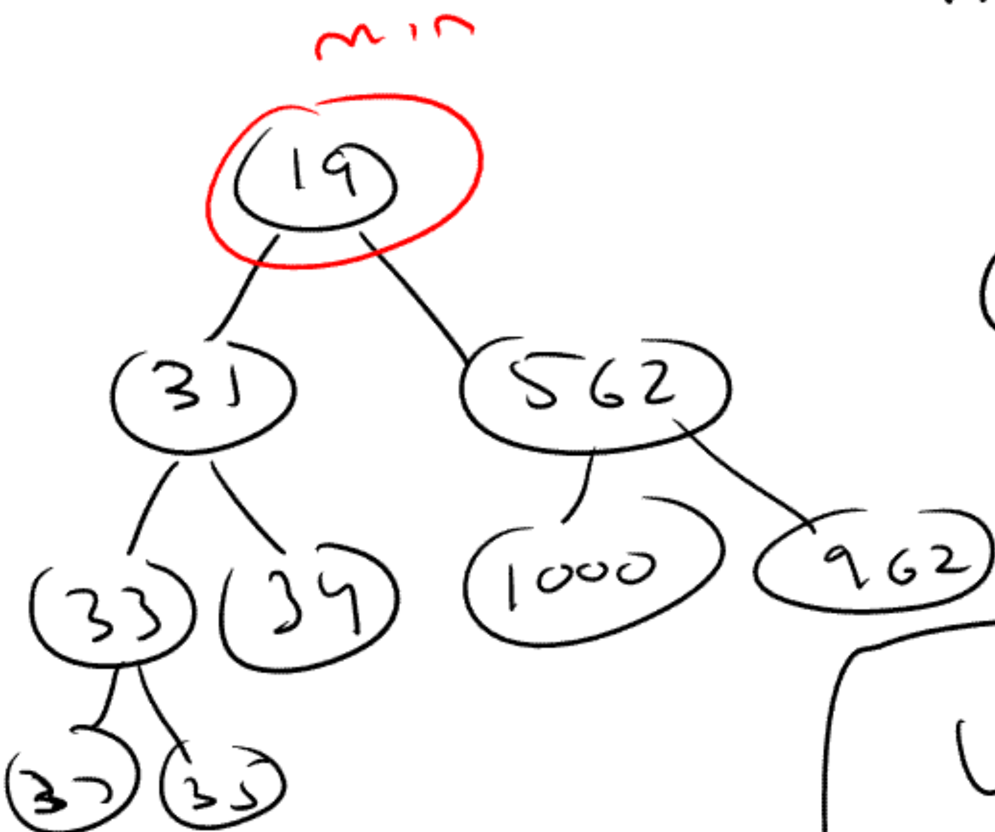
partial order

every node in min heap less than its children

$\leq$ if you have duplicates

heap:

1. partially ordered
2. complete tree
3. implemented as array



Why? next time...