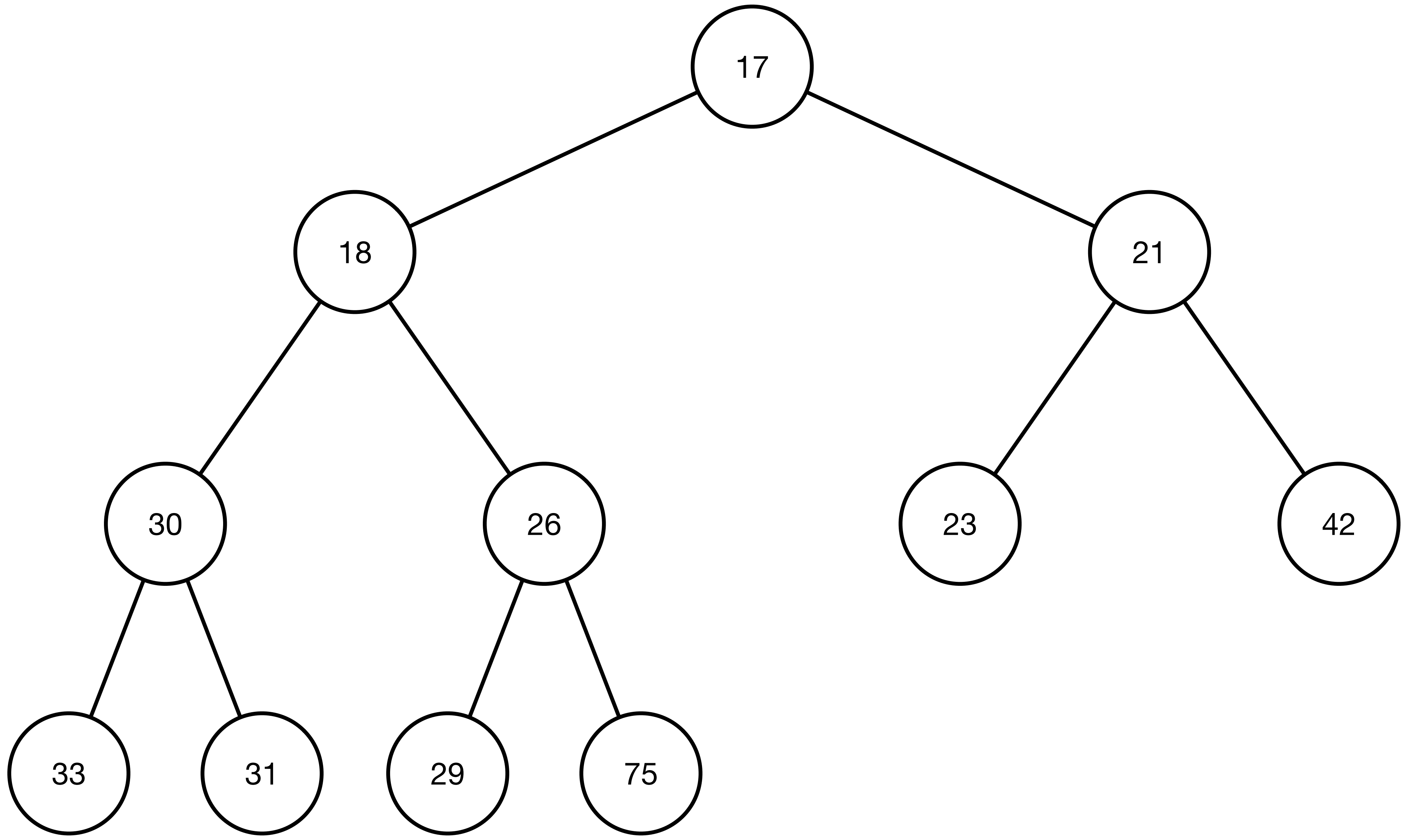
THE UNIVERSITY OF VERMONT
COLLEGE OF ENGINEERING &
MATHEMATICAL SCIENCES

BINARY HEAP

AN INTRODUCTION



CS 124 / Department of Computer Science

Motivation

A *binary heap* is a widely used data structure.

- Heapsort algorithm (Williams 1964)
- Graph algorithms (*e.g.*, shortest path, spanning tree)
- Priority queue (which itself has abundant applications)

What is a binary heap?

A *binary heap* is a binary tree with *heap properties*:

- The tree is *complete*. This means that each level is full with the possible exception of the last level, which may be incomplete, but should be filled from left to right. This is called the *structure property* (sometimes called the *shape property*).

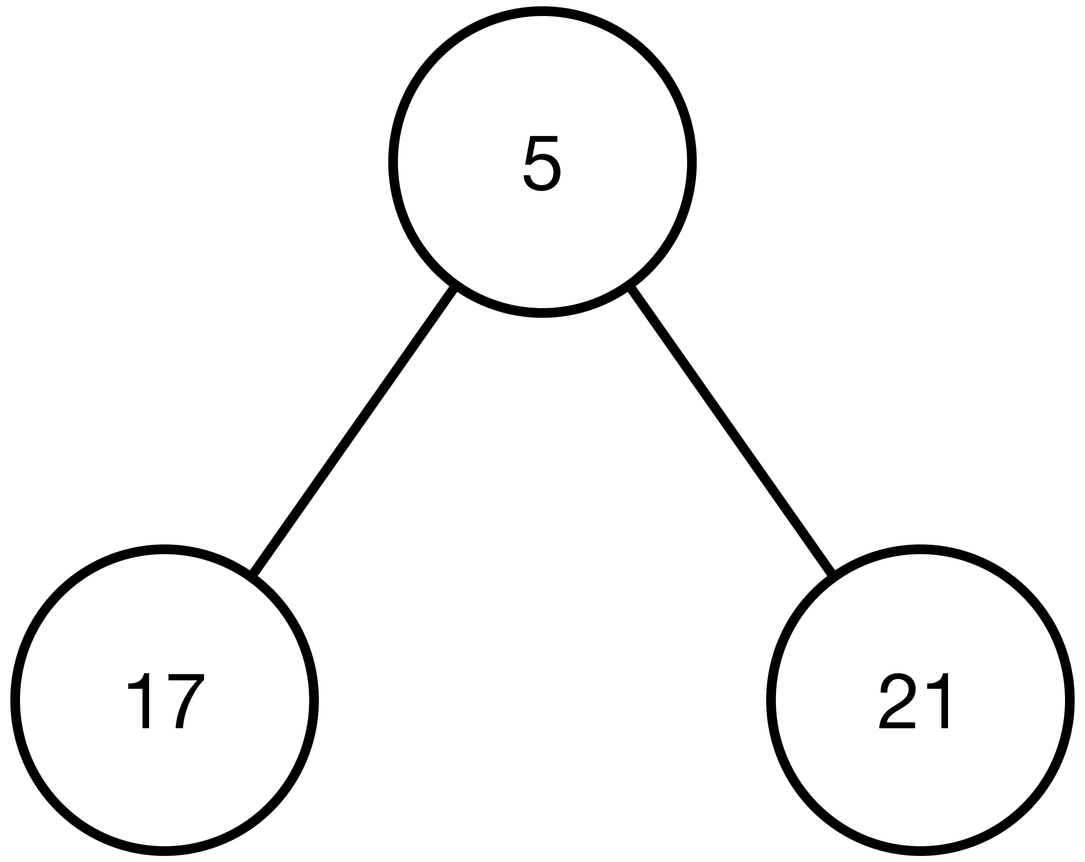
- With the exception of the root, every node must be ordered with respect to its parent. This is called the *heap order property* (or simply the *heap property*).

That's it!

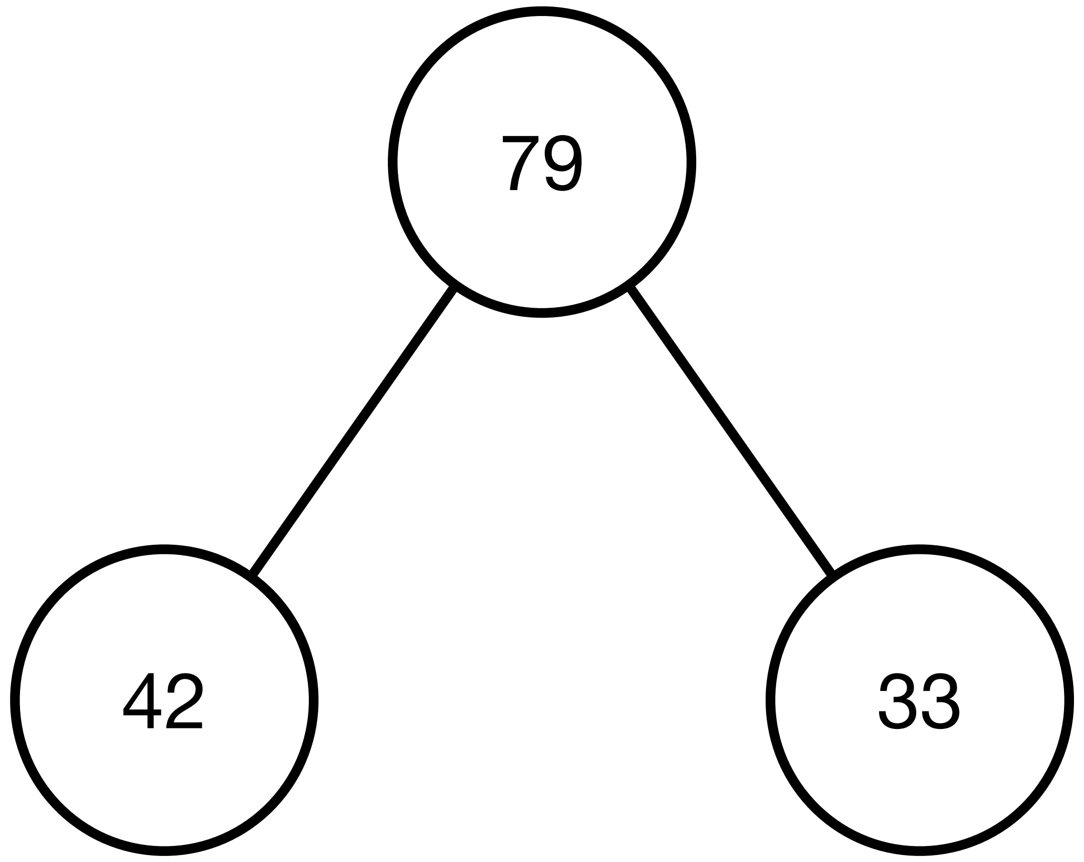
Heap order property

When implementing a binary heap, we can choose one of two orderings — but once we choose, we *must remain consistent*.

- *Minimal value* is at the root, and child nodes must have values *greater than or equal to* that of the parent node.



- *Maximal value* is at the root, and child nodes must have values *less than or equal to* that of the parent node.



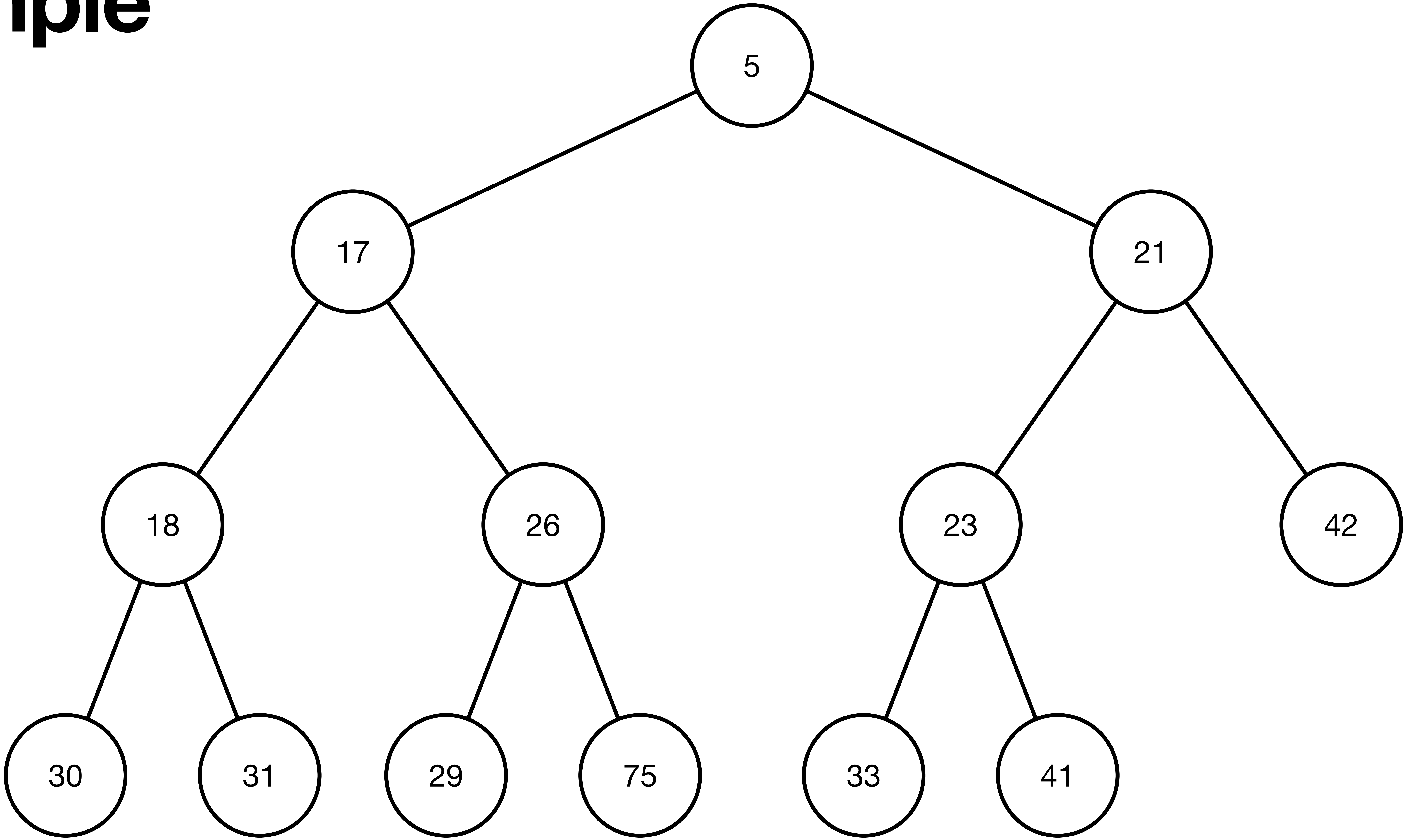
Heap order property

It is important to keep in mind the distinction between a binary heap and a binary search tree (BST). They are *not* the same thing.

