ICS 240
Introduction to Data Structures

Jessica Maistrovich

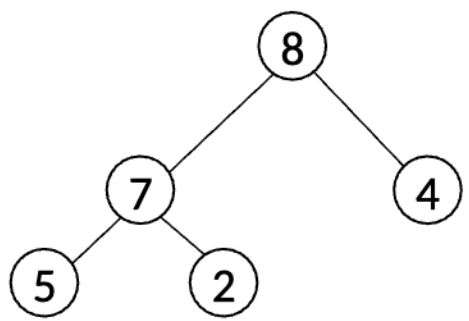
Metropolitan State University

Heap

Abstract Data Type

The Heap Data Structure

- A **heap** is a complete binary tree with the following two properties:
  - **Structural property:** all levels are full, except possibly the last one, which is filled from left to right
  - **Order (heap) property:** for any node x ⇒ Parent(x) ≥ x

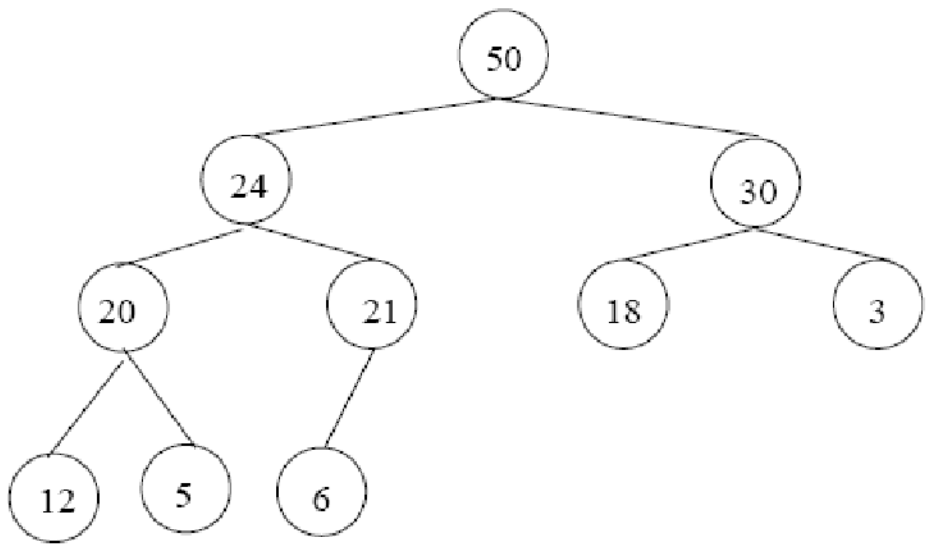


Heap

From the heap property, it follows that:
"The root is the maximum element of the heap"

Adding/Deleting Nodes

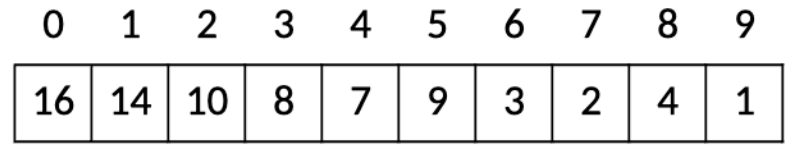
- To maintain the structure property of the heap (i.e., complete binary tree):
  - New nodes are always inserted at the bottom level (left to right)
  - Nodes are removed from the bottom level (right to left)

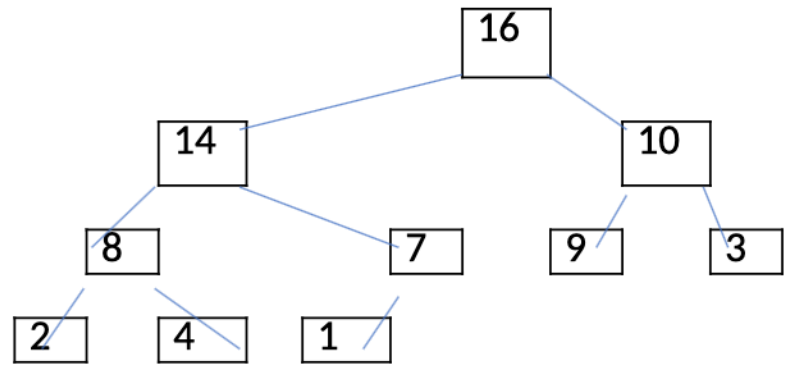


Array Representation of Heaps

- A heap can be stored as an array *A*
  - Root of tree is `A[0]`
  - Left child of `A[i] = A[2i+1]`
  - Right child of `A[i] = A[2i + 2]`
  - Parent of `A[i] = A[ ⌊(i-1)/2⌋ ]`

