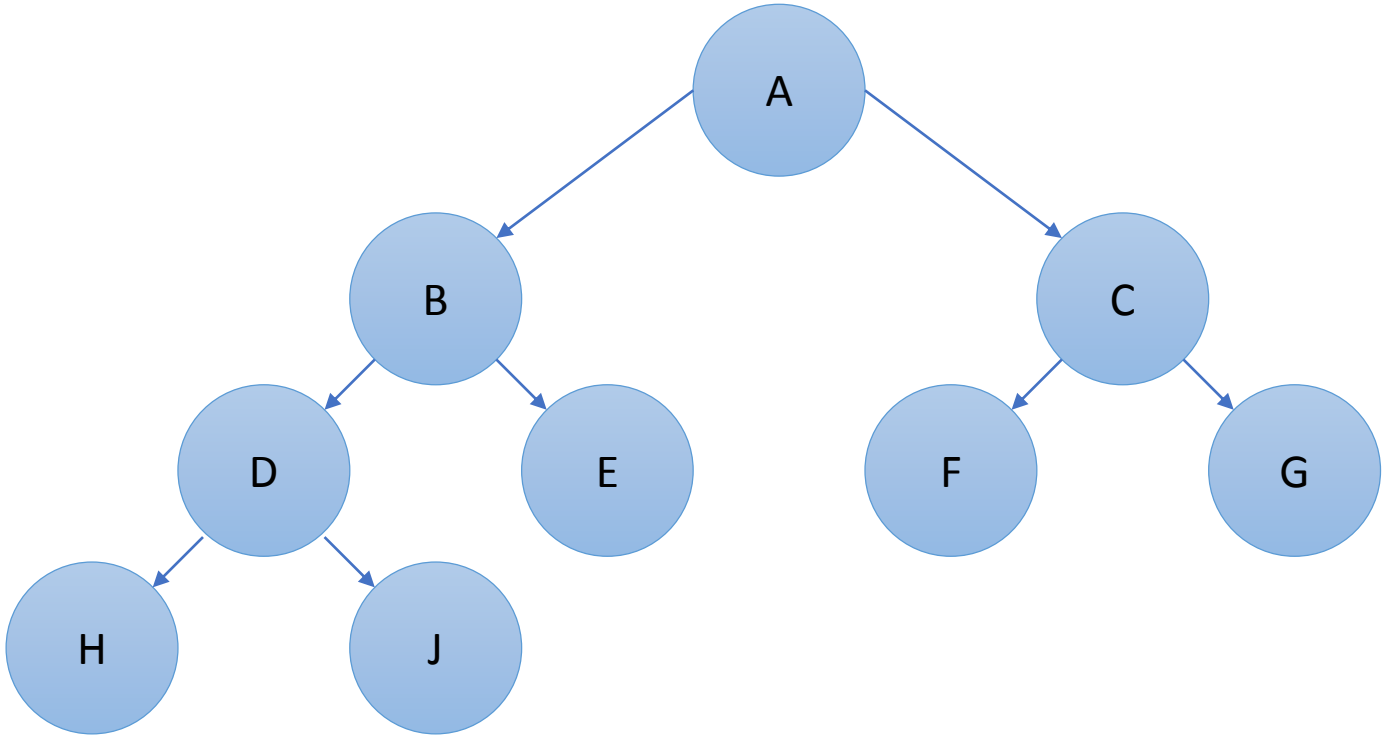
Warmup

Draw a picture of the Binary Tree represented by the following List

0	1	2	3	4	5	6	7	8
A	B	C	D	E	F	G	H	J



Today

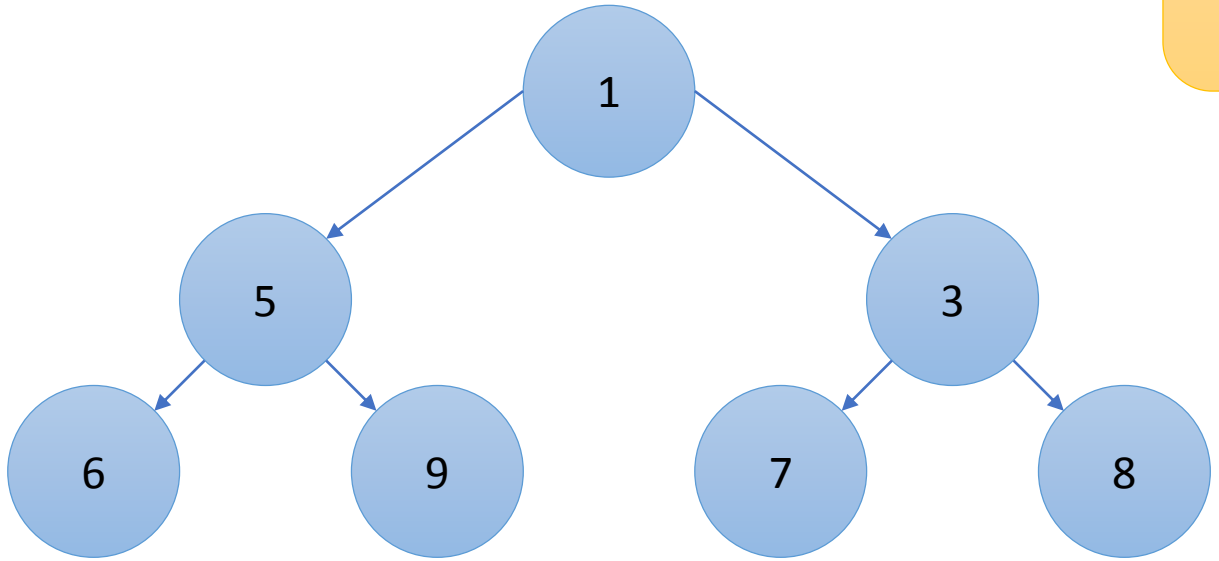
- Heaps
  - What are they?
  - What are they good for?
- How to add to a heap
- How to remove from a heap?
- How to heapify a tree!

Heap

A Heap is a complete **Binary Tree** that follows the following rule:

- The root of a Heap is smaller than all of its children.
  (this rule also applies to all subtrees in the Heap)

This is called the **"heap property"**



Why are Heaps cool?

- The root of the heap is always the smallest value
- Consider the following Tree method (which is inherited by a heap)

```
public E removeRoot()
```

What do you know about the value that `removeRoot` will return when called on a heap?

Min-heap vs Max-heap

We are talking about Min Heaps, where **the root will always be the smallest (minimum) value**.

A Max Heap is the same as a min heap except its heap property is "**the root of the heap is the largest value**".

Activity 1: Sort the following list using a Heap

```
List<Integer> nums ← {3, 1, 7, 2, 9, 6, 4}

Heap h = new Heap();

//your code goes here
```

A `Heap` is-a `Tree`, so it has all the same methods as a `Tree`

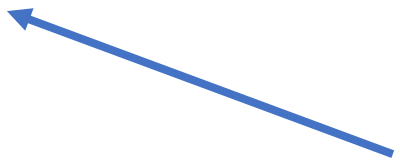
You have access to methods like:

```
add(E data)
size()
removeRoot()
```

Activity 1: Sort the following list using a Heap

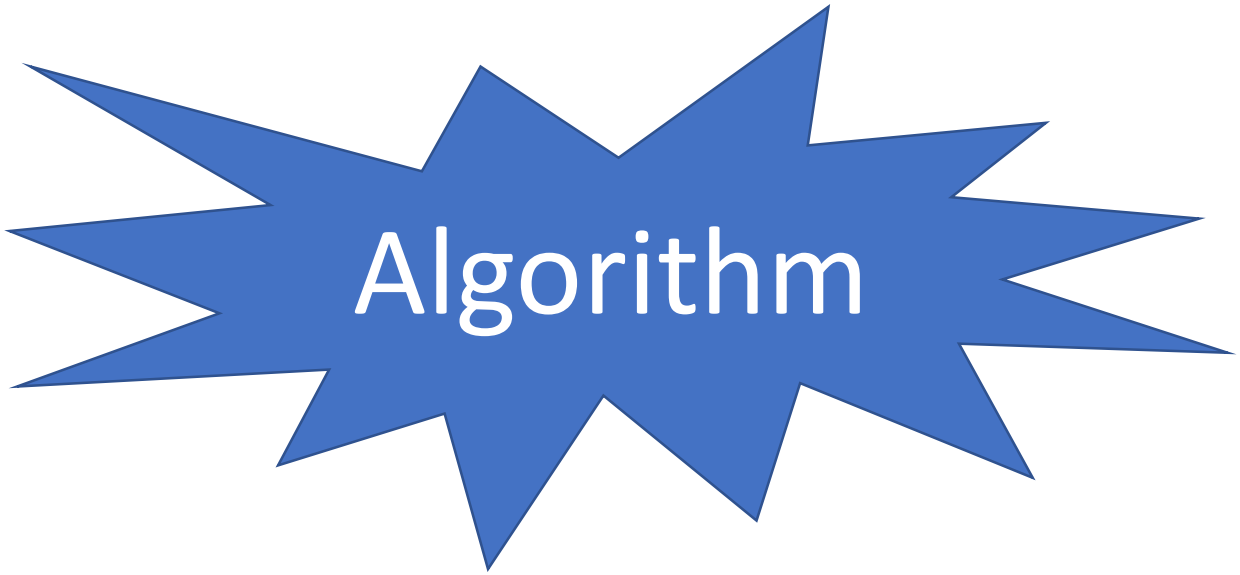
```
List<Integer> nums ← {3, 1, 7, 2, 9, 6, 4}

Heap h = new Heap();

while(nums.size() > 0)
  h.add(nums.remove());

while(h.size() > 0)
  nums.add(h.removeRoot());
```

A `Heap` is-a `Tree`, so it has all the same methods as a `Tree`



When this loop finishes, `nums` is empty!

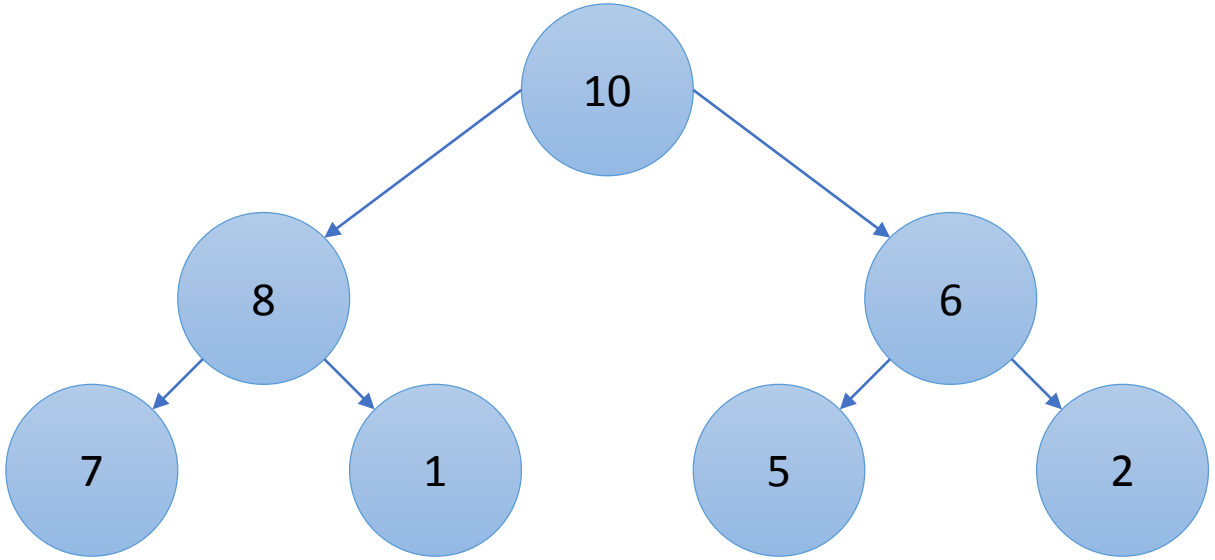
Adding to a Heap



- Add the element to the bottom level of the heap
- Compare the added element with its parent
  - If they are in the correct order, stop
  - If not, swap the element with its parent and return to the previous step

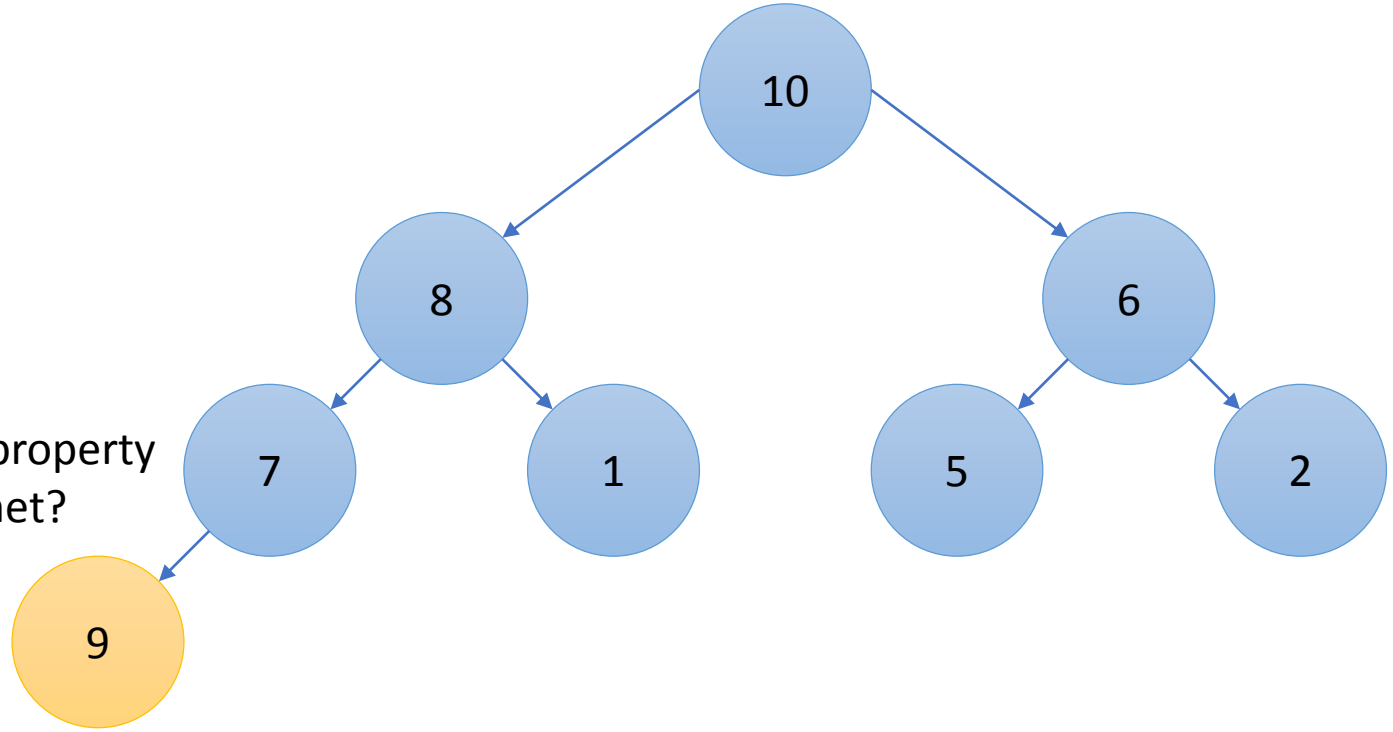
Adding to a Heap

Add this: 9



This is a MAX HEAP

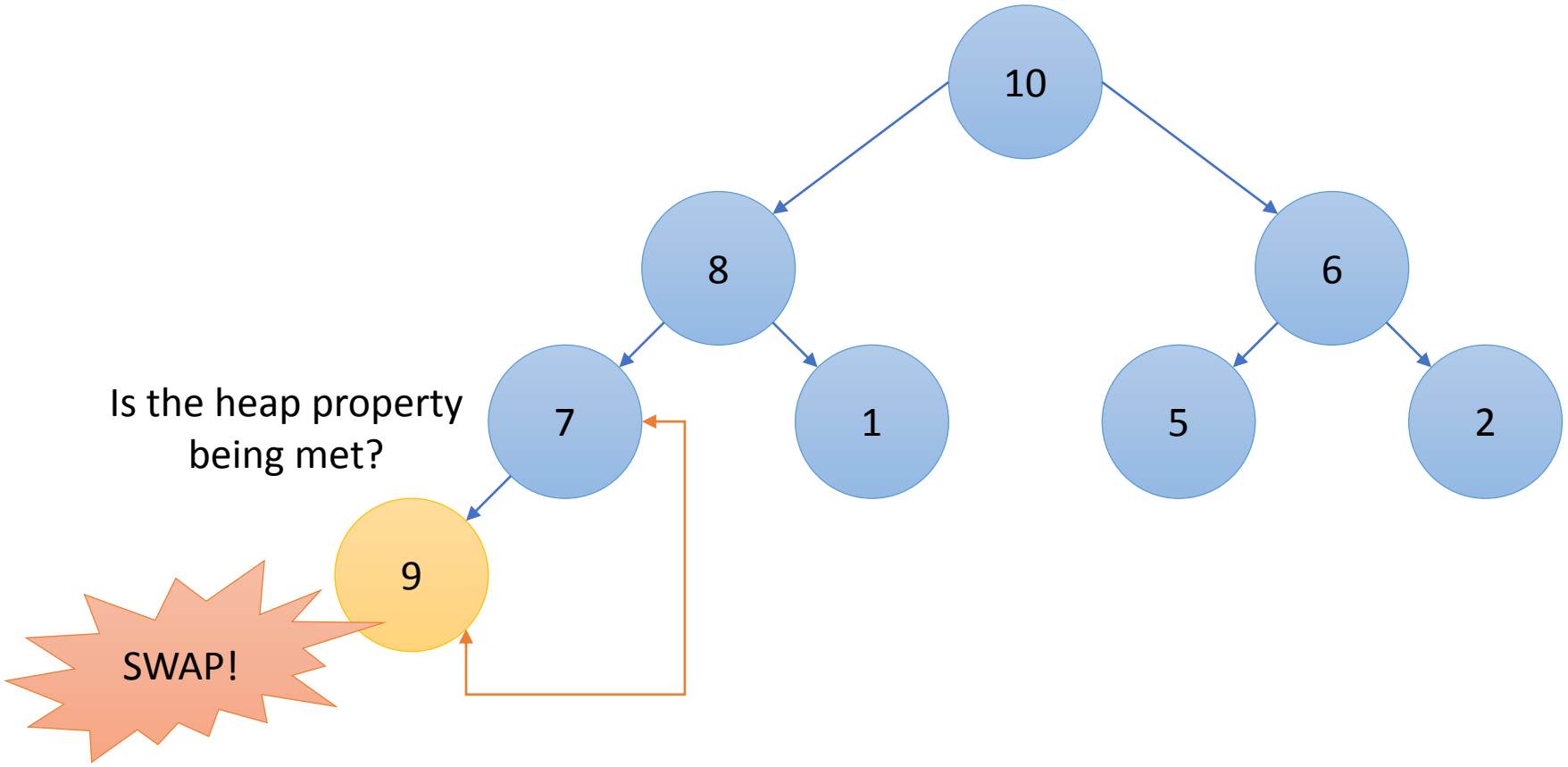
Adding to a Heap



Is the heap property being met?

