CS 2133
Heaps

Tree

A tree is a finite set of one or more nodes such that:

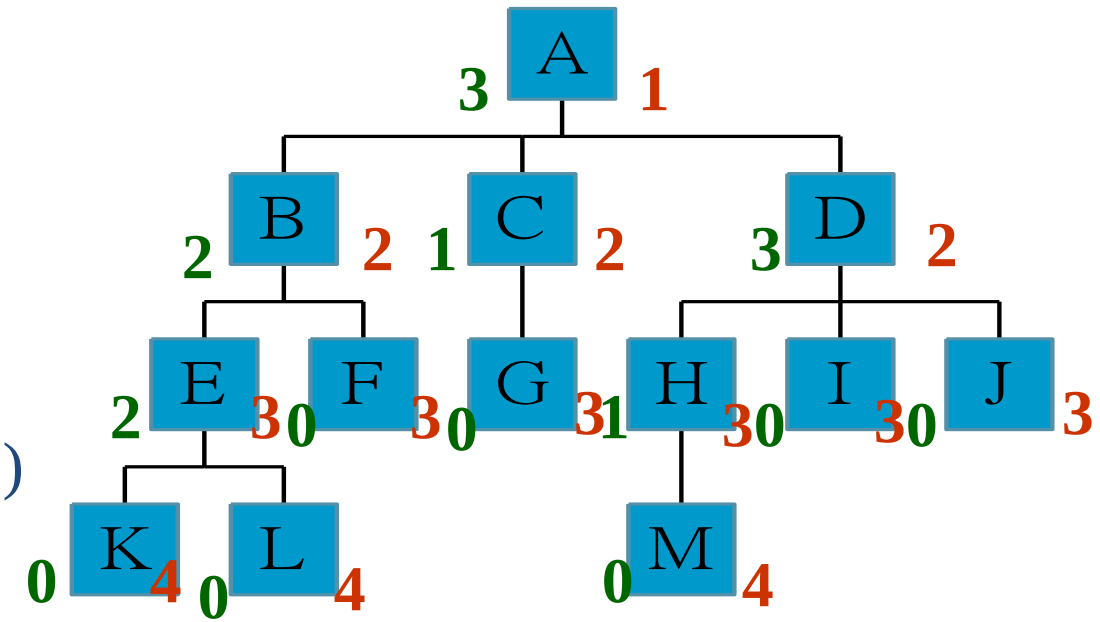
- There is a specially designated node called the root.
- The remaining nodes are partitioned into n>=0 disjoint sets T_1, ..., T_n, where each of these sets is a tree.
- We call T_1, ..., T_n the subtrees of the root.

Terminology

- The degree of a node is the number of subtrees of the node
- The node with degree 0 is a leaf or terminal node.
- A node that has subtrees is the *parent* of the roots of the subtrees.
- The roots of these subtrees are the *children* of the node.
- Children of the same parent are *siblings*.
- The ancestors of a node are all the nodes along the path from the root to the node.

Example

- node (13)
- degree of a node
- leaf (terminal)
- nonterminal
- parent
- children
- sibling
- degree of a tree (3)
- ancestor
- level of a node
- height of a tree (4)



Level

1

2

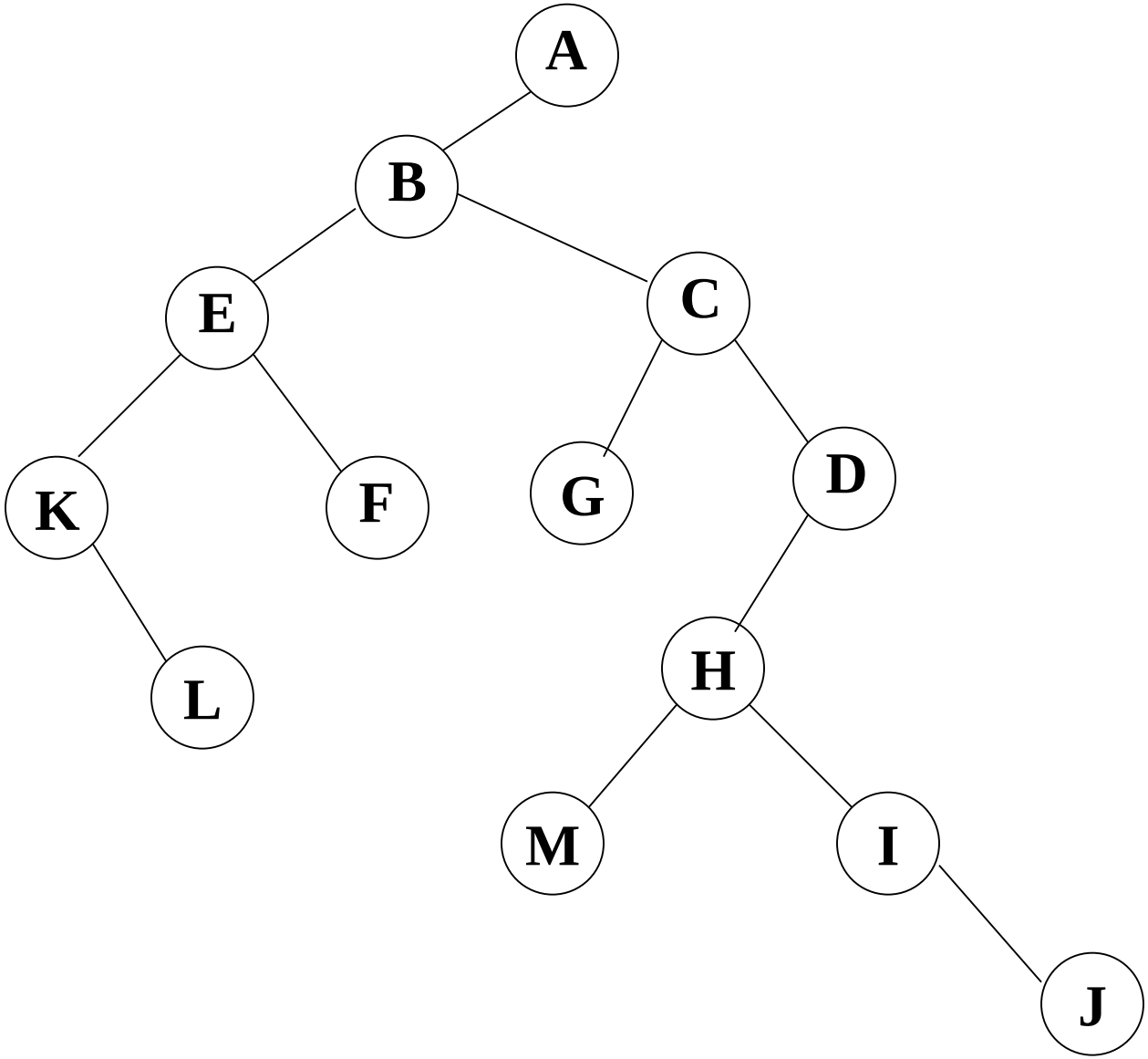
3

4

Binary Tree

- A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called *the left subtree* and *the right subtree*.
- Any tree can be transformed into binary tree.
  - by left child-right sibling representation
- The left subtree and the right subtree are distinguished.

Example of Binary Tree



Number of Nodes in BT

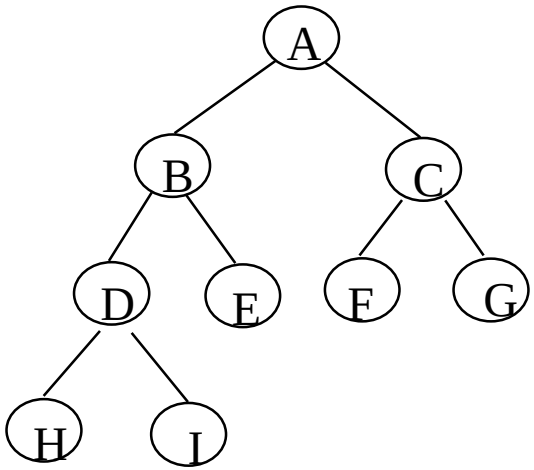
- The maximum number of nodes on level i of a binary tree is 2^{i-1}, i>=1.
- The maximum number of nodes in a binary tree of depth k is 2^k-1, k>=1.

Prove by induction.

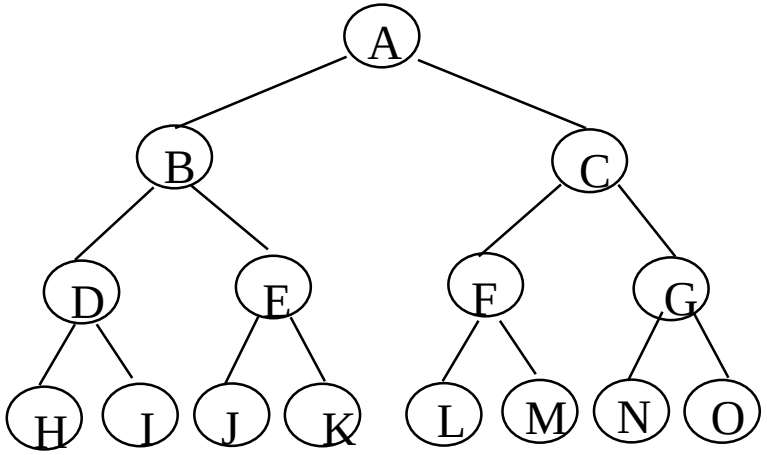
$$\sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

Full BT vs. Complete BT

- A full binary tree is a tree in which every node other than the leaves has two children.
- A complete binary tree is a binary tree in which every level, except possible the last, is completely filled, and all nodes are as far left as possible.



Complete binary tree

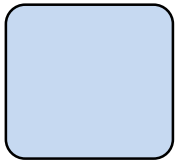


Full binary tree of depth 4

Heaps Definition

A **heap** is a certain kind of complete binary tree.

Heaps

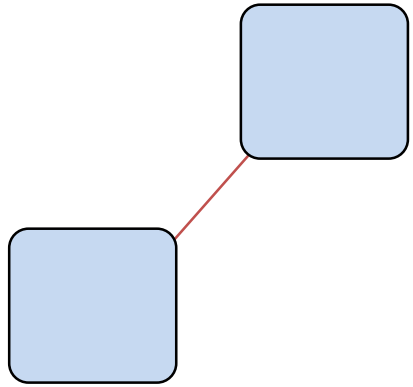


A **heap** is a certain kind of complete binary tree.

When a complete binary tree is built, its first node must be the root.

Heaps

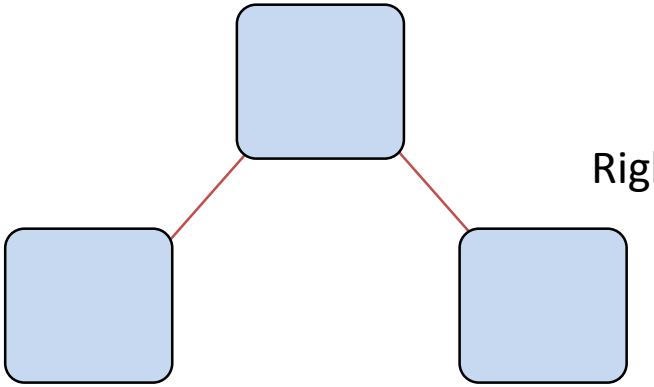
Complete binary tree.



The second node is always the left child of the root.