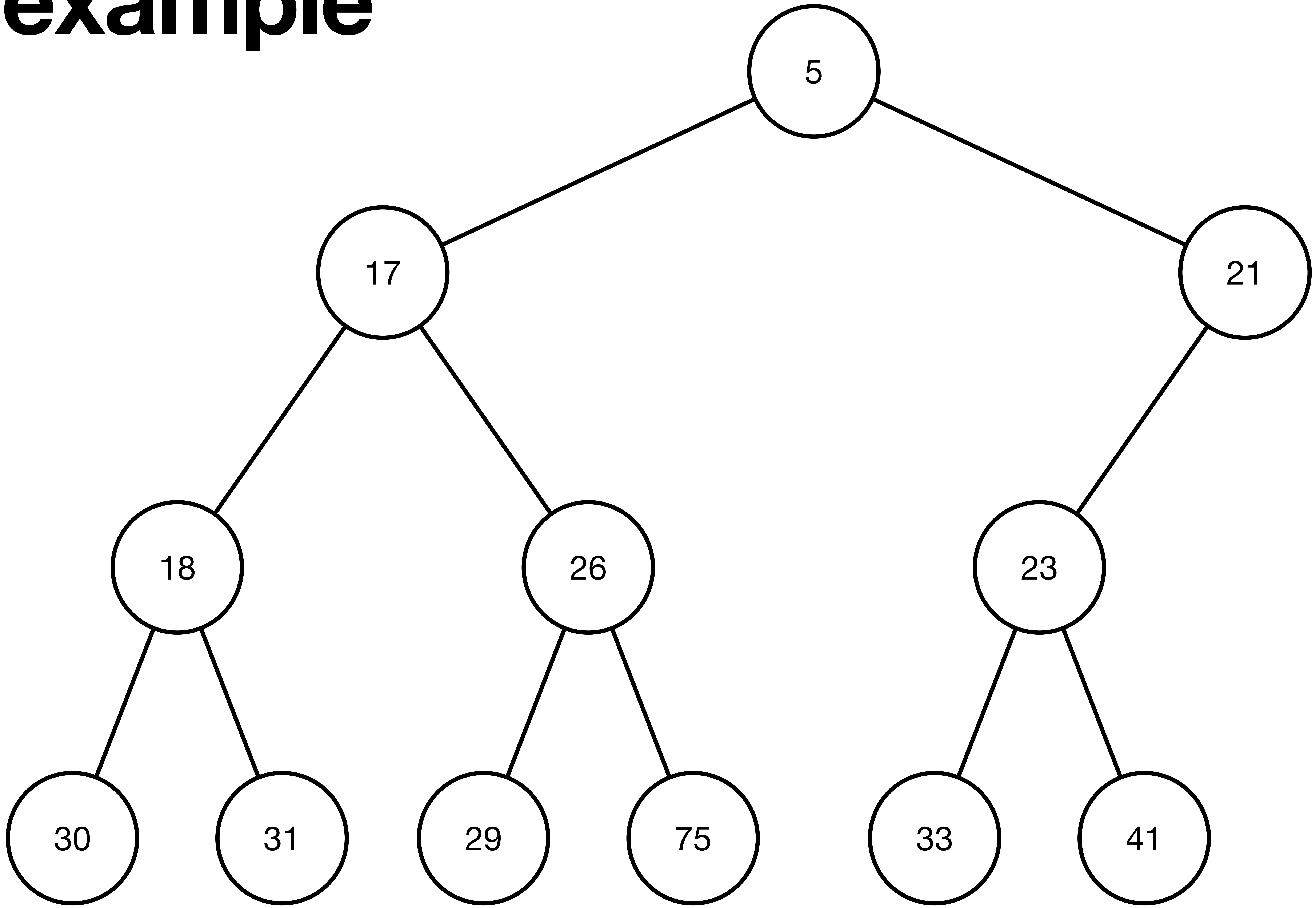
A binary search tree requires that values of a left subtree must be ordered with respect to the right subtree. This is *not* the case with binary heaps. The heap order property is a little more relaxed.

Binary heap ≠ binary search tree

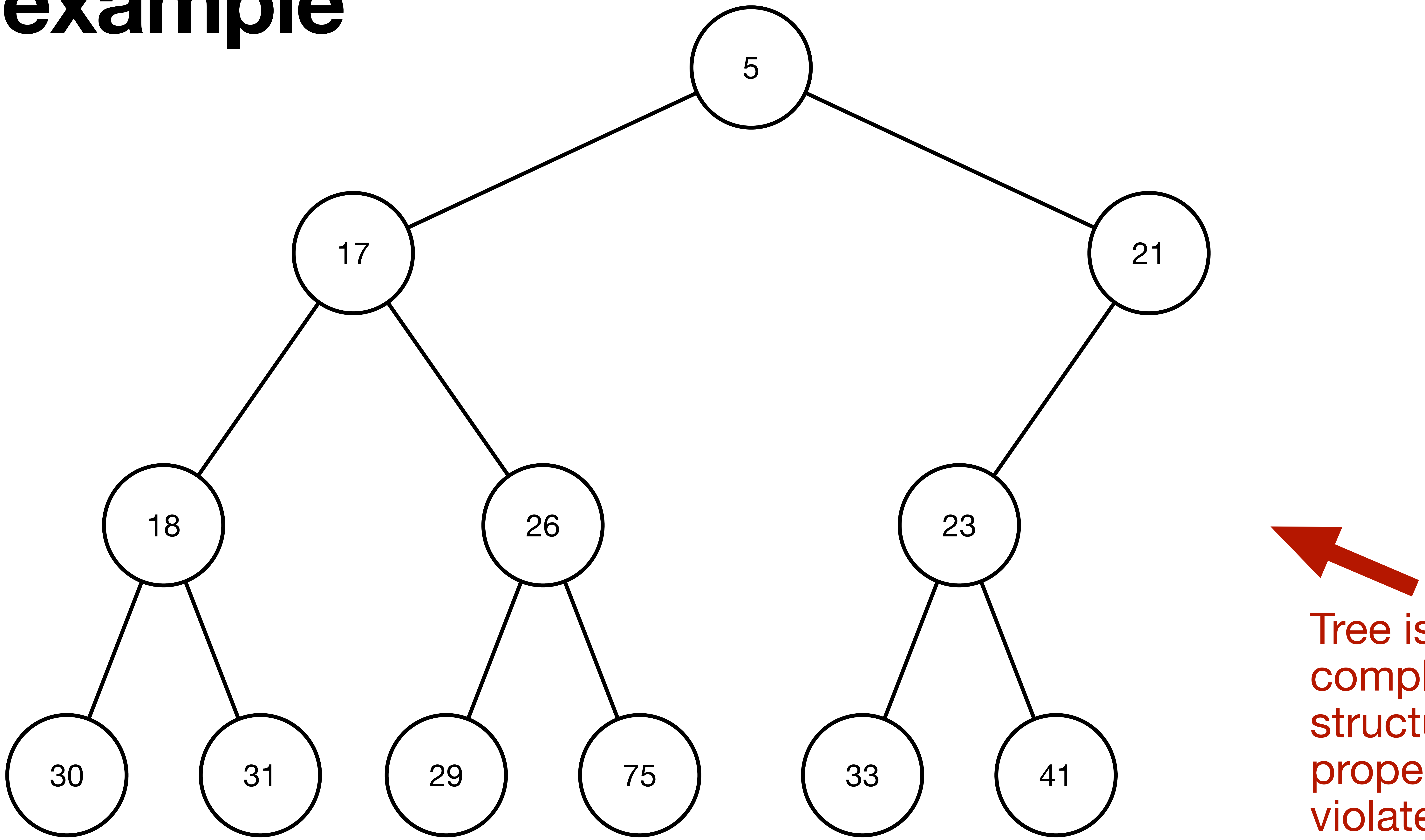
Example



Non-example

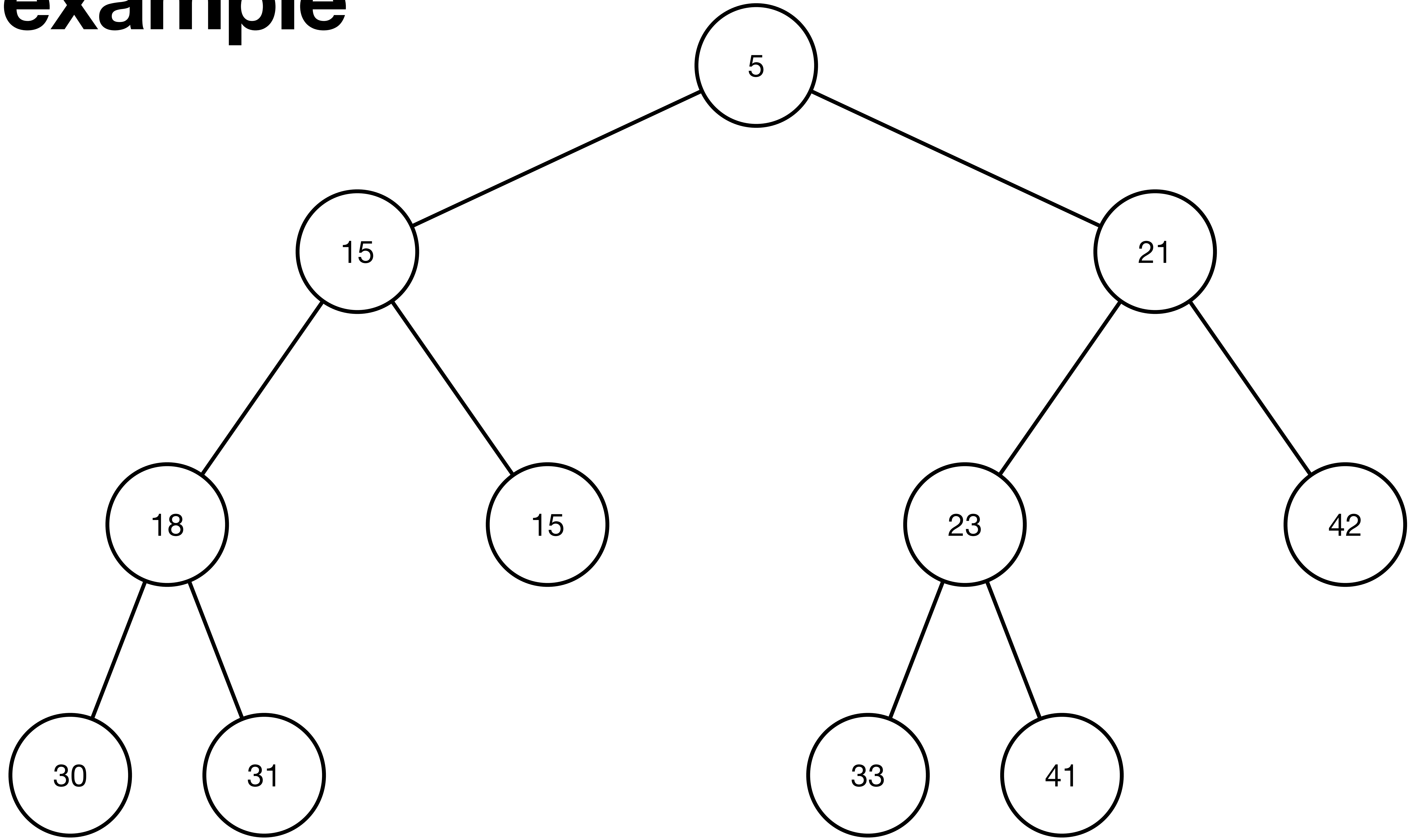


Non-example

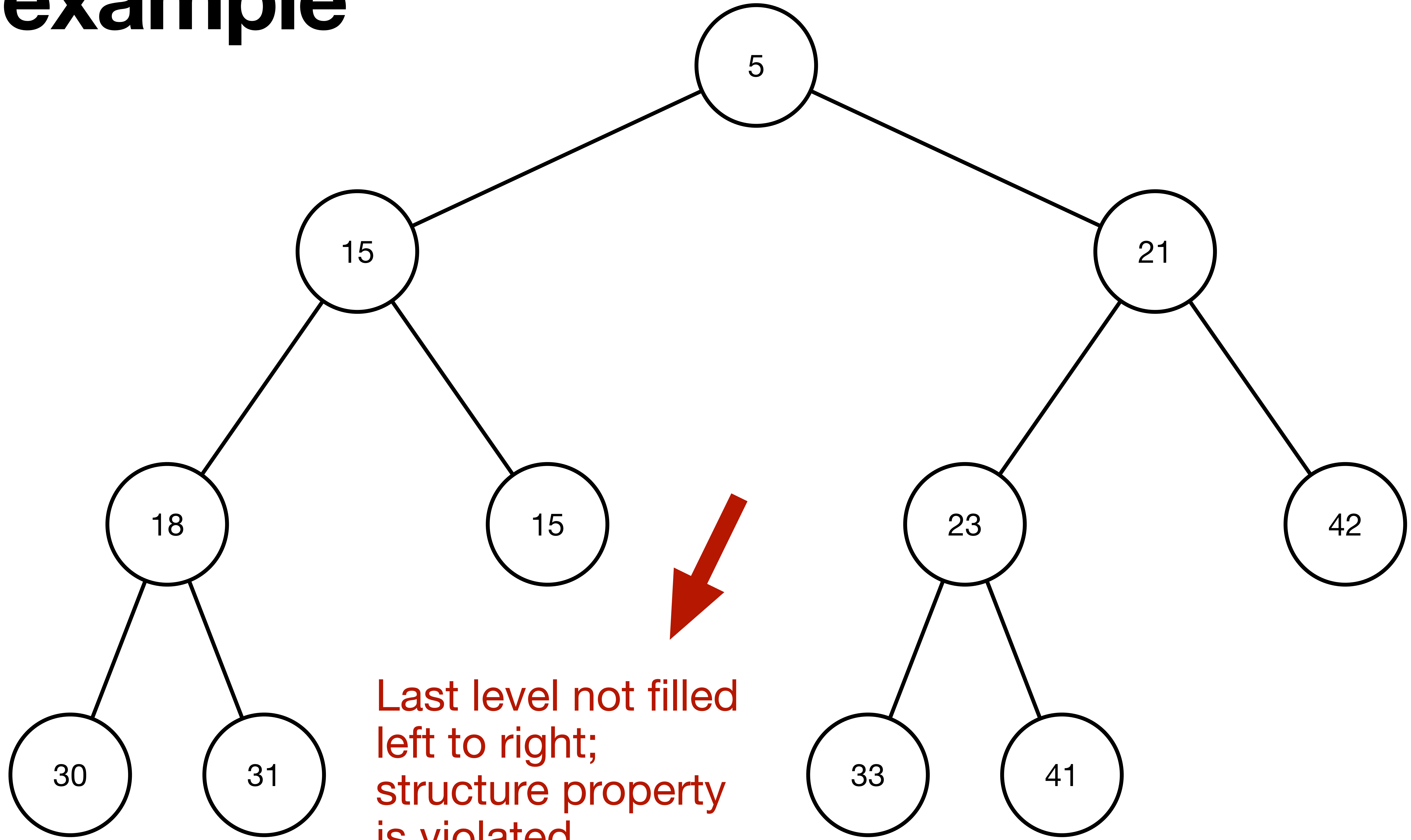


Tree is not complete; structure property is violated.