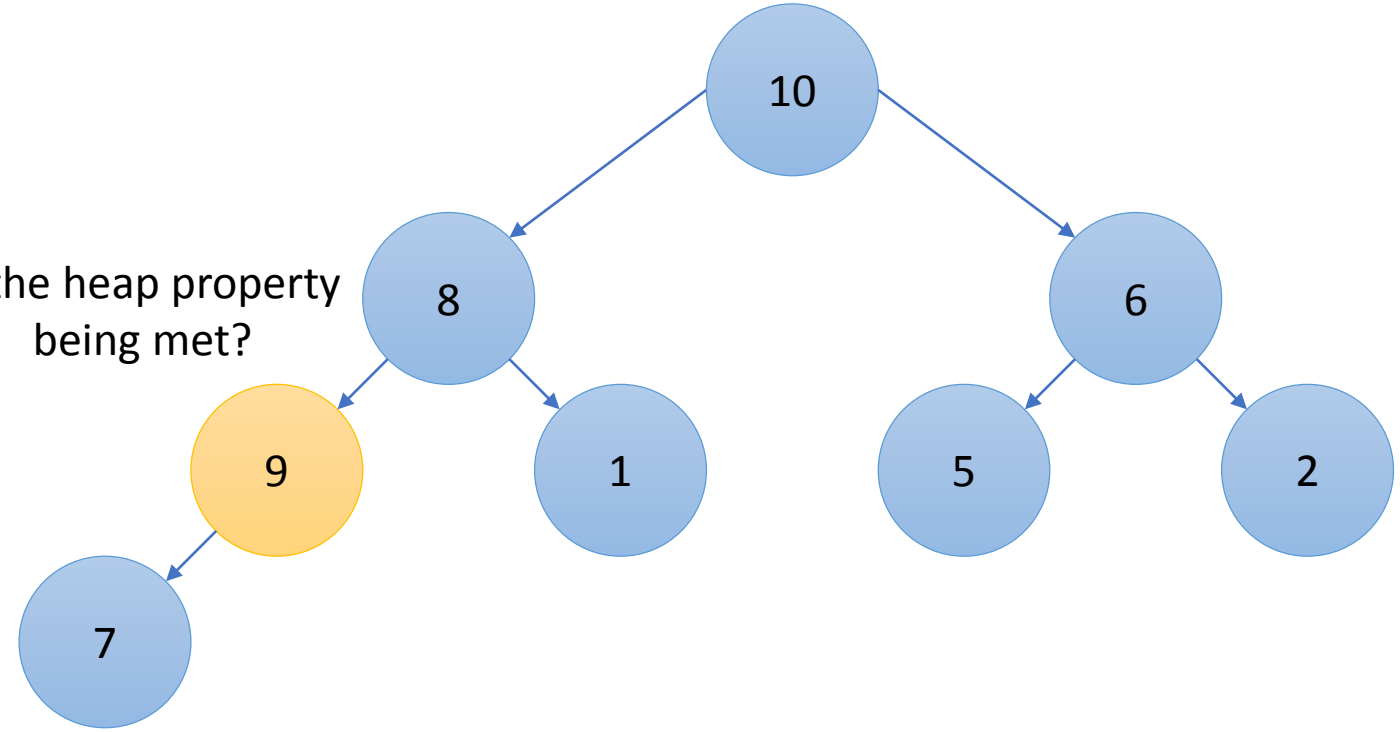
Add the new element in such a way that the Heap remains complete!

Adding to a Heap

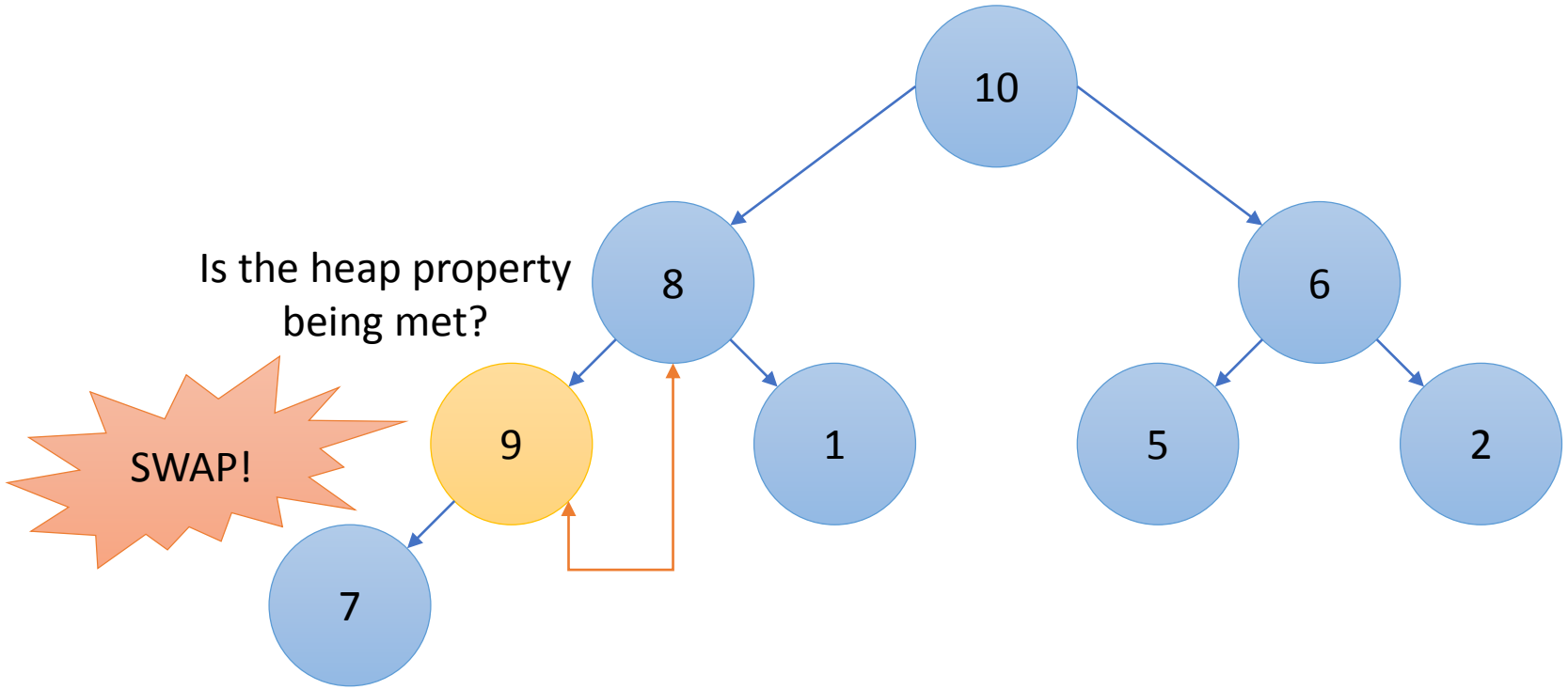


Adding to a Heap

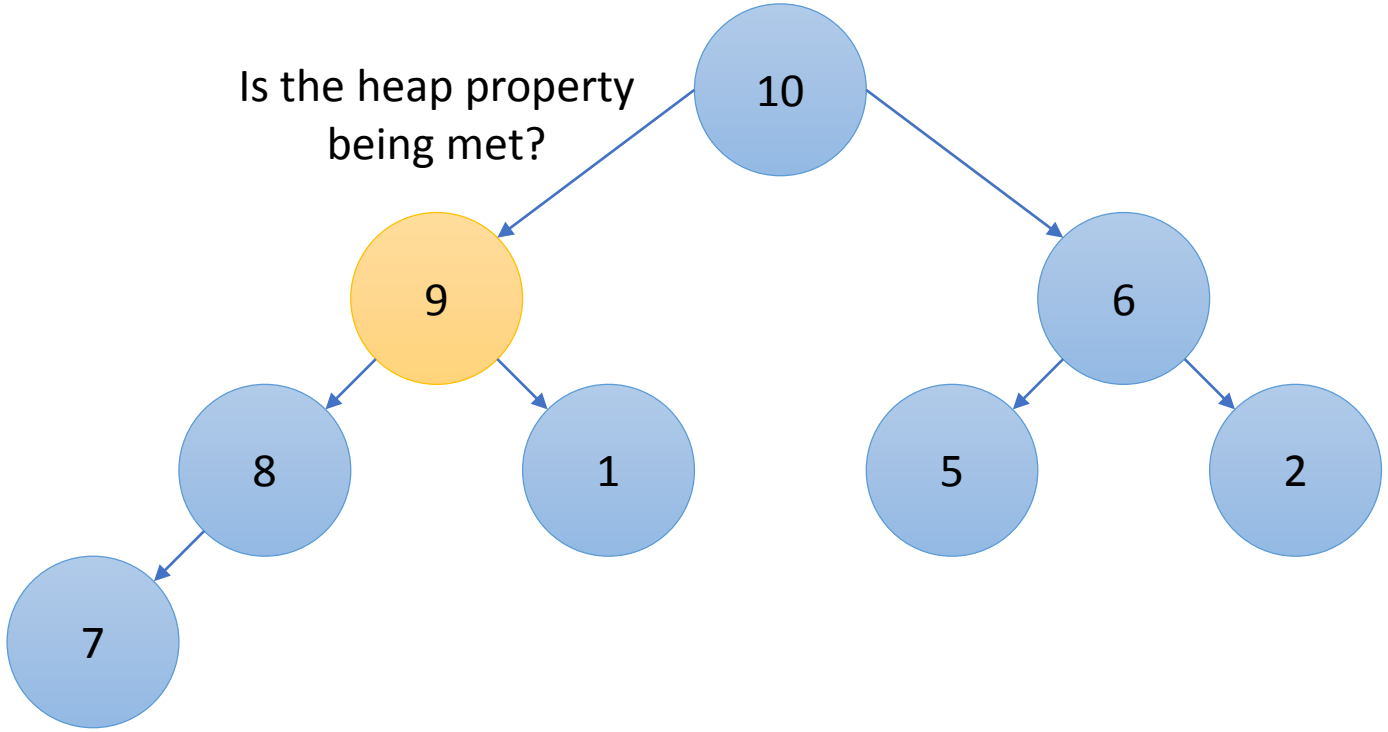
Is the heap property being met?



