



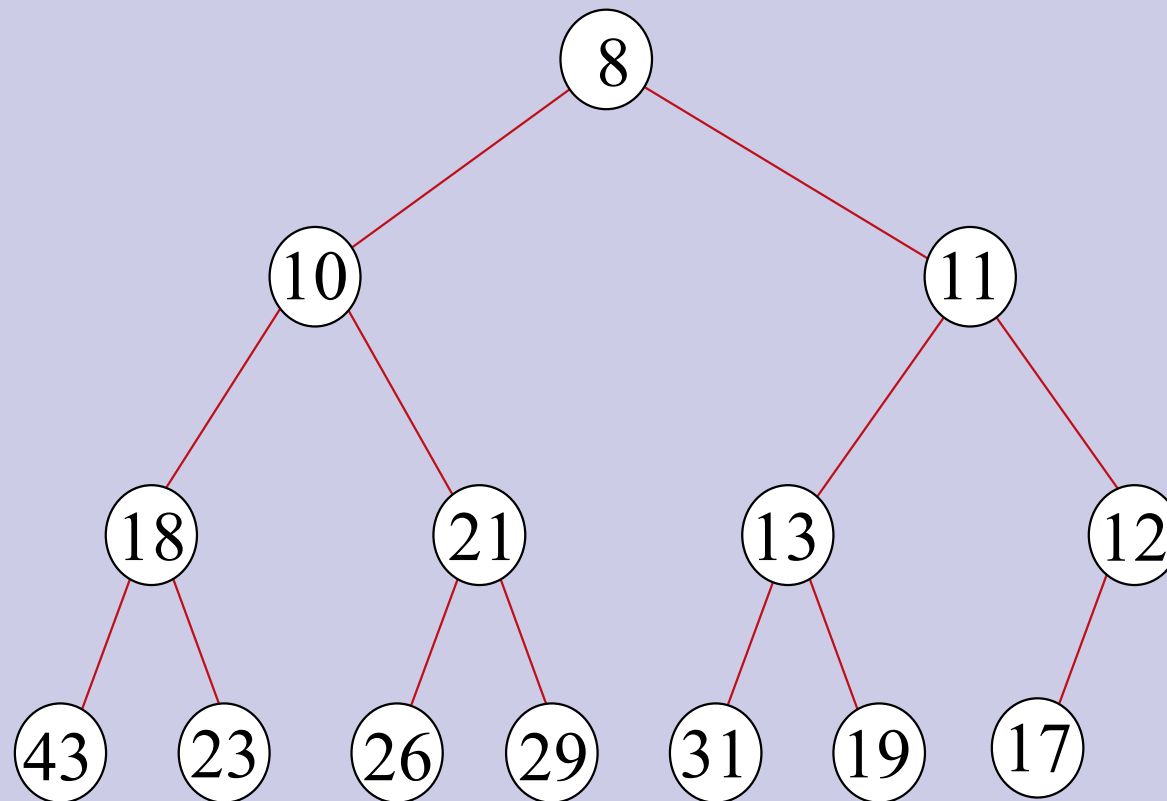
Binary Heaps

- ☐ Delete-min
- ☐ Building a heap in $O(n)$ time
- ☐ Heap Sort

Delete-min

- The minimum element is the one at the top of the heap.
- We can delete this and move one of its children up to fill the space.
- Empty location moves down the tree.
- Might end up at any position on last level.
- Resulting tree would not be left filled.