Non-example

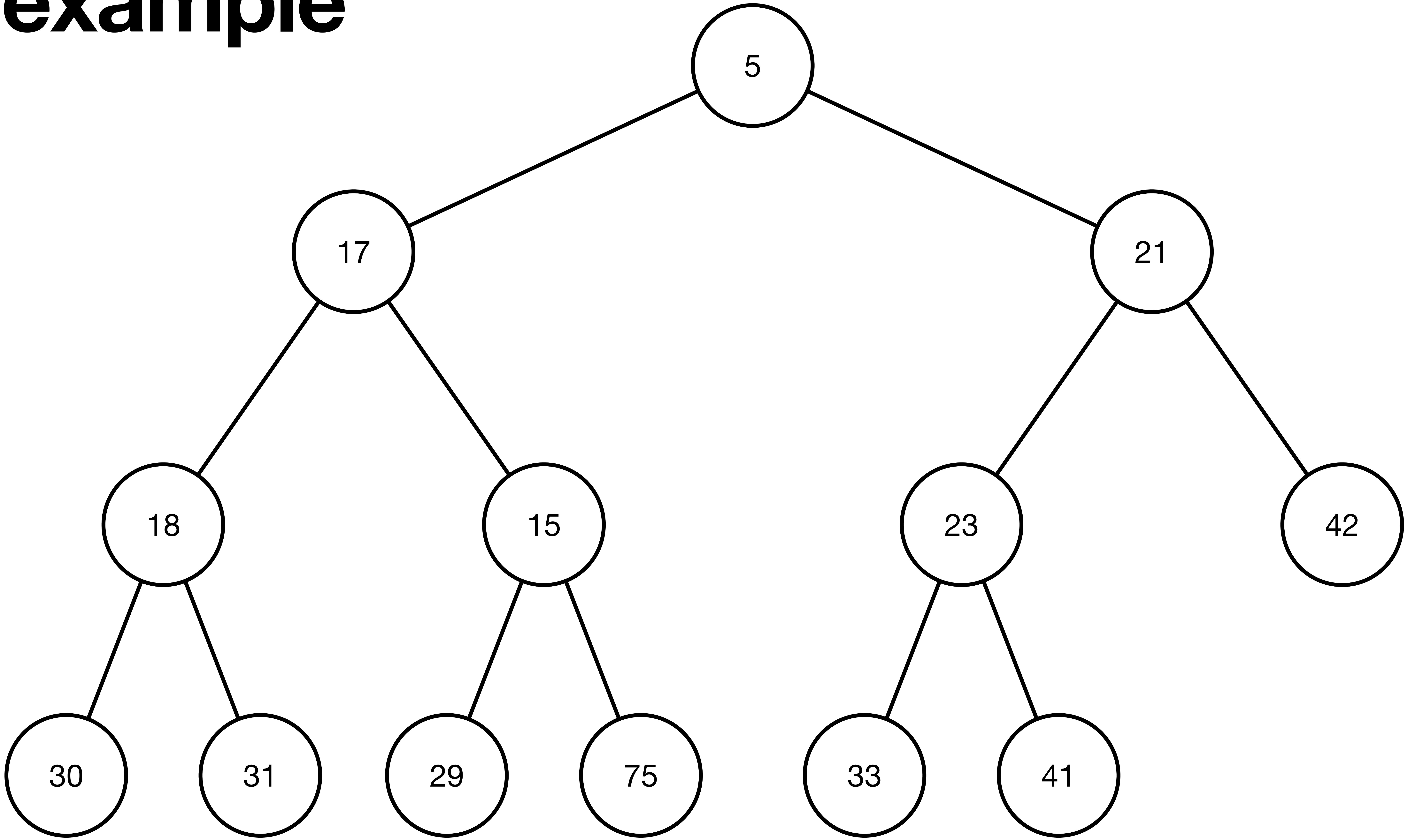


Non-example

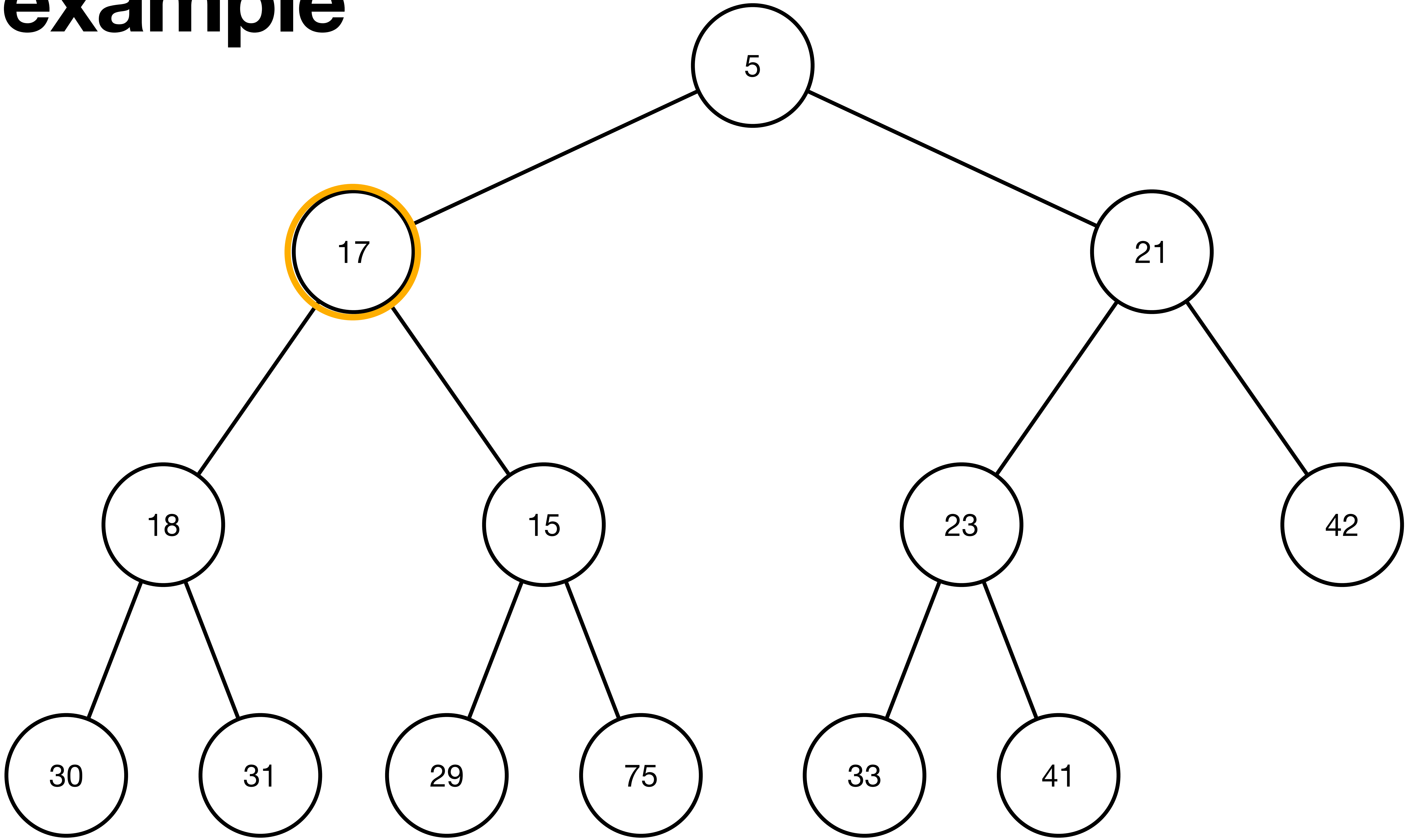


Last level not filled left to right; structure property is violated.

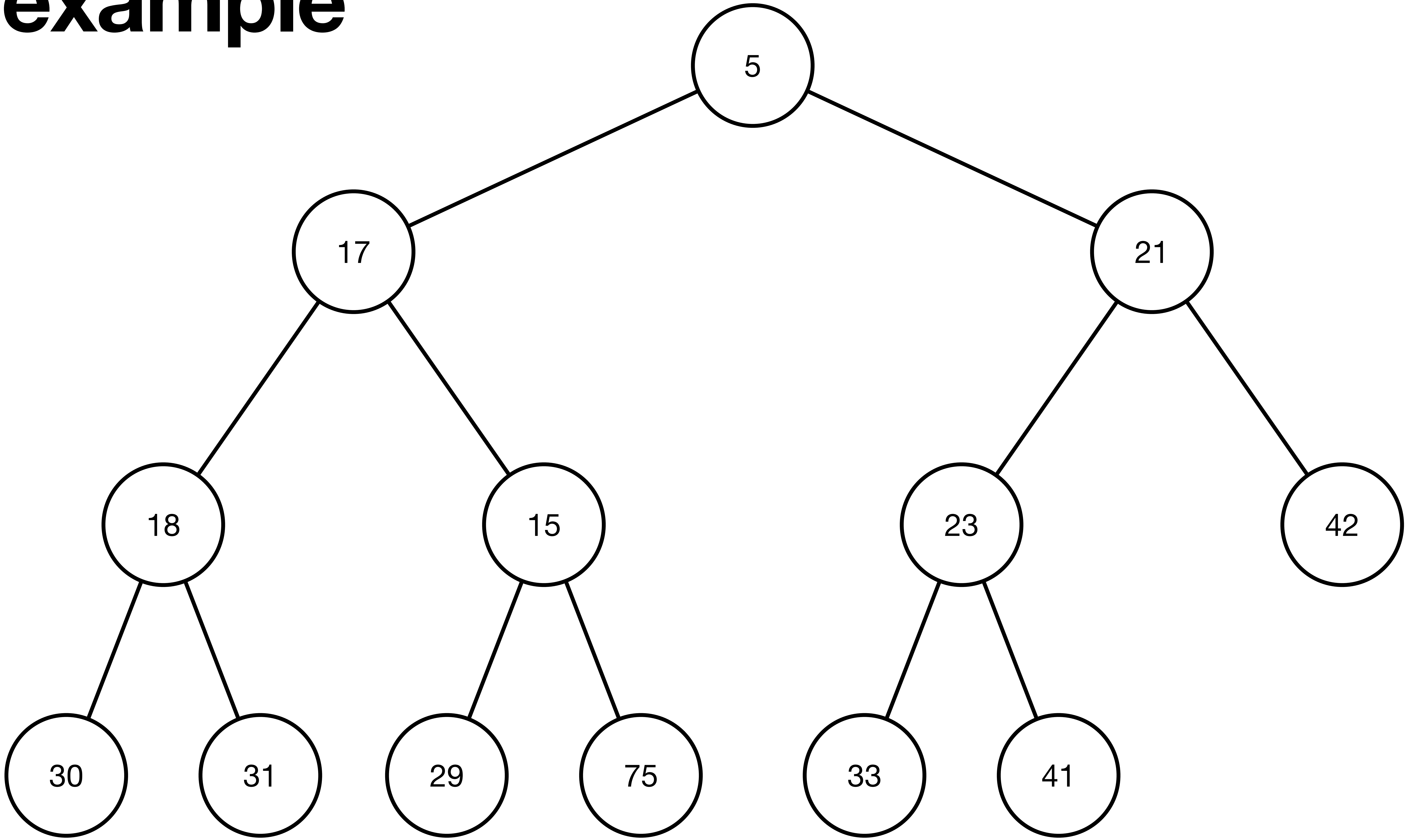
Non-example



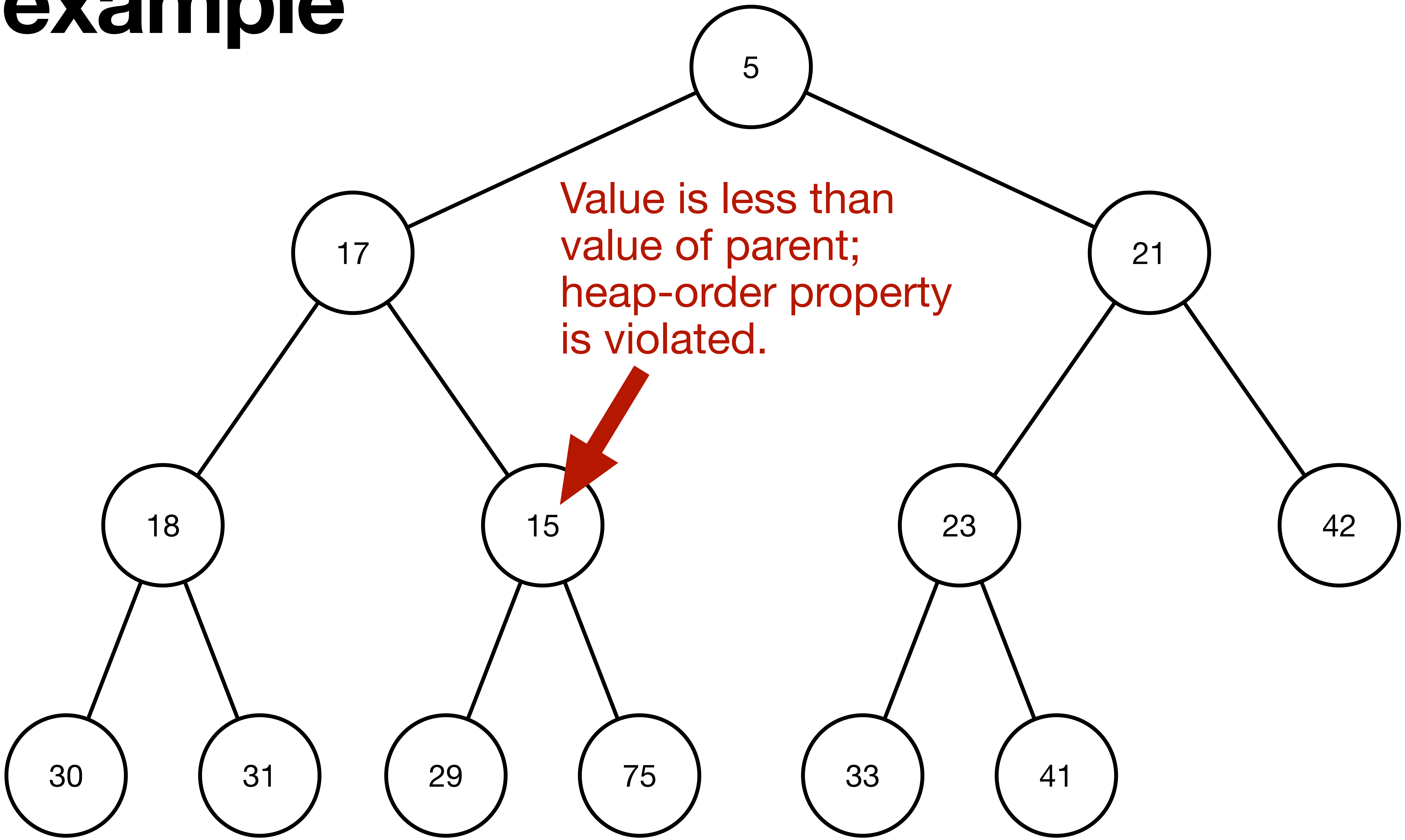
Non-example



Non-example



Non-example



5
17
21
Value is less than value of parent; heap-order property is violated.
18
15
23
42
30
31
29
75
33
41

Number of nodes in a binary heap

Recall that the *height* of a tree, h, is the length of the longest path from the root node to a leaf node.

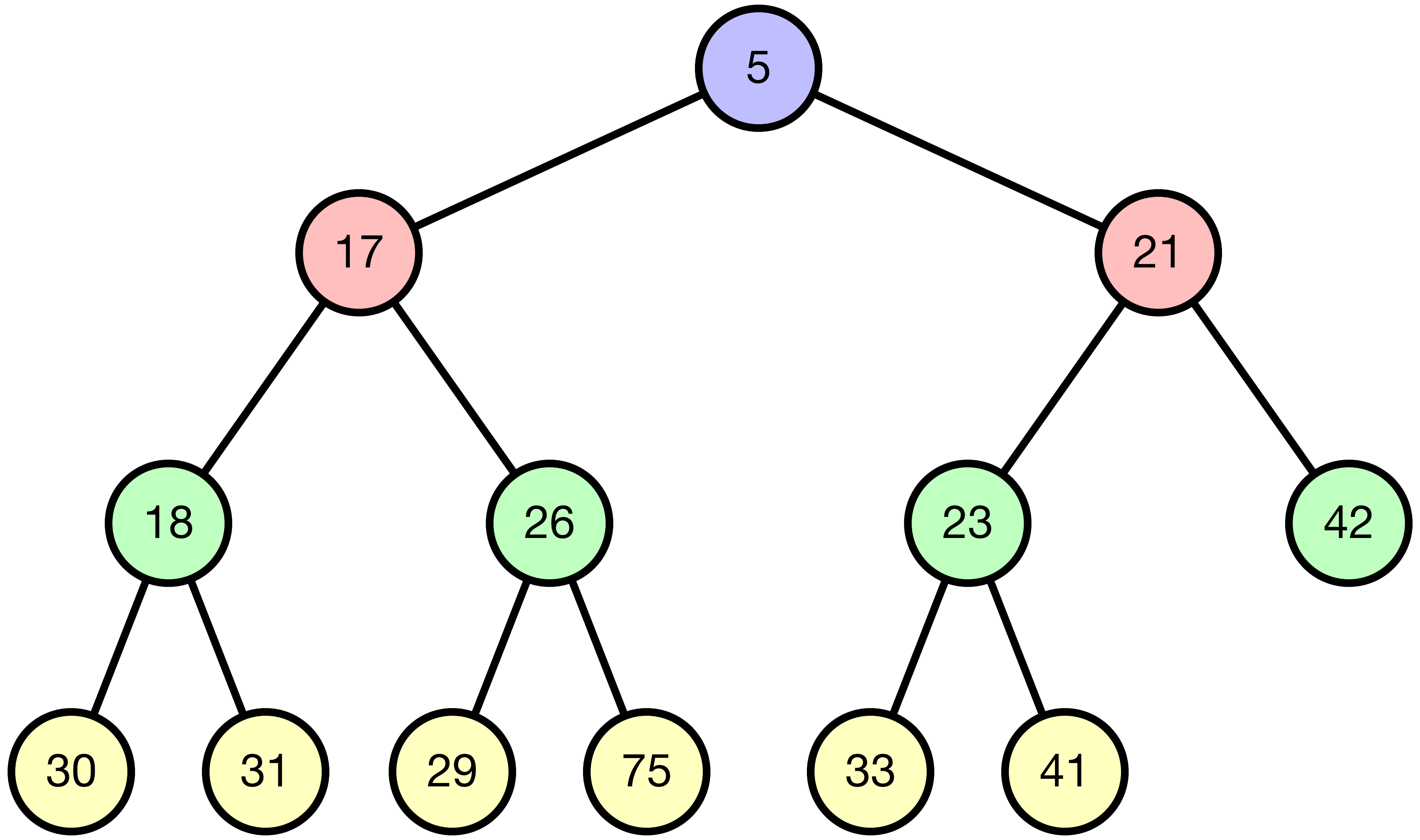
Recall also that the structure property of a binary heap requires that the tree be *complete*. A complete binary tree must have between 2^h and 2^{h+1} -1 nodes.

