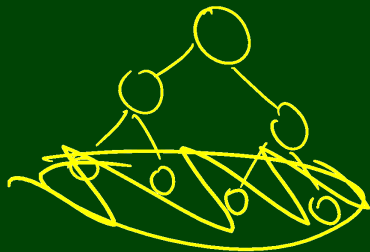


Suppose a heap has n nodes.

- How many nodes on the bottom level?

$$\begin{aligned}h &= 4 + 2 + 1 \\ &= 7\end{aligned}$$



Suppose a heap has n nodes.

- How many nodes on the bottom level? $\frac{n}{2}$
- And the level above? $\frac{n}{4}$
- etc.

Suppose we have a random value, x , in the heap.

- How often is x in the bottom level?



What else can we do with a heap?

Given a particular index i into the array...

- `decreaseKey(i, newPriority)`: Change priority, percolate up
- `increaseKey(i, newPriority)`: Change priority, percolate down
- `remove(i)`: Call `decreaseKey(i, $-\infty$)`, then `deleteMin`

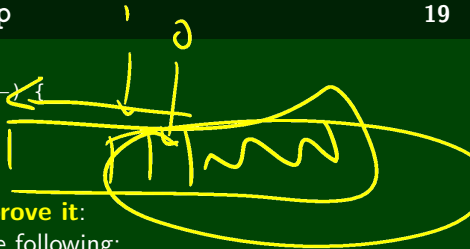


What are the running times of these operations?

```

1 void buildHeap(int[] input) {
2     for (i = (size + 1)/2; i >= 0; i--) {
3         percolateDown(i);
4     }
5 }

```



The algorithm seems to work. Let's **prove it**:

To prove that it works, we'll prove the following:

Before loop iteration i , all $\text{arr}[j]$ where $j > n/2 - i$ have the heap property

Formally, we'd do this by induction. Here's a sketch of the proof:

- Base Case: All $j > (\text{size} + 1) / 2$ **have no children**.
- Induction Step:

We know that `percolateDown` **preserves the heap property** and makes its argument also have the heap property. So, after the $(i+1)$ st iteration, we know i is less than all its children and by the IH, we know that all of the children past $\text{arr}[i]$ already had the heap property (and `percolateDown` didn't break it).

So, since the loop ends with index 0, once we're done all the elements of the array will have the heap property.

```
1 void buildHeap(int[] input) {  
2     for (i = (size + 1)/2; i >= 0; i--) {  
3         percolateDown(i);  
4     }  
5 }
```

Was this even worth the effort?

The loop runs $n/2$ iterations and each one is $\mathcal{O}(\lg n)$; so, the algorithm is $\mathcal{O}(n \lg n)$.