- The elements in the subarray `A[(⌊n/2⌋) .. n-1]` are leaves





Heap Types

- **Max-heaps** (largest element at root), have the *max-heap property:*
  - for all nodes i, excluding the root:

$$A[parent(i)] \geq A[i]$$

- **Min-heaps** (smallest element at root), have the *min-heap property:*
  - for all nodes i, excluding the root:

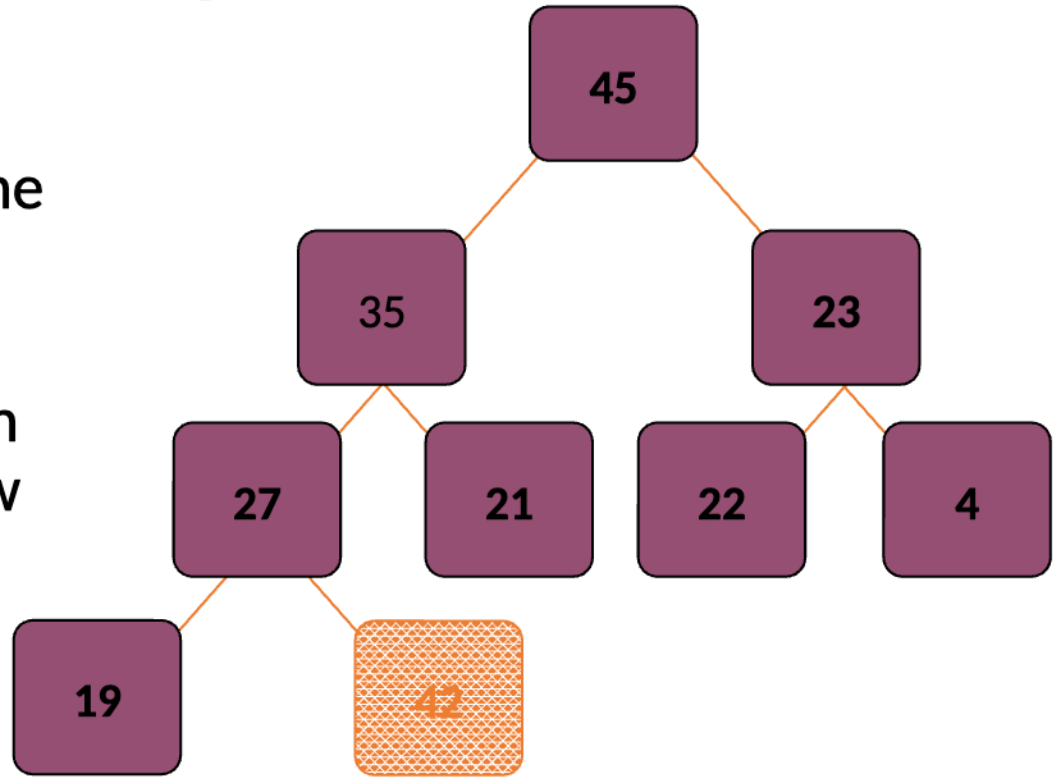
$$A[parent(i)] \leq A[i]$$

Adding to a Heap