Given a tree of N nodes, its height is $\lfloor log_2 N \rfloor$.

How can we represent a binary heap?

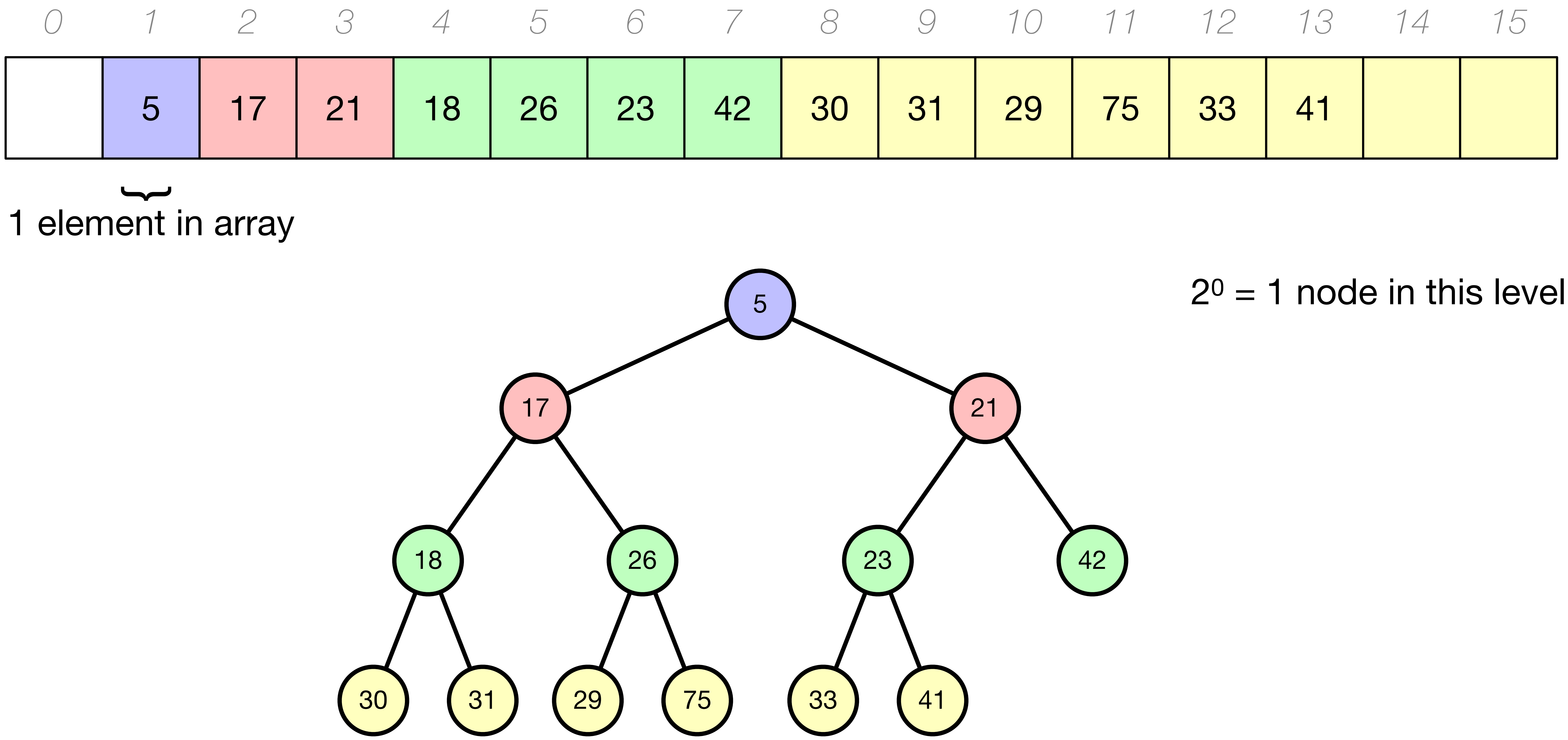
Because a binary heap has a highly regular structure, we can represent it with an *array*.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	5	17	21	18	26	23	42	30	31	29	75	33	41		



How can we represent a binary heap?

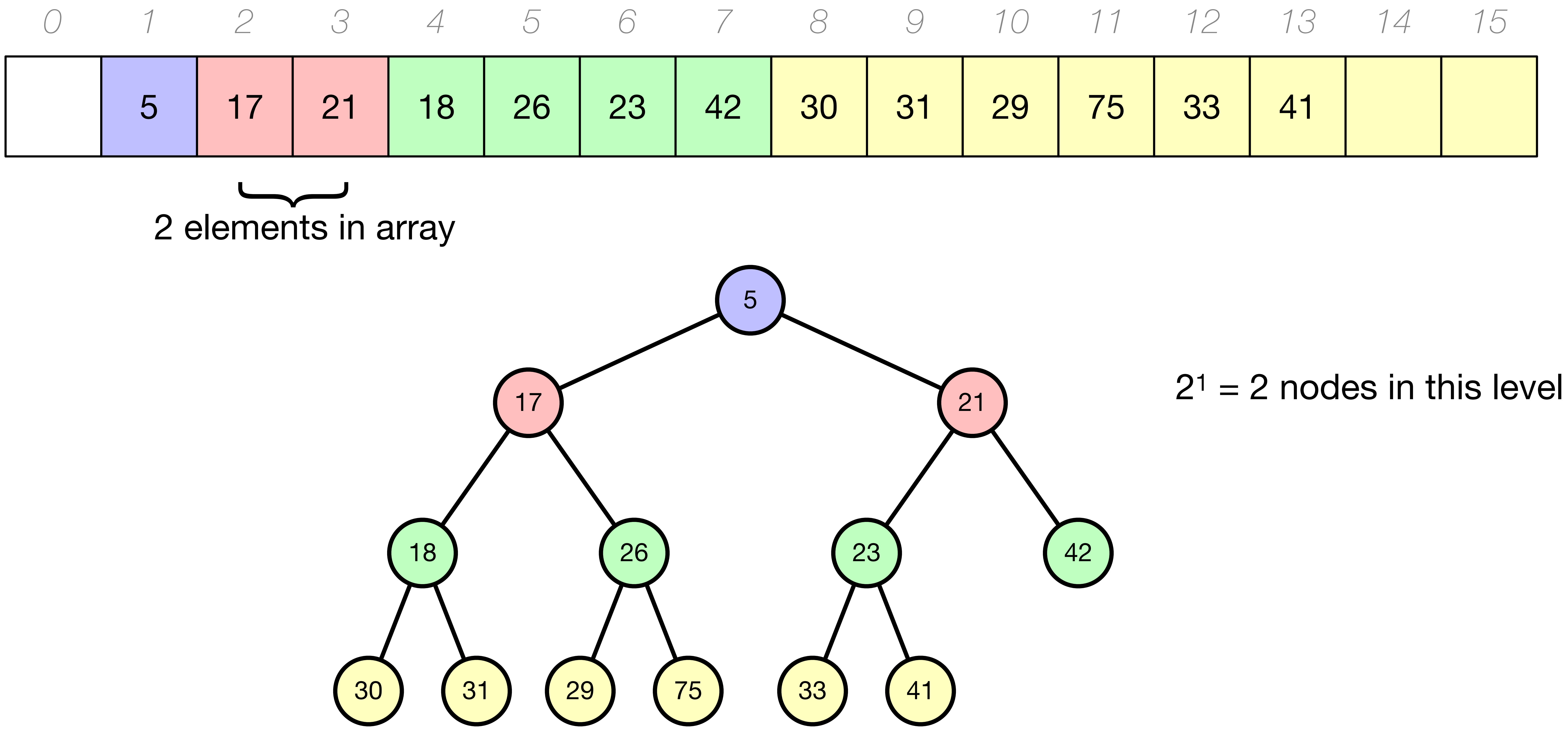
Because a binary heap has a highly regular structure, we can represent it with an *array*.



2^0 = 1 node in this level

How can we represent a binary heap?

Because a binary heap has a highly regular structure, we can represent it with an *array*.

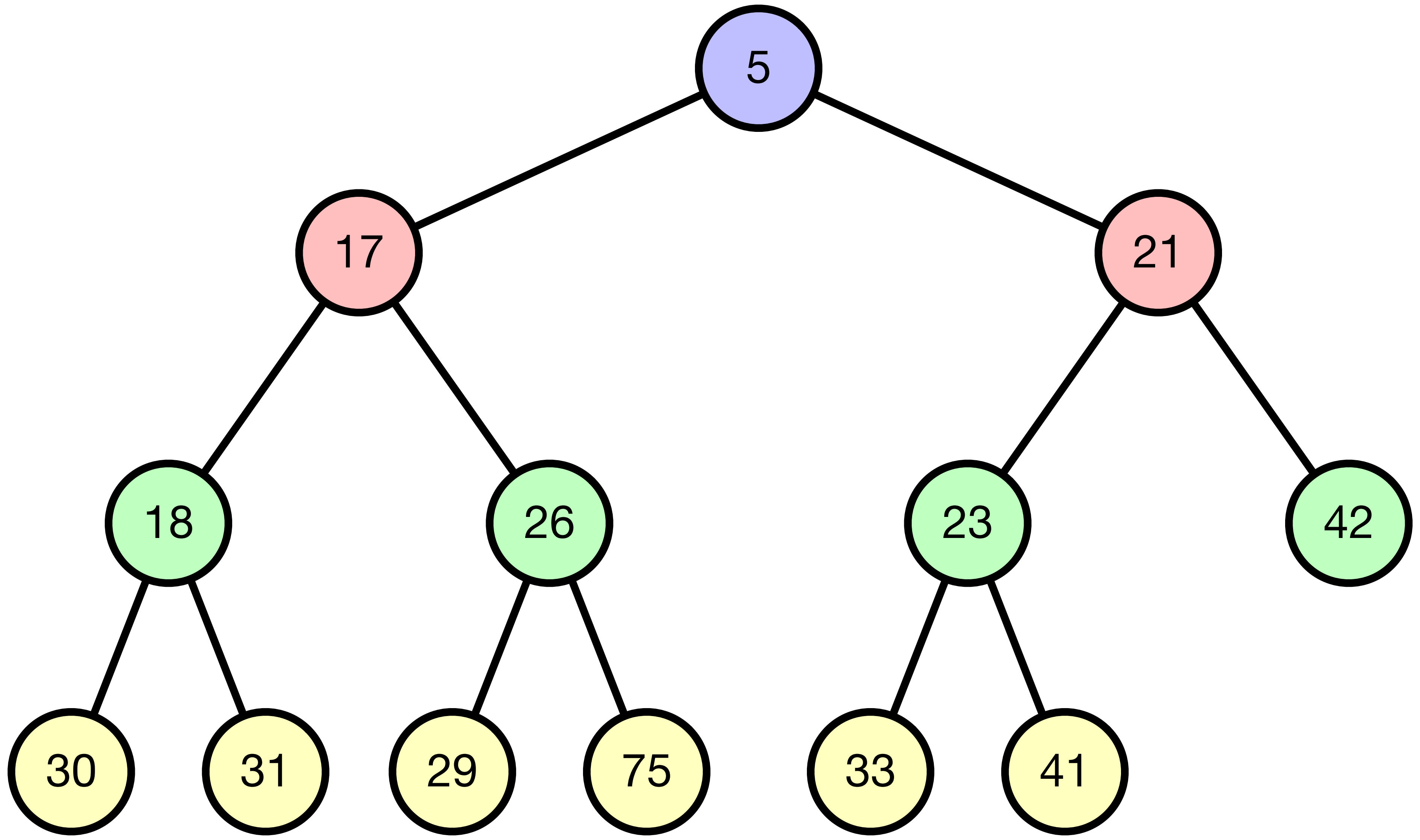


How can we represent a binary heap?

Because a binary heap has a highly regular structure, we can represent it with an *array*.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	5	17	21	18	26	23	42	30	31	29	75	33	41		

4 elements in array



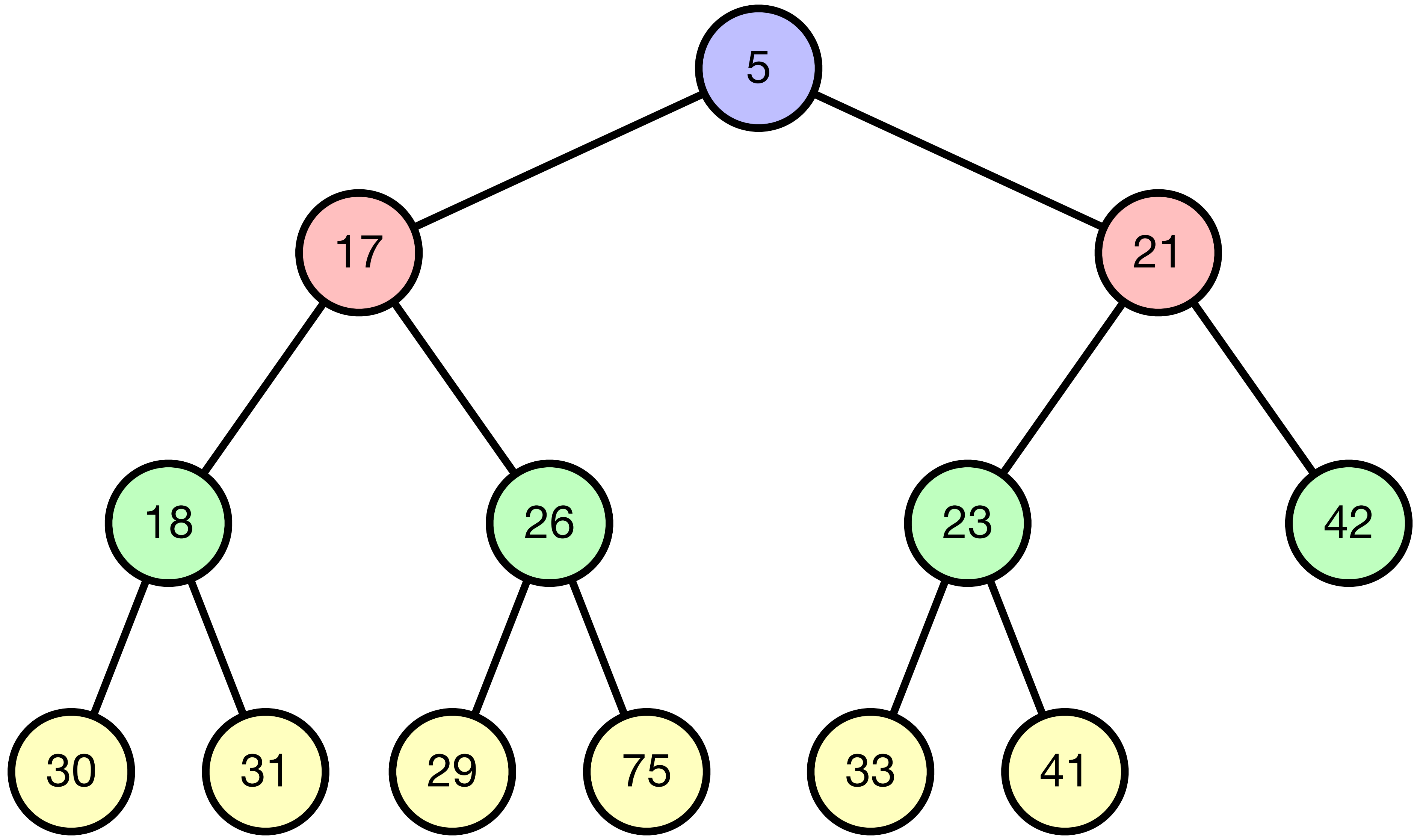
2^2 = 4 nodes in this level

How can we represent a binary heap?