Heaps

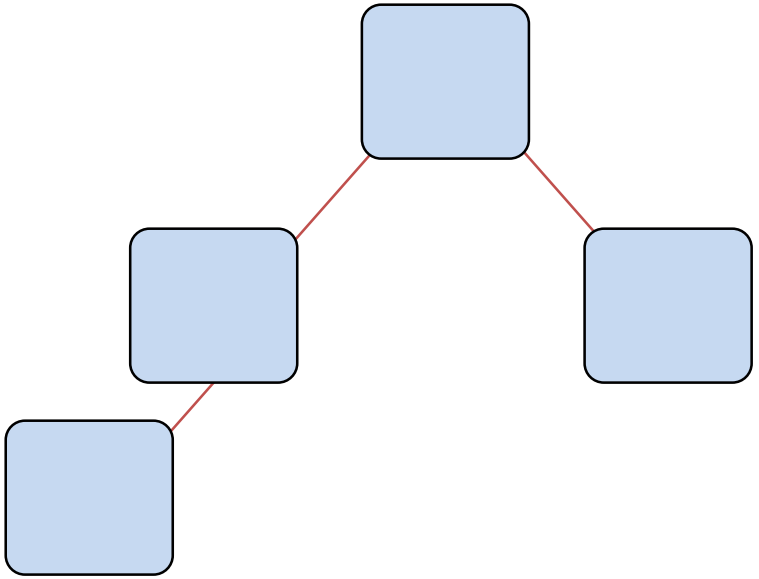
Complete binary tree.



The third node is always the right child of the root.

Heaps

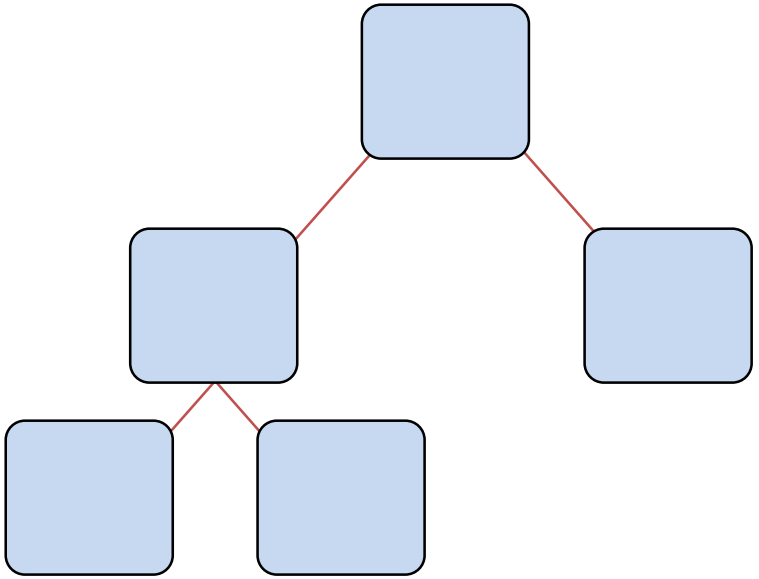
Complete binary tree.



The next nodes always fill the next level from left-to-right.

Heaps

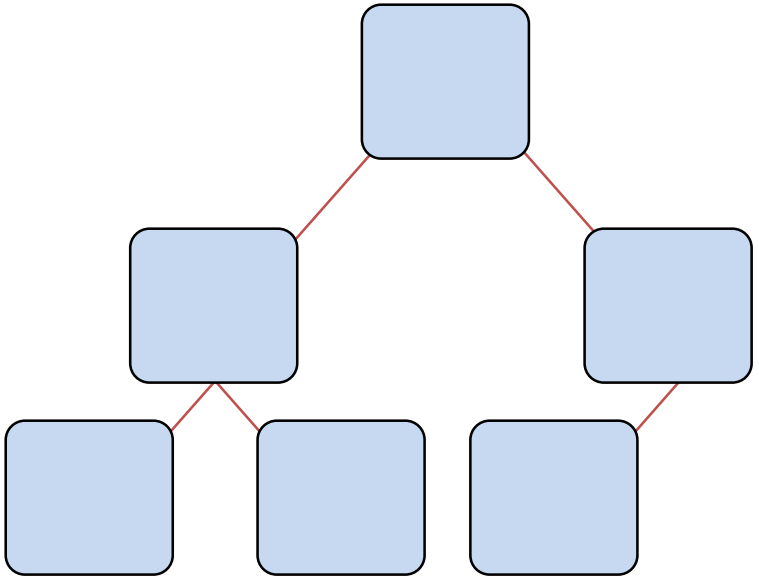
Complete binary tree.



The next nodes always fill the next level from left-to-right.

Heaps

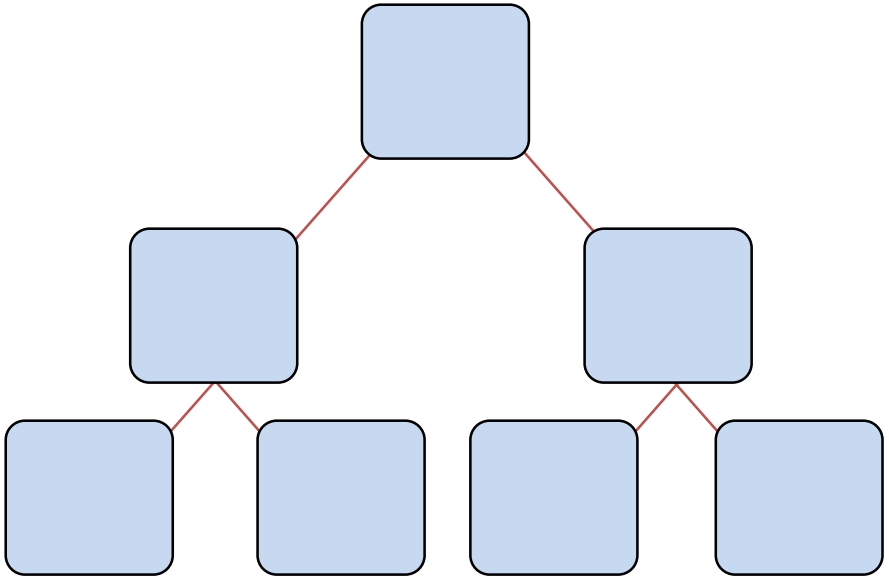
Complete binary tree.



The next nodes always fill the next level from left-to-right.

Heaps

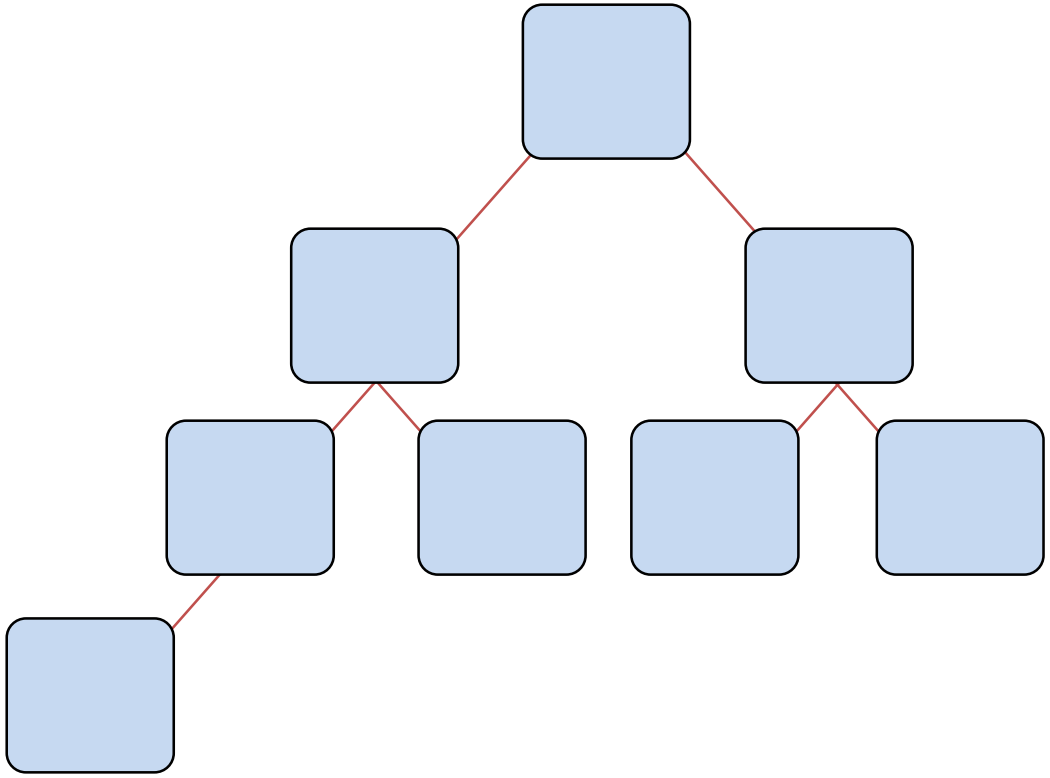
Complete binary tree.



The next nodes always fill the next level from left-to-right.

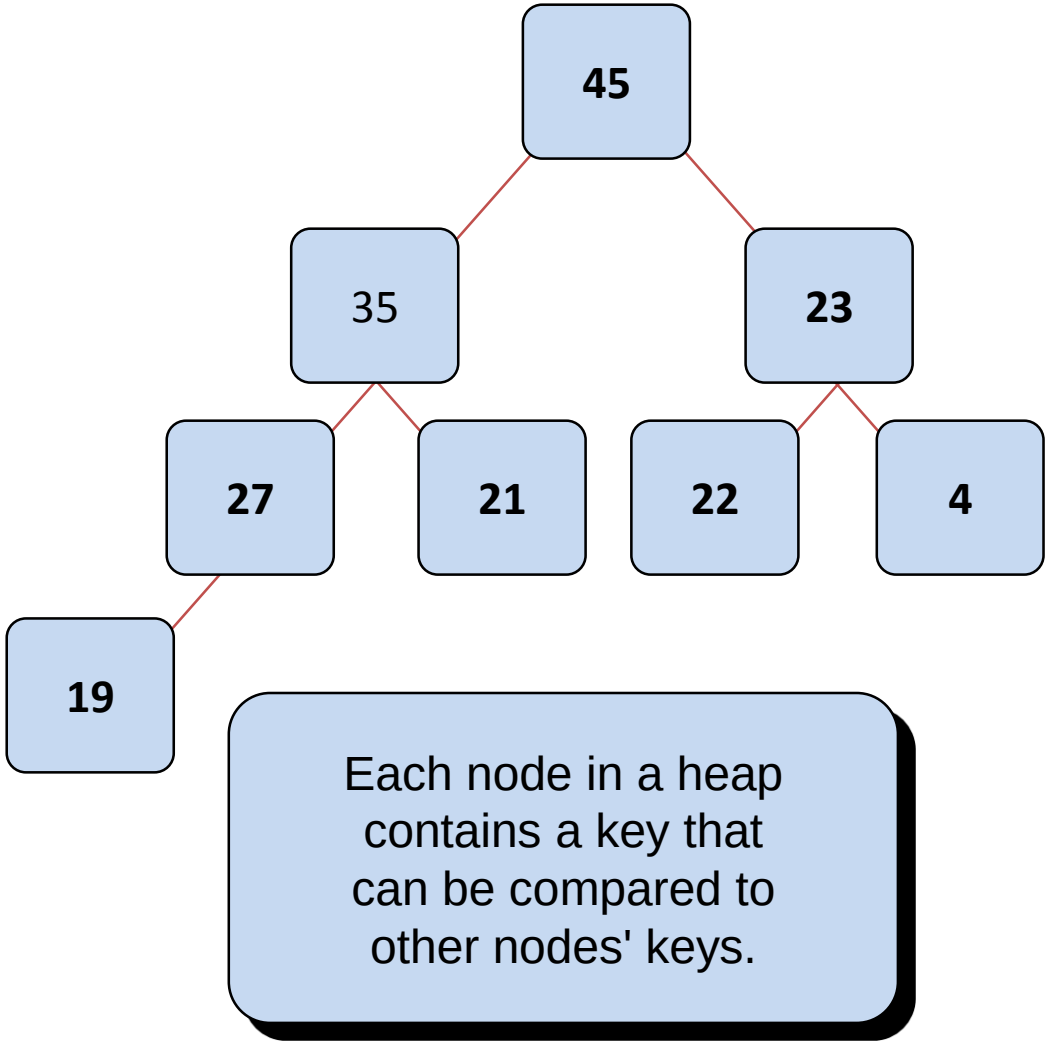
Heaps

Complete binary tree.



