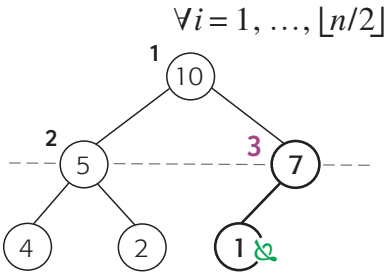
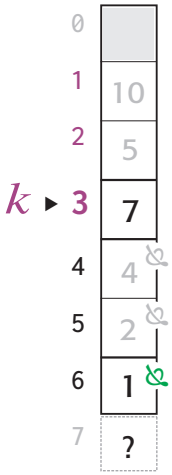
Heap Array Representation
Operation Analysis



$$H[i] \geq \max\{H[2i], H[2i+1]\}$$

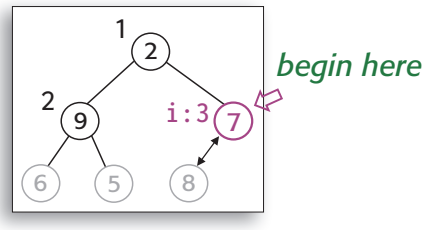


Exercise
Write conditions to test if a parent node *k* has none, 1, or 2 children. **Hint**: compare to number of nodes.

- **Parent and leaf indices (previously)**
- **Node count in parental level (review)**

Parent node cases

Heap Construction
Bottom-Up Heapify



Exercise
Trace the instance (code, generate a log to study, see *heapsort* slide).

Algorithm *HeapBottomUp*
Input Array $H[1..n]$ of ordarable keys
Output A heap in $H[1..n]$

```
1:  for i ← ⌊n/2⌋ downto 1 do                ▷ Scan parental nodes from last up to root
2:      k ← i, v ← H[k]                      ▷ Pick next node, save key
3:      heap ← false
4:      while not heap and 2*k ≤ n do        ▷ skip if k has no child (2k > n)
5:          j ← 2*k                          ▷ Pick first (left) child of k (assume largest)
6:          if j < n then                    ▷ 2nd child? (see case analysis)
7:              if H[j] < H[j+1] then j ← j+1   ▷ Pick 2nd child if larger
8:          if v ≥ H[j] then heap ← true     ▷ Confirm heap condition and exit
9:          else H[k] ← H[j], k ← j          ▷ Otherwise, sink parent (replace by largest)
10:     H[k] ← v
```

Stop heapify if either heap condition true or no children to check.

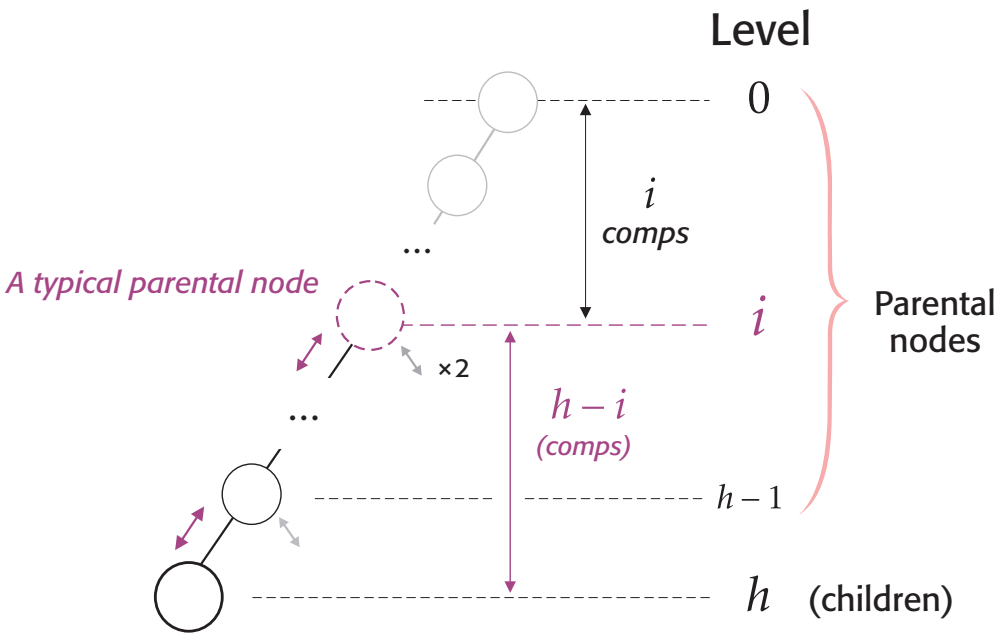
Quiz
Suggesst a suitable basic operation?

Bottom-Up Construction
Efficiency Summation

Quiz
How many nodes are in a **complete** tree (worst case heap shape)? Determine the height.