Adding a Node to a Heap

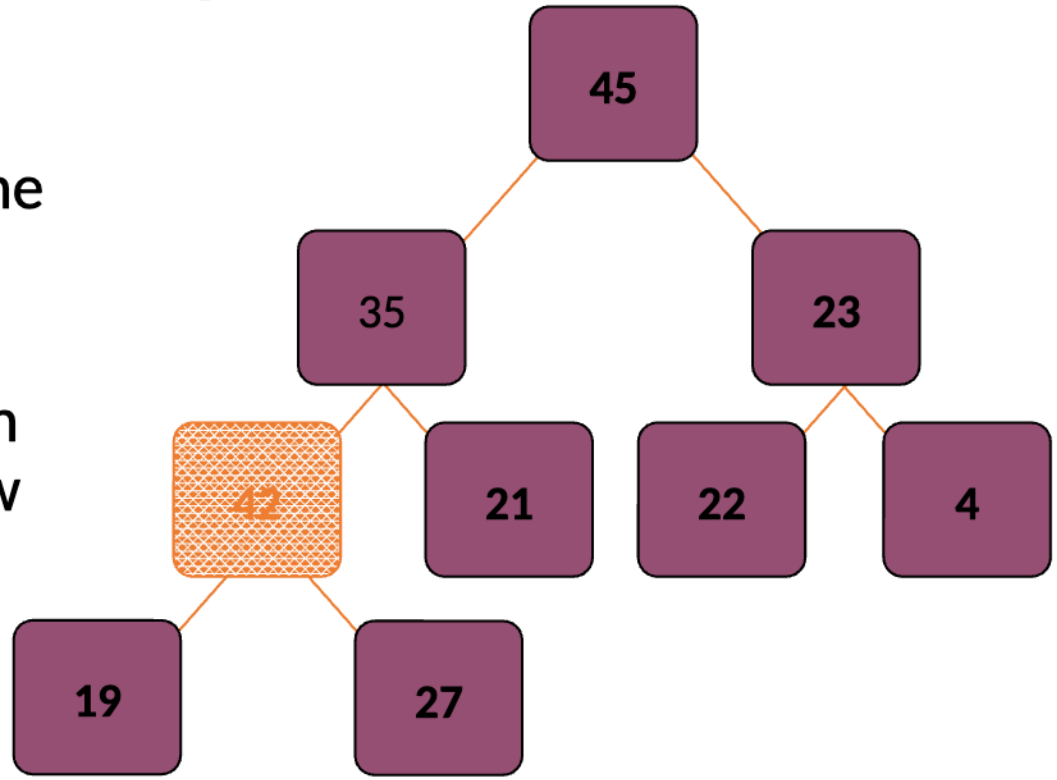
❶Put the new node in the next available spot

❷Push the new node upward, swapping with its parent until the new node reaches an acceptable location



Adding a Node to a Heap

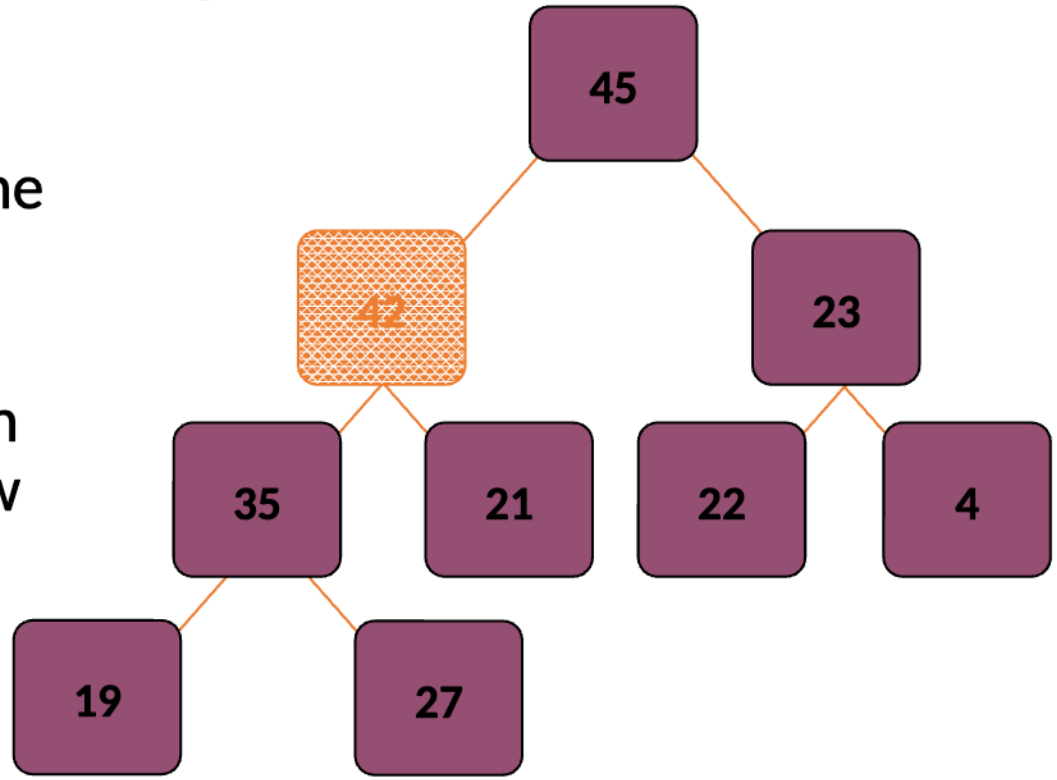
1. Put the new node in the next available spot
2. Push the new node upward, swapping with its parent until the new node reaches an acceptable location