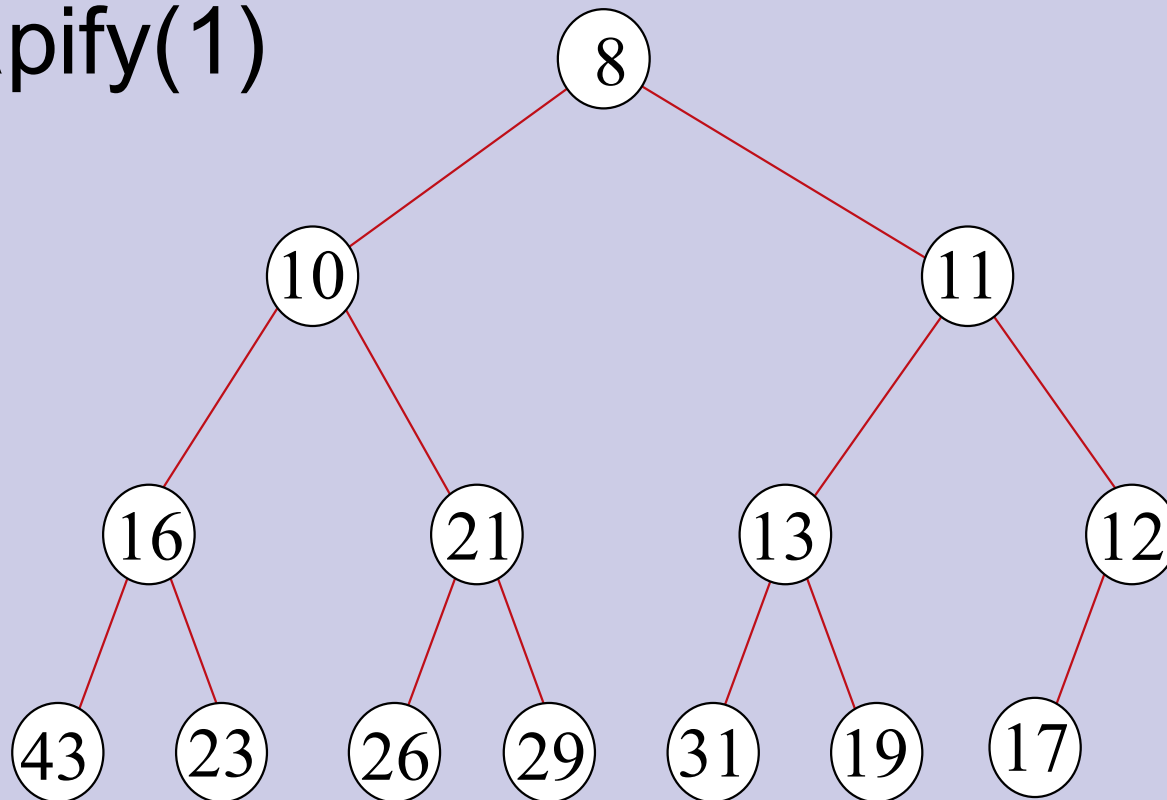
Delete-min in a Heap



This is not a heap

Delete-min in a Heap (2)