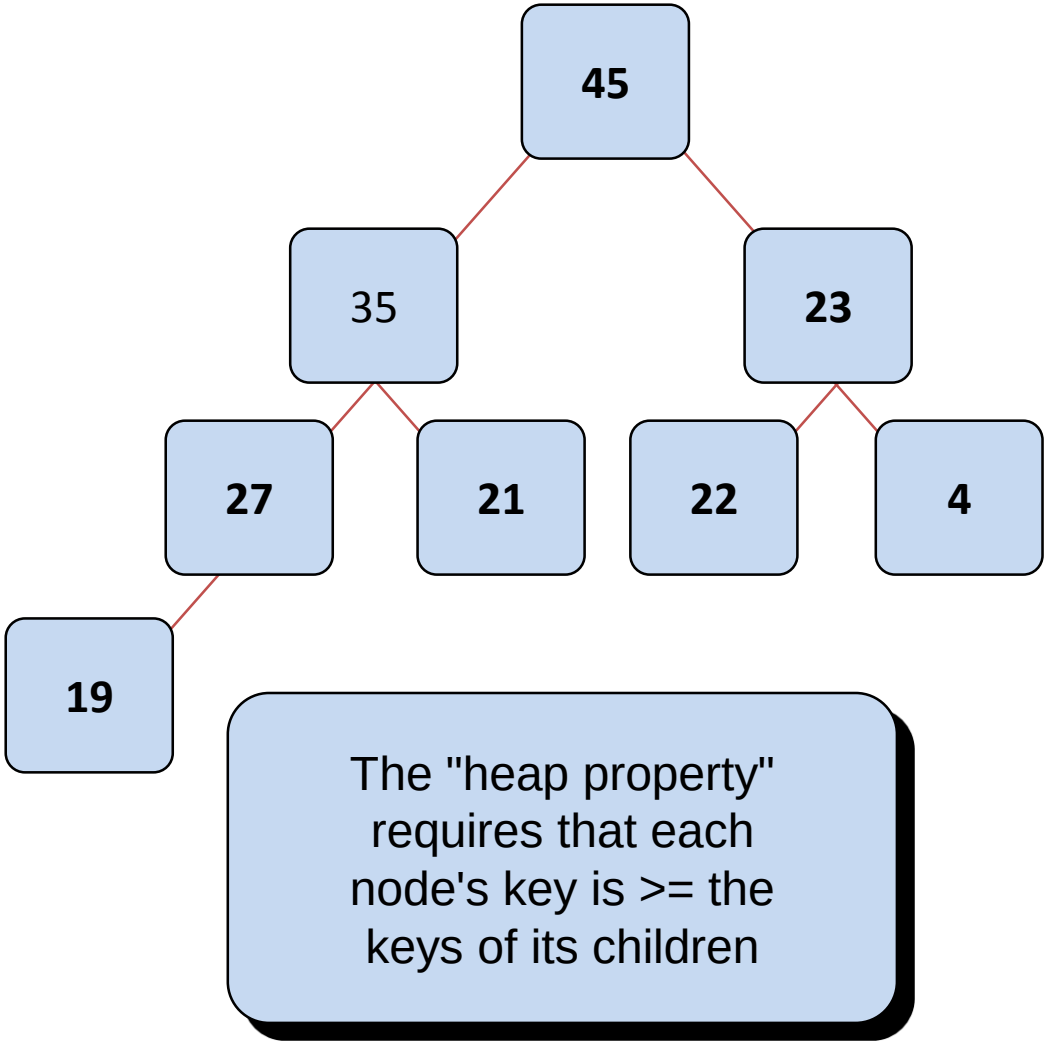
Heaps

A heap is a **certain** kind of complete binary tree.



Heaps

A heap is a **certain** kind of complete binary tree.



Application : Priority Queues

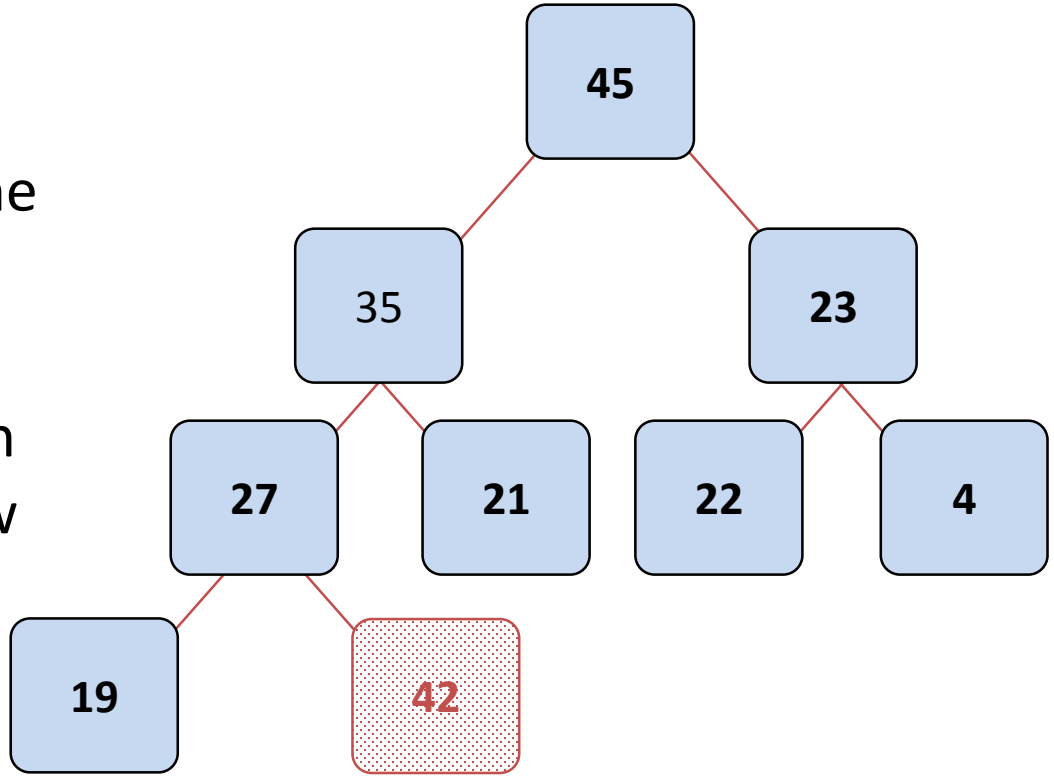
- A priority queue is a container class that allows entries to be retrieved according to some specific priority levels
  - The highest priority entry is removed first
  - If there are several entries with equally high priorities, then the priority queue's implementation determines which will come out first (e.g. FIFO)
- Heap is suitable for a priority queue

The Priority Queue ADT with Heaps

- The entry with the highest priority is always at the root node
- Focus on two priority queue operations
  - adding a new entry
  - remove the entry with the highest priority
- In both cases, we must ensure the tree structure remains to be a heap
  - we are going to work on a conceptual heap without worrying about the precise implementation
  - later I am going to show you how to implement...