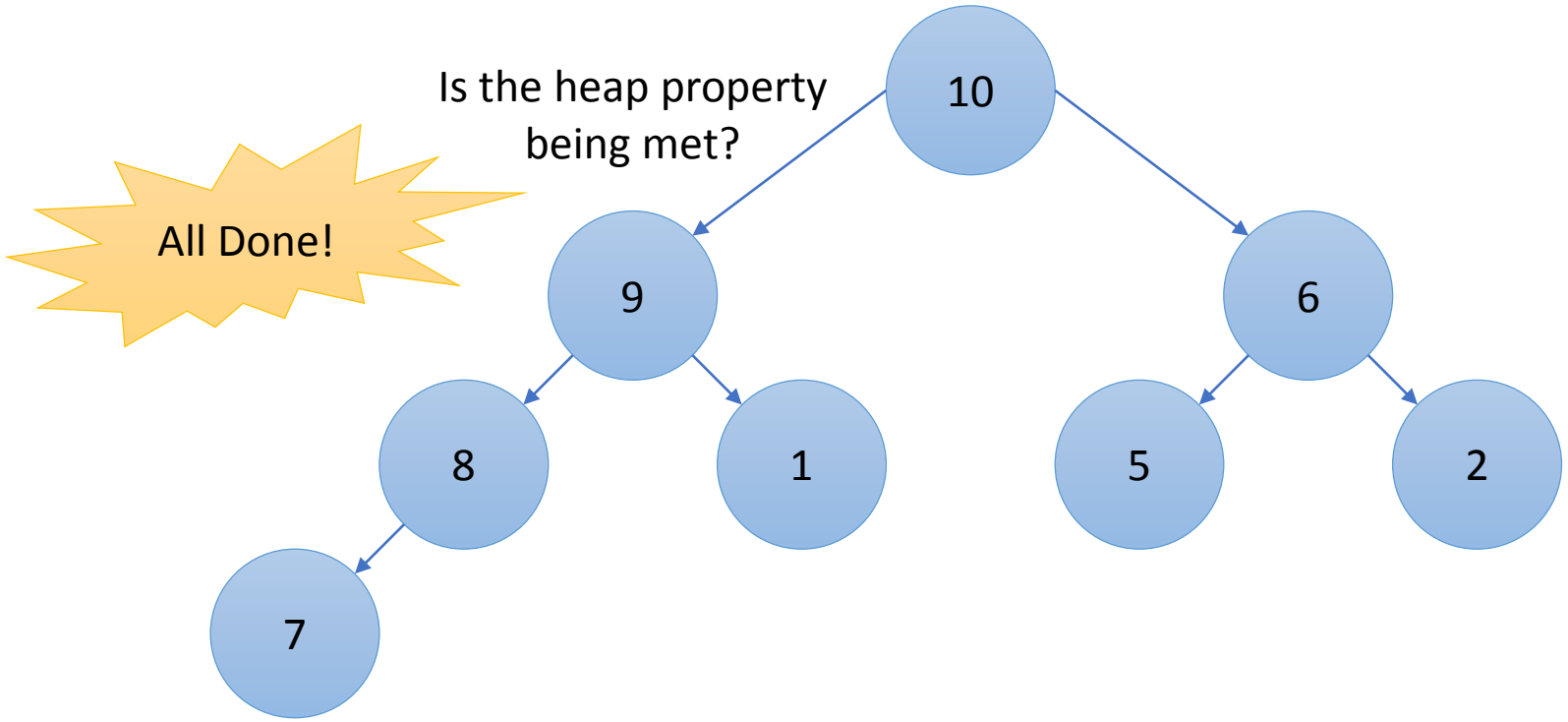
Adding to a Heap



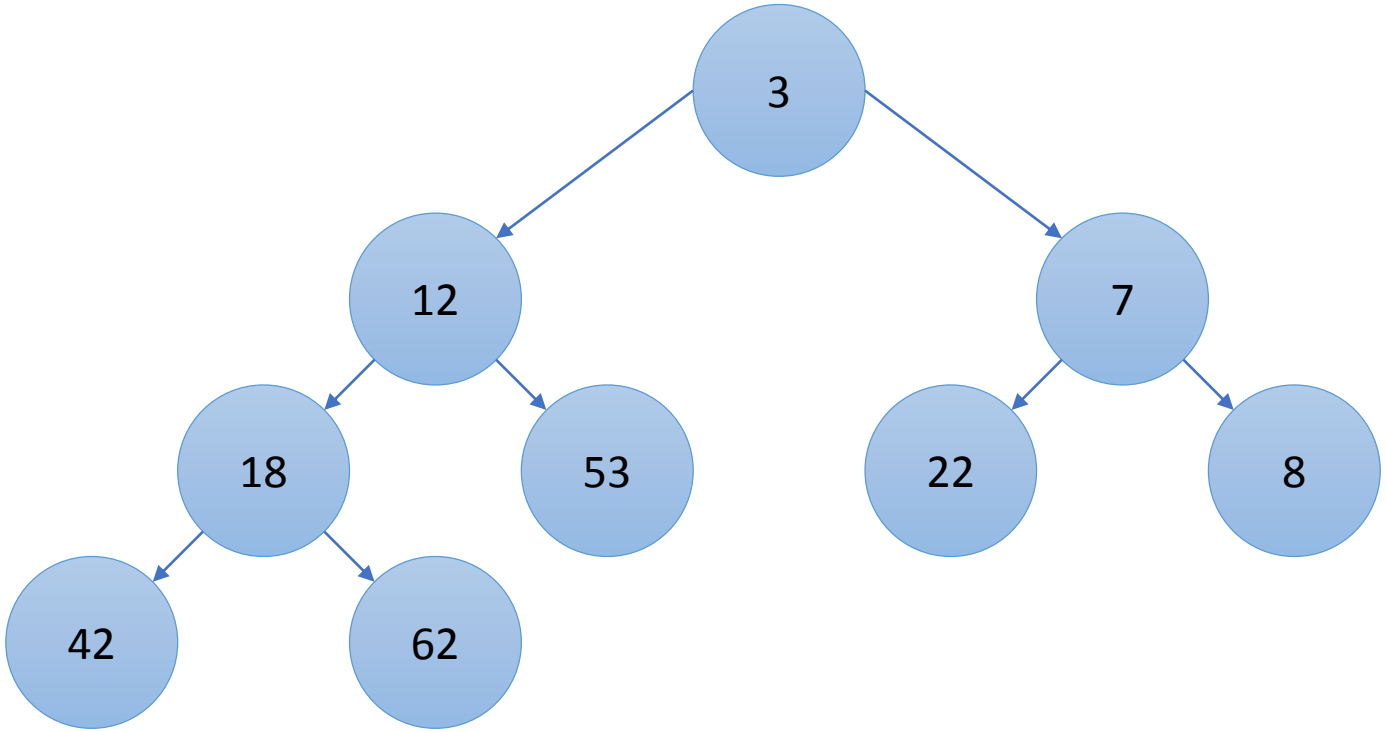
Adding to a Heap



Adding to a Heap

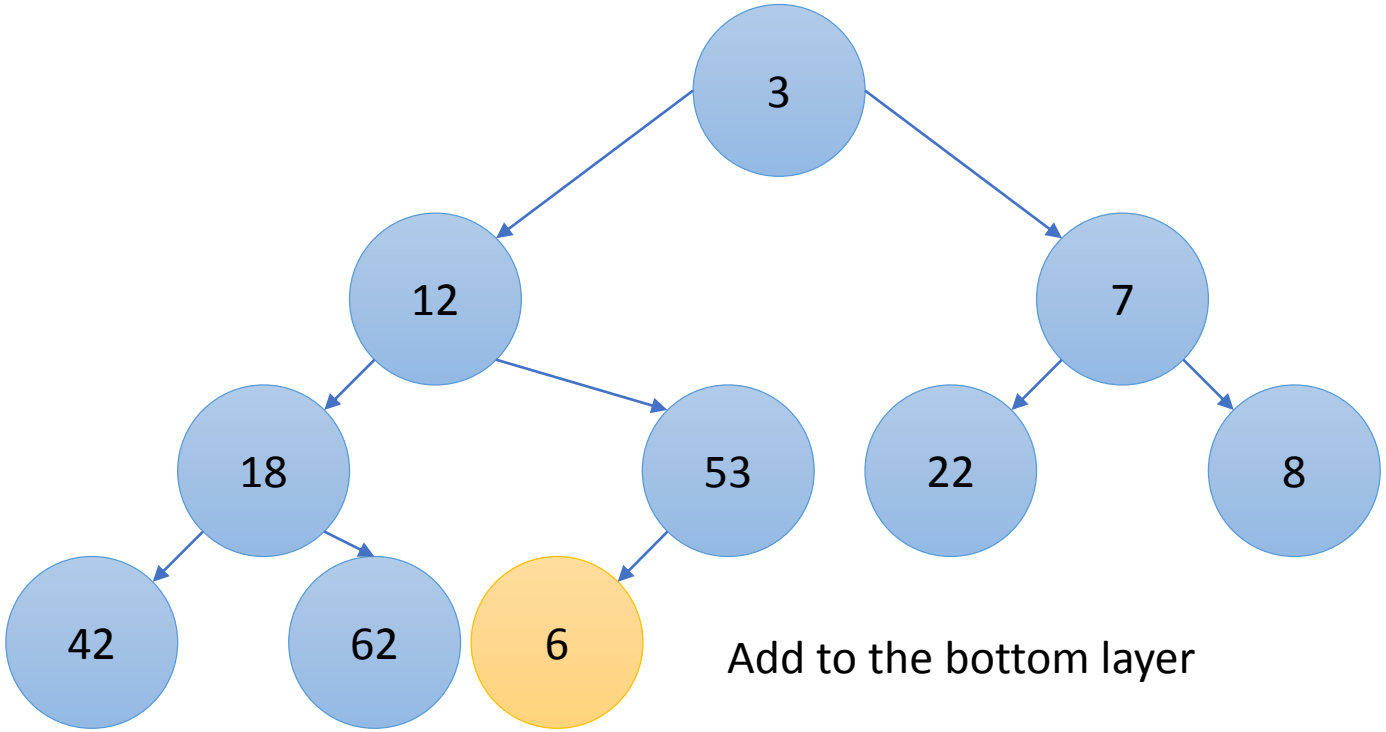


Activity 2: Add the following value to this Min Heap

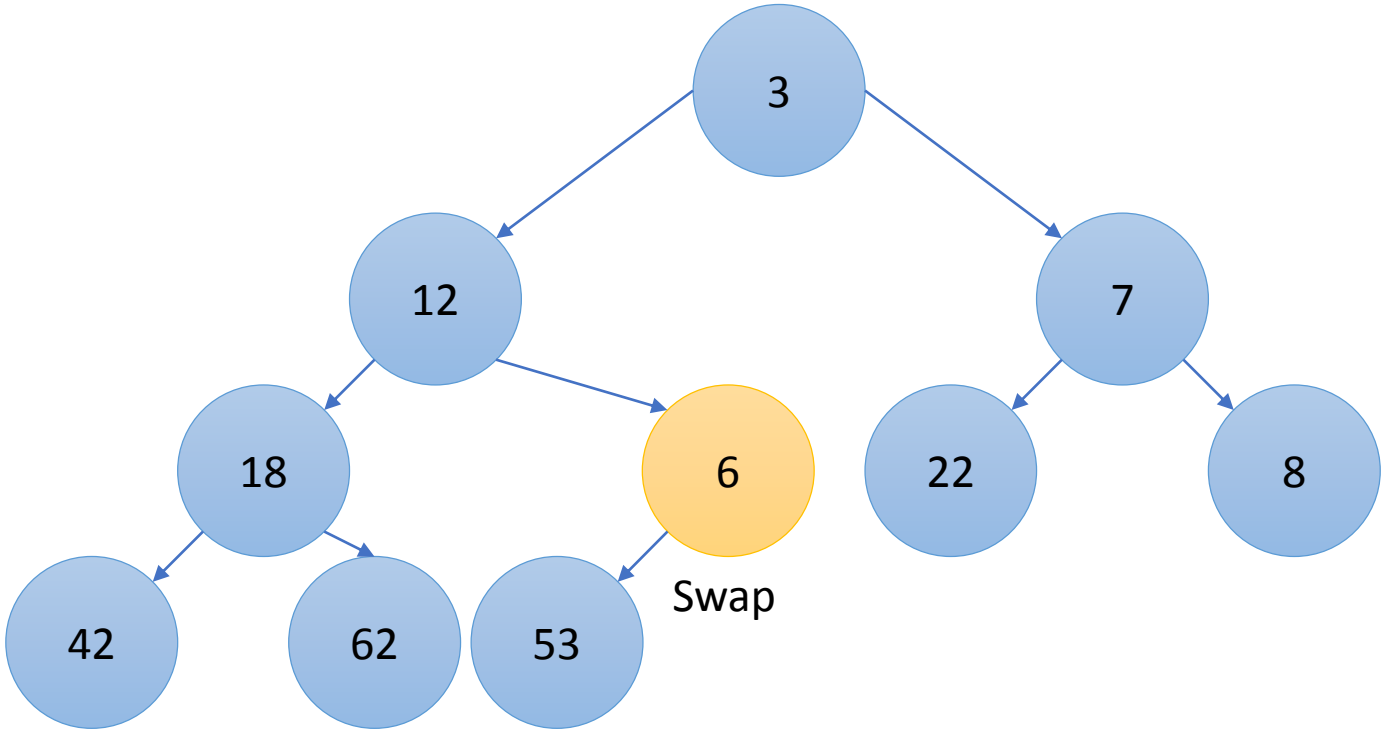


Add this: 6

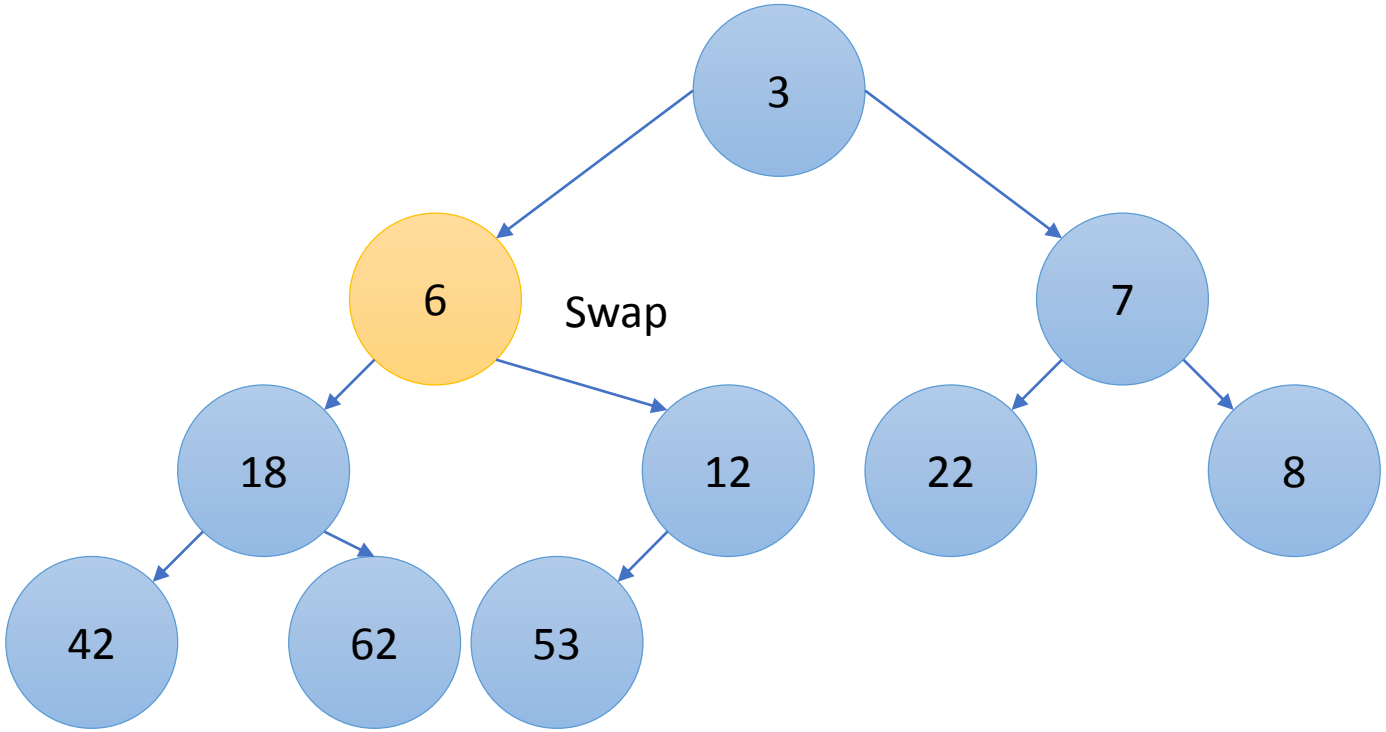
Activity 2: Add the following value to this Min Heap



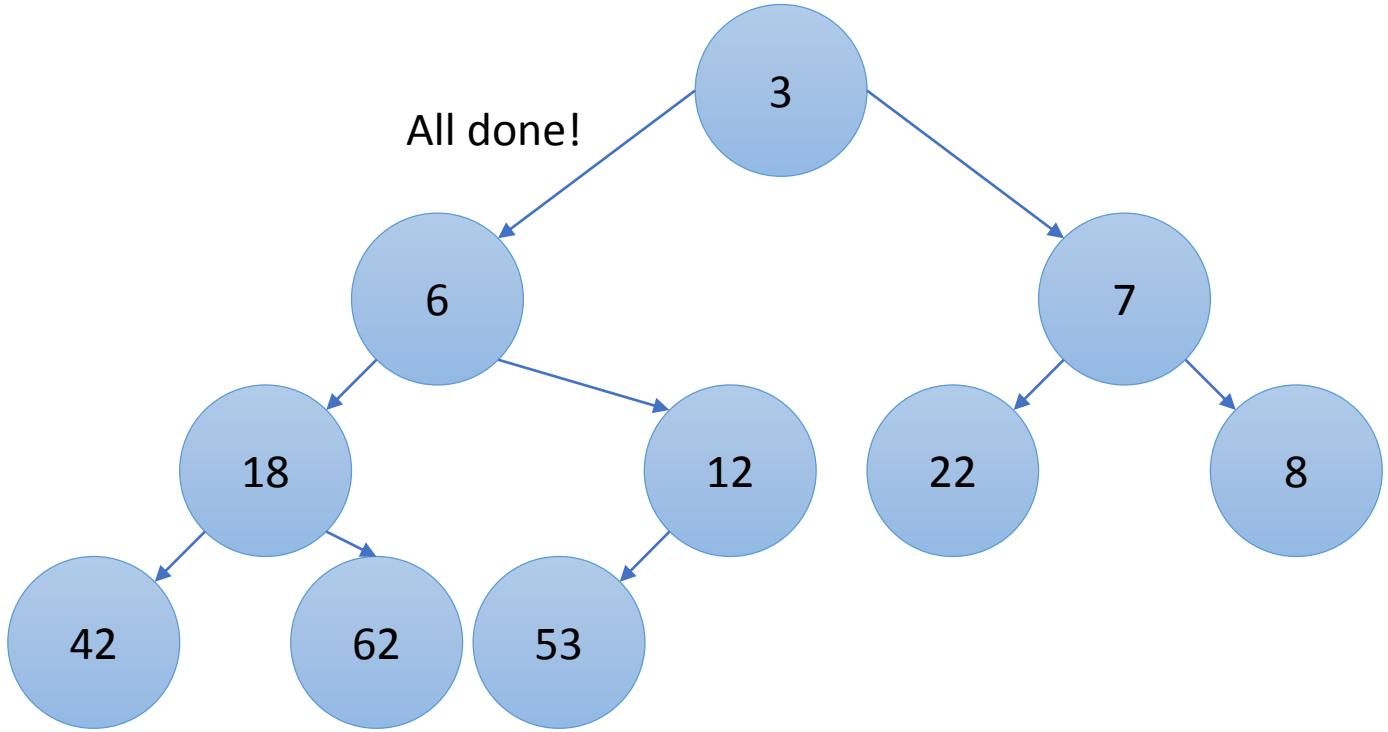
Activity 2: Add the following value to this Min Heap



Activity 2: Add the following value to this Min Heap



Activity 2: Add the following value to this Min Heap

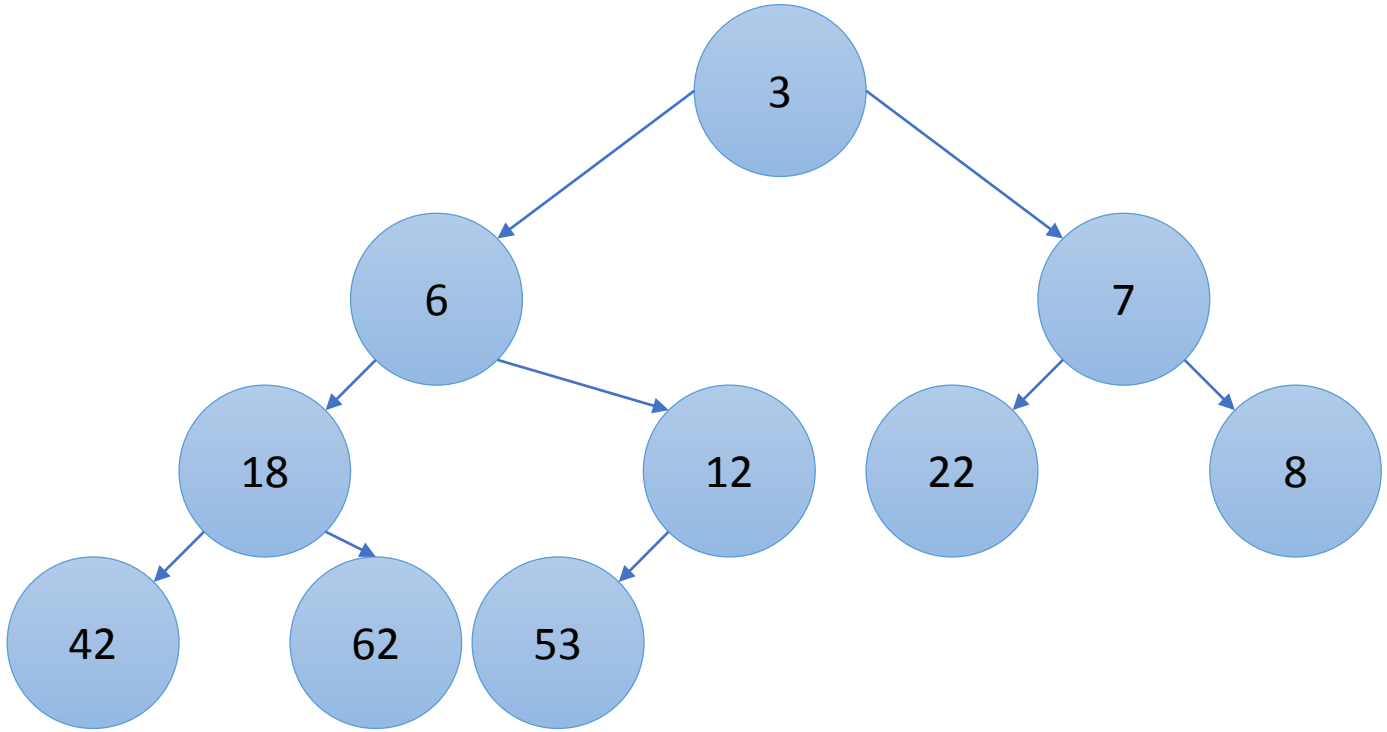


Removing from a Heap

You always remove the root node from a Heap

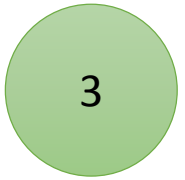
1. Replace the root with the last element on the last level
2. Compare the new root with its children
    1. If the heap property is met, stop
    2. If not, swap the element with one of its children and repeat step 2
       (min-heap swaps with smaller child, max-heap swaps with larger child)