Because a binary heap has a highly regular structure, we can represent it with an *array*.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	5	17	21	18	26	23	42	30	31	29	75	33	41		

8 elements in array

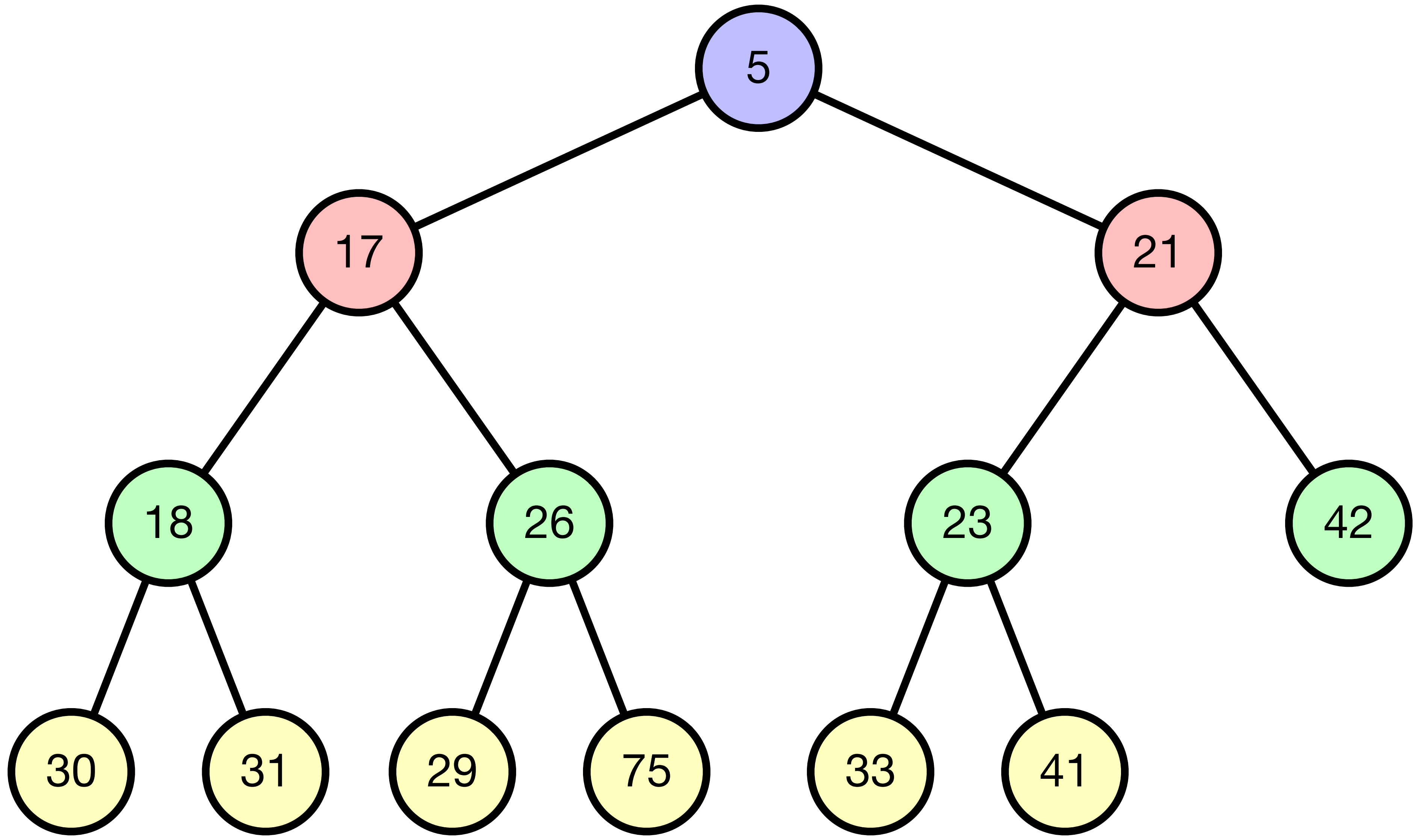


Up to $2^3 = 8$ nodes in this level

How can we represent a binary heap?

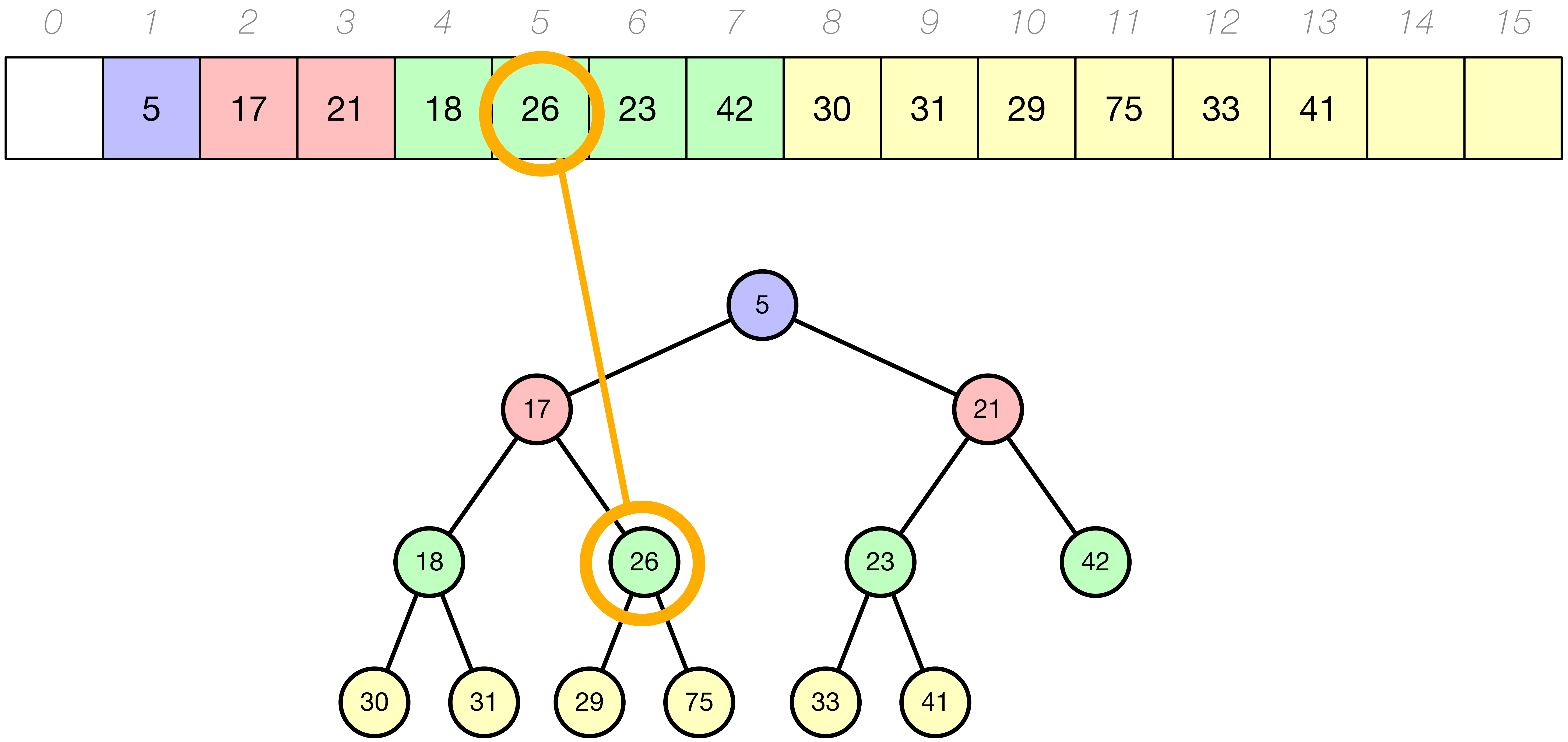
Because a binary heap has a highly regular structure, we can represent it with an *array*.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	5	17	21	18	26	23	42	30	31	29	75	33	41		



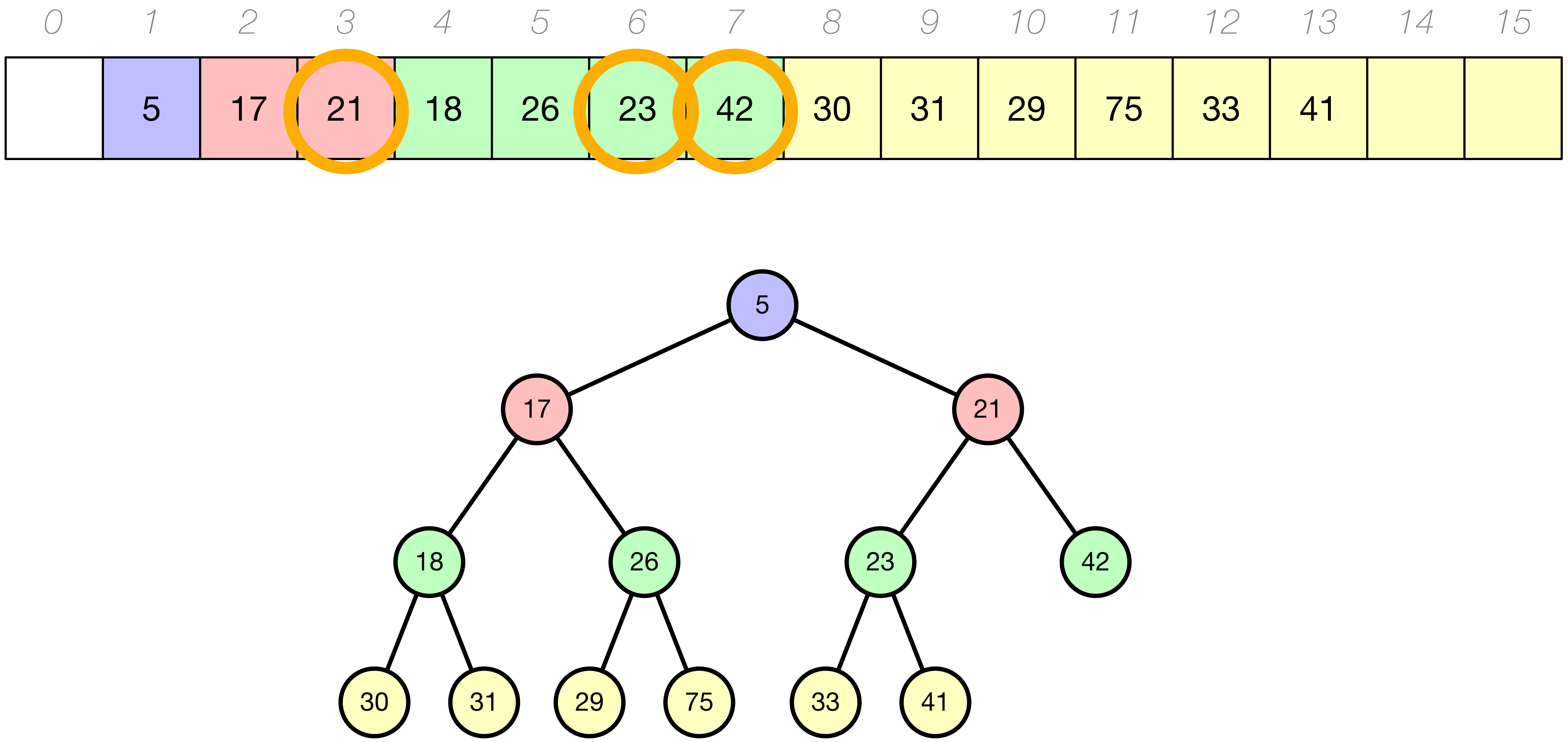
How can we represent a binary heap?

Because a binary heap has a highly regular structure, we can represent it with an *array*.



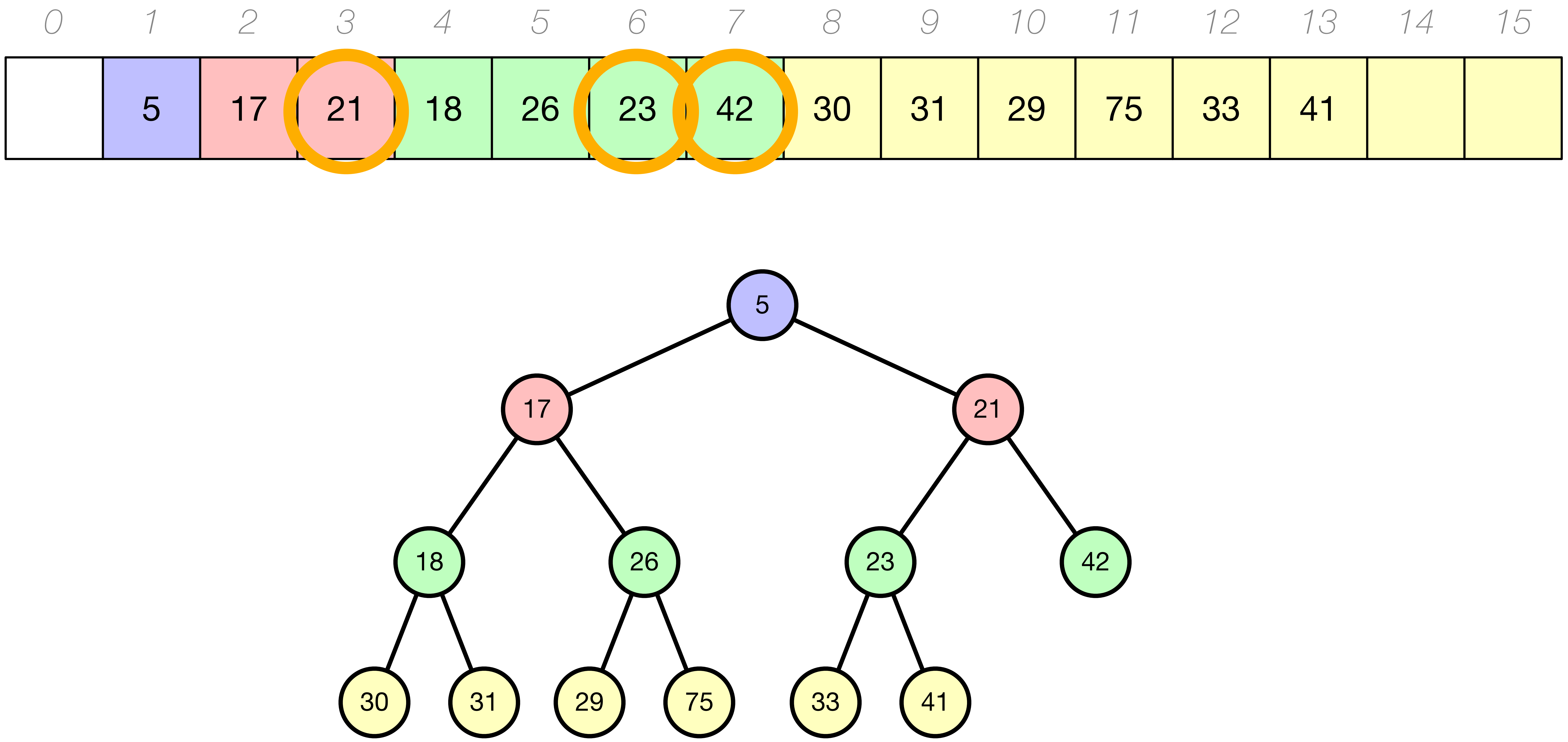
How can we find a node's children?

Node at index *i* has children at indices 2*i* and 2*i*+1.



How can we find a node's parent?

Unless we're at the root, given a node with index, *i*, a node's parent is at index *i* / 2, if *i* is even. If a node's index is odd, the node's parent is at index (*i* - 1) / 2.



How can we modify a binary heap?