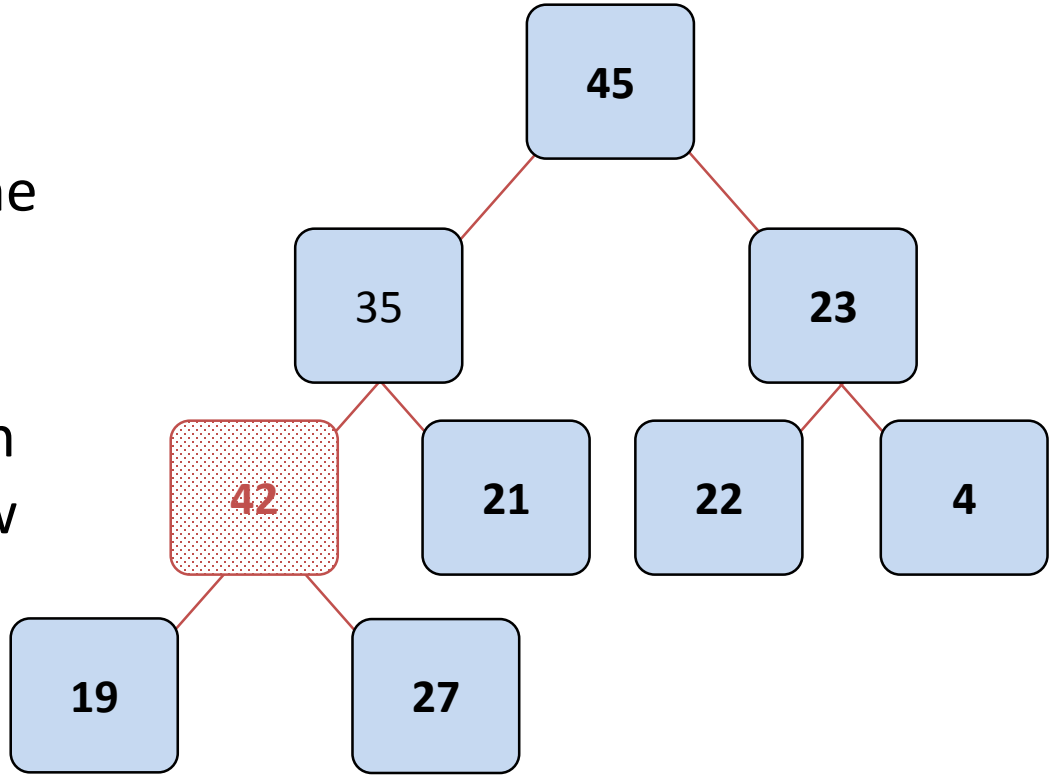
Adding a Node to a Heap

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



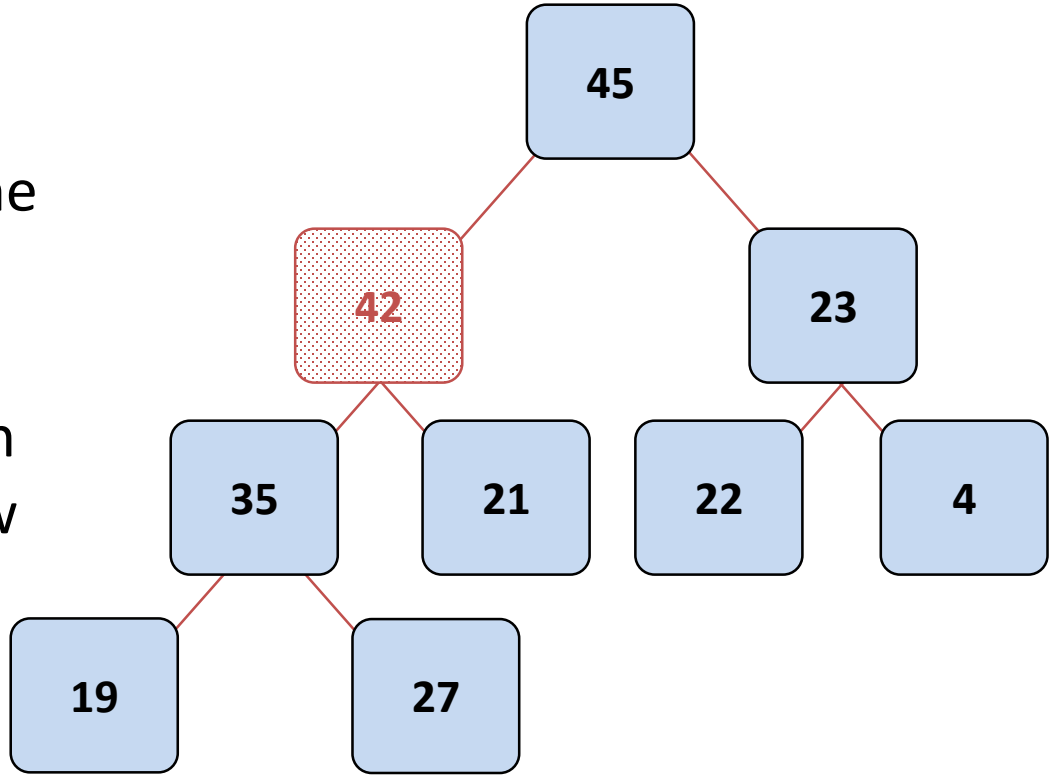
Adding a Node to a Heap

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



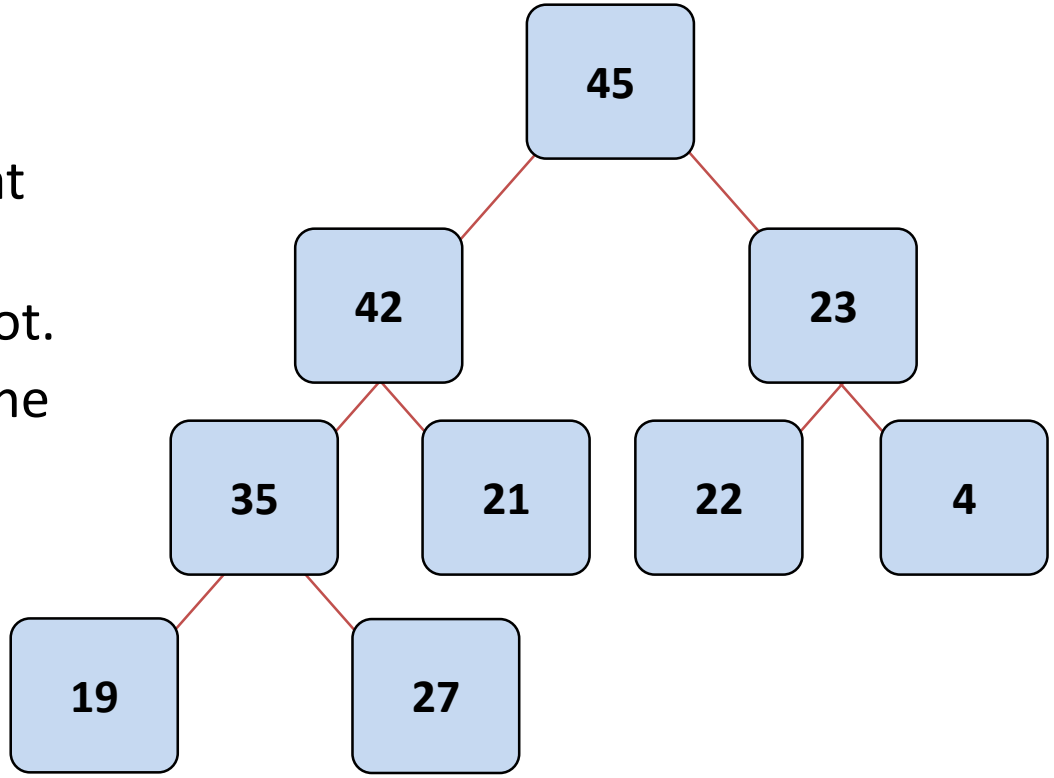
Adding a Node to a Heap

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



Adding a Node to a Heap

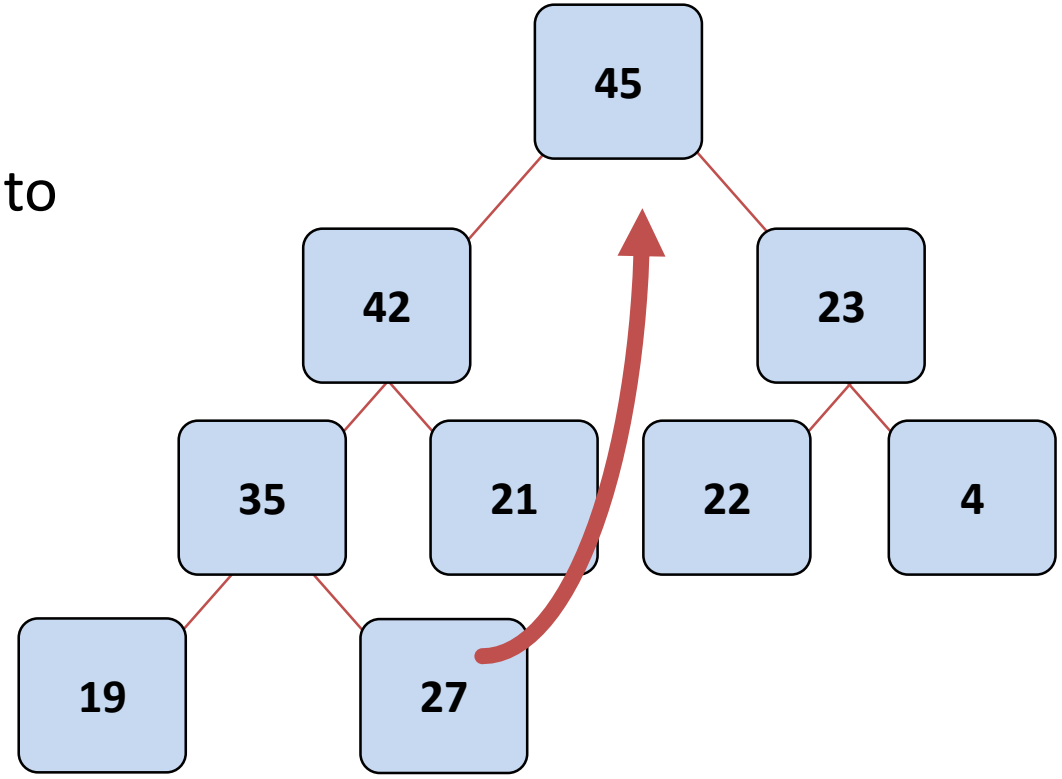
- The parent has a key that is >= new node, or
- The node reaches the root.
- The process of pushing the new node upward is called **reheapification upward**.



Note: Note: we need to easily go from child to parent as well as parent to child.

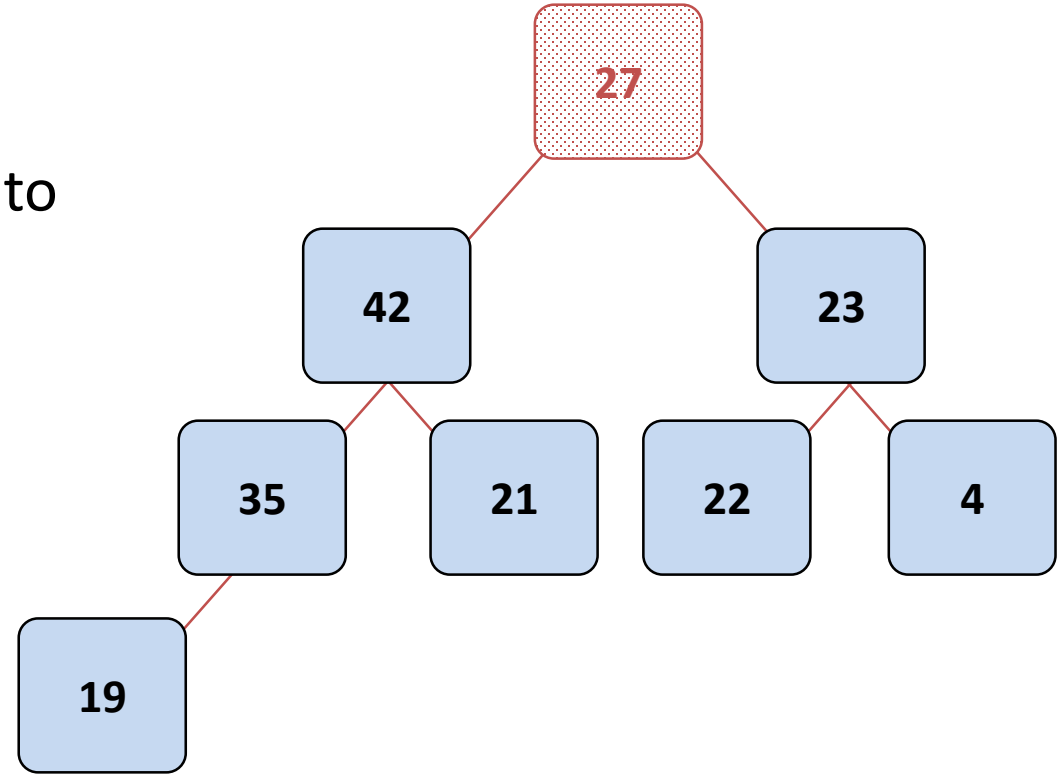
Removing the Top of a Heap

- ☆ Move the last node onto the root.



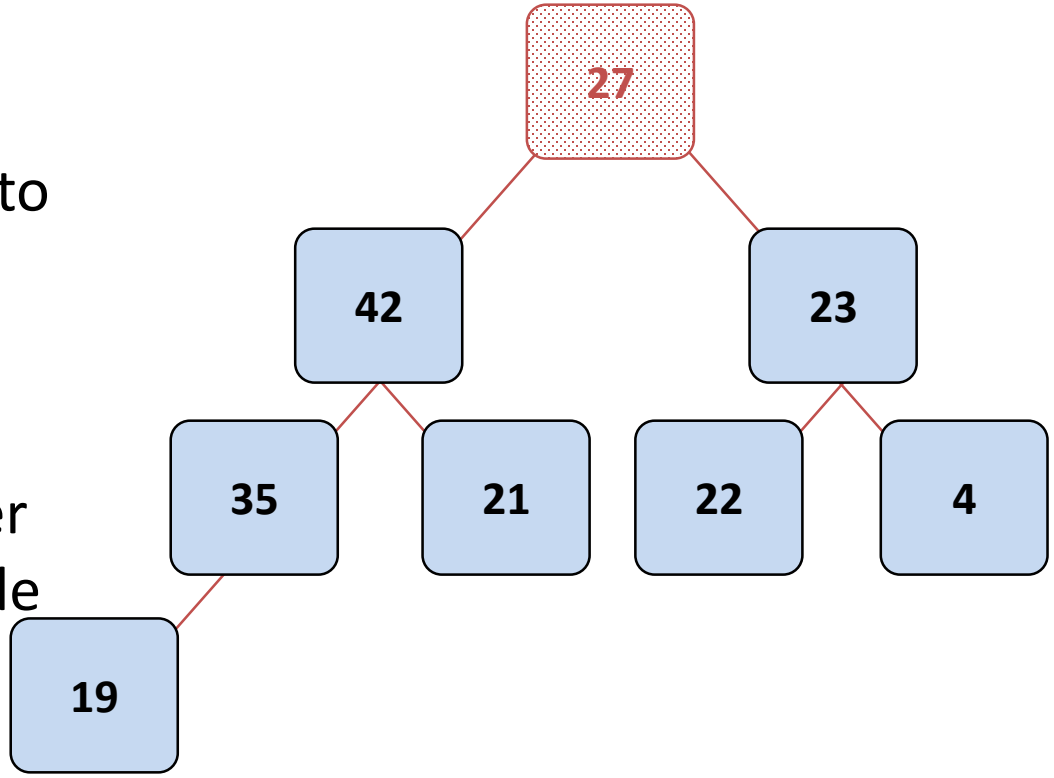
Removing the Top of a Heap

- ☆ Move the last node onto the root.