Adding a Node to a Heap

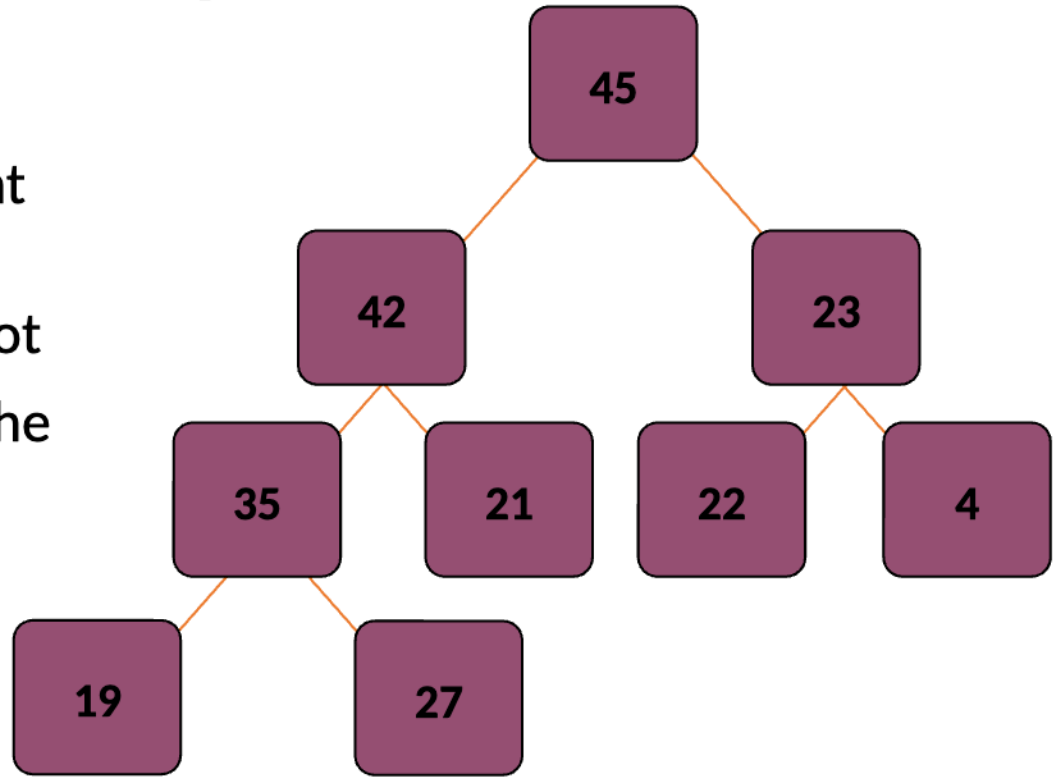
❶Put the new node in the next available spot.

❷Push the new node upward, swapping with its parent until the new node reaches an acceptable location