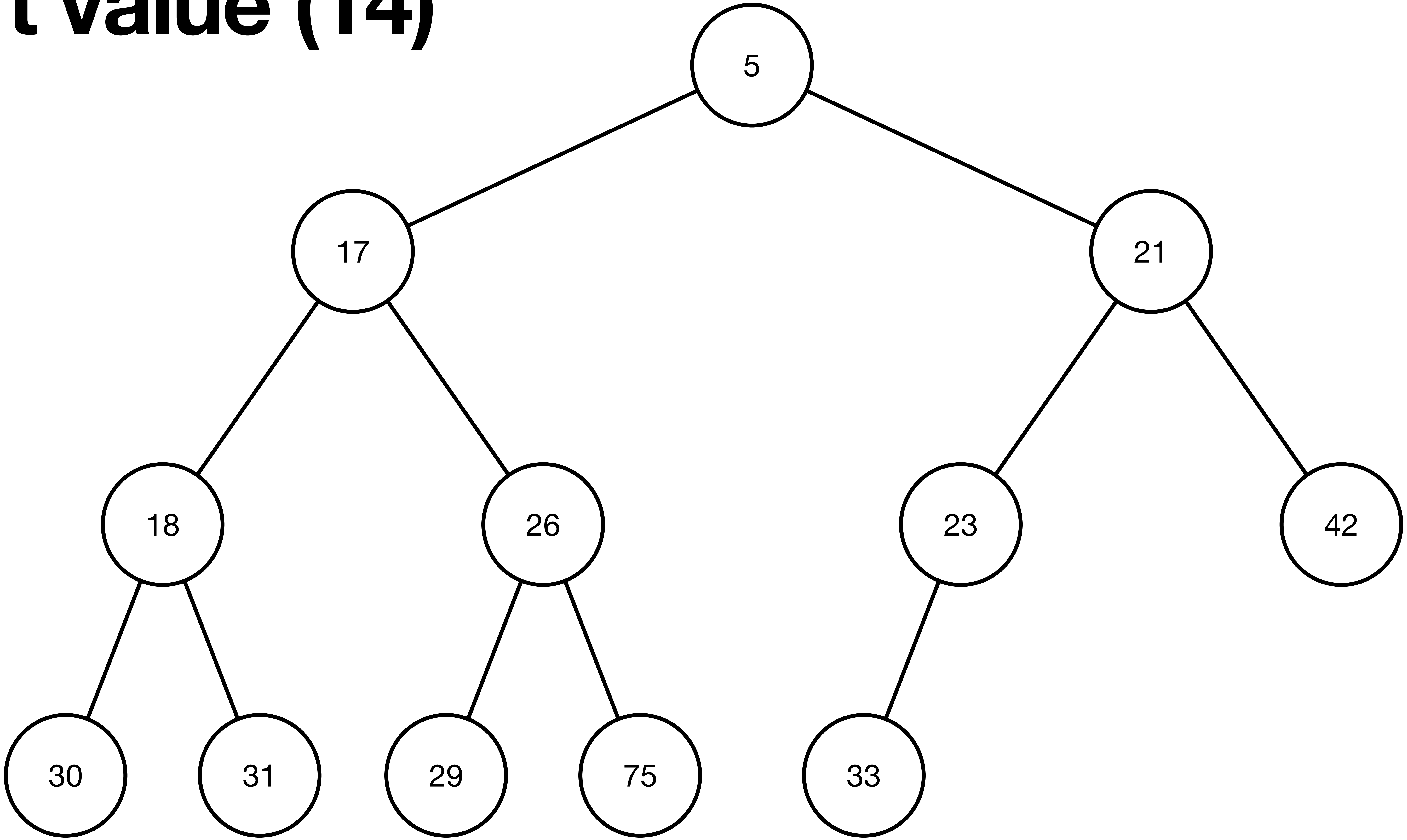
As a static object, a binary heap is of little utility. We need to be able to modify it by inserting and deleting nodes. But we must insert and delete in ways that *preserve the required properties*.

Let's look at the following operations:

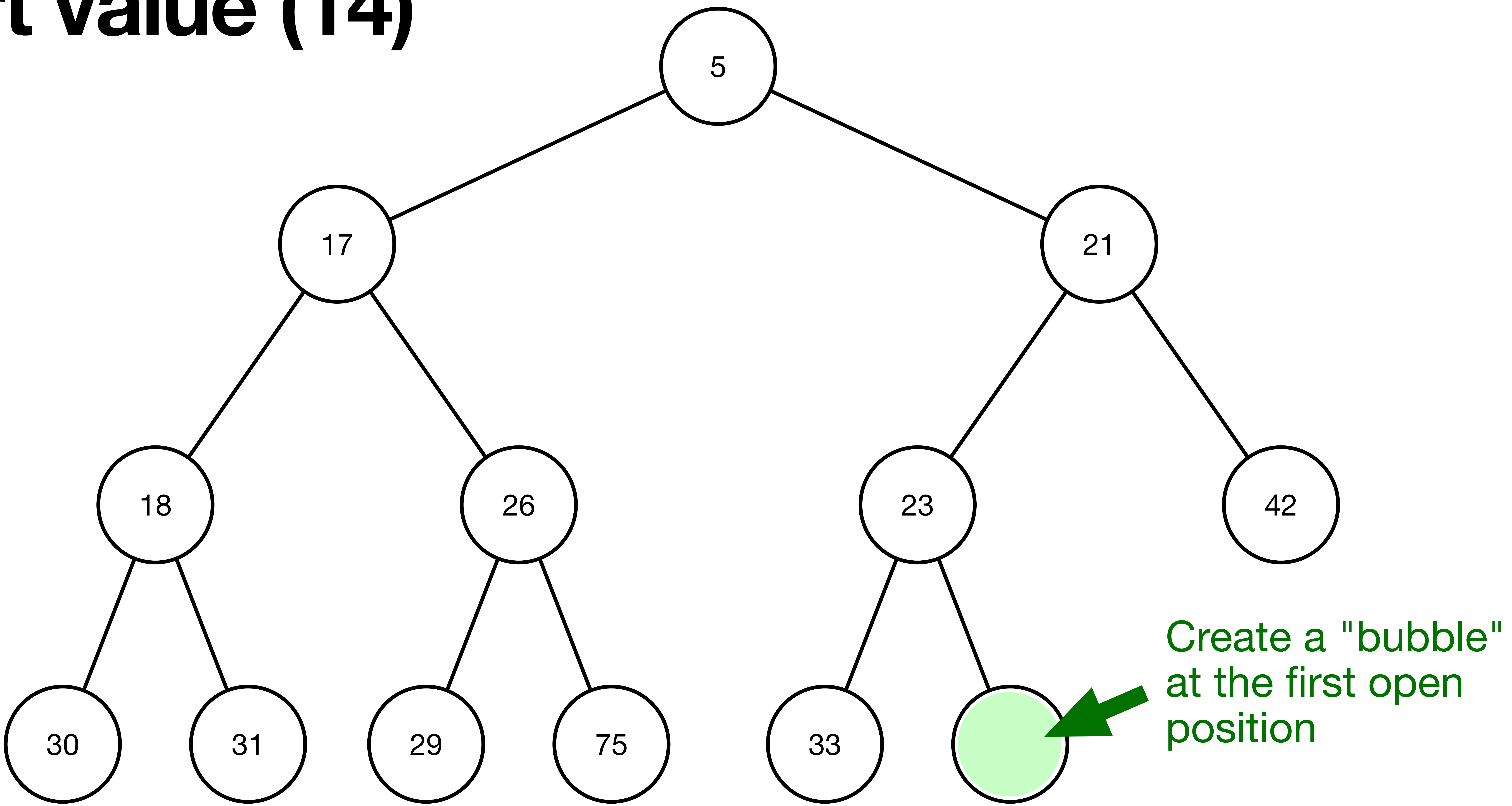
- Insert
- Delete minimum

When inserting and deleting we use one of two strategies to preserve structure and heap-order property. We use the metaphor of a "bubble" *percolating up* or *percolating down* within the heap.

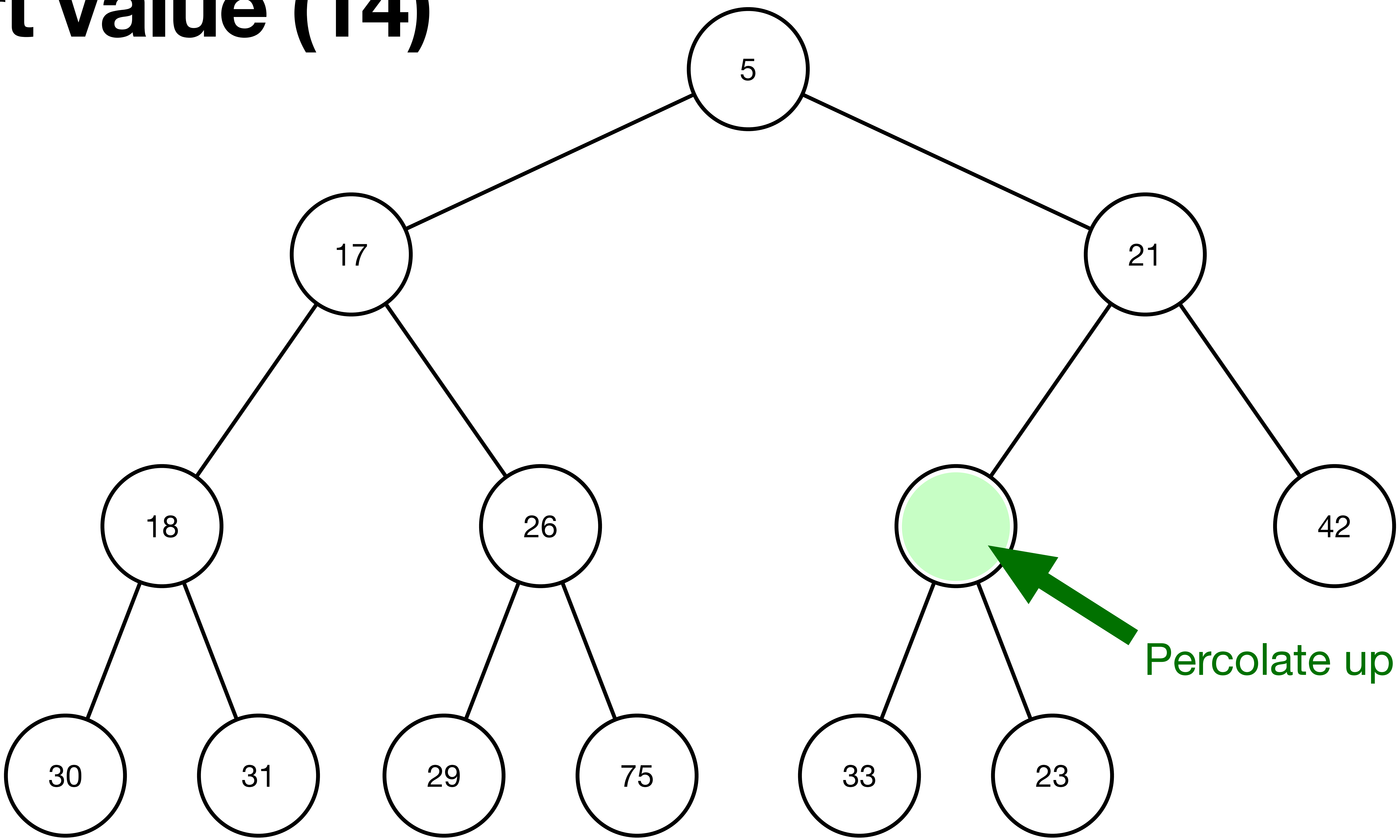
Insert value (14)



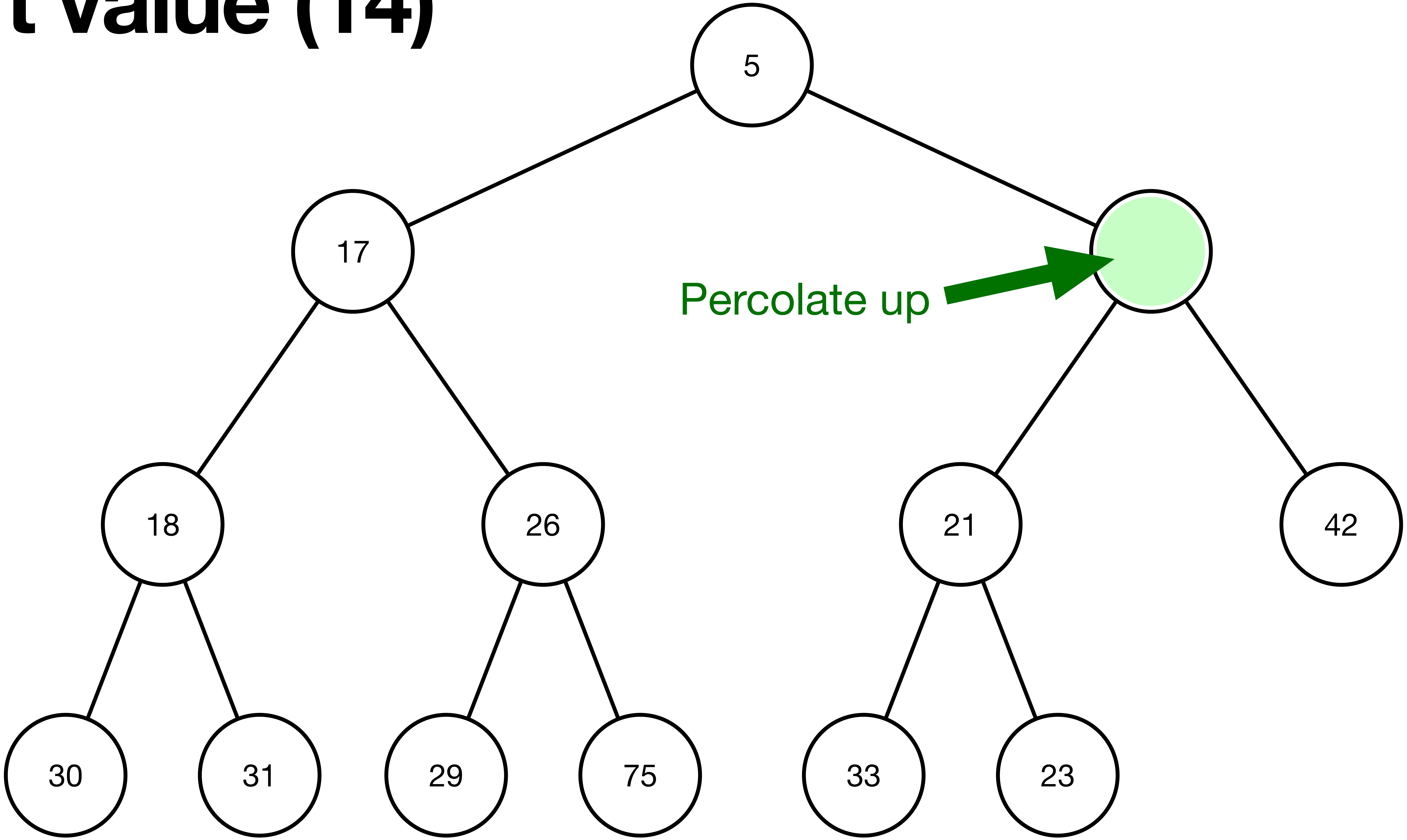
Insert value (14)



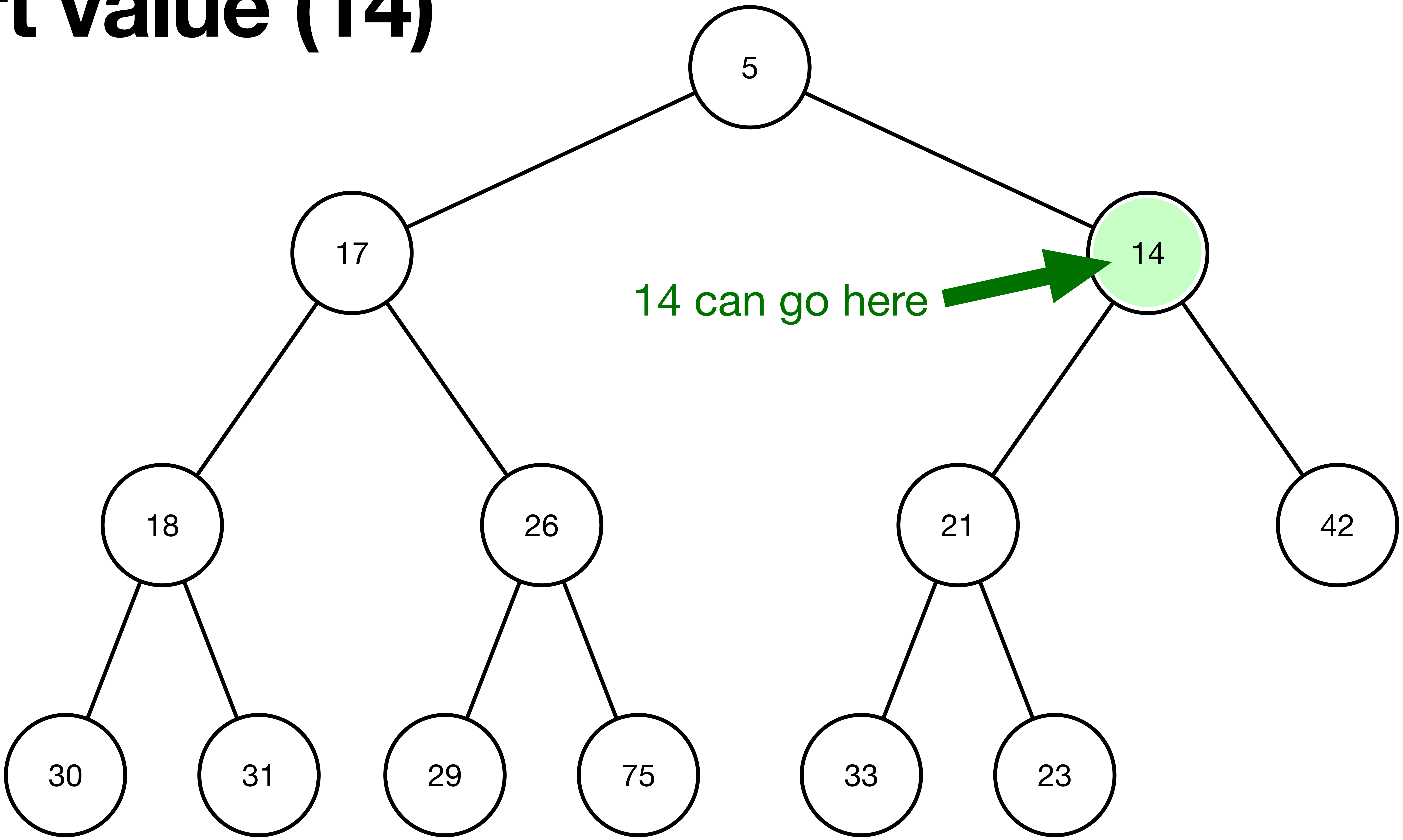
Insert value (14)