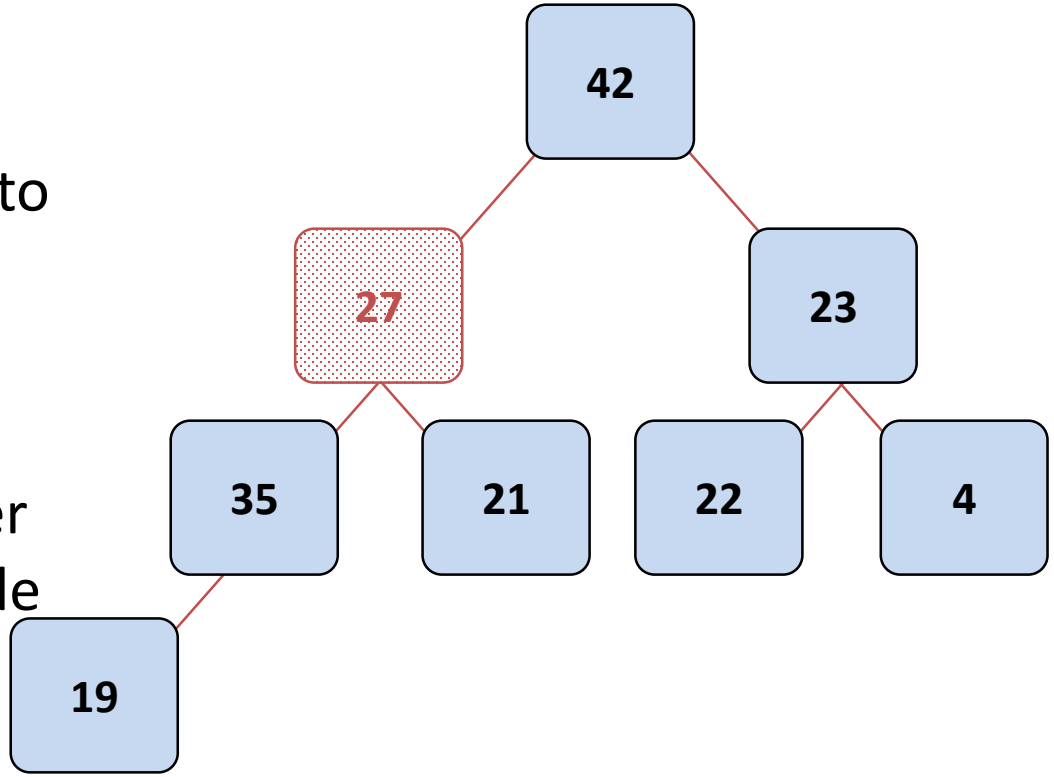
Removing the Top of a Heap

- ☆ Move the last node onto the root.
- 🕐 Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.



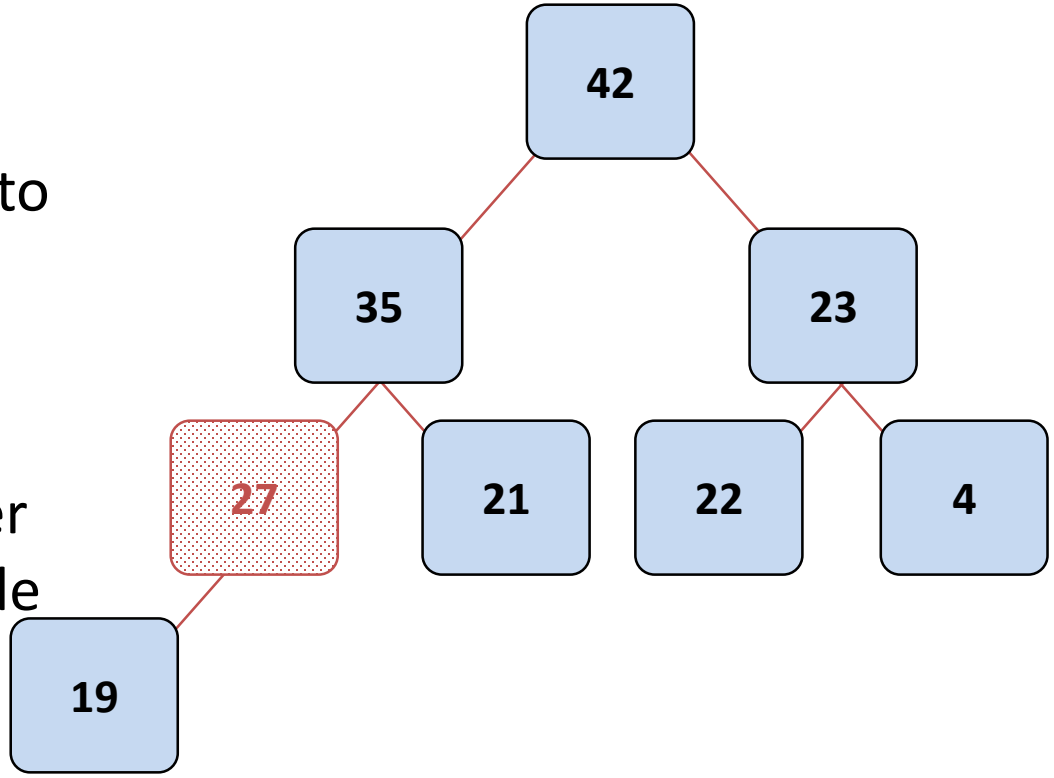
Removing the Top of a Heap

- ☆ Move the last node onto the root.
- 🕐 Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.



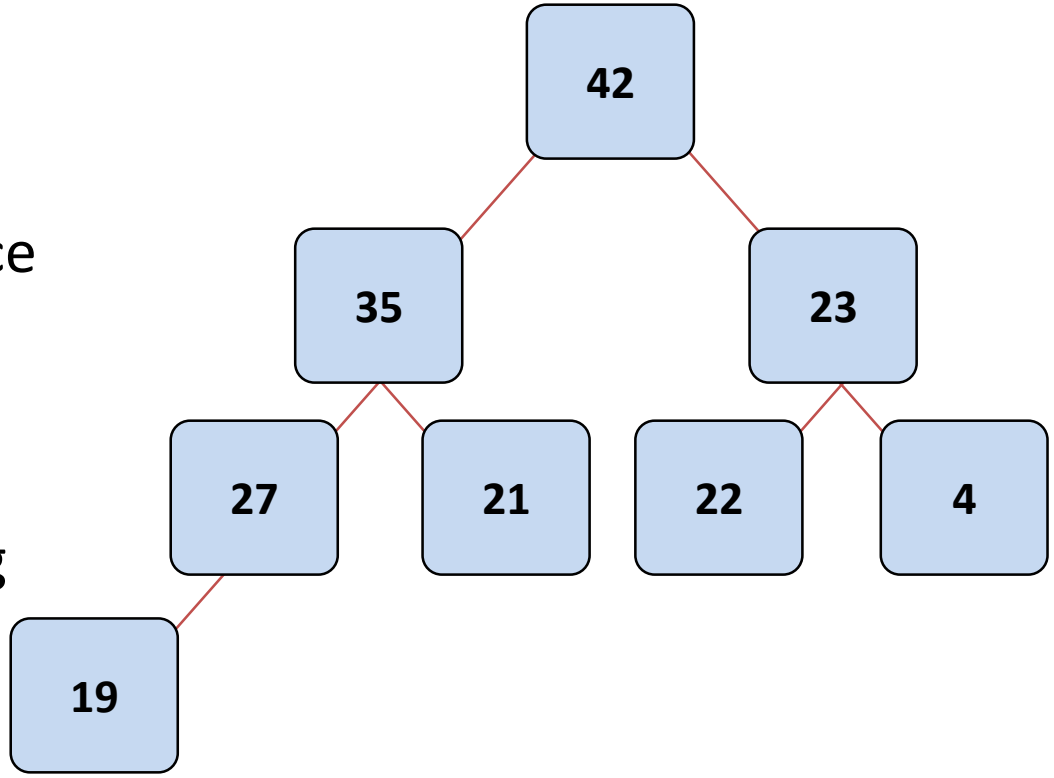
Removing the Top of a Heap

- ☆ Move the last node onto the root.
- 🕐 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 the leaf.
- The process of pushing the new node downward is called **reheapification downward**.

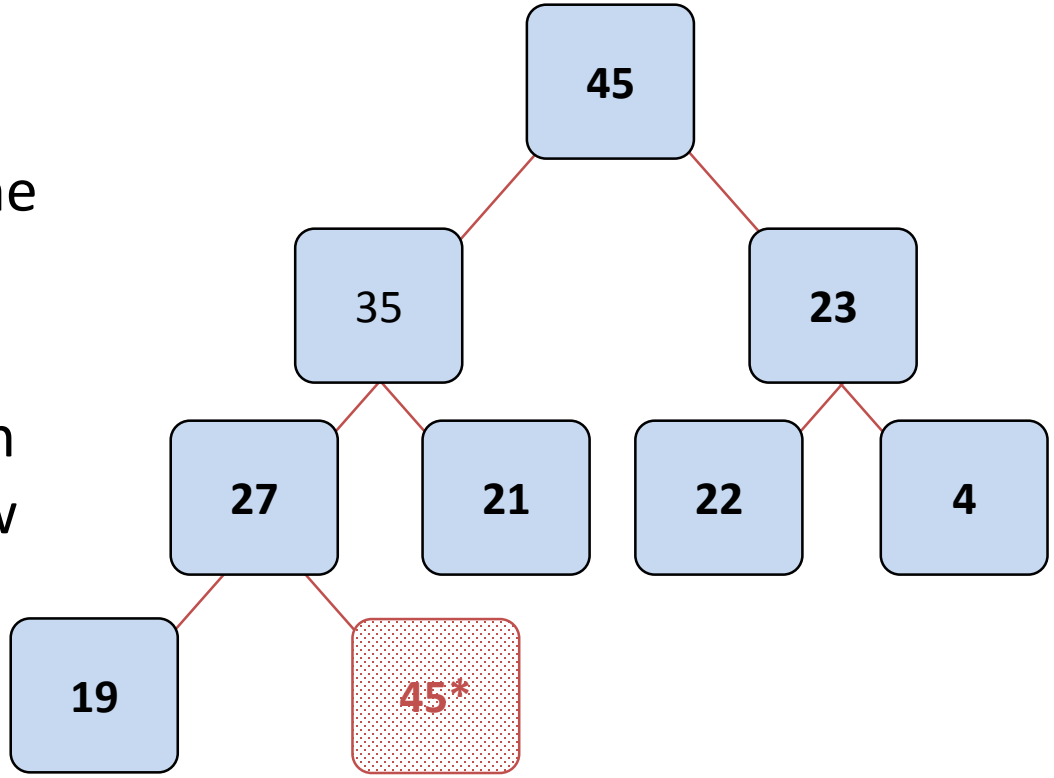


Priority Queues Revisited

- A priority queue is a container class that allows entries to be retrieved according to some specific priority levels
  - The highest priority entry is removed first
  - **If there are several entries with equally high priorities, then the priority queue's implementation determines which will come out first (e.g. FIFO)**
- Heap is suitable for a priority queue