Remove the root from this Min-Heap

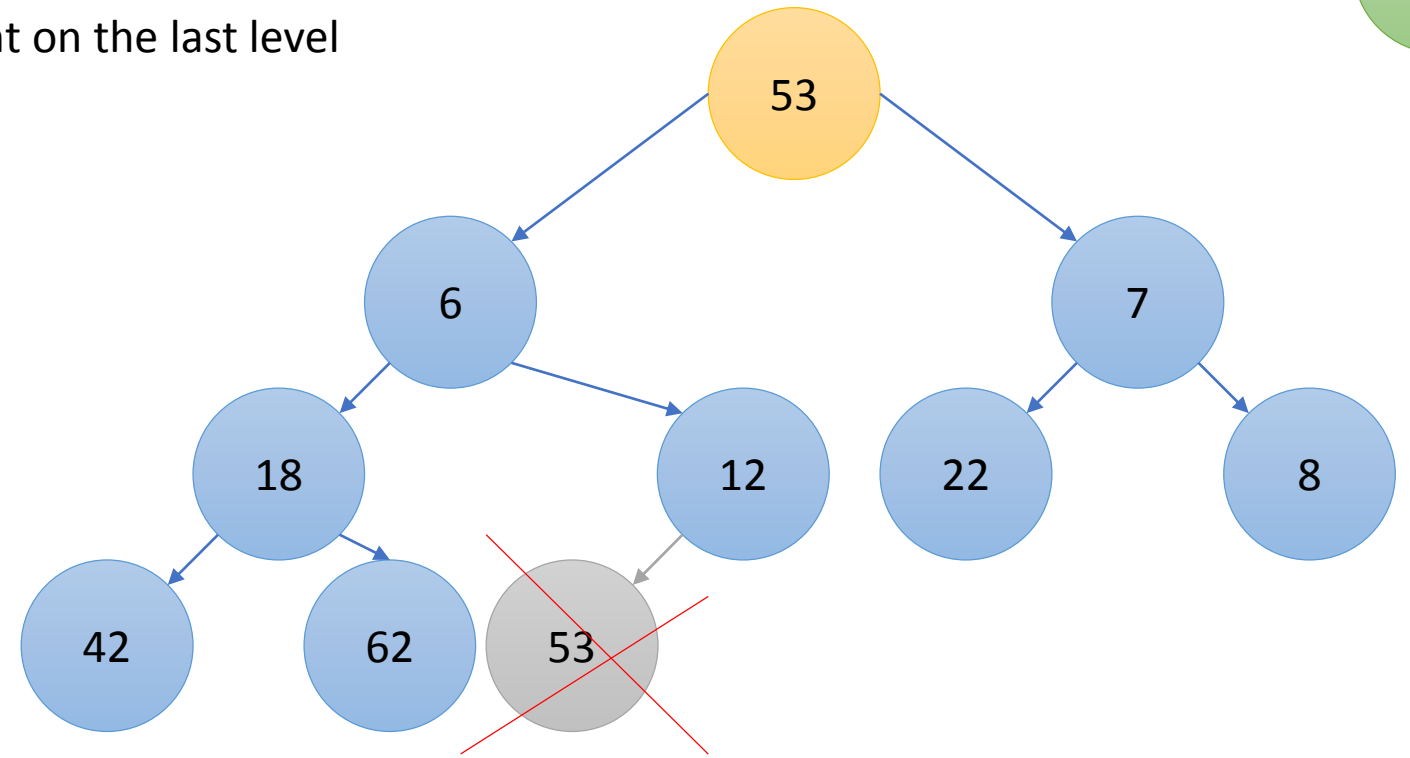


Remove the root from this Min-Heap

removed

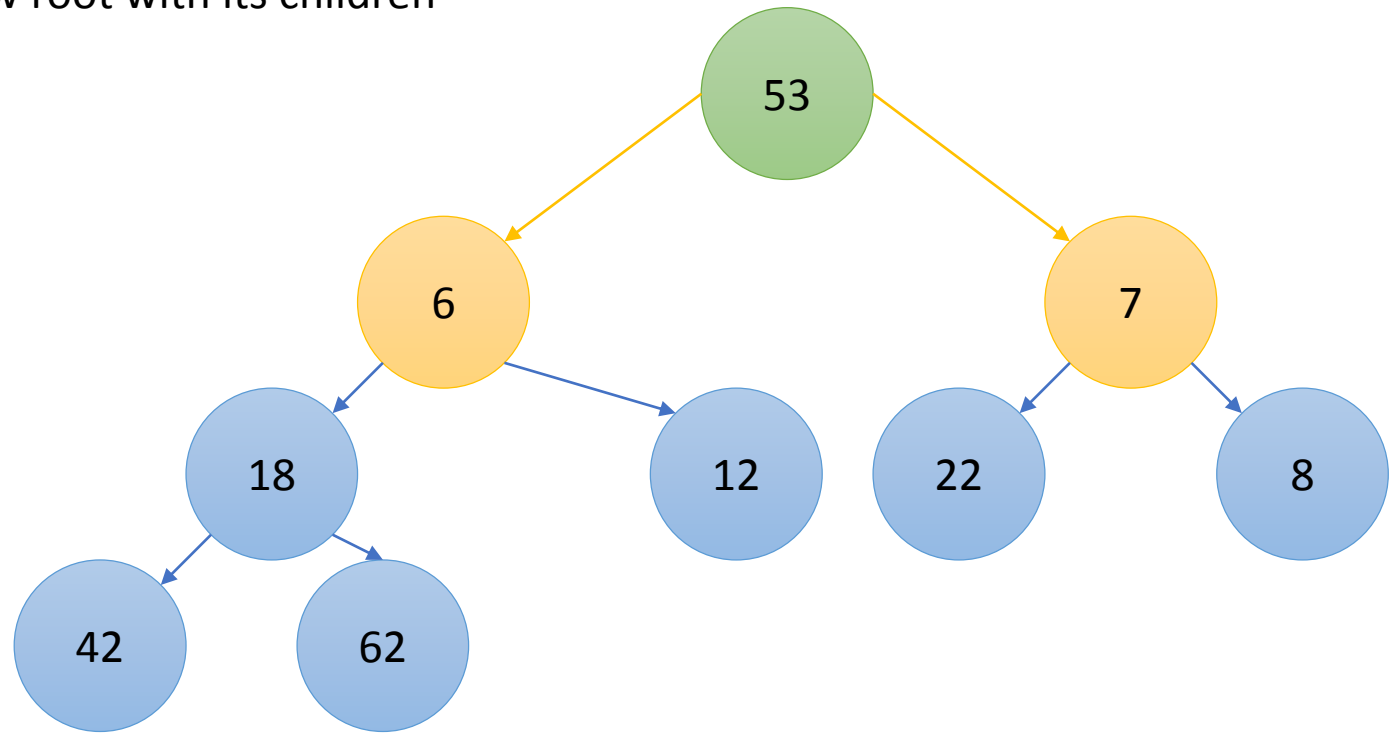


1. Replace the root node with the last element on the last level



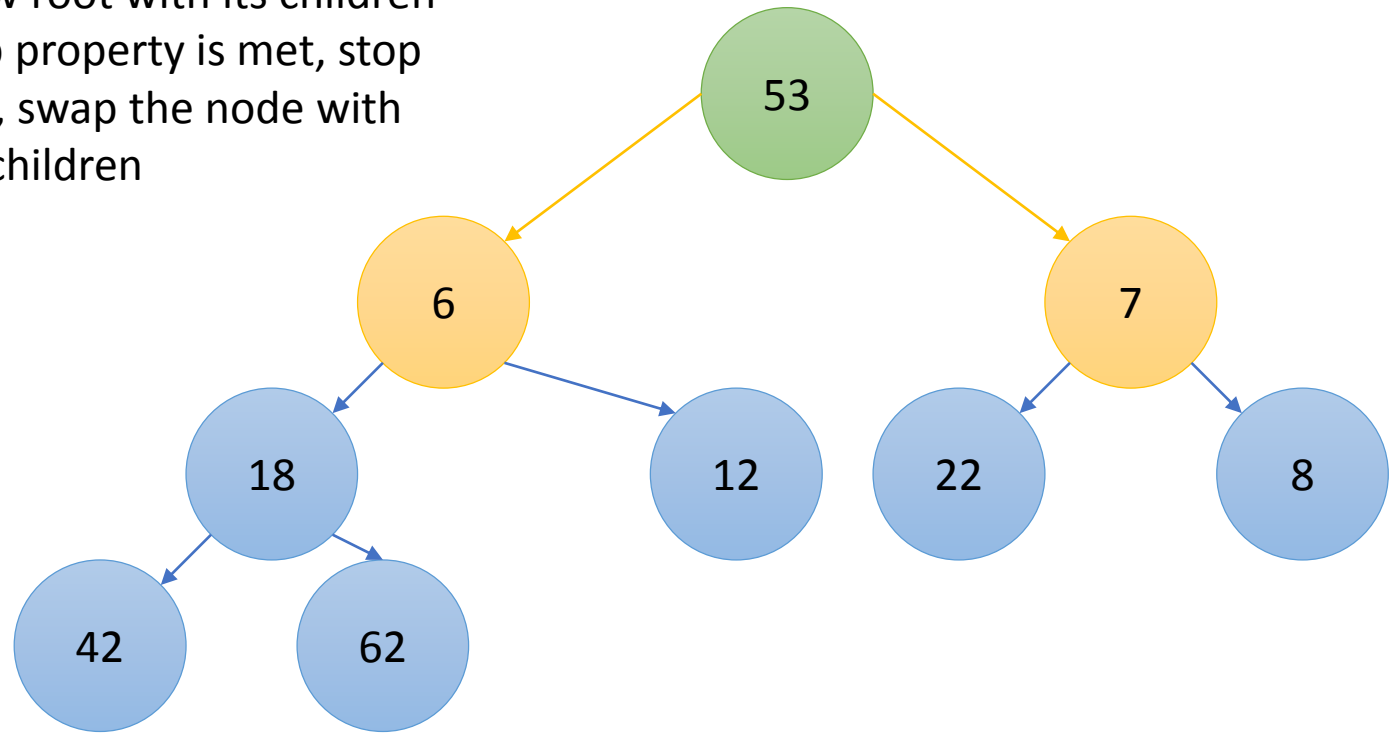
Remove the root from this Min-Heap

2. Compare the new root with its children



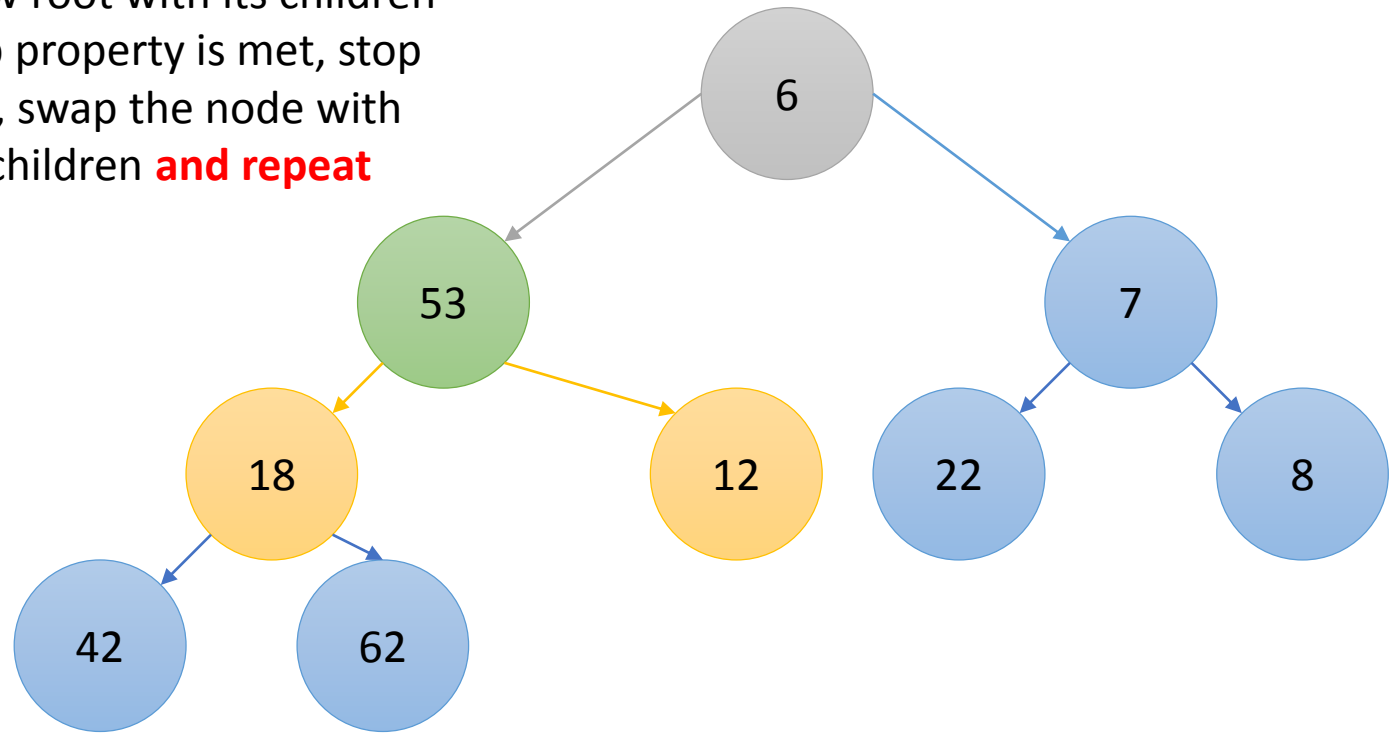
Remove the root from this Min-Heap

2. Compare the new root with its children
    1. If the heap property is met, stop
    2. Otherwise, swap the node with one of its children



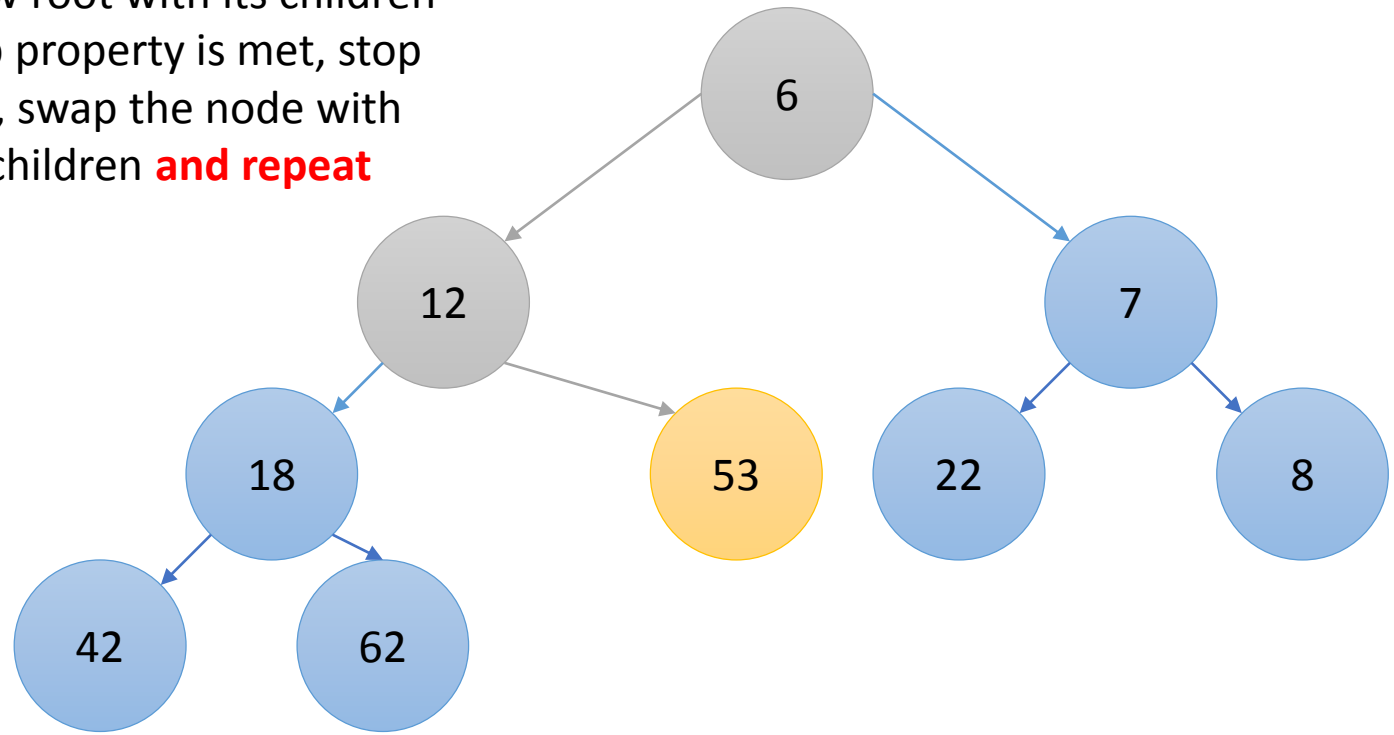
Remove the root from this Min-Heap

2. Compare the new root with its children
    1. If the heap property is met, stop
    2. Otherwise, swap the node with one of its children **and repeat**



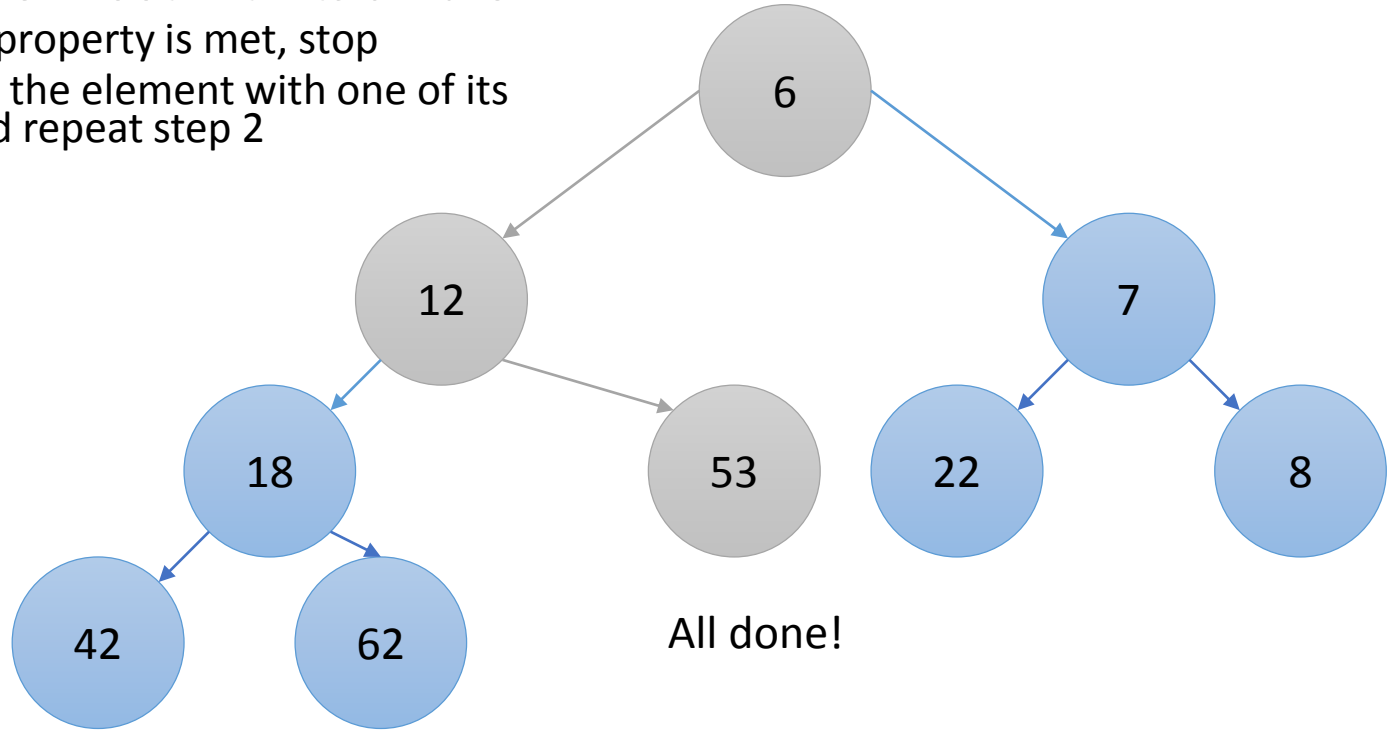
Remove the root from this Min-Heap

2. Compare the new root with its children
    1. If the heap property is met, stop
    2. Otherwise, swap the node with one of its children **and repeat**