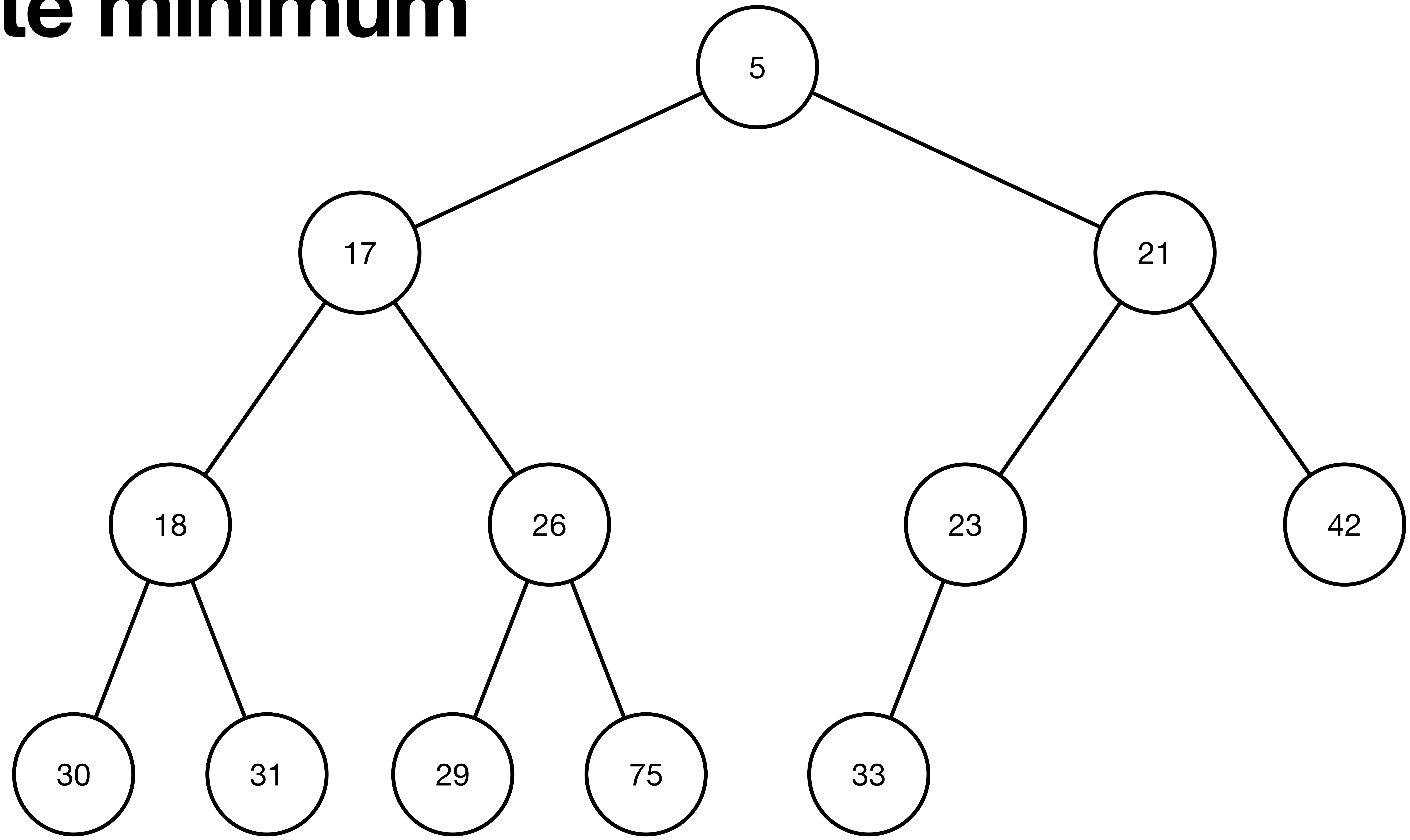
Insert value (14)



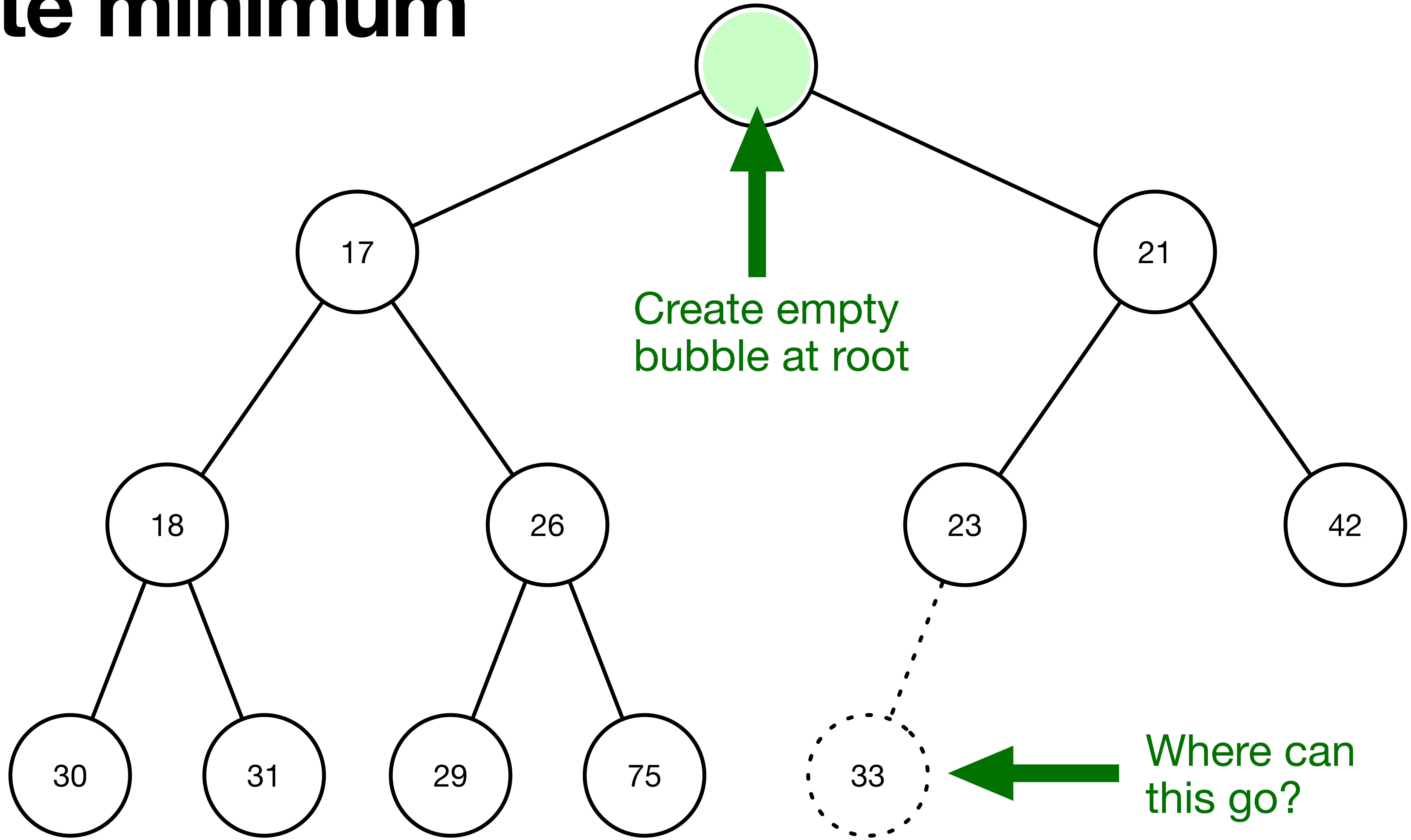
Insert value (14)