Adding a Node to a Heap

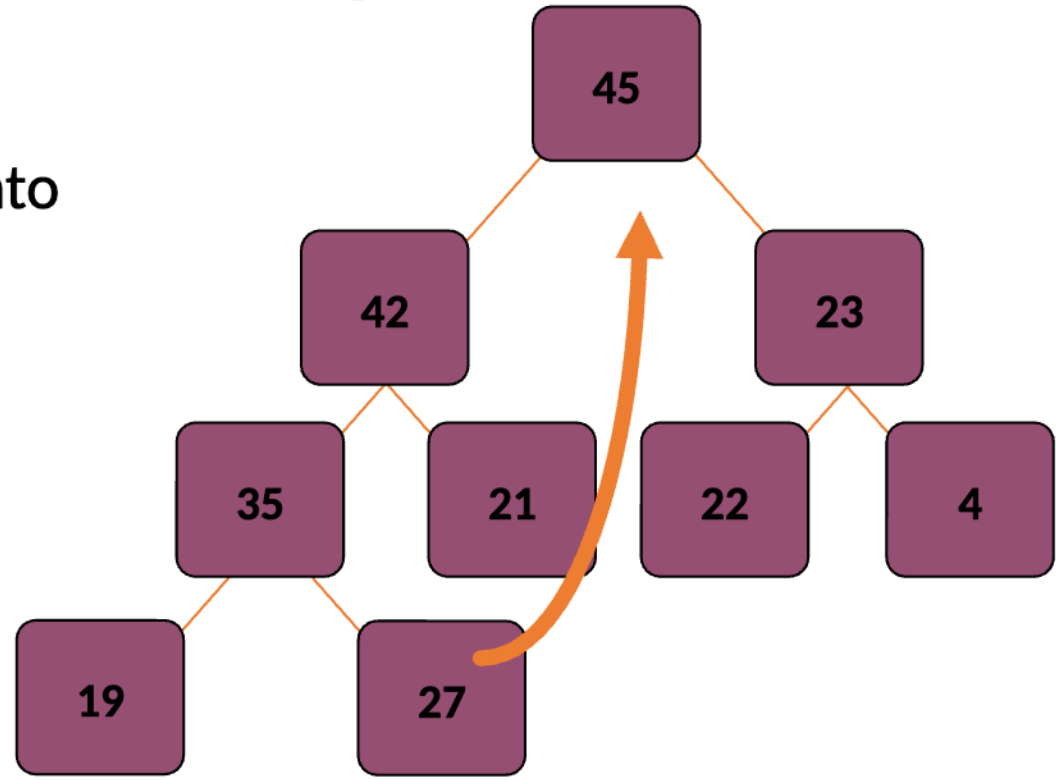
- ✔ The parent has a key that is >= new node, or
- ✔ The node reaches the root
- ↗ The process of pushing the new node upward is called **reheapification upward**