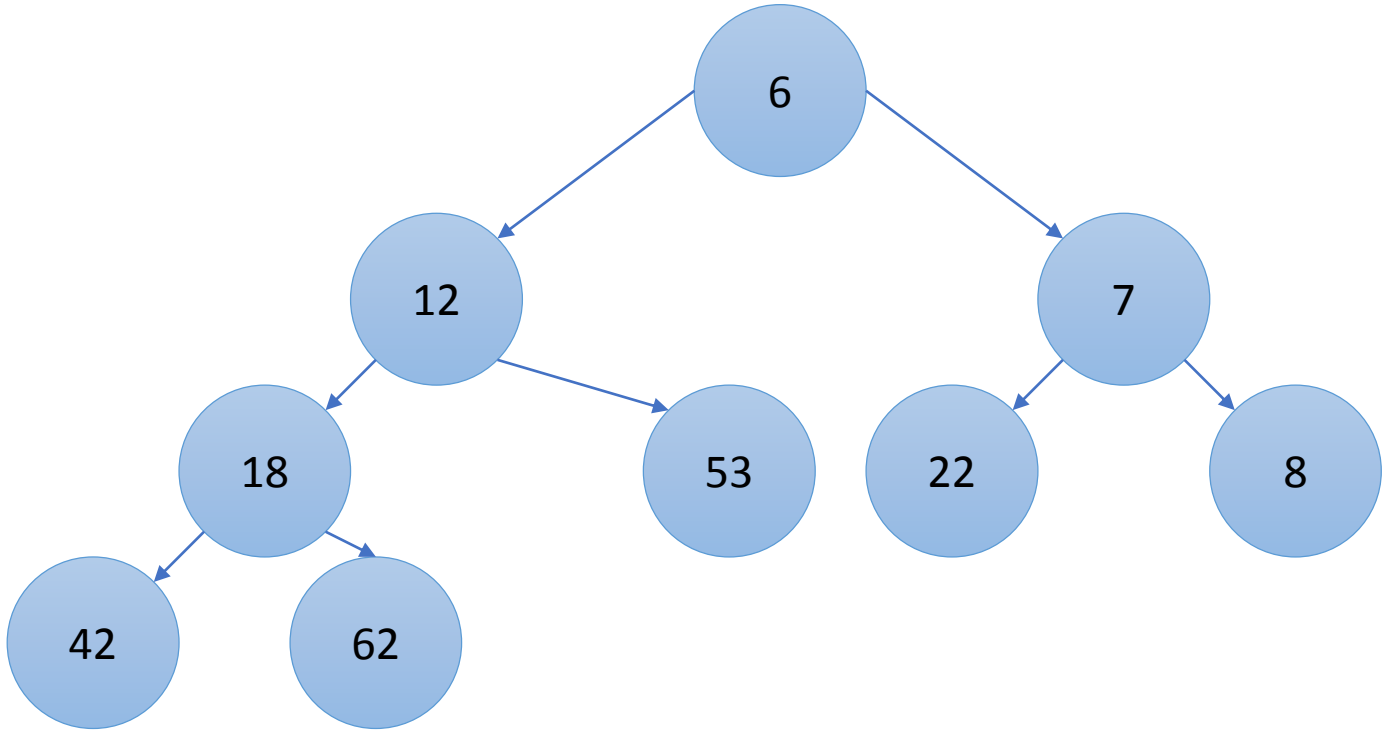


Remove the root from this Min-Heap

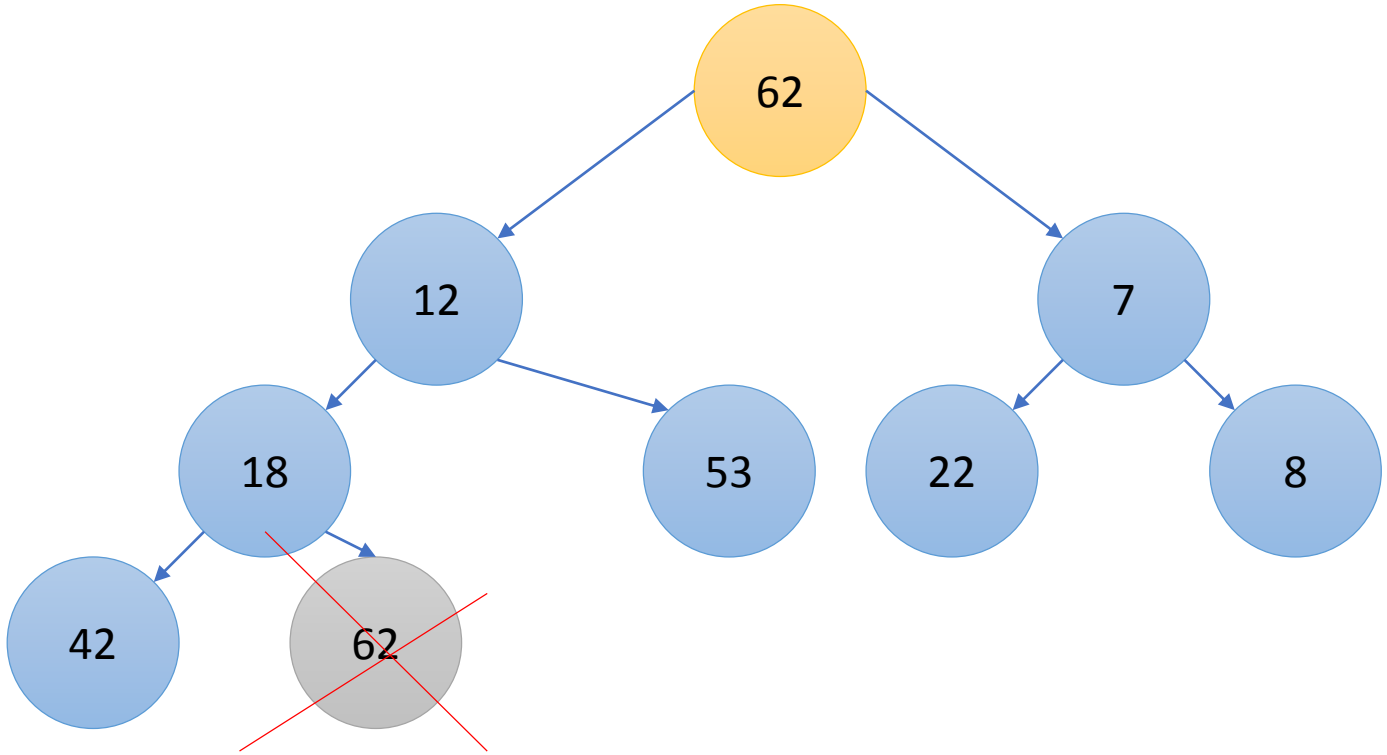
1. Replace the root with the last element on the last level
2. Compare the new root with its children
    1. If the heap property is met, stop
    2. If not, swap the element with one of its children and repeat step 2



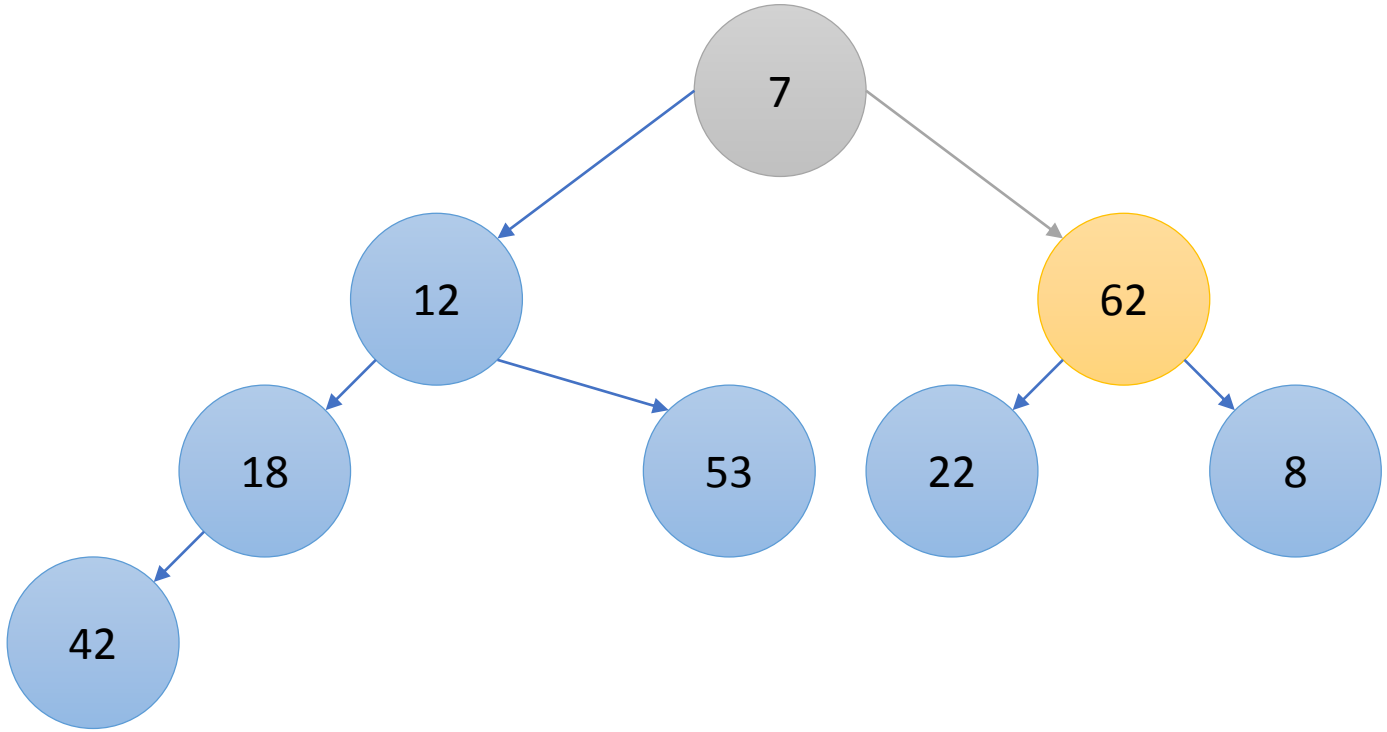
Activity 3: Remove the root from this Min-Heap



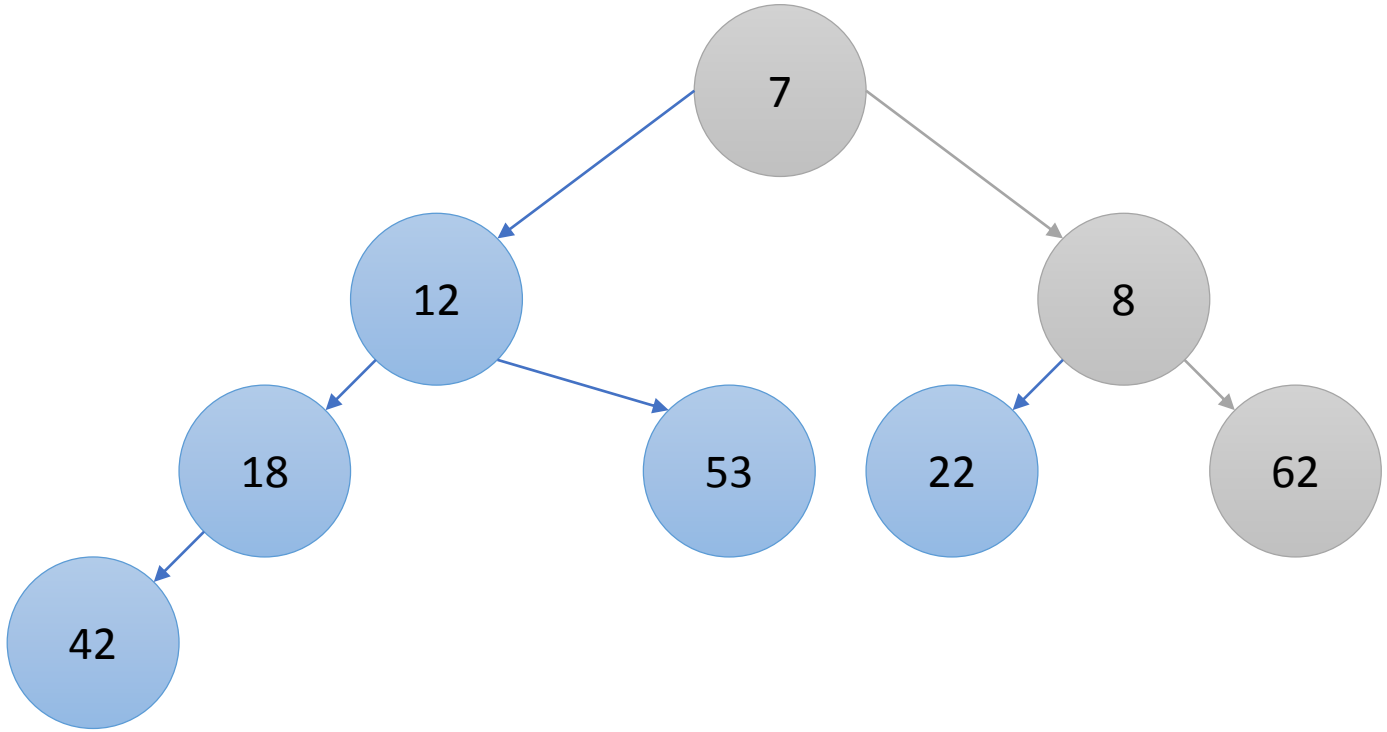
Activity 3: Remove the root from this Min-Heap



Activity 3: Remove the root from this Min-Heap



Activity 3: Remove the root from this Min-Heap



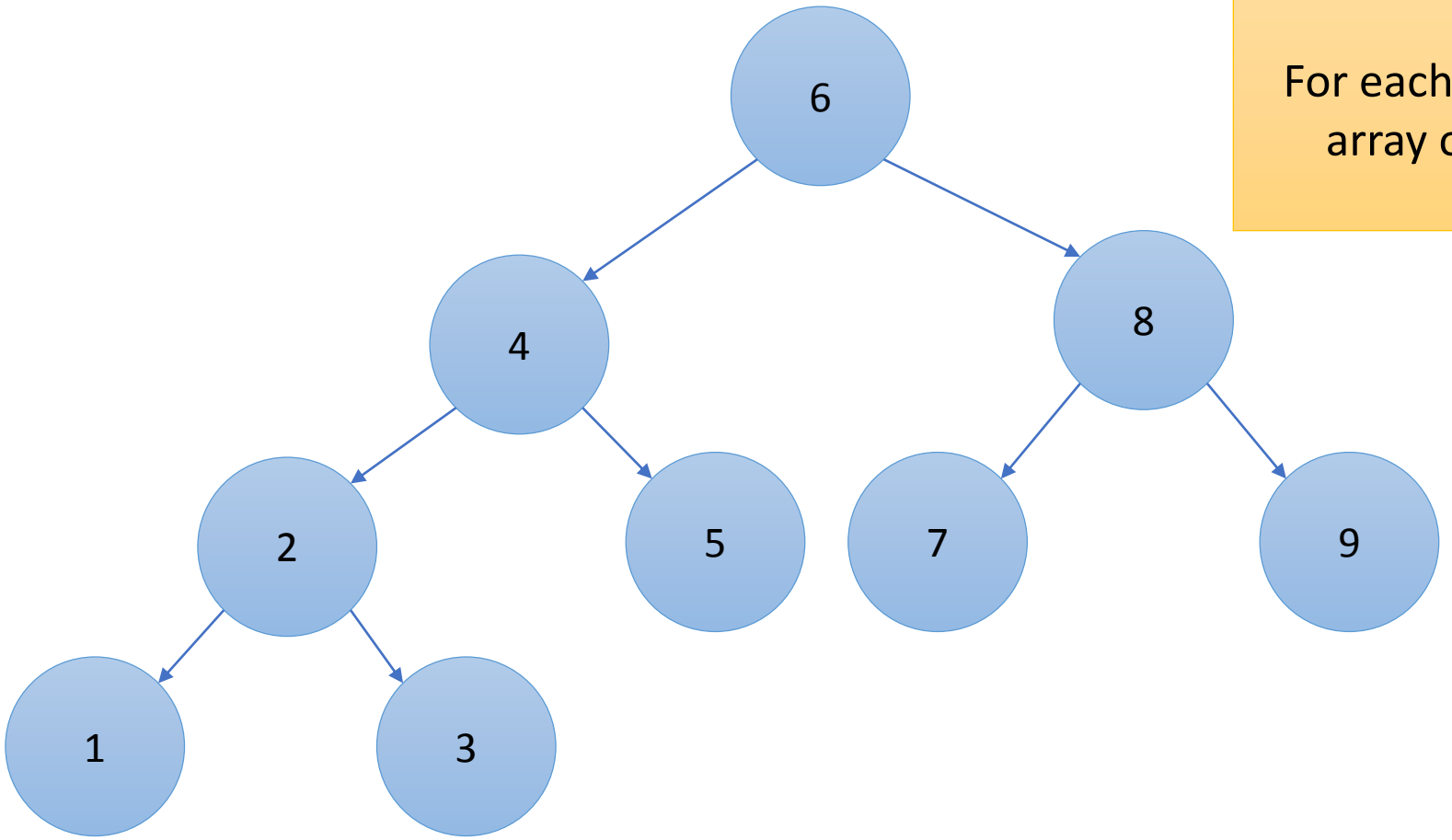
Heapify

If you have an *arbitrary Binary Tree*, how do you turn it into a Heap?

The Heapify Method

For each element in reverse-array order, sink it down

Convert a Complete Tree into a Min-Heap using Heapify



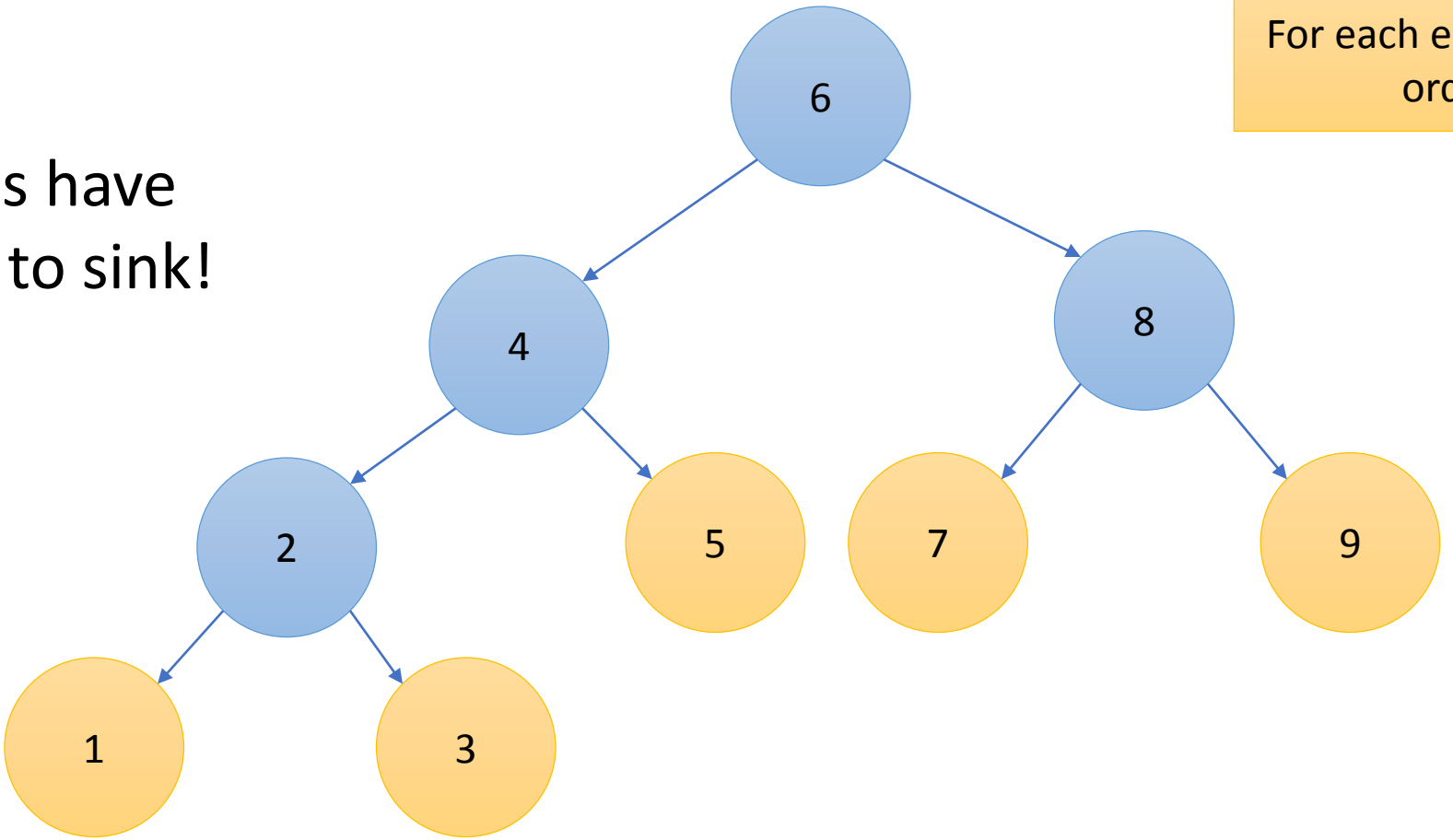
For each element in reverse-array order, sink it down

0	1	2	3	4	5	6	7	8
6	4	8	2	5	7	9	1	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

Leaf nodes have no where to sink!