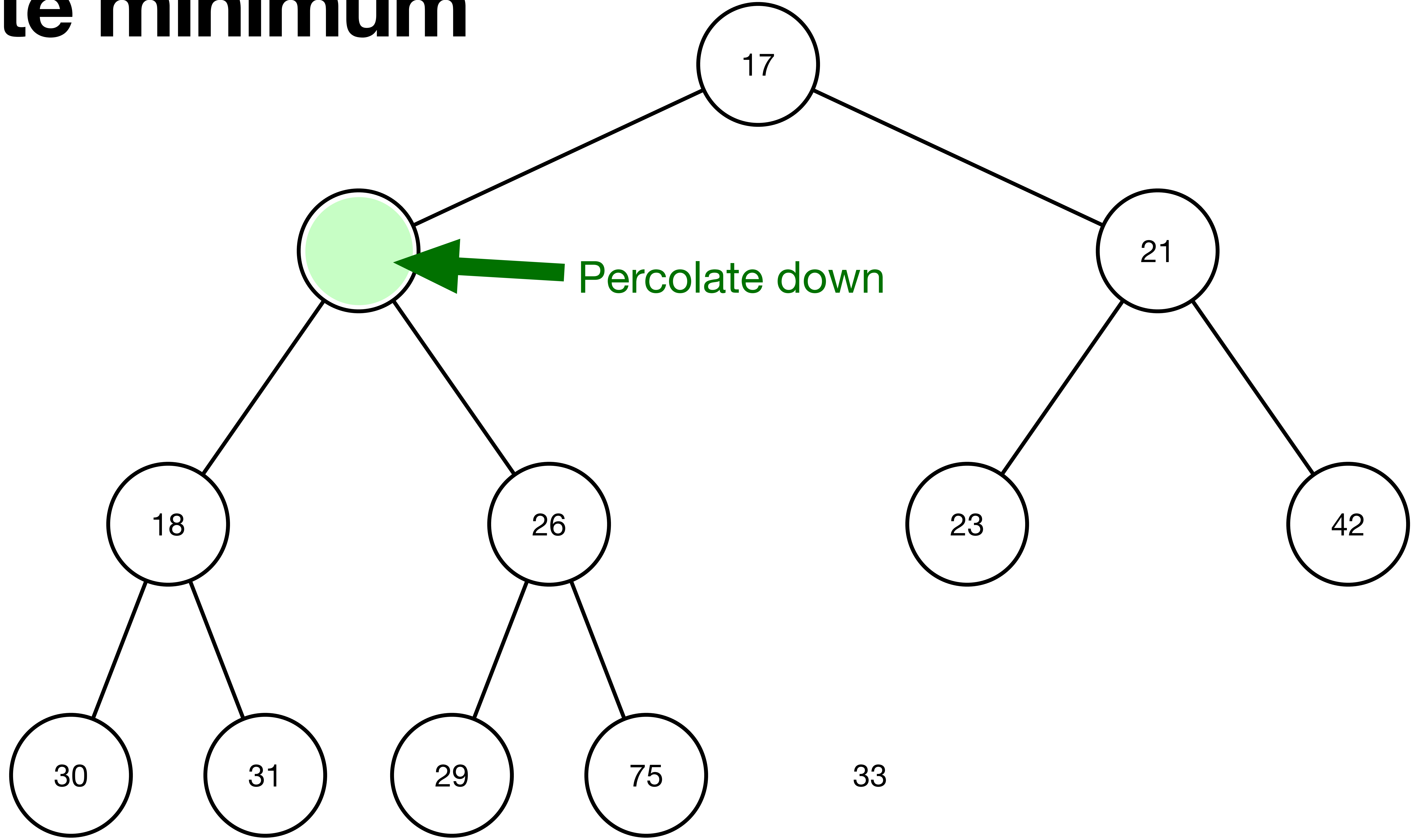
Delete minimum



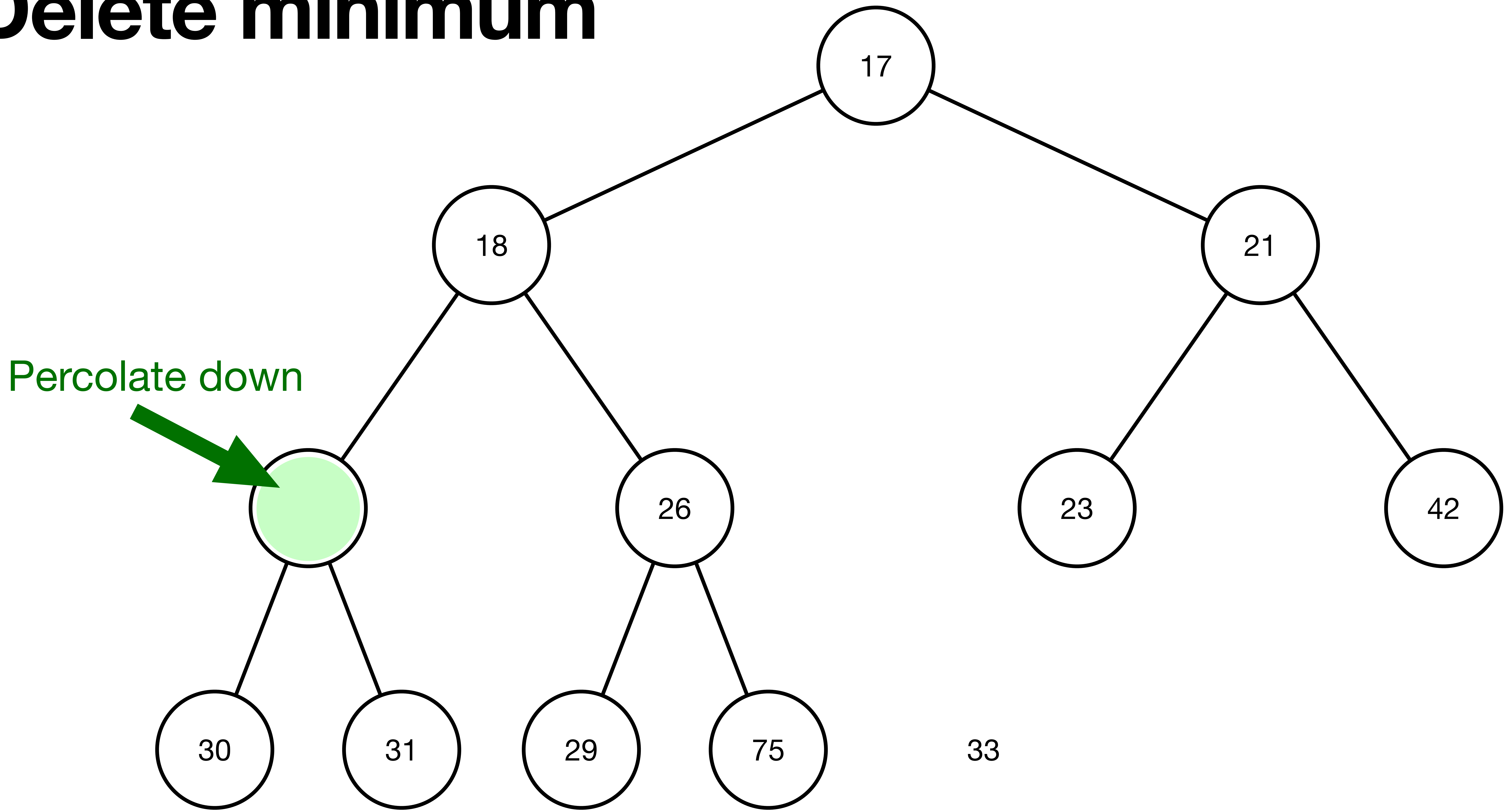
Delete minimum



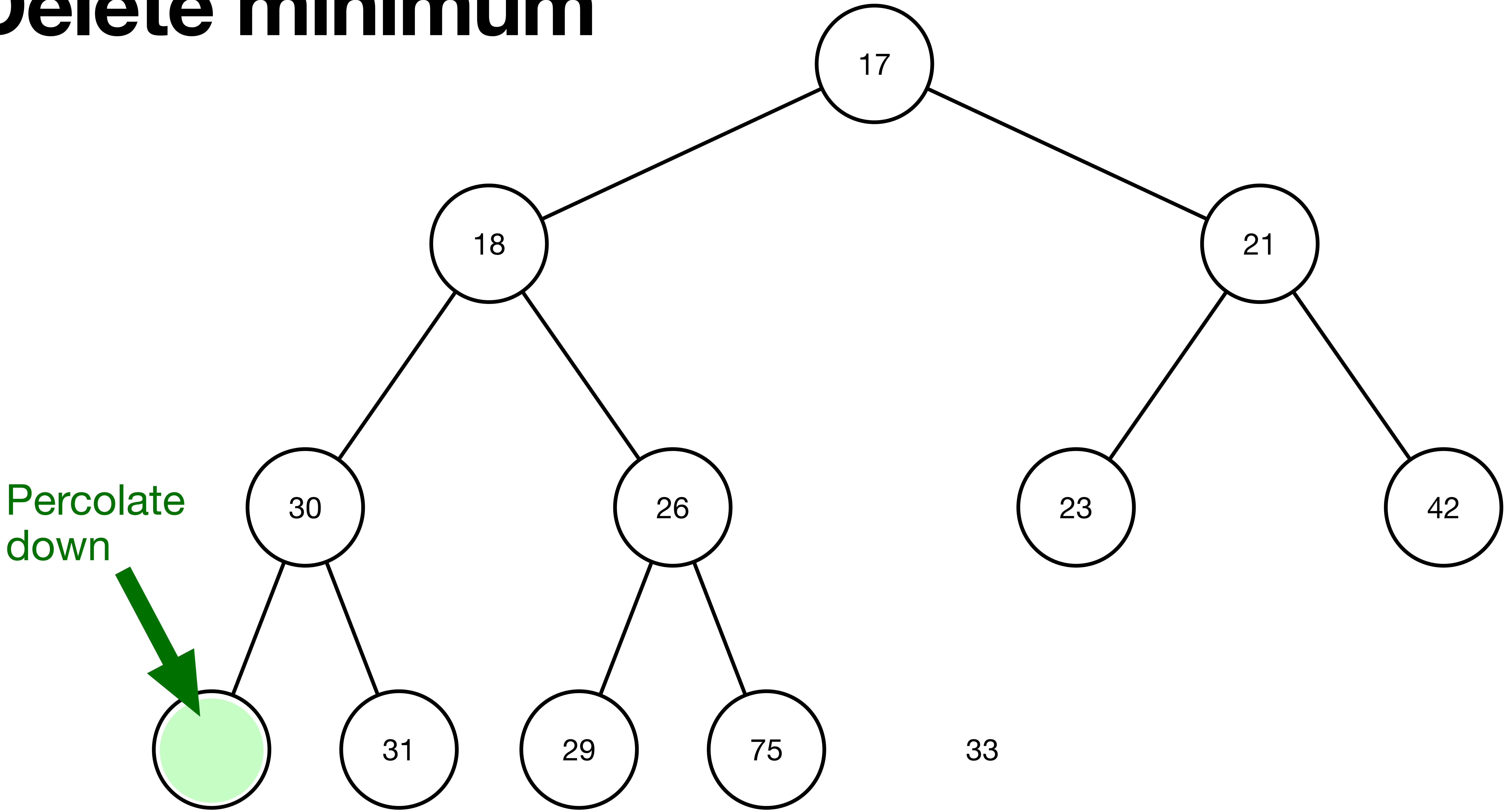
Delete minimum



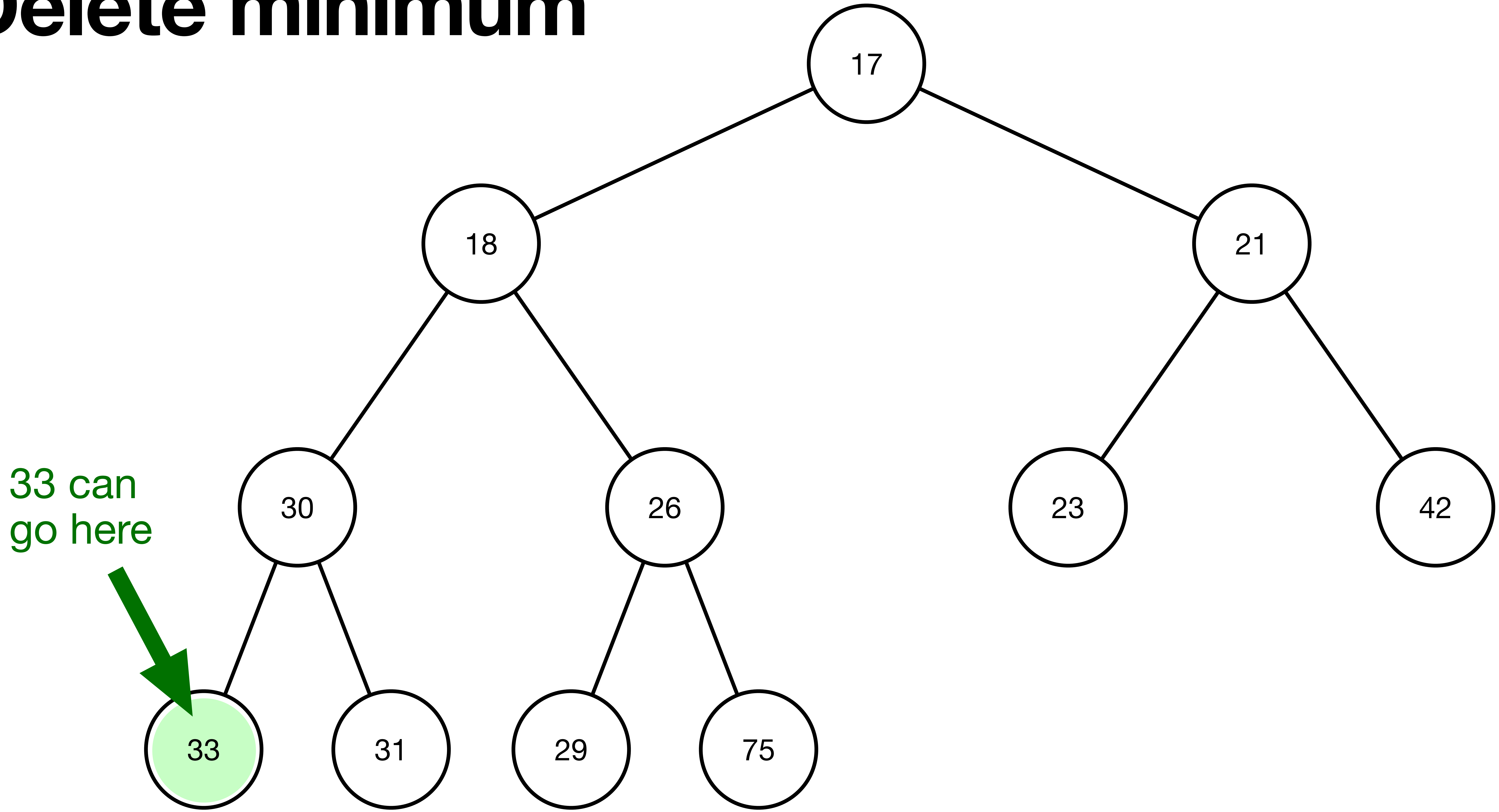
Delete minimum



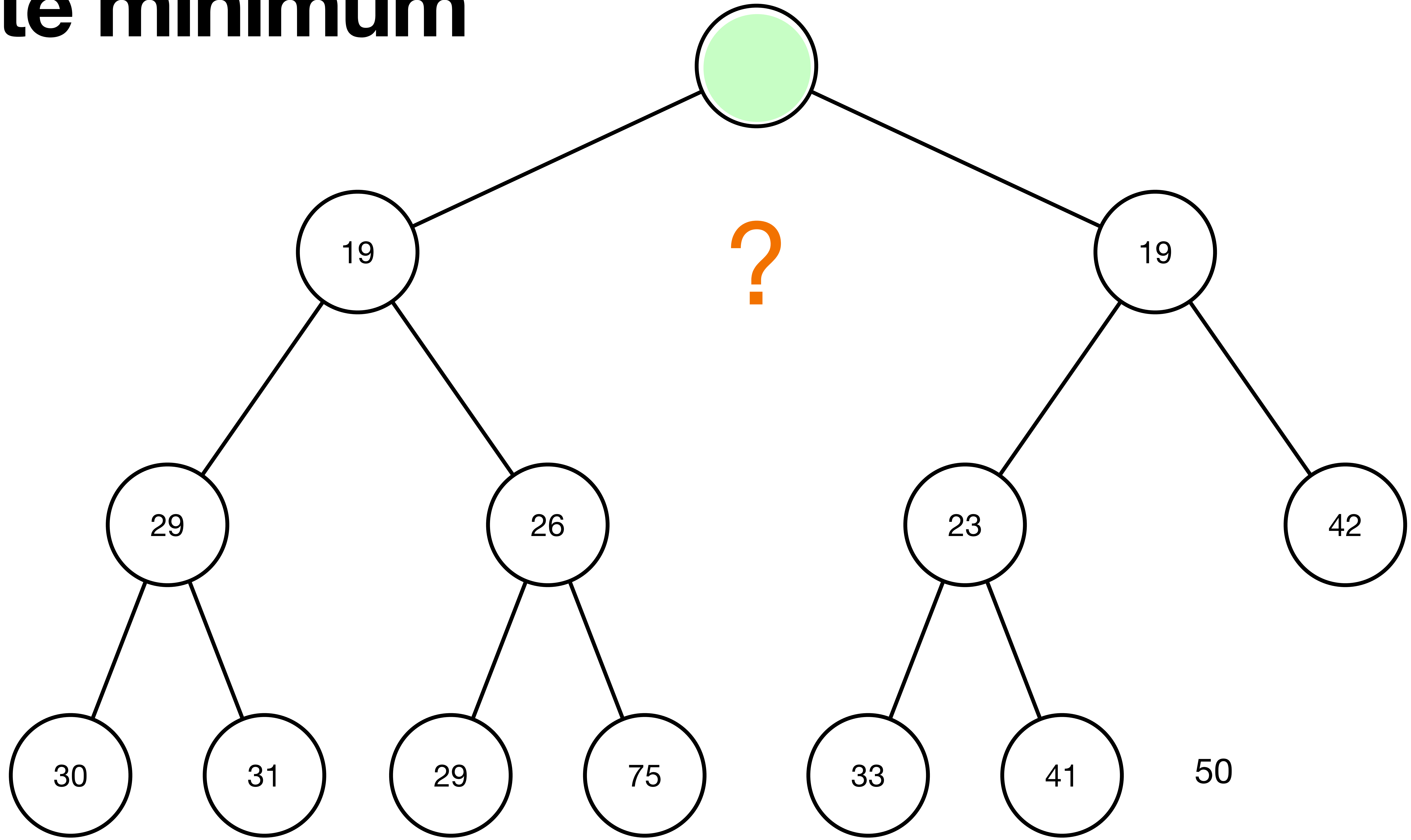
Delete minimum



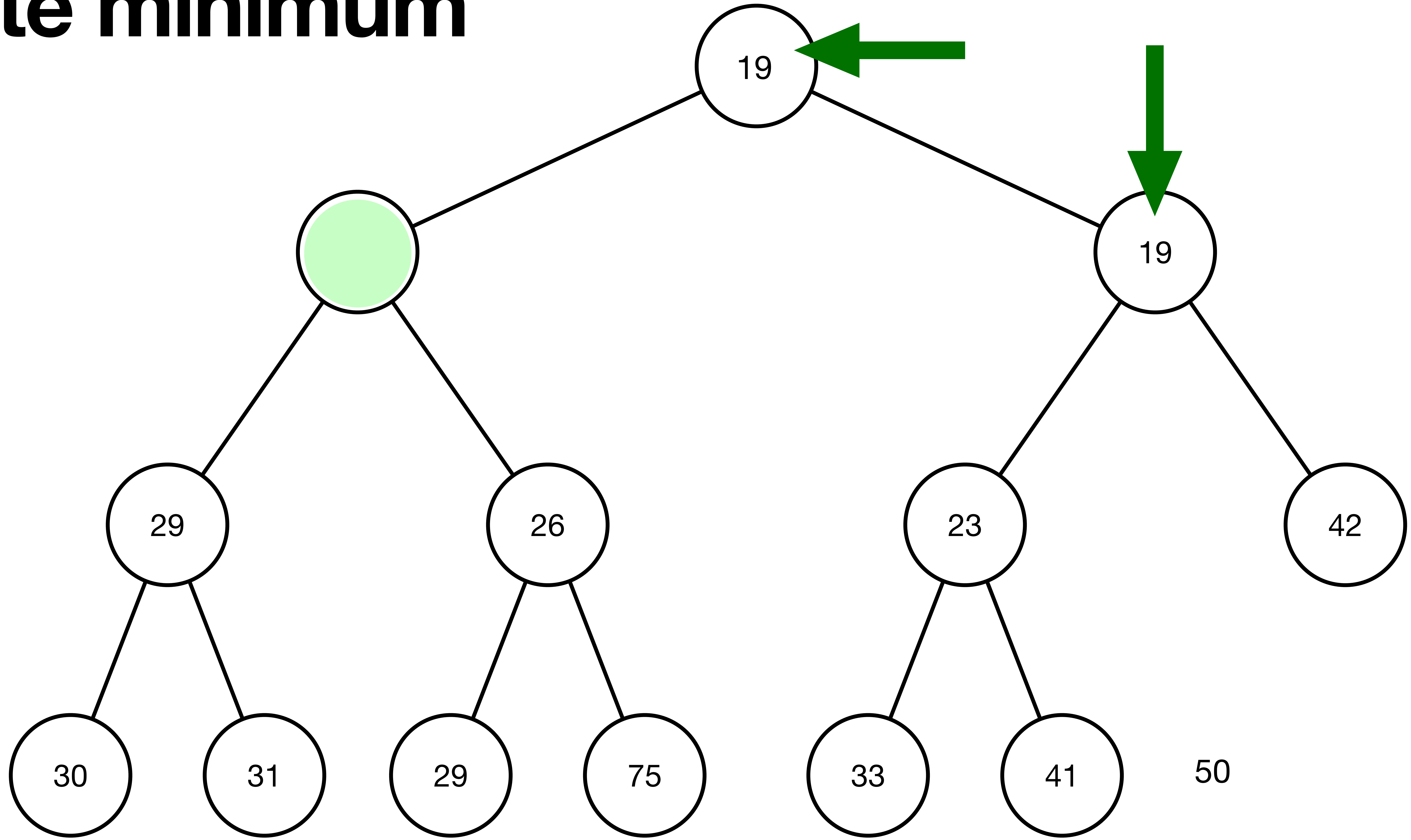
Delete minimum



Delete minimum



Delete minimum



What's a binary heap good for?

We've already mentioned that binary heap is used in Williams' *heapsort* algorithm and is used in many *graph algorithms*.

Because a binary heap maintains the smallest value at the root, it is well-suited as a data structure for a *priority queue*.

With the ordering we've chosen here, jobs or processes with lower numbers have higher priority, and we can consume the binary heap by continuing to remove the lowest valued node (at the root).

Summary

- Binary heap is a *complete* binary tree with the *heap-order property*.
- Structure property and heap-order property must be preserved.
- Binary heap can be represented with an *array*.
- We modify the heap by creating *bubbles* and *percolating up or down* until a new (or orphaned) value can find a suitable node in the tree.
- Insert and delete operations have O(*log N*) complexity.
- Binary heaps are well-suited to *priority queue* and other applications.