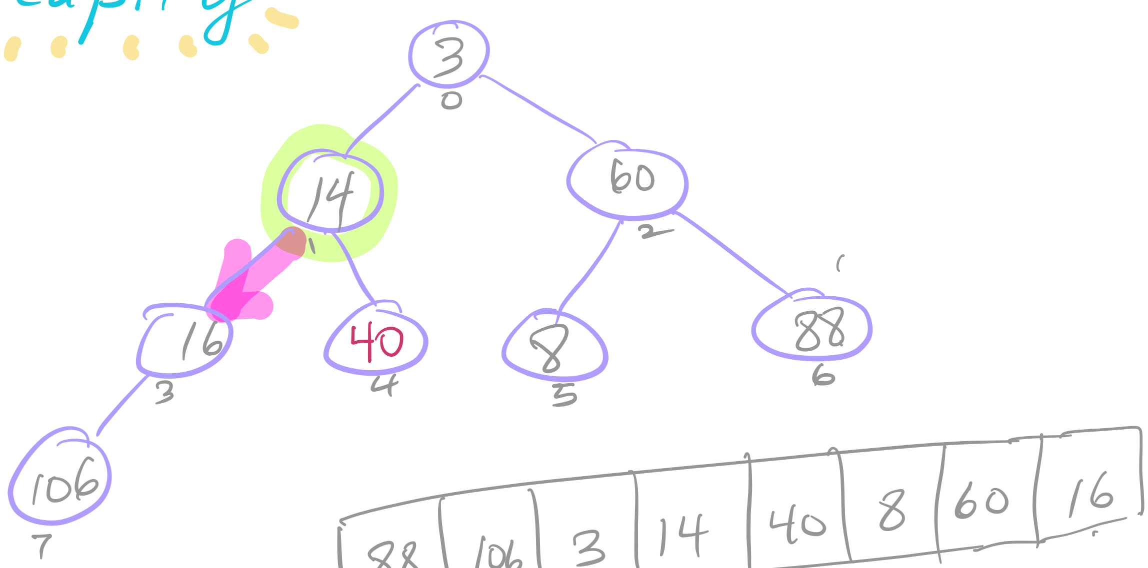
More on Heaps

Suppose we are to initialize a new, empty heap and insert {40, 88, 106, 14, 3, 16, 60, 8} into it.

We could start with an empty heap and do 8 inserts.

It is more efficient, however, to simply write those values into our array and

Heapify



Heapify(1)

- assume it's two child subtrees are heapified

- if value at node 1 is out of order with its children, send it down (swap with smallest child)

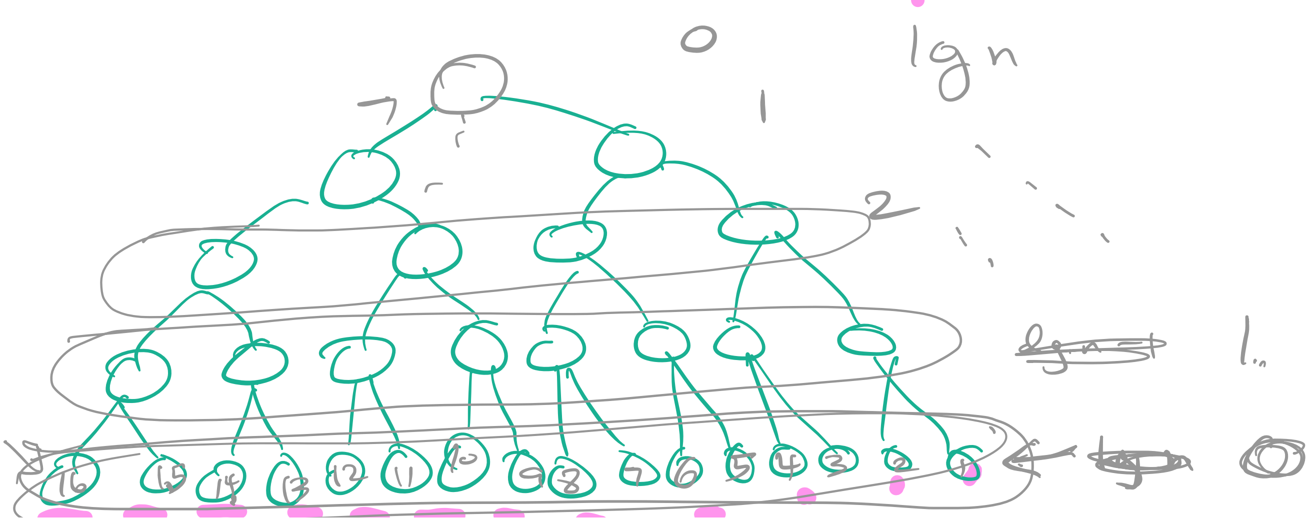
Clearly, this is a bottom-up approach.