Removing from a Heap

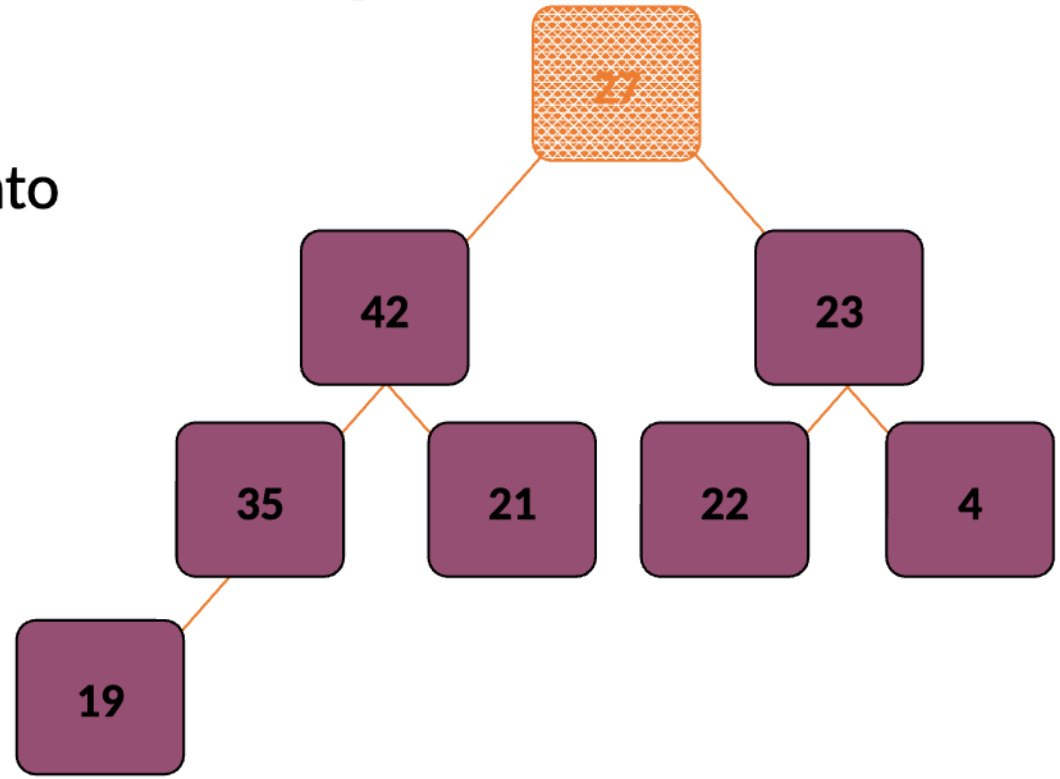
Removing the Top of a Heap

❶ Move the last node onto the root



Removing the Top of a Heap

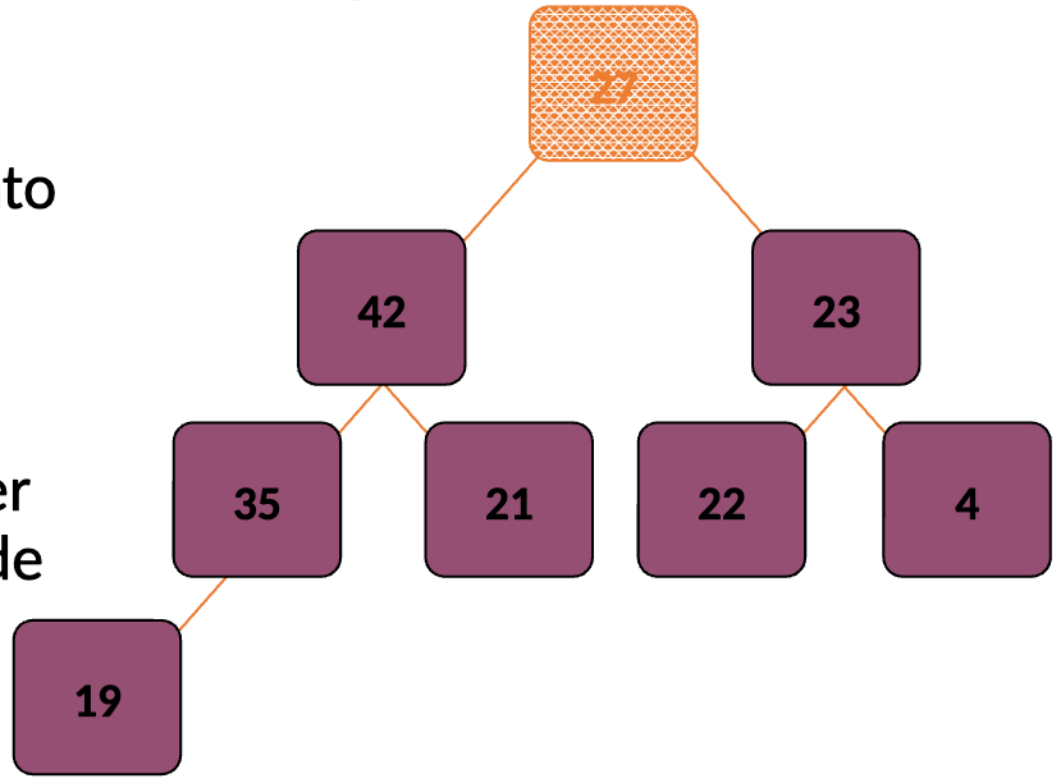
❶ Move the last node onto the root



Removing the Top of a Heap

❶ Move the last node onto the root

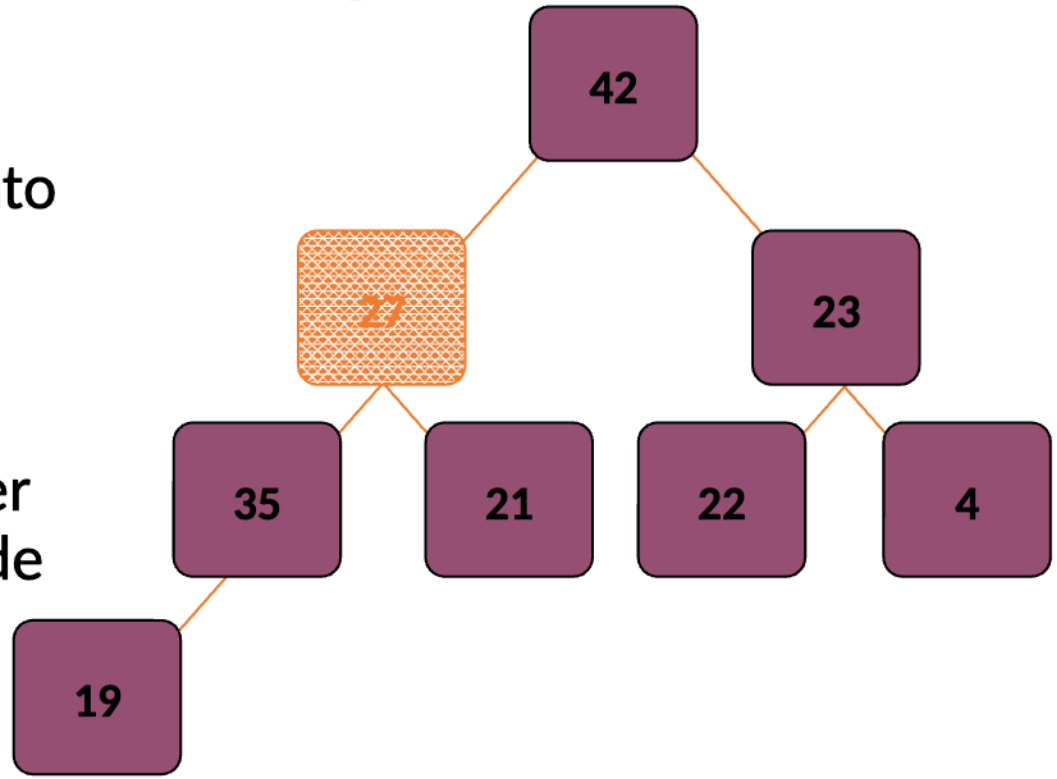
❷ Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location