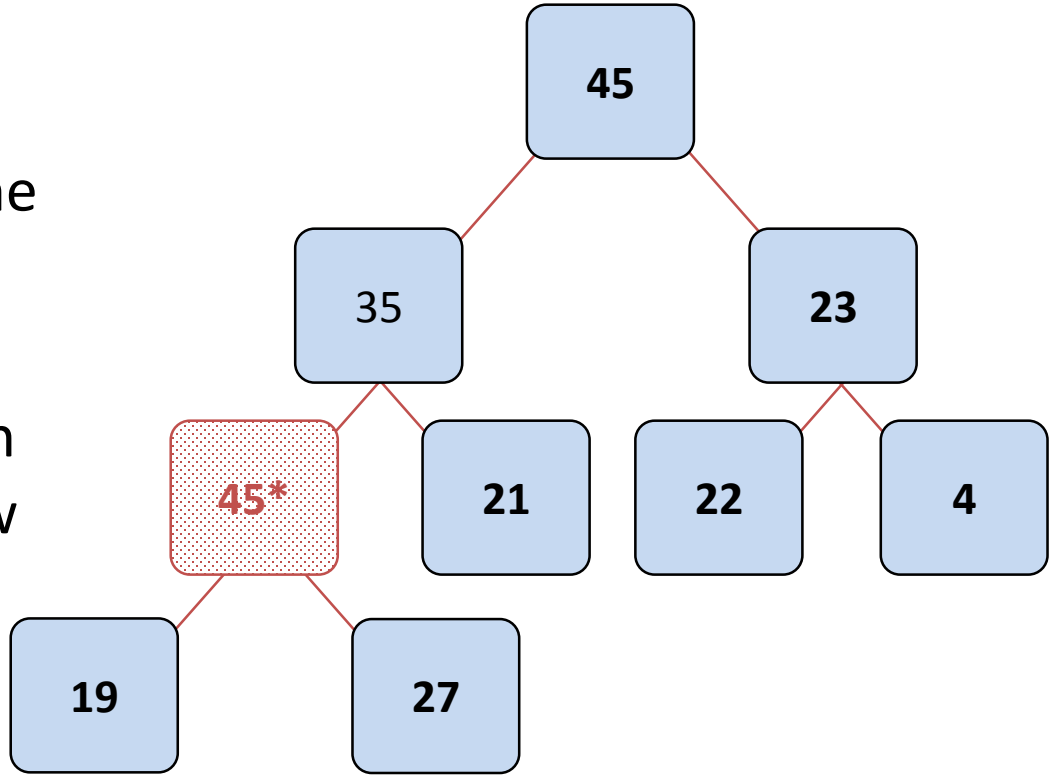
Adding a Node: same priority

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



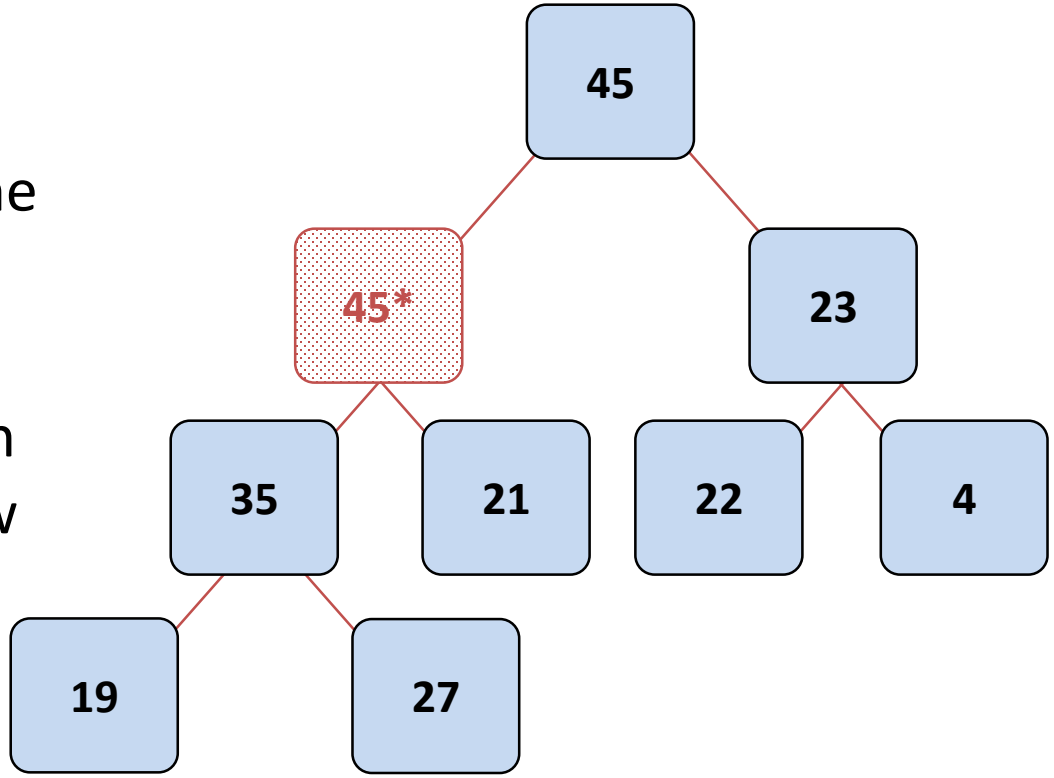
Adding a Node : same priority

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



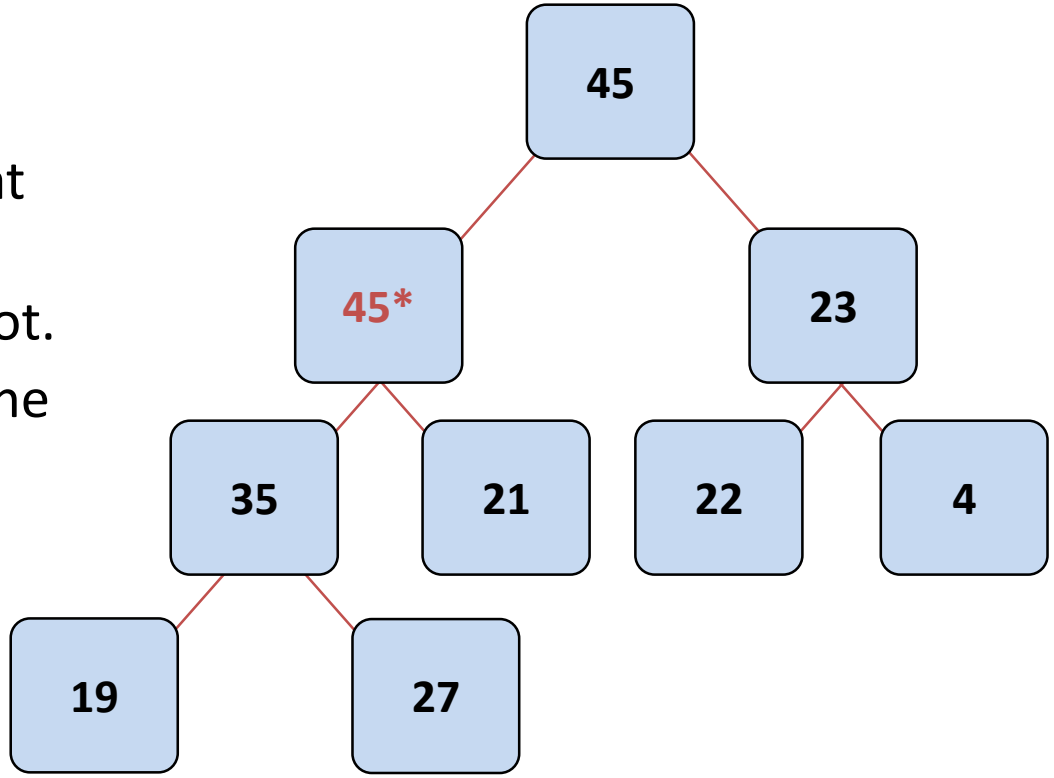
Adding a Node : same priority

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



Adding a Node : same priority

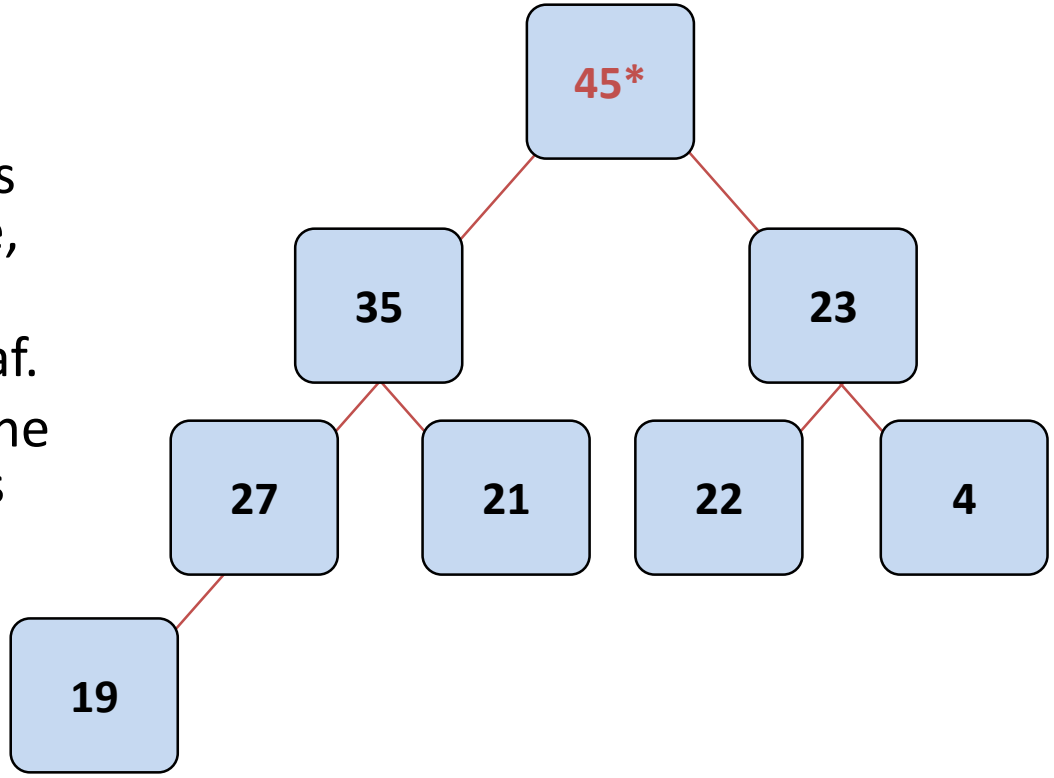
- The parent has a key that is >= new node, or
- The node reaches the root.
- The process of pushing the new node upward is called **reheapification upward**.