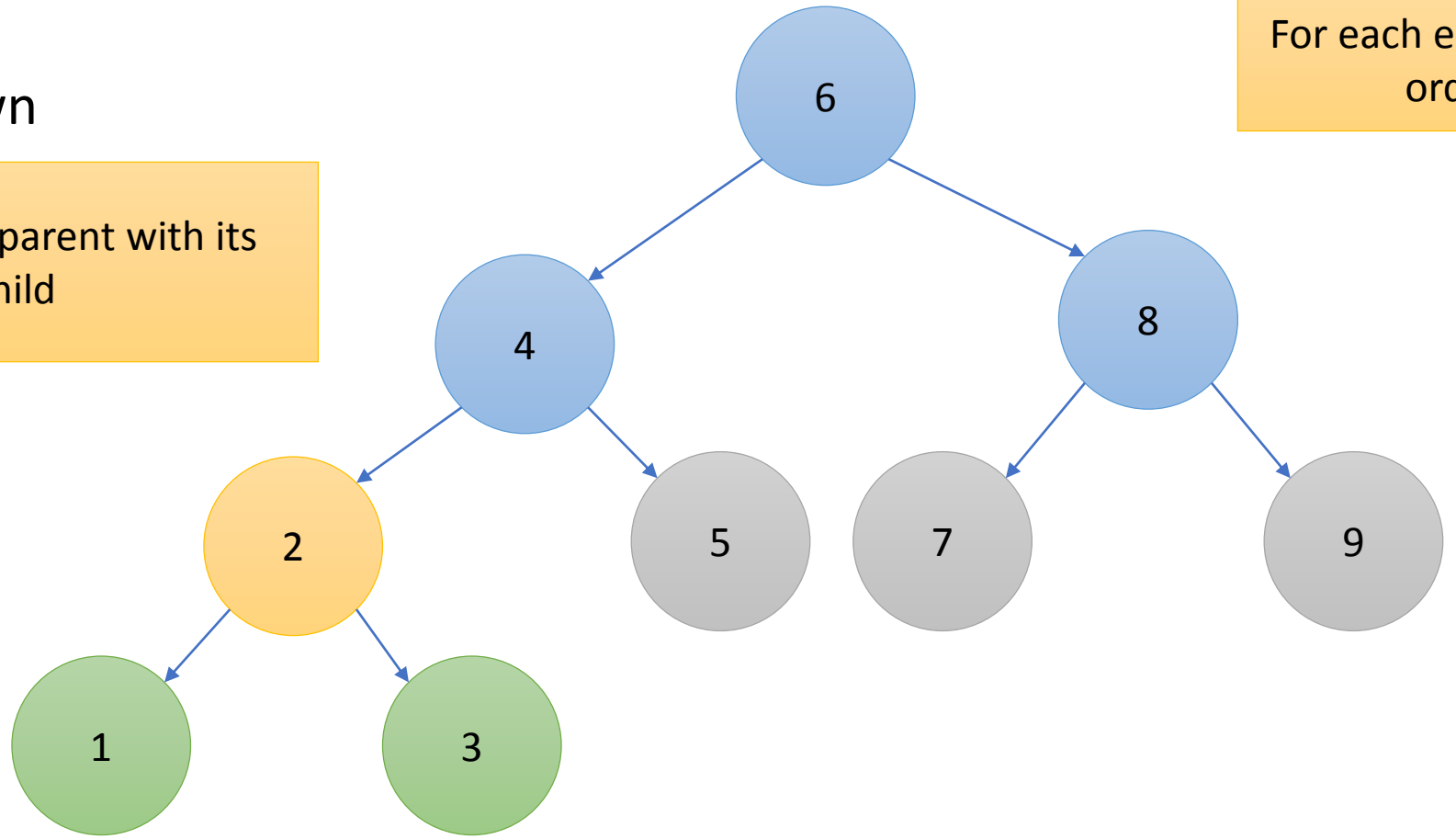
0	1	2	3	4	5	6	7	8
6	4	8	2	5	7	9	1	3

Convert a Complete Tree into a Min-Heap using Heapify

Sink 2 down

Sink: Swap the parent with its smallest child

For each element in reverse-array order, sink it down



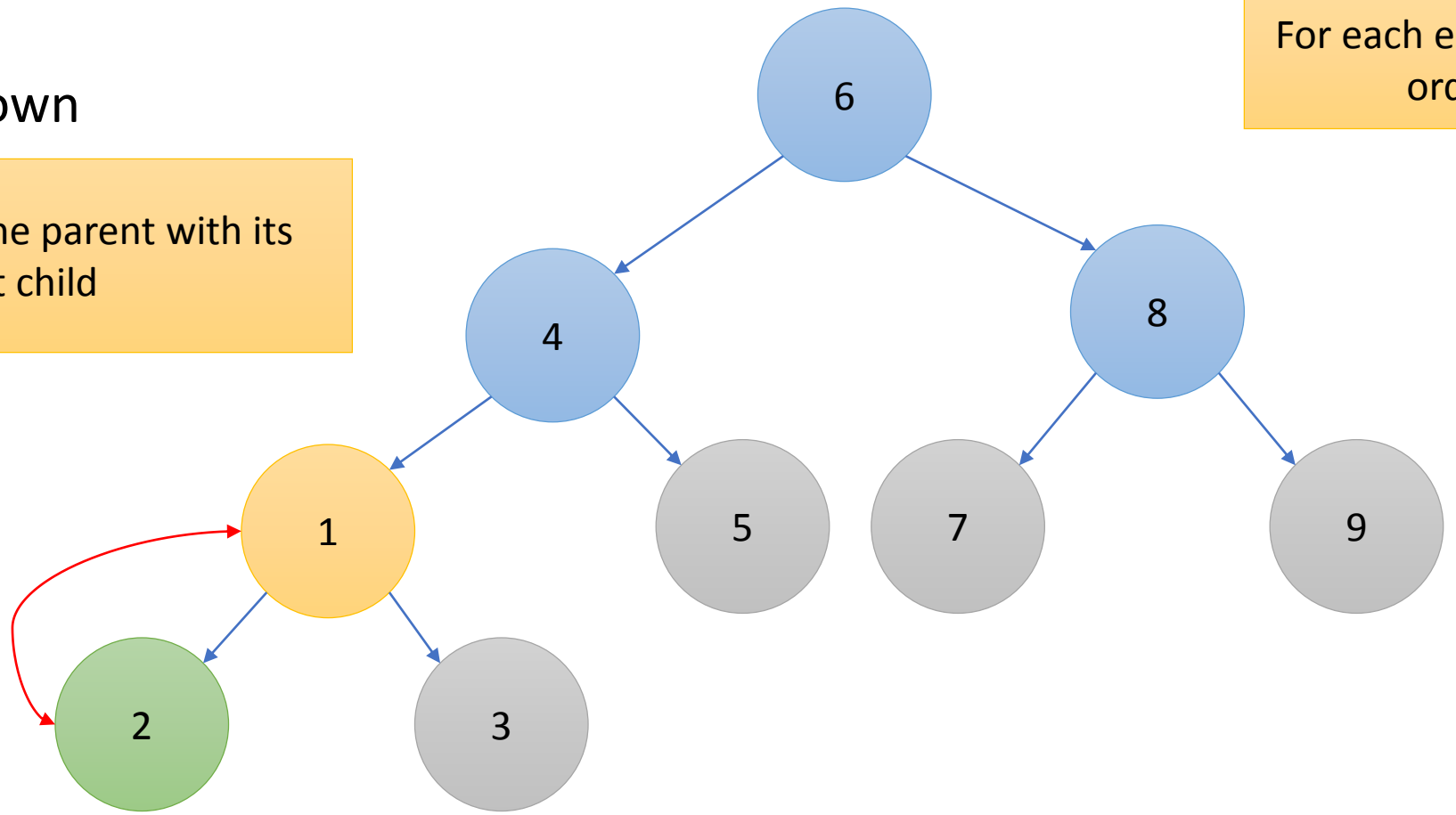
0	1	2	3	4	5	6	7	8
6	4	8	2	5	7	9	1	3

Convert a Complete Tree into a Min-Heap using Heapify

Sink 2 down

Sink: Swap the parent with its smallest child

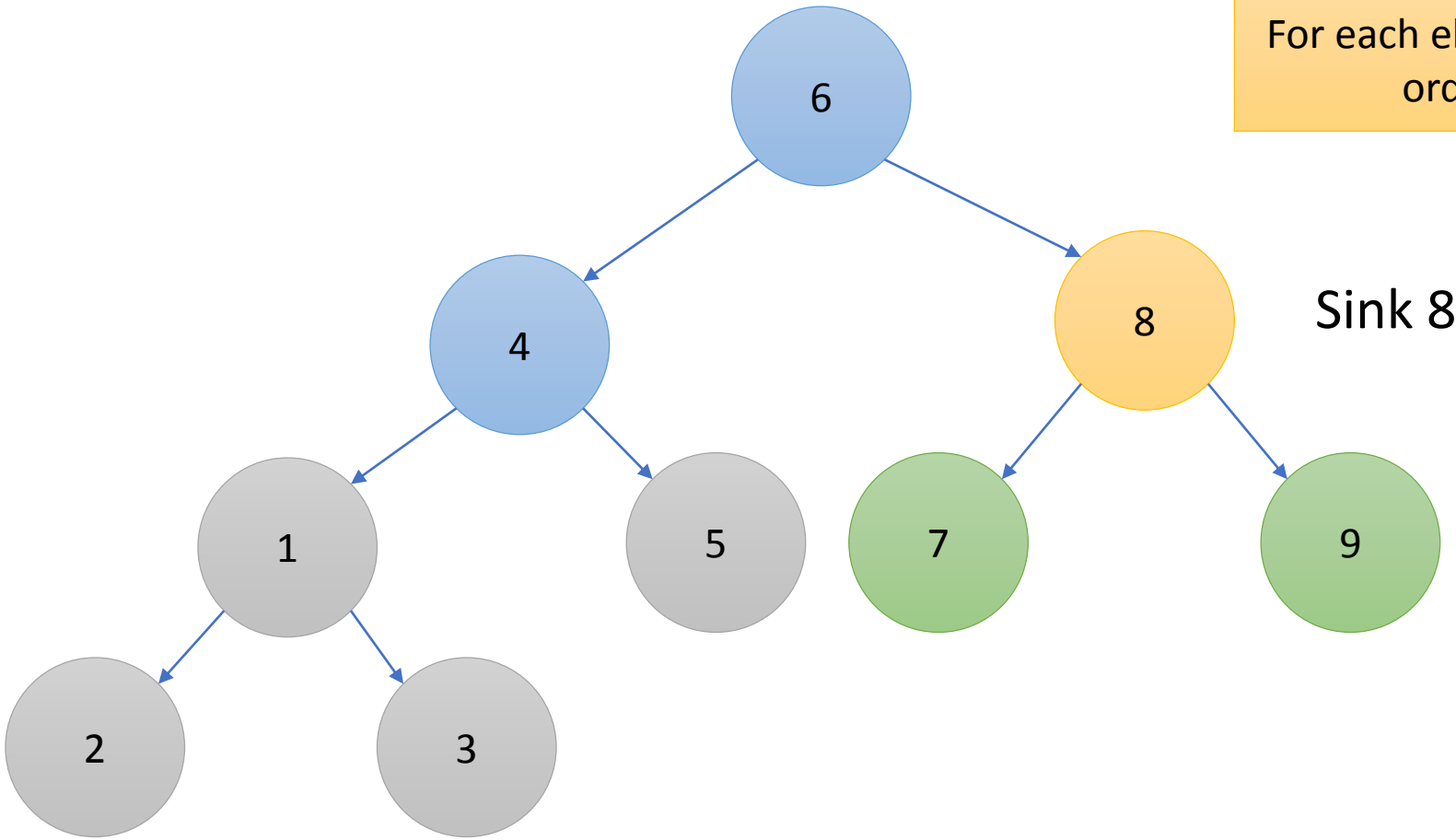
For each element in reverse-array order, sink it down



0	1	2	3	4	5	6	7	8
6	4	8	1	5	7	9	2	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

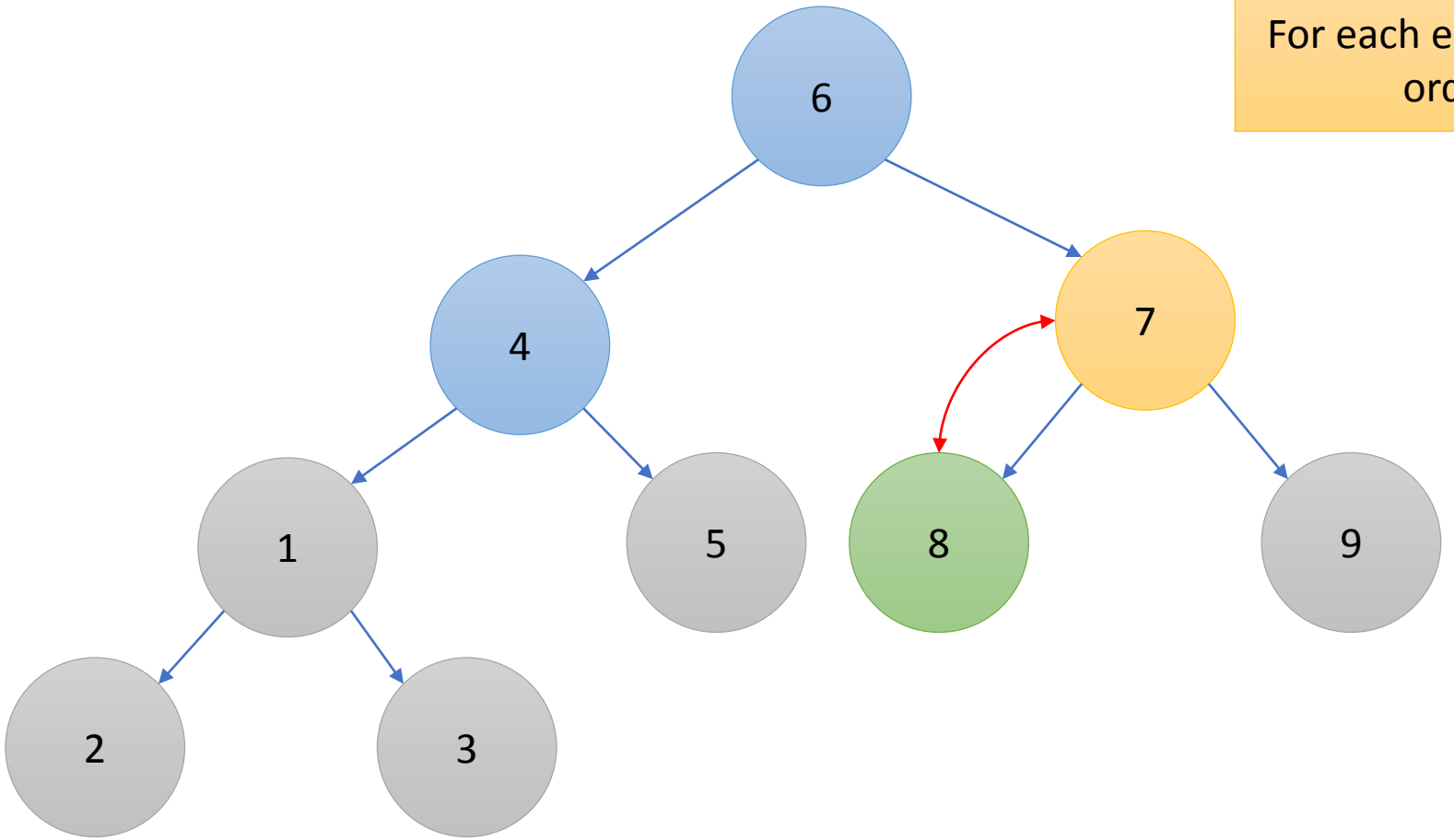


Sink 8 down

0	1	2	3	4	5	6	7	8
6	4	8	1	5	7	9	2	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

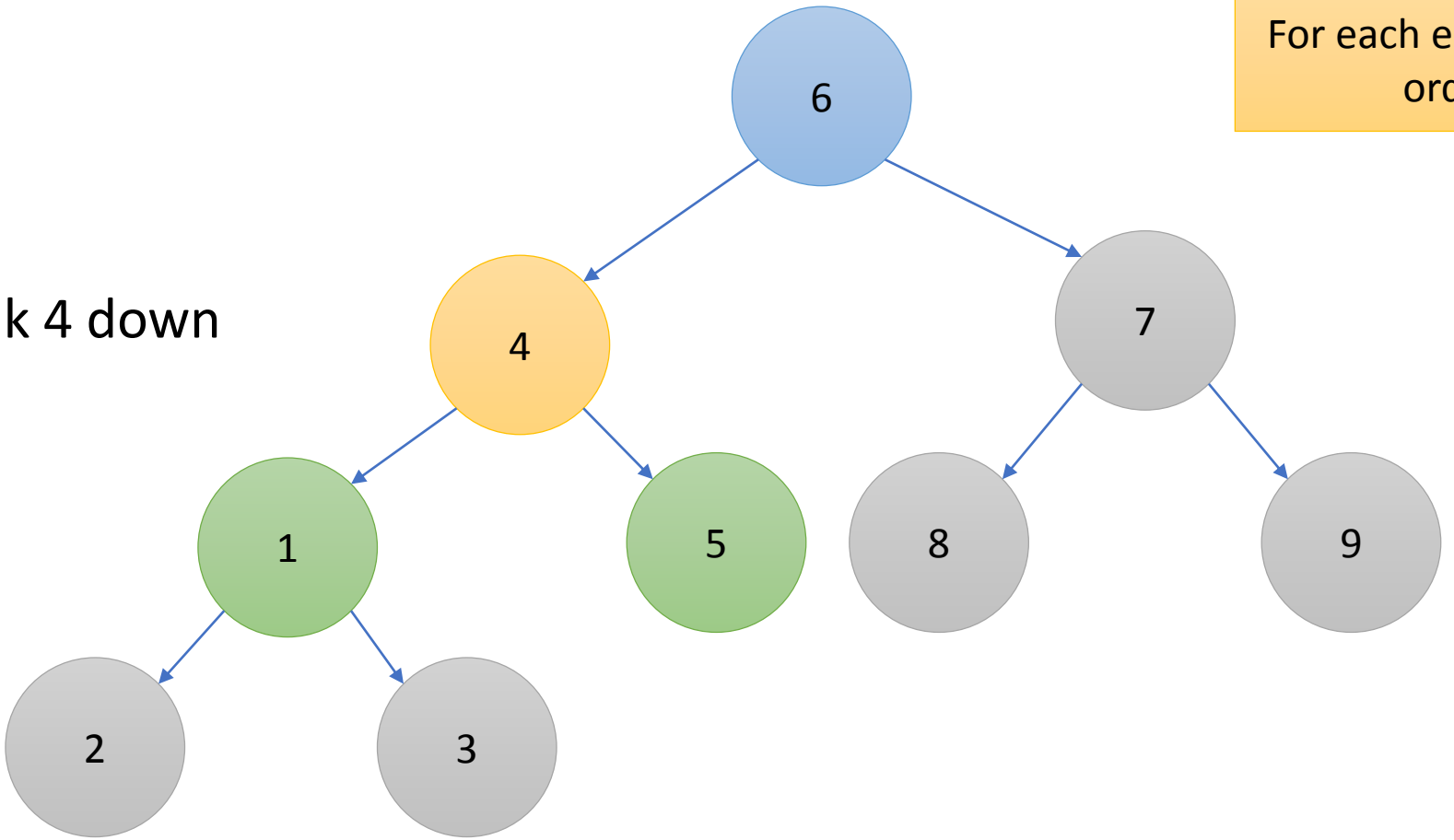


0	1	2	3	4	5	6	7	8
6	4	7	1	5	8	9	2	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