Removing the Top of a Heap

❶ Move the last node onto the root

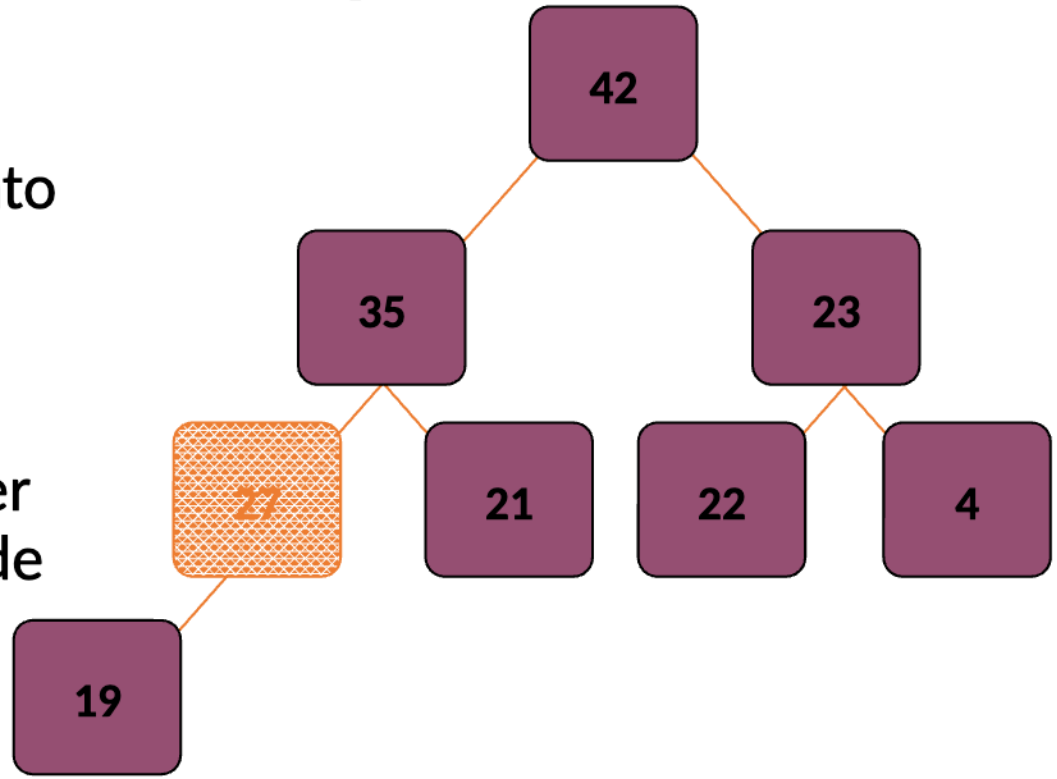
❷ Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location



Removing the Top of a Heap

❶ Move the last node onto the root

❷ Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location



Removing the Top of a Heap

- ✔ The children all have keys <= the out-of-place node, or
- ✔ The node reaches a leaf
- ↘ The process of pushing the new node downward is called **reheapification downward**