Note: Implementation determines which 45 will be in the root, and will come out first when popping.

Removing the Top of a Heap

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



Note: Implementation determines which 45 will be in the root, and will come out first when popping.

Heap Implementation

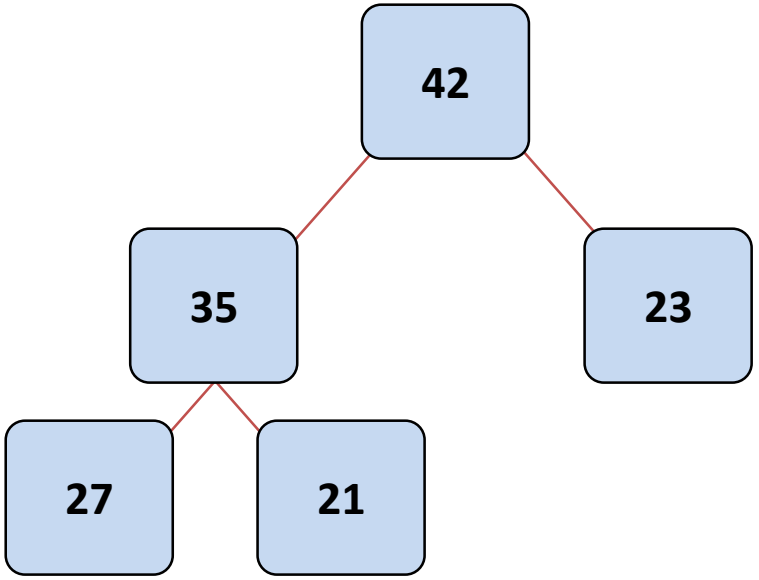
- Use tree structure
  - node implementation is for a general binary tree
  - need to have doubly linked node
- Use arrays
  - A heap is a complete binary tree
  - which can be implemented more easily with an array than with the node class
  - and do two-way links

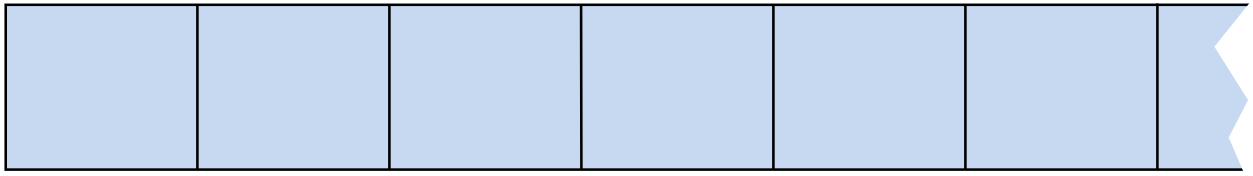
Formulas for location children and parents in an array representation

- Root at location [0]
- Parent of the node in [i] is at [(i-1)/2]
- Children of the node in [i] (if exist) is at [2i+1] and [2i+2]
- Test:
  - complete tree of 10, 000 nodes
  - parent of 4999 is at (4999-1)/2 = 2499
  - children of 4999 is at 9999 (V) and 10,000 (X)

Implementing a Heap

- We will store the data from the nodes in a partially-filled array.

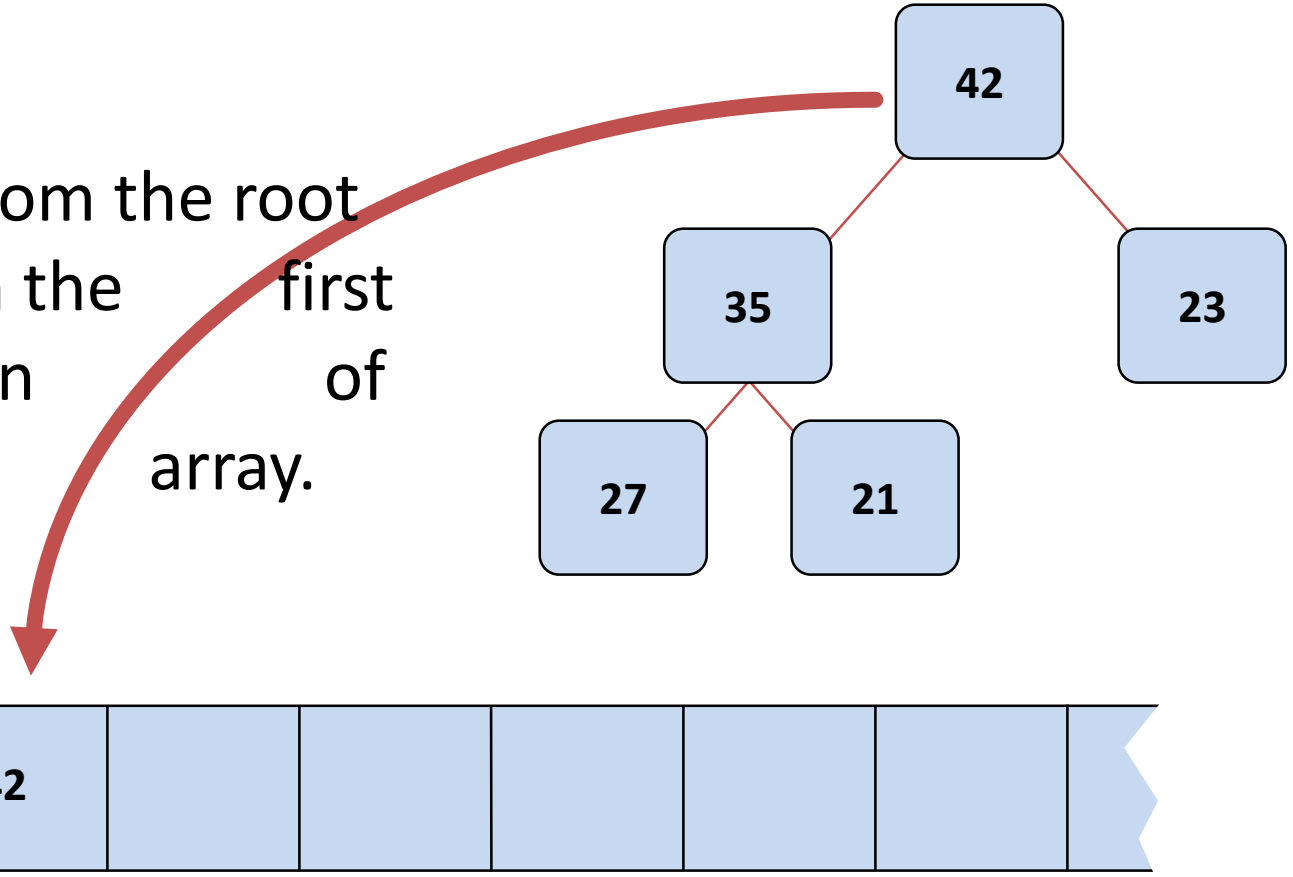




An array of data

Implementing a Heap

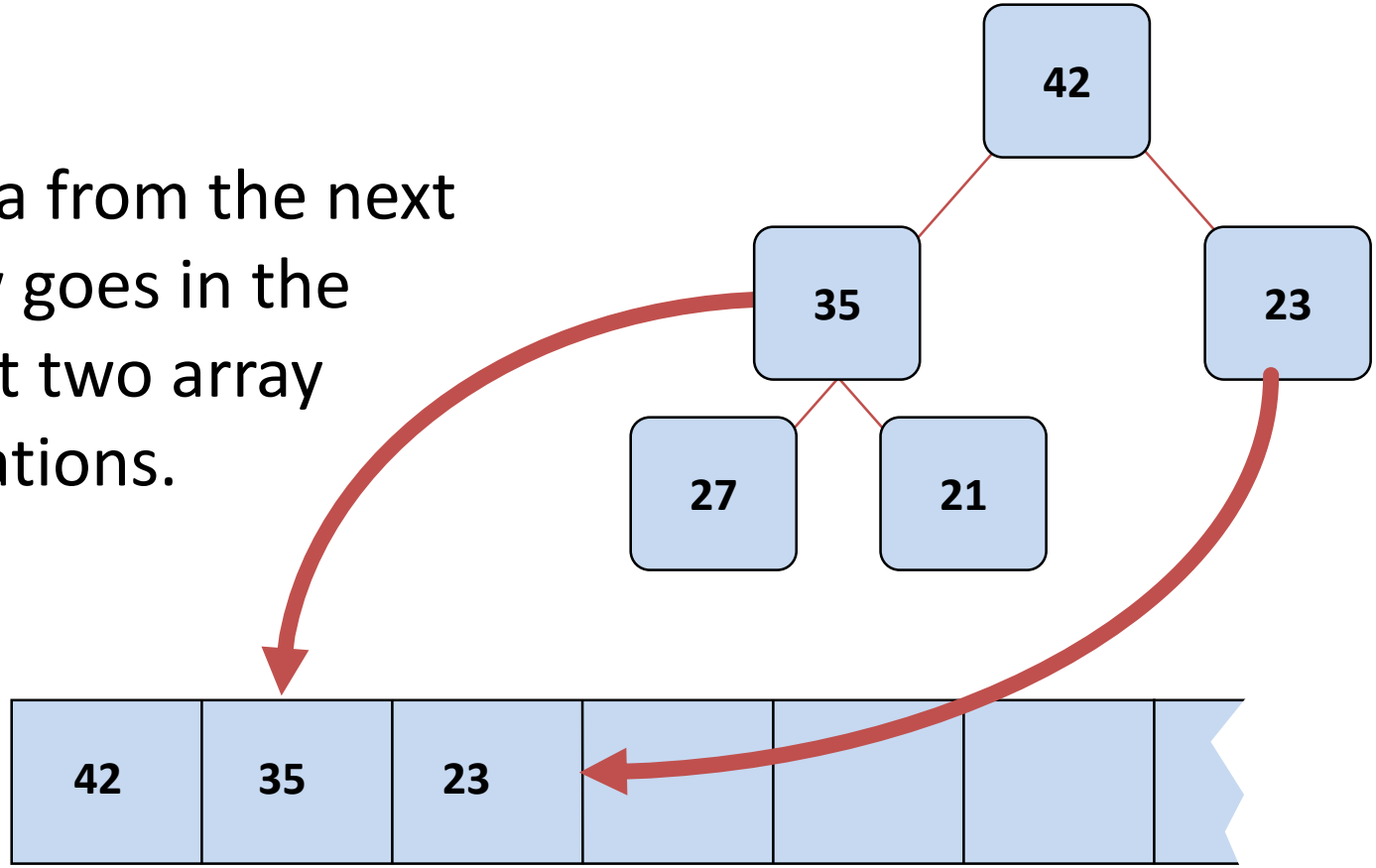
- Data from the root goes in the first location of the array.



