Heaps I

- Heaps
- Properties
- Insertion and Deletion
Heaps

• A **heap** is a binary tree $T$ that stores a collection of keys (or key-element pairs) at its internal nodes and that satisfies two additional properties:
  - **Order Property**: $\text{key}(\text{parent}) \leq \text{key}(\text{child})$
  - **Structural Property**: all levels are full, except the last one, which is left-filled (**complete binary tree**)
Not Heaps

- bottom level is not left-filled

- key(parent) > key(child)
Height of a Heap

A heap $T$ storing $n$ keys has height $h = \lceil \log(n + 1) \rceil$, which is $O(\log n)$

- $n \geq 1 + 2 + 4 + ... + 2^{h-2} + 1 = 2^{h-1} - 1 + 1 = 2^{h-1}$
  - $n \leq 1 + 2 + 4 + ... + 2^{h-1} = 2^h - 1$
- Therefore $2^{h-1} \leq n \leq 2^h - 1$
- Taking logs, we get $\log(n + 1) \leq h \leq \log n + 1$
- Which implies $h = \lceil \log(n+1) \rceil$
Heap Insertion

So here we go ...

The key to insert is 6

```
    3
   /|
  4 7
 /|
21 10 20 8
/|
22 28 13 19 25
/|
  8
```

Heaps I
Heap Insertion

Add the key in the *next available position* in the heap.

Now begin *Upheap*.
Upheap

- *Swap parent-child keys out of order*
Upheap Continues

Heaps I
End of Upheap

- *Upheap* terminates when new key is greater than the key of its parent or the top of the heap is reached

- (total #swaps) $\leq (h - 1)$, which is $O(\log n)$
Removal From a Heap

RemoveMin()

- The removal of the top key leaves a hole
- We need to fix the heap
- First, replace the hole with the last key in the heap
- Then, begin *Downheap*
Downheap compares the parent with the smallest child. If the child is smaller, it switches the two.
Downheap Continues

Heaps I
Downheap Continues

Heaps I

6.13
• *Downheap* terminates when the key is greater than the keys of both its children or the bottom of the heap is reached.

• (total #swaps) $\leq (h - 1)$, which is $O(\log n)$