Quiz
How many key comps made by a node in level i until its key sinks down to a leaf? (Again, a worst case for heapify).

Exercise
Use $h = 5$ instance to check the figure (assume $i = 2$), write the worst-case efficiency summation.



$$= \log_2(n+1) - 1$$

$$\Leftrightarrow\ n = 2^{h+1} - 1$$

i

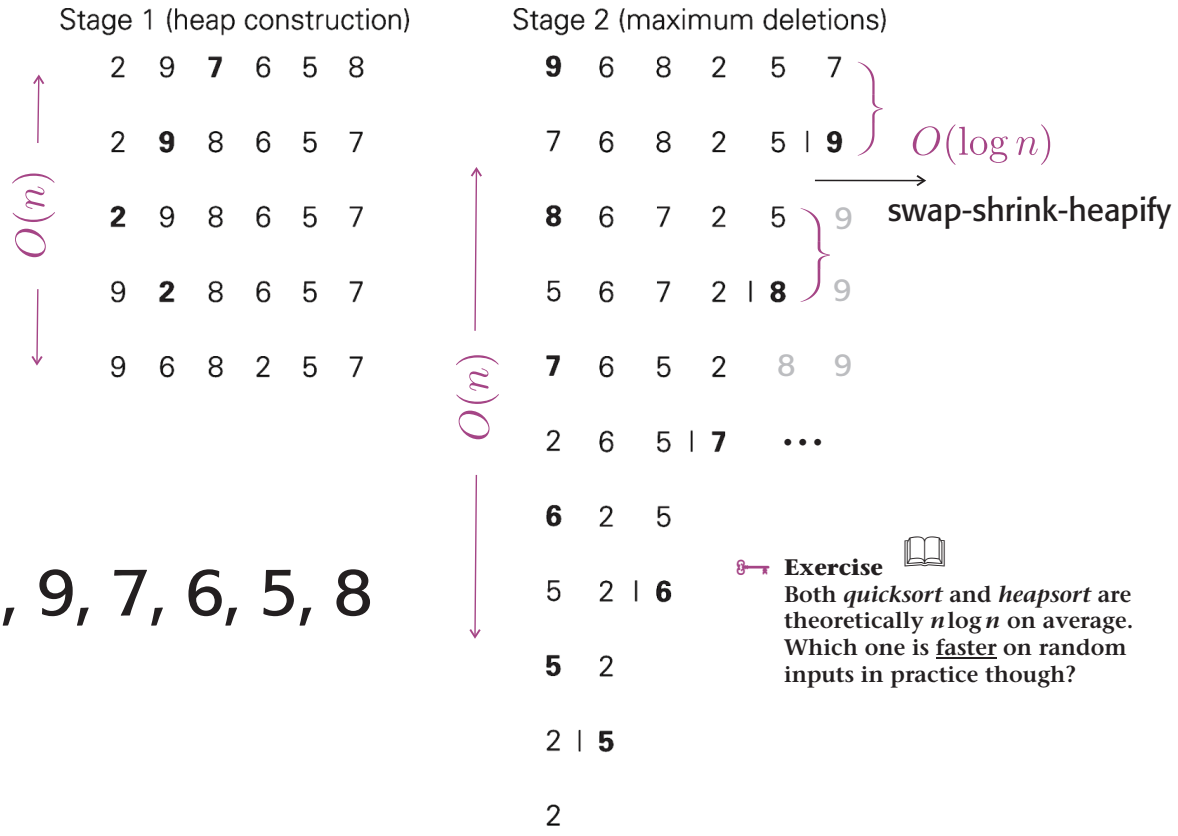
Heapsort

Exercise
Use your code to generate a log of construction steps for this instance.

```
bot-up (3): 2,9,7,6,5,8
> heapify: 2,9,8,6,5,7
bot-up (2): 2,9,8,6,5,7
bot-up (1): 2,9,8,6,5,7
> heapify: 9,2,8,6,5,7
> heapify: 9,6,8,2,5,7
```

Hint
Comparisons and sink logic in array-based pseudocode different than logical tree diagrams. (Note how max delete and *BottomUp* share logic! Explain).

$O(?)$ Sort 2, 9, 7, 6, 5, 8



Exercise
Both *quicksort* and *heapsort* are theoretically $n \log n$ on average. Which one is faster on random inputs in practice though?

The Heap
Performance Review

Quiz
Where does the extra efficiency come from (why not *n*log *n*)?

⇨ **Heap construction** $O(n)$

Quiz
What's the efficiency if heap is constructed by successive (repeated) inserts?

⇨ **Insert/max delete** $O(\log n)$

Exercise
Compare to *mergesort*.

⇨ **Heapsort** ? $n \log n$

Exercise
Refine your pen-paper *insert* and *max-delete* pseudocode in terms of array operations.

⇨ **Programming exercises**