An array of data

Implementing a Heap

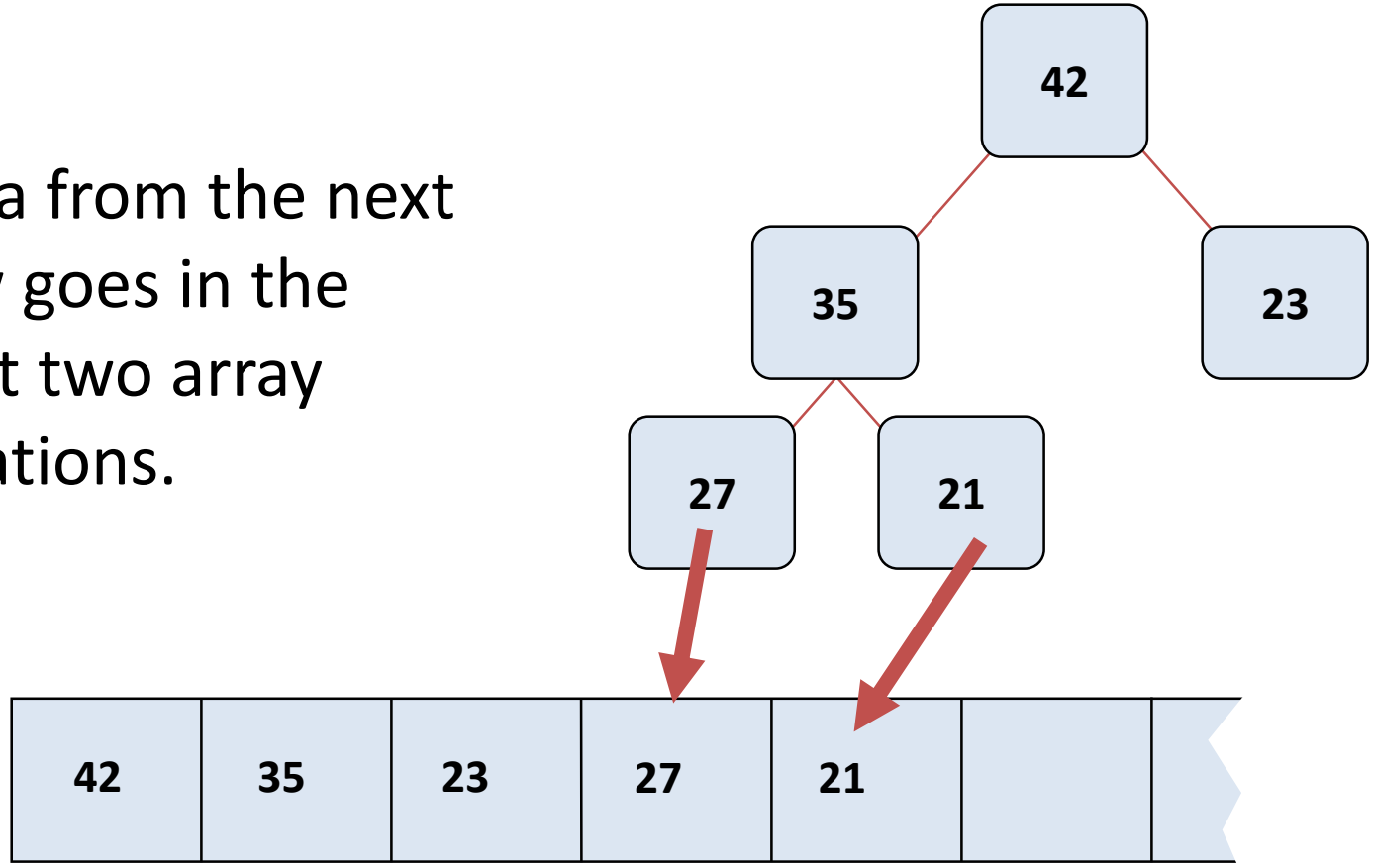
- Data from the next row goes in the next two array locations.



An array of data

Implementing a Heap

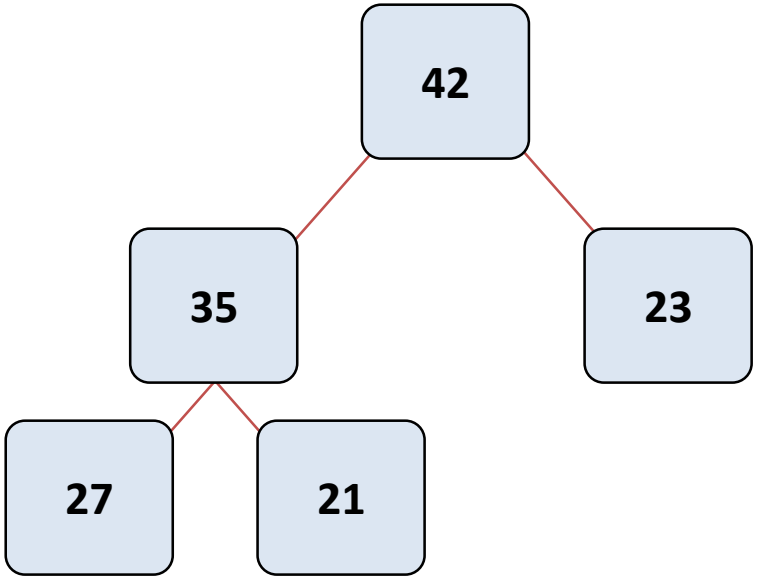
- Data from the next row goes in the next two array locations.

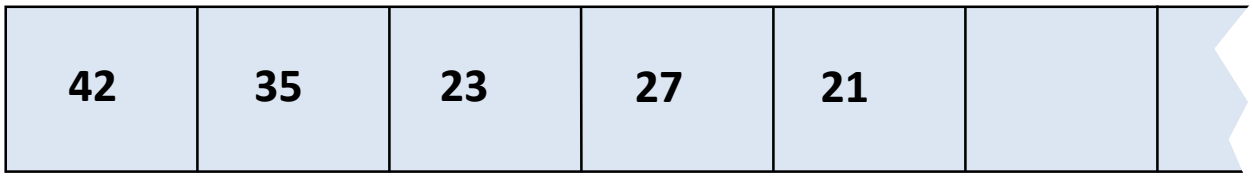


An array of data

Implementing a Heap

- Data from the next row goes in the next two array locations.



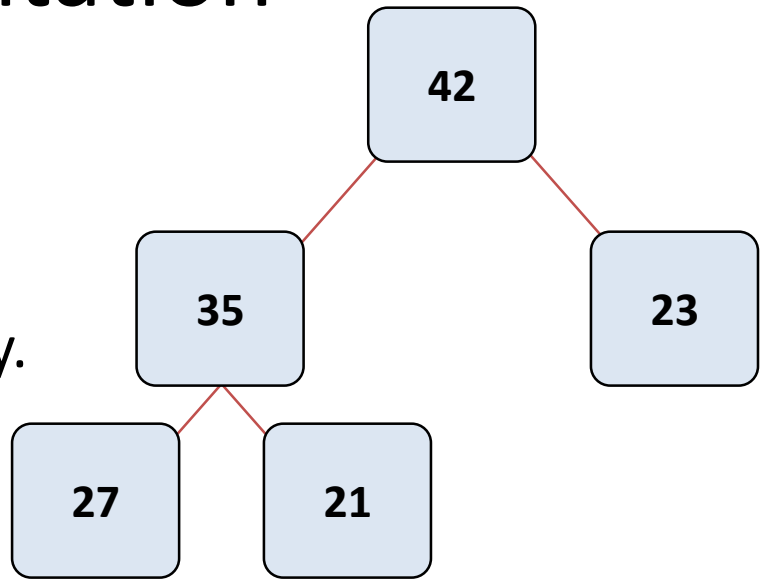


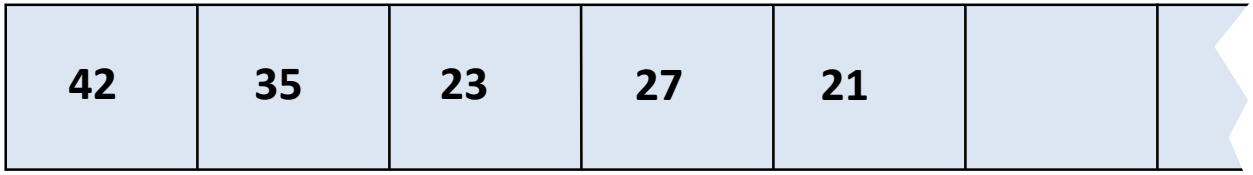
An array of data

We don't care what's in this part of the array.

Important Points about the Implementation

- The links between the tree's nodes are **not** actually stored as pointers, or in any other way.
- The only way we "know" that "the array is a tree" is from the way we manipulate the data.

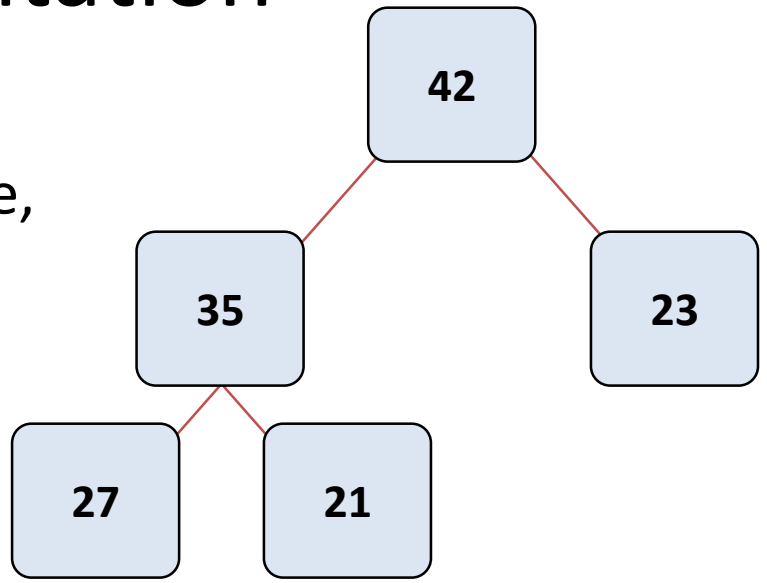


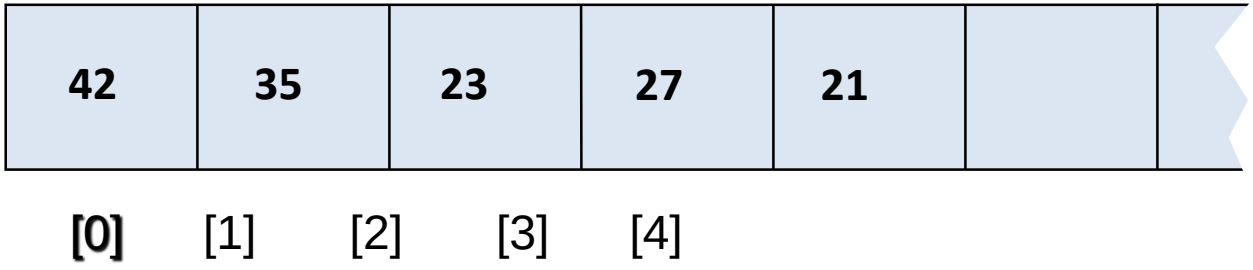


An array of data

Important Points about the Implementation

- If you know the index of a node, then it is easy to figure out the indexes of that node's parent and children.





Formulas for location children and parents in an array representation

- Root at location [0]
- Parent of the node in [i] is at [(i-1)/2]
- Children of the node in [i] (if exist) is at [2i+1] and [2i+2]
- Test:
  - complete tree of 10, 000 nodes
  - parent of 4999 is at (4999-1)/2 = 2499
  - children of 4999 is at 9999 (V) and 10,000 (X)

Summary

- A heap is a complete binary tree, where the entry at each node is greater than or equal to the entries in its children.
- To add an entry to a heap, place the new entry at the next available spot, and perform a reheapification upward.
- To remove the biggest entry, move the last node onto the root, and perform a reheapification downward.