Why is bottom-up preferred over top down in this DS?

Analysis:

- the max an element will have to move up is its depth.

- the max an element will have to move down is its lgn - depth



How many nodes have great depth?

How many nodes have great height?

Analysis of worst case for bottom up $\rightarrow$ layers (heading downwards)

MakeHeap:

$$\frac{n}{2}\times 0 + \frac{n}{4}\times 1 + \frac{n}{8}\times 2 + \cdots + \frac{n}{2^{\lg n}}\cdot(\lg n - 1)$$

Other case is much worse

$$\frac{n}{2}\cdot \lg n + \frac{n}{4}\times \lg n - 1 + \cdots + \frac{n}{2^{\lg n}}\cdot 0$$

$$\leq n \cdot \underbrace{\left[\frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots + \cdots\cdots\right]}$$

$$S = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \frac{4}{32} + \frac{5}{64} + \cdots$$

$$2S = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} + \cdots$$

$$2S - S = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \cdots$$

$$= 1$$

$\therefore$ max # swaps if heapify downward, all elements, bottom up: $1 \cdot n = n$.

If we to heapify upwards, what is worst case?

$$\frac{n}{2} \cdot \lg n + \frac{n}{4}$$

MakeHeap(A, n)

// A an array elements, A[0..n-1]

$O(n)$

for $i = \lfloor \frac{n}{2} - 1 \rfloor$ downto 0

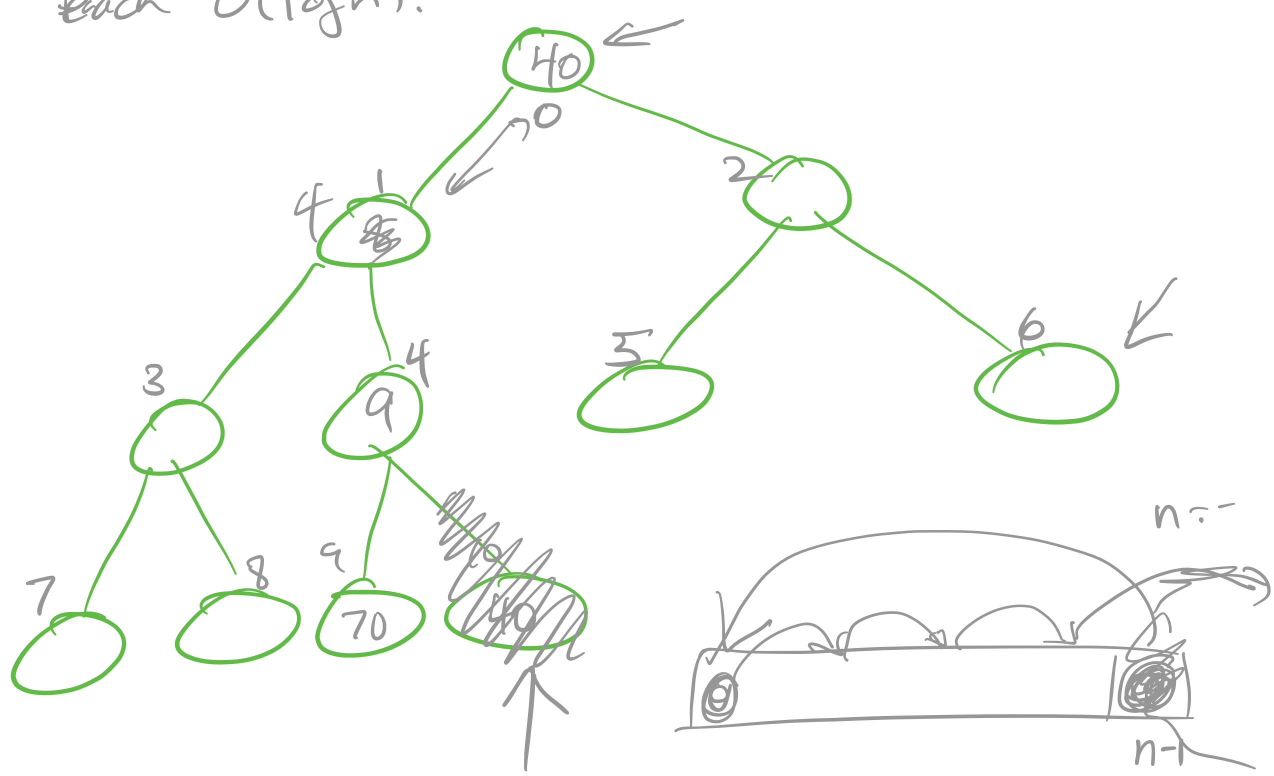
Heapify(A, i)

n operation

Extract + RemoveMin

each $O(\log n)$.

3



$n = \cancel{15}\ 14$