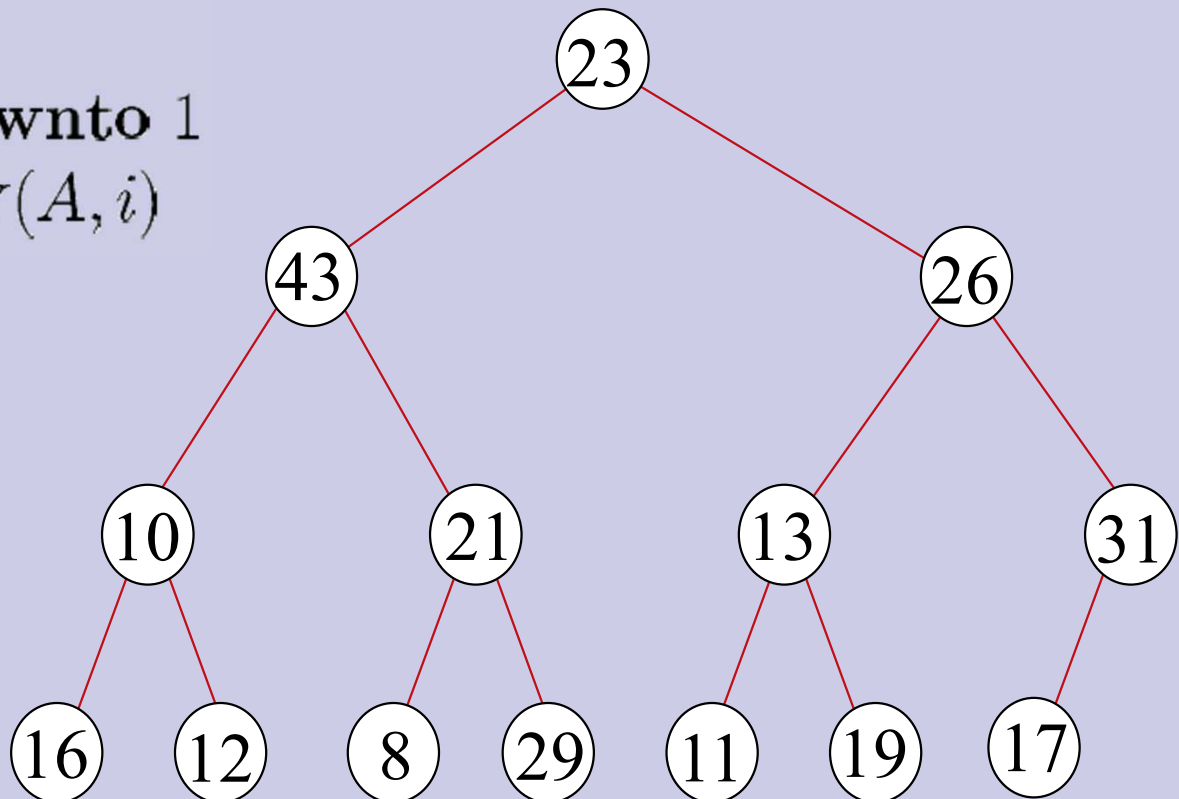
- Replace top element with last element of heap.
- Heapify(1)



Building a heap

- We start from the bottom and move up
- All leaves are heaps to begin with

```
BUILD-HEAP( $A$ )  
  1 for  $i \leftarrow \lfloor n/2 \rfloor$  downto 1  
  2   do HEAPIFY( $A, i$ )
```



Building a Heap: Analysis

- Correctness: induction on i , all trees rooted at $m > i$ are heaps
- Running time: n calls to Heapify = $n O(\lg n) = O(n \lg n)$
- We can provide a better $O(n)$ bound.
 - Intuition: for most of the time Heapify works on smaller than n element heaps

Building a Heap: Analysis (2)

- height of node: length of longest path from node to leaf
- height of tree: height of root
- time for $\text{Heapify}(i) = O(\text{height of subtree rooted at } i)$
- assume $n = 2^k - 1$ (a complete binary tree)

Building a heap: Analysis (3)

- For the $n/2$ nodes of height 1, heapify() requires at most 1 swap each.
- For the $n/4$ nodes of height 2, heapify() requires at most 2 swaps each.
- For the $n/2^i$ nodes of height i , heapify() requires at most i swaps each.
- So total number of swaps required is

$$\begin{aligned} T(n) &= O\left(\frac{n+1}{2} + \frac{n+1}{4} \cdot 2 + \frac{n+1}{8} \cdot 3 + \dots + 1 \cdot k\right) \\ &= O\left((n+1) \cdot \sum_{i=1}^{\lfloor \lg n \rfloor} \frac{i}{2^i}\right) \text{ since } \sum_{i=1}^{\lfloor \lg n \rfloor} \frac{i}{2^i} = \frac{1/2}{(1-1/2)^2} = 2 \\ &= O(n) \end{aligned}$$

Building a Heap: Analysis (4)

- How? By using the following "trick"

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \quad \text{if } |x| < 1 \quad // \text{differentiate}$$

$$\sum_{i=1}^{\infty} i \cdot x^{i-1} = \frac{1}{(1-x)^2} \quad // \text{multiply by } x$$