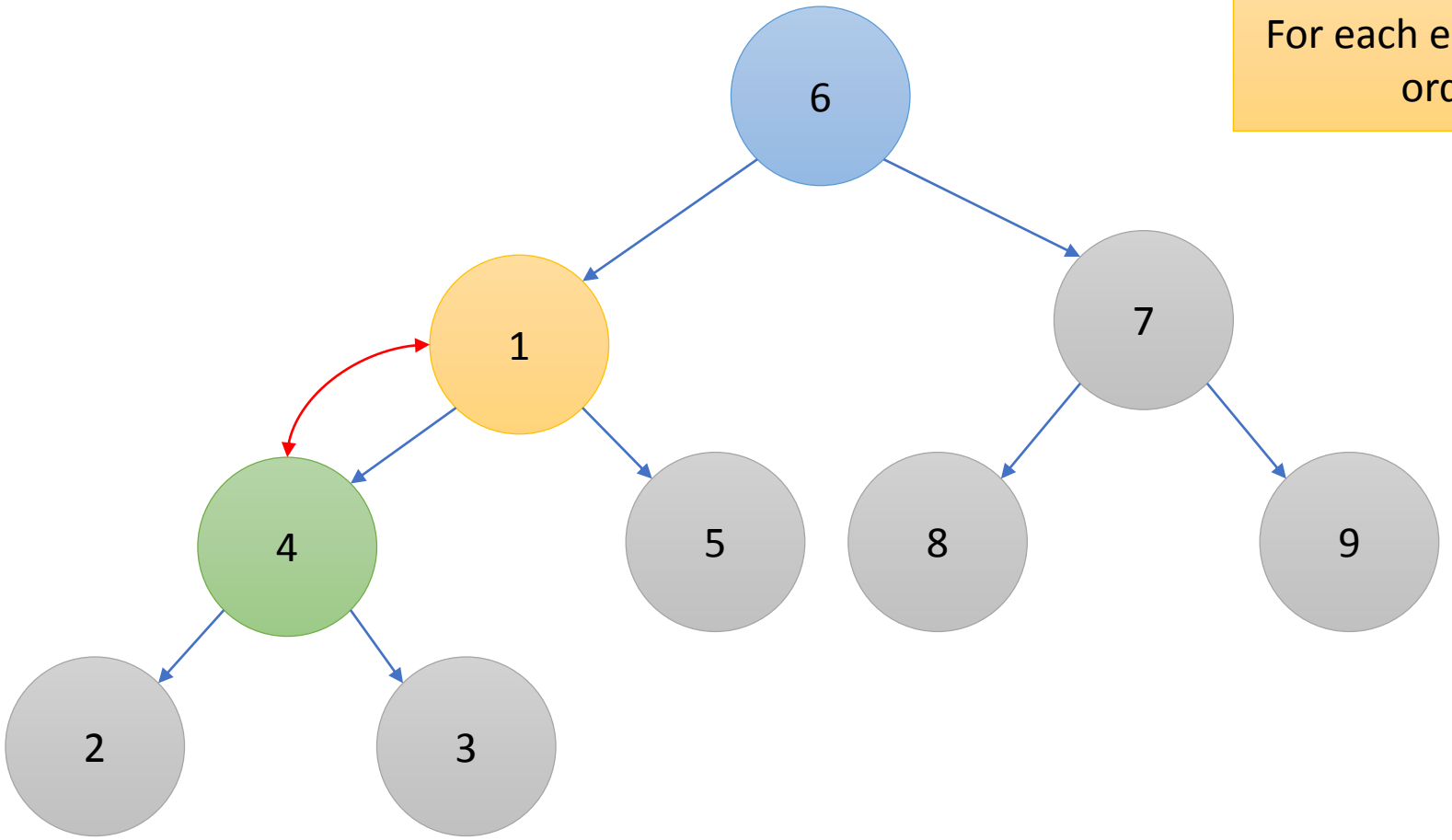
Sink 4 down



0	1	2	3	4	5	6	7	8
6	4	7	1	5	8	9	2	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

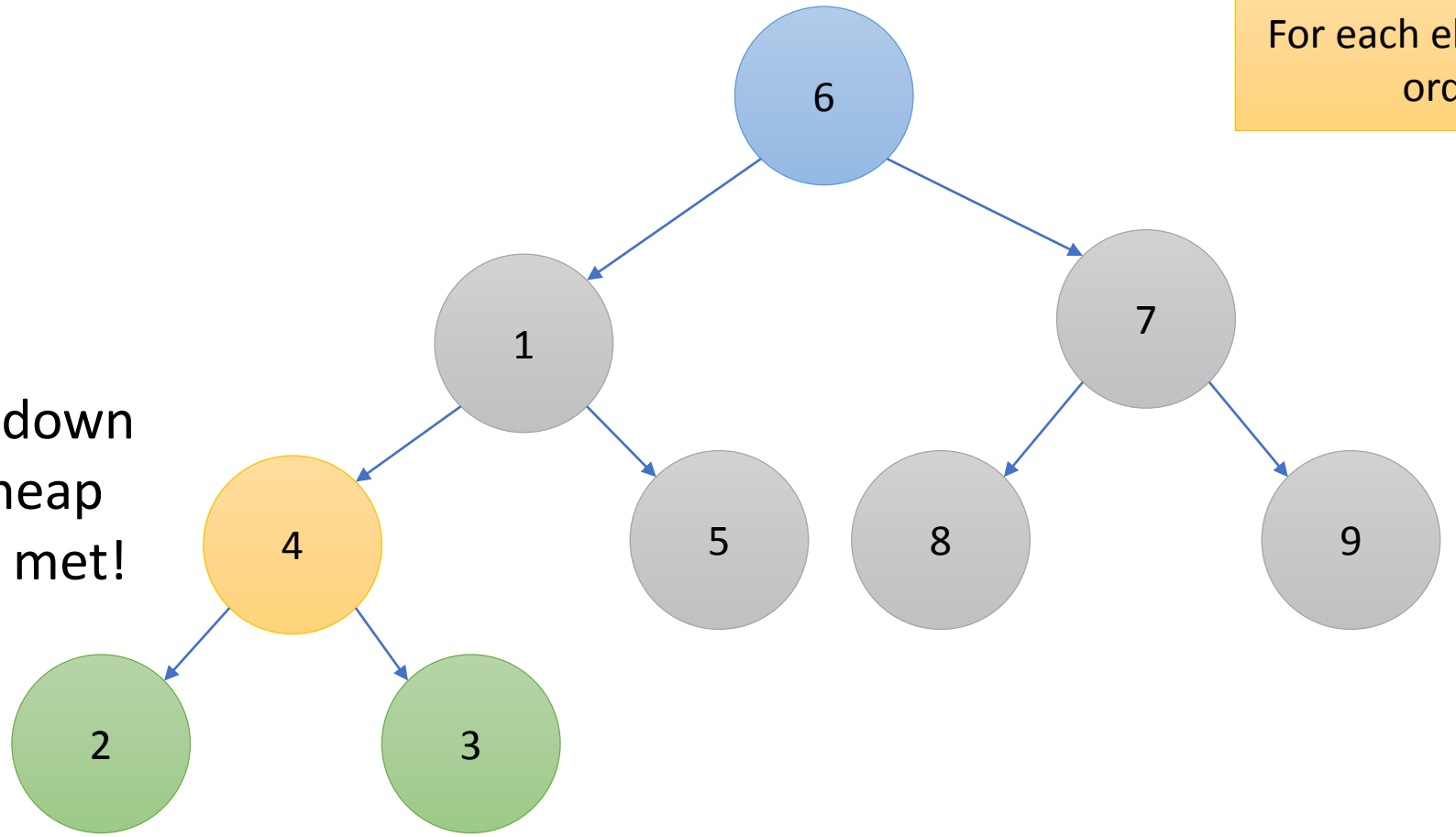


0	1	2	3	4	5	6	7	8
6	1	7	4	5	8	9	2	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

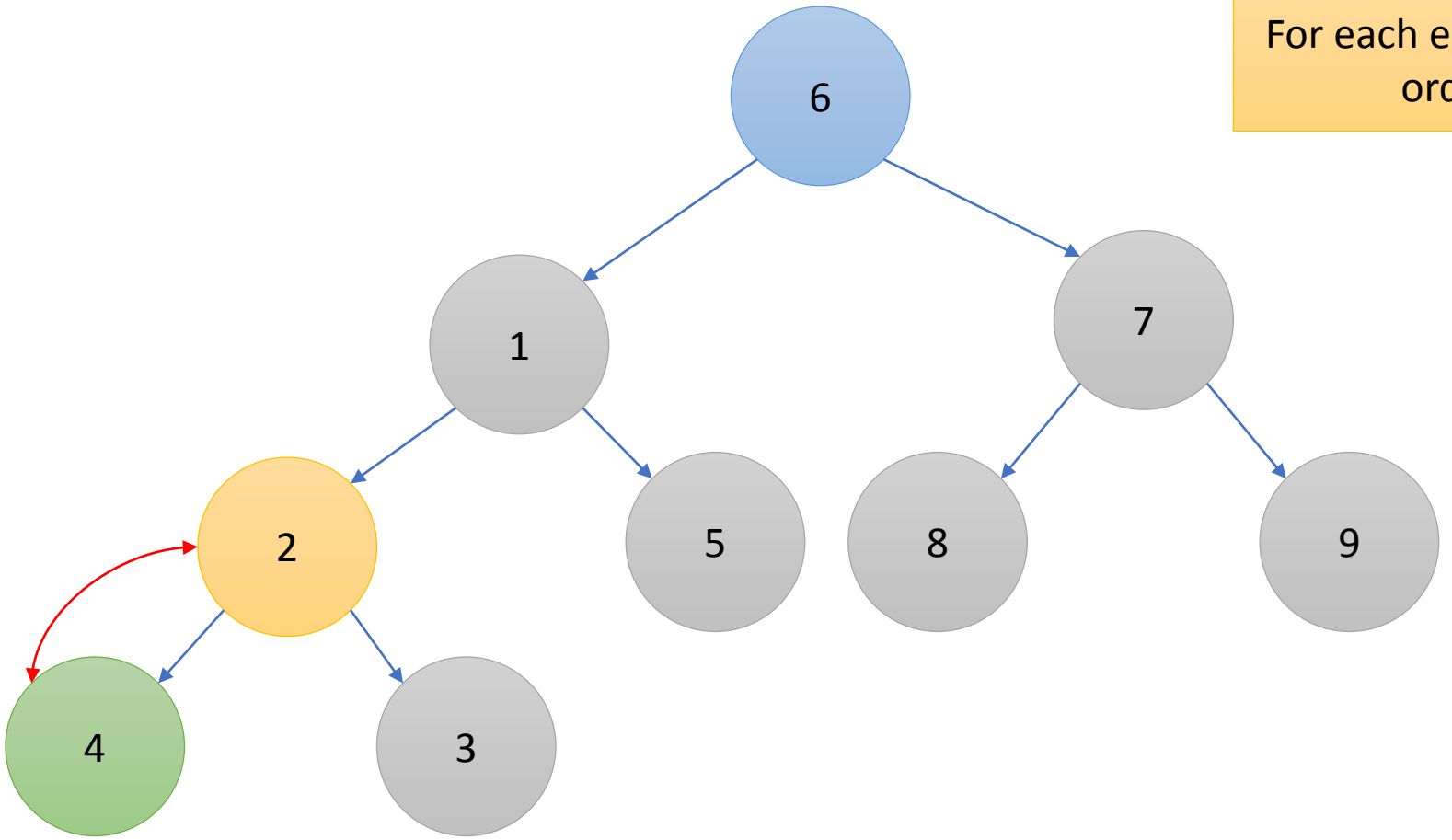
Keep sinking 4 down as long as the heap property is not met!



0	1	2	3	4	5	6	7	8
6	1	7	4	5	8	9	2	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

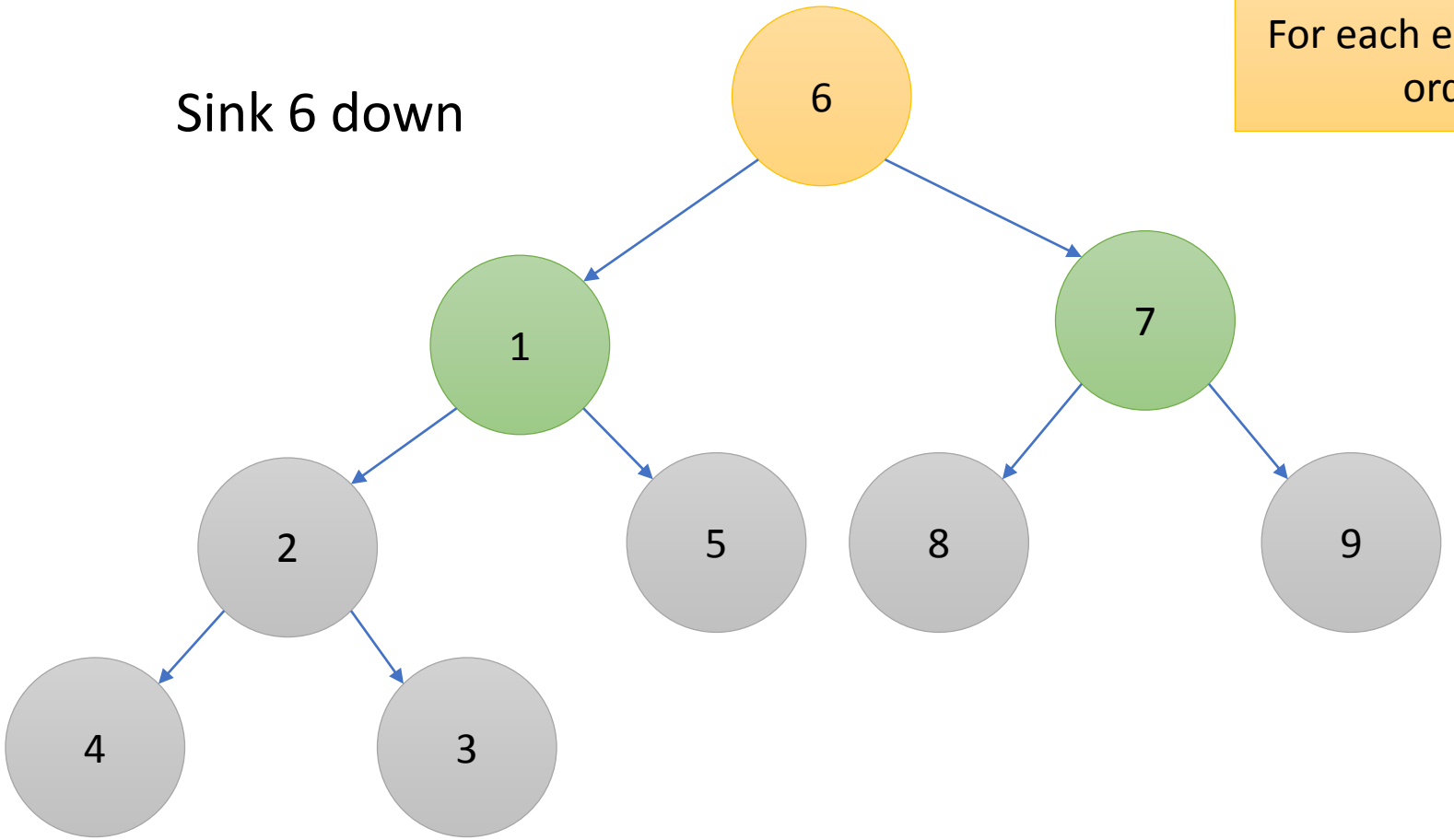


0	1	2	3	4	5	6	7	8
6	1	7	2	5	8	9	4	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

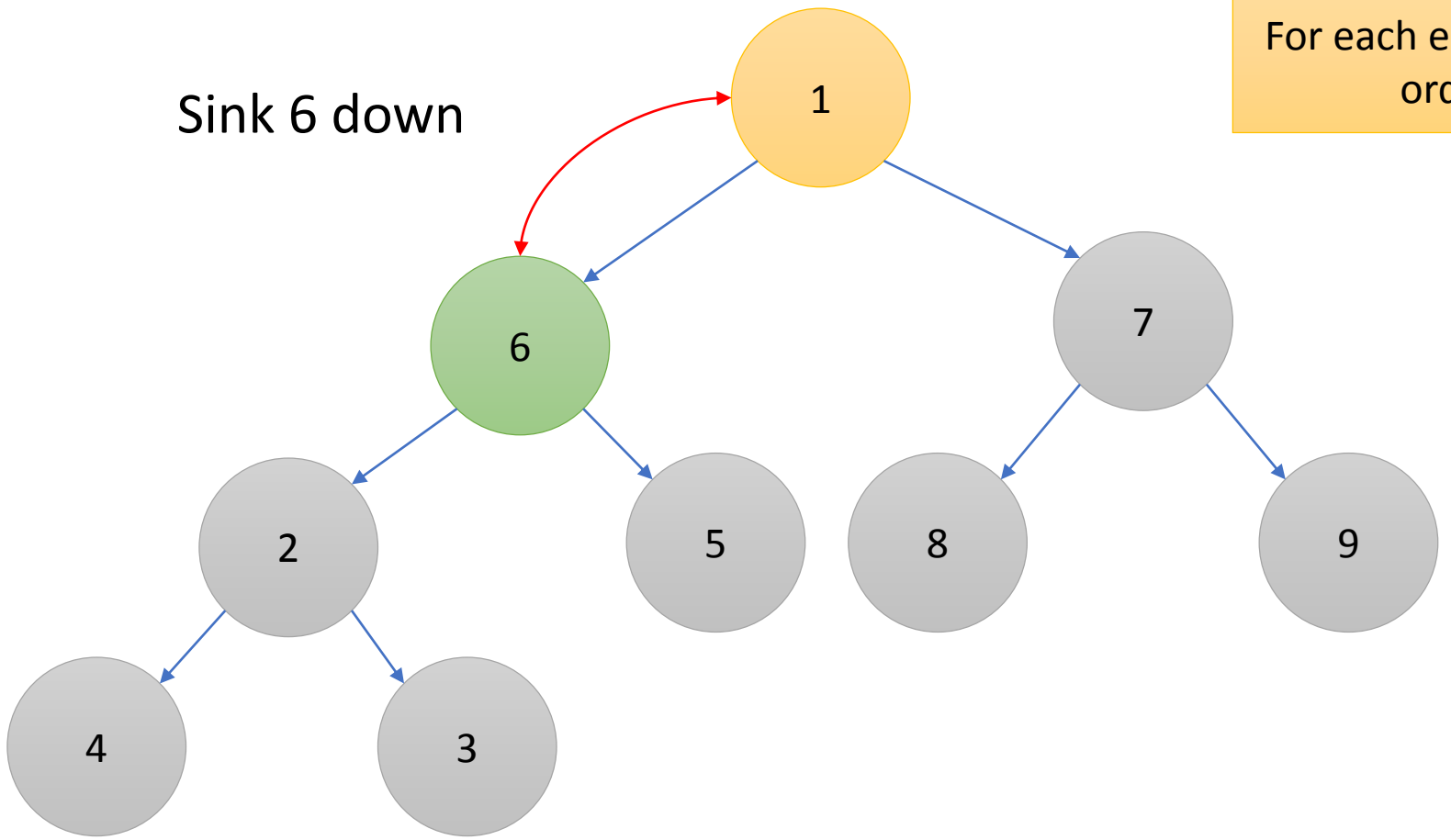
Sink 6 down



0	1	2	3	4	5	6	7	8
6	1	7	2	5	8	9	4	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