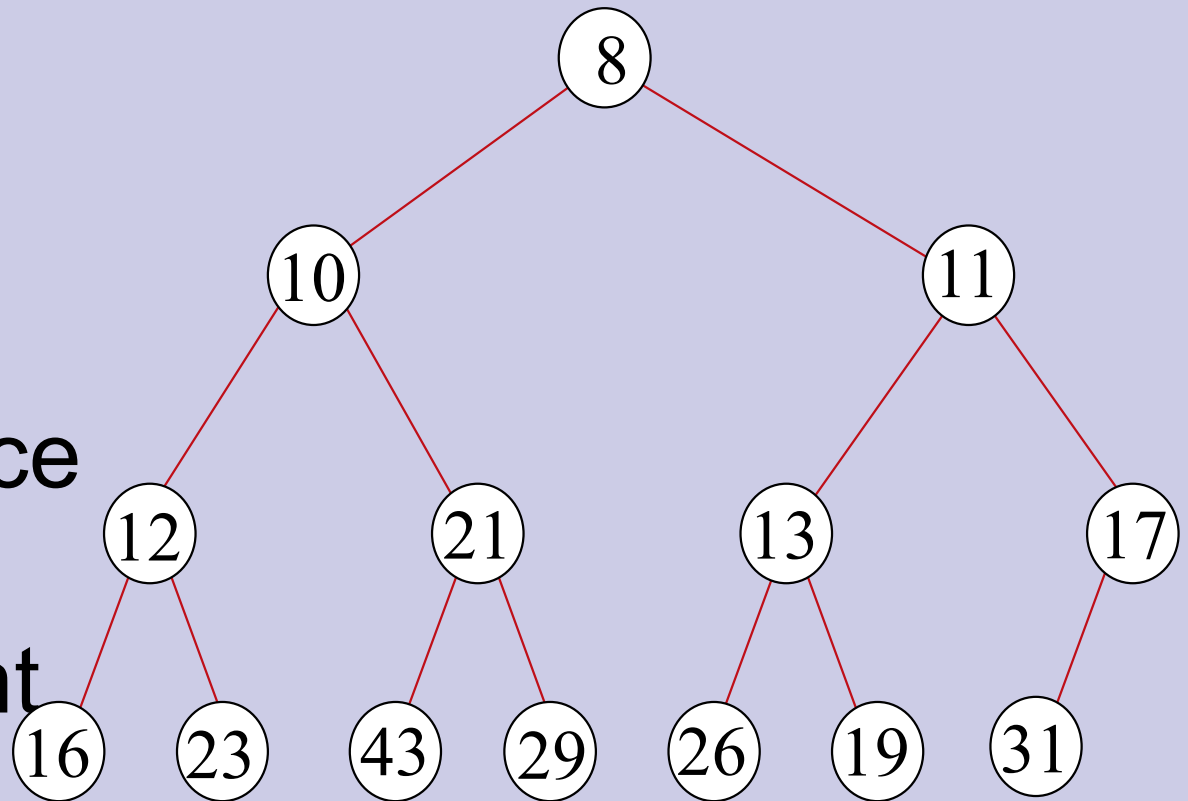
$$\sum_{i=1}^{\infty} i \cdot x^i = \frac{x}{(1-x)^2} \quad // \text{plug in } x = \frac{1}{2}$$

$$\sum_{i=1}^{\infty} \frac{i}{2^i} = \frac{1/2}{1/4} = 2$$

- Therefore Build-Heap time is $O(n)$

Heap Sort

- Create a heap.
- Do delete-min repeatedly till heap becomes empty.
- To do an in place sort, we move deleted element to end of heap.



Running times of heap operations

- Insert: $O(\log n)$
- Heapify: $O(\log n)$
- Find minimum: $O(1)$
- Delete-min: $O(\log n)$
- Building a heap: $O(n)$
- Heap Sort: $O(n \log n)$