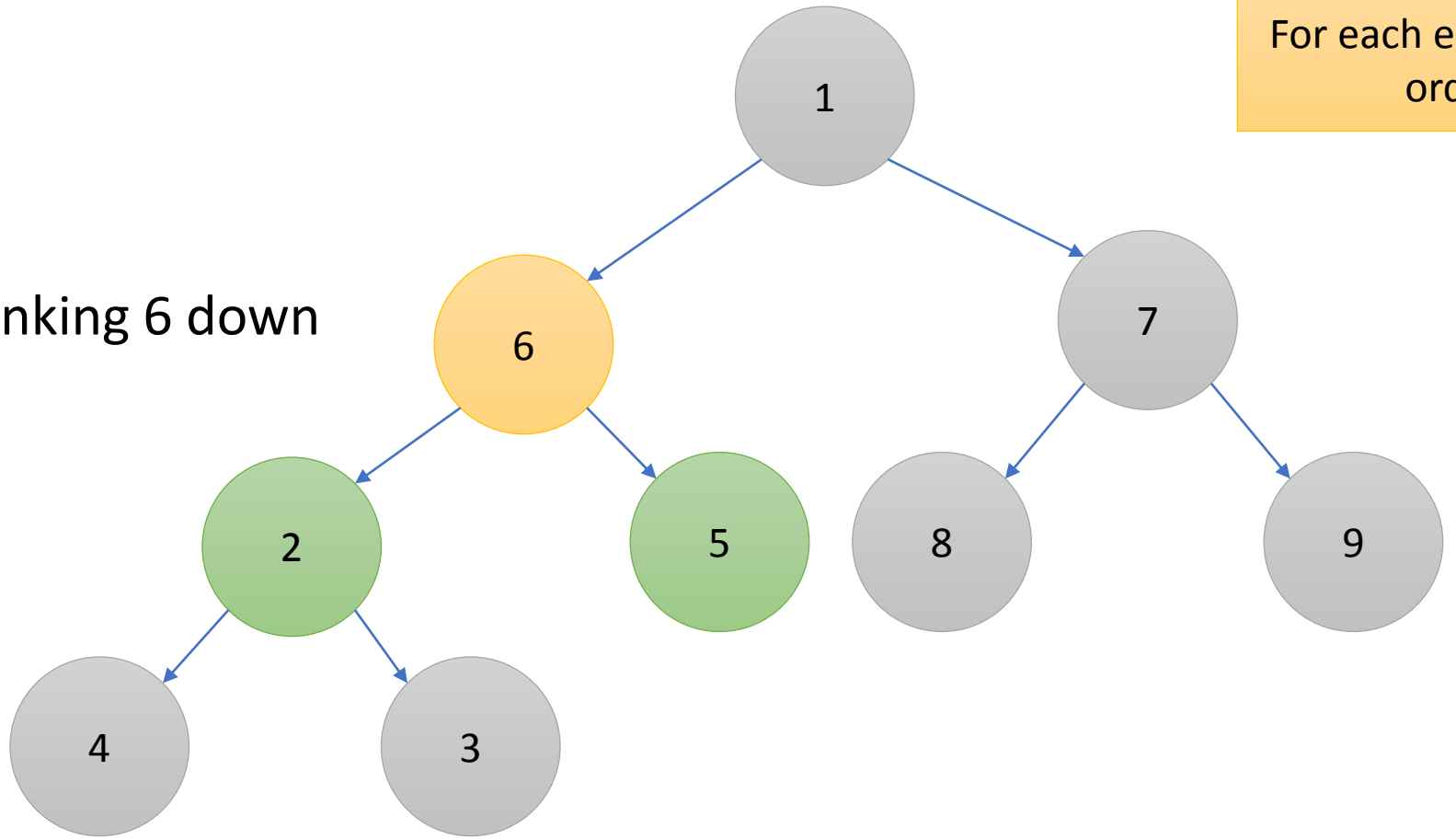


0	1	2	3	4	5	6	7	8
1	6	7	2	5	8	9	4	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

Keep sinking 6 down

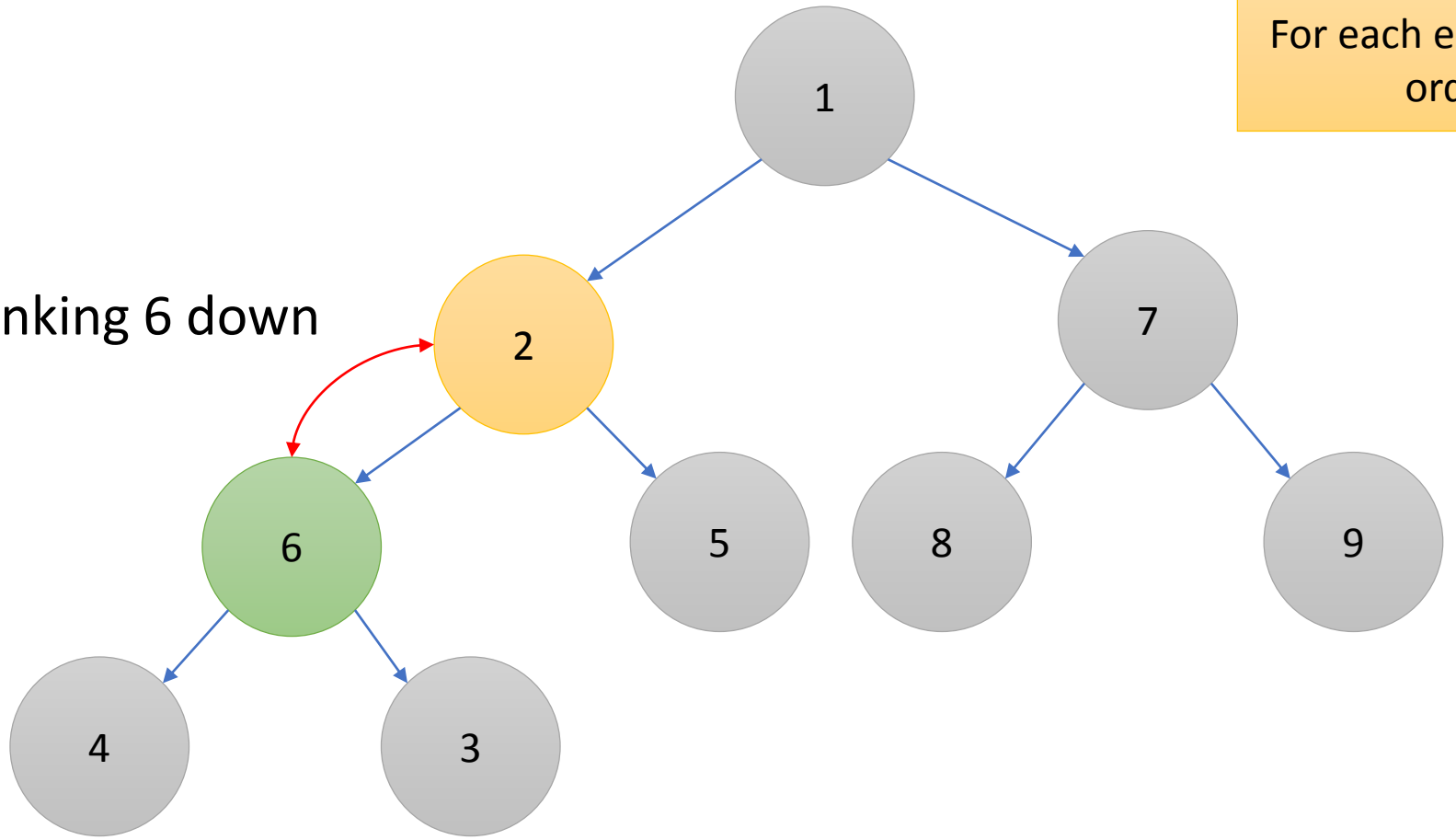


0	1	2	3	4	5	6	7	8
1	6	7	2	5	8	9	4	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

Keep sinking 6 down

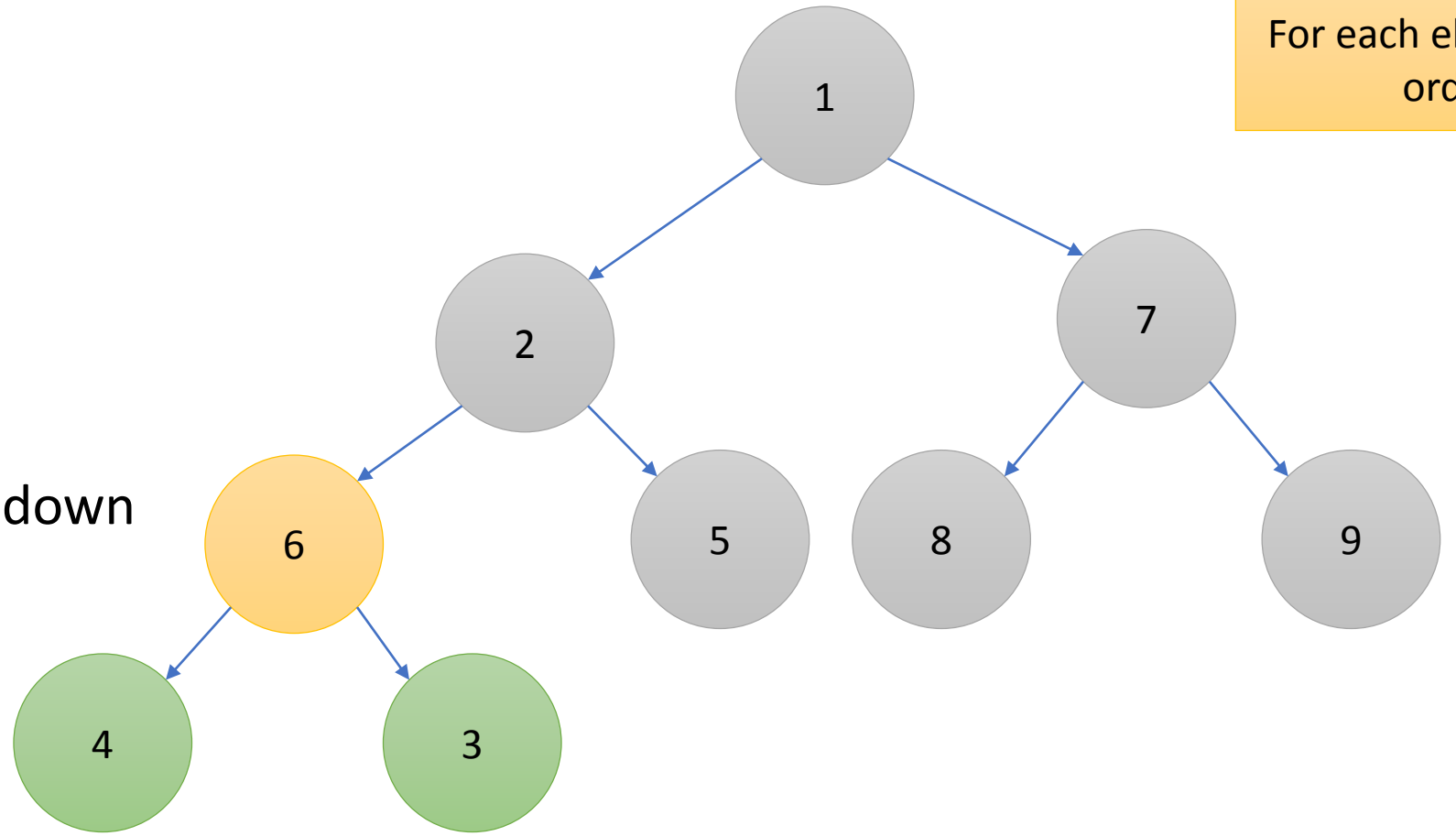


0	1	2	3	4	5	6	7	8
1	2	7	6	5	8	9	4	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

Keep sinking 6 down

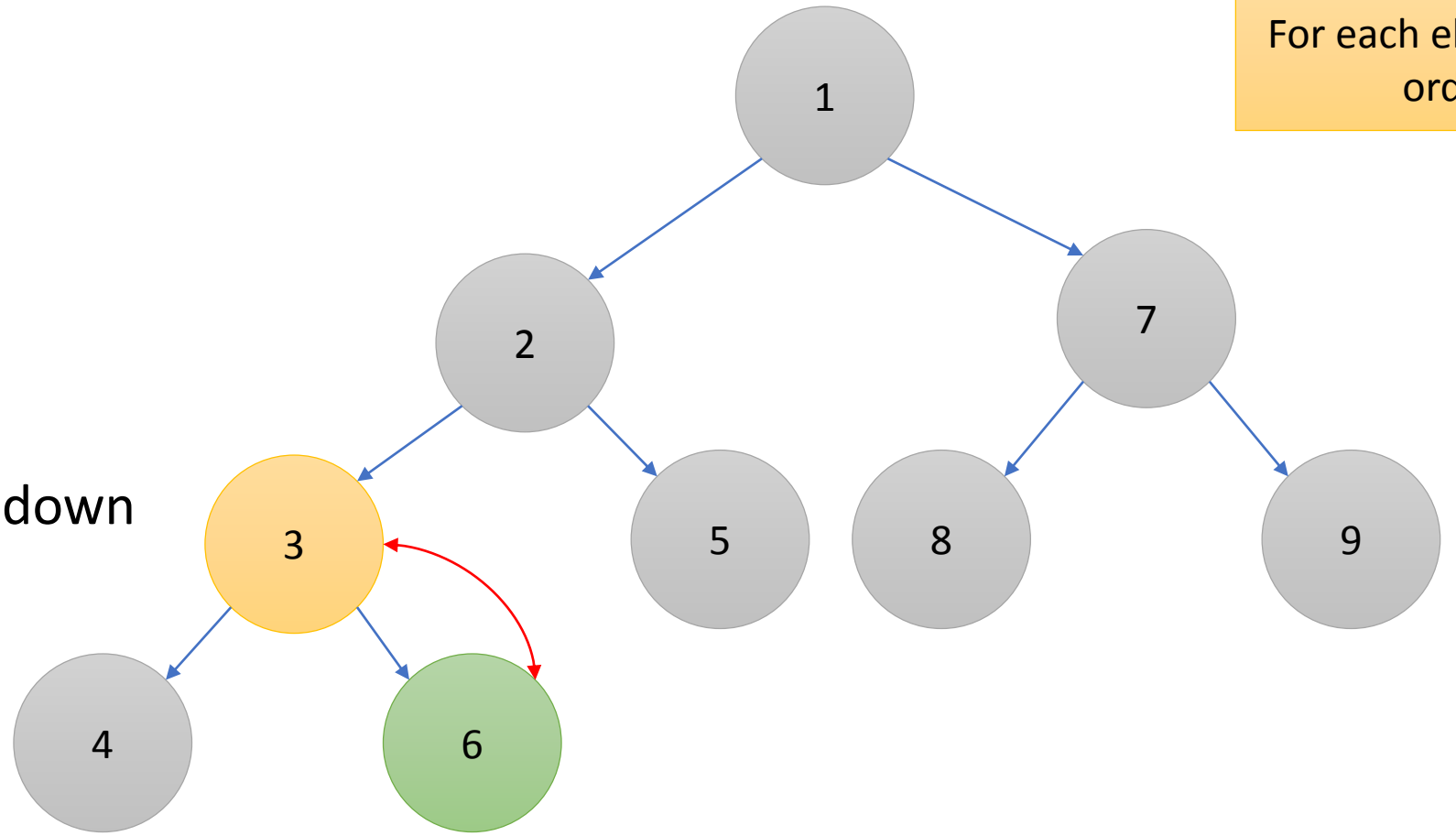


0	1	2	3	4	5	6	7	8
1	2	7	6	5	8	9	4	3

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

Keep sinking 6 down

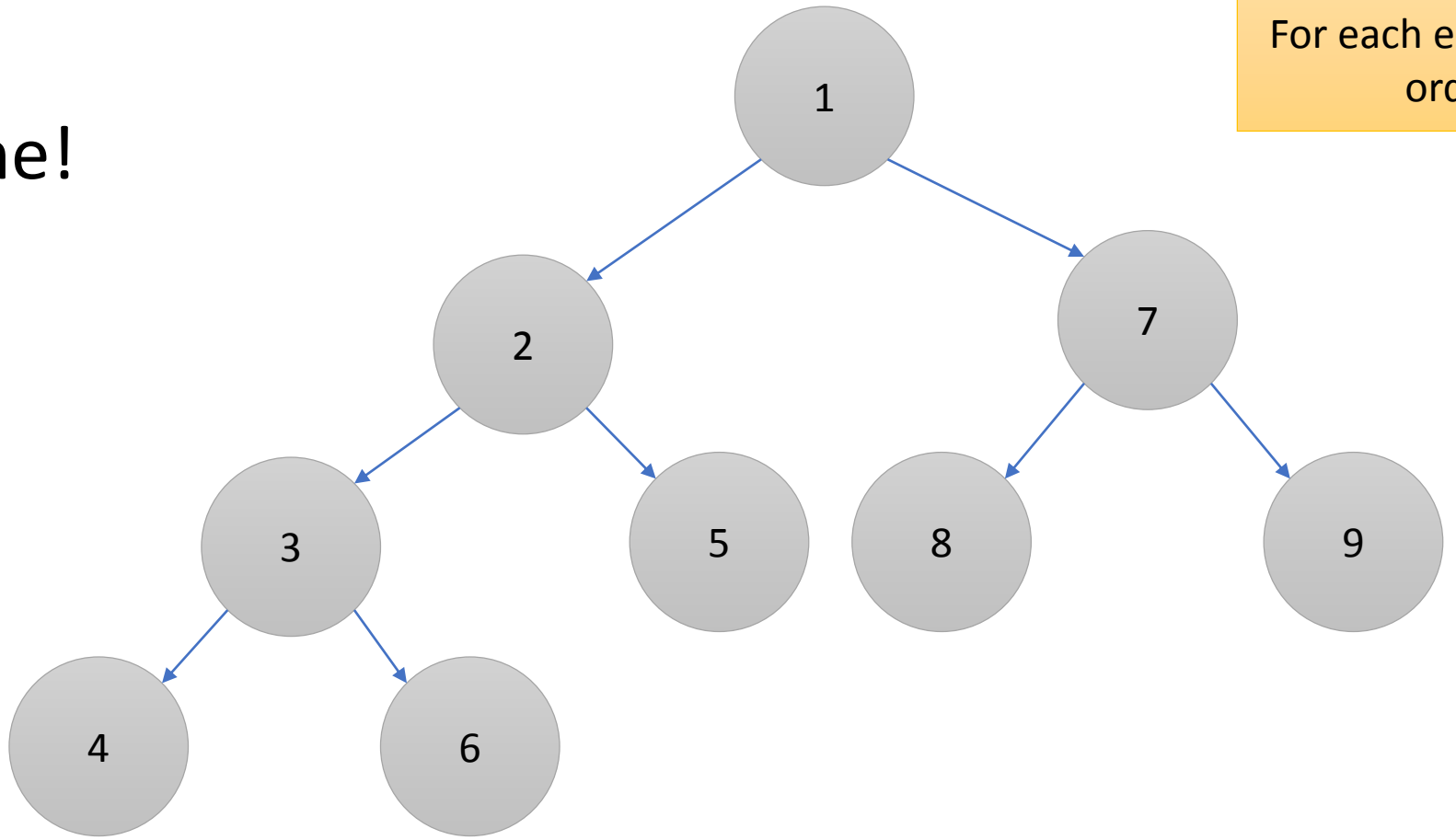


0	1	2	3	4	5	6	7	8
1	2	7	3	5	8	9	4	6

Convert a Complete Tree into a Min-Heap using Heapify

For each element in reverse-array order, sink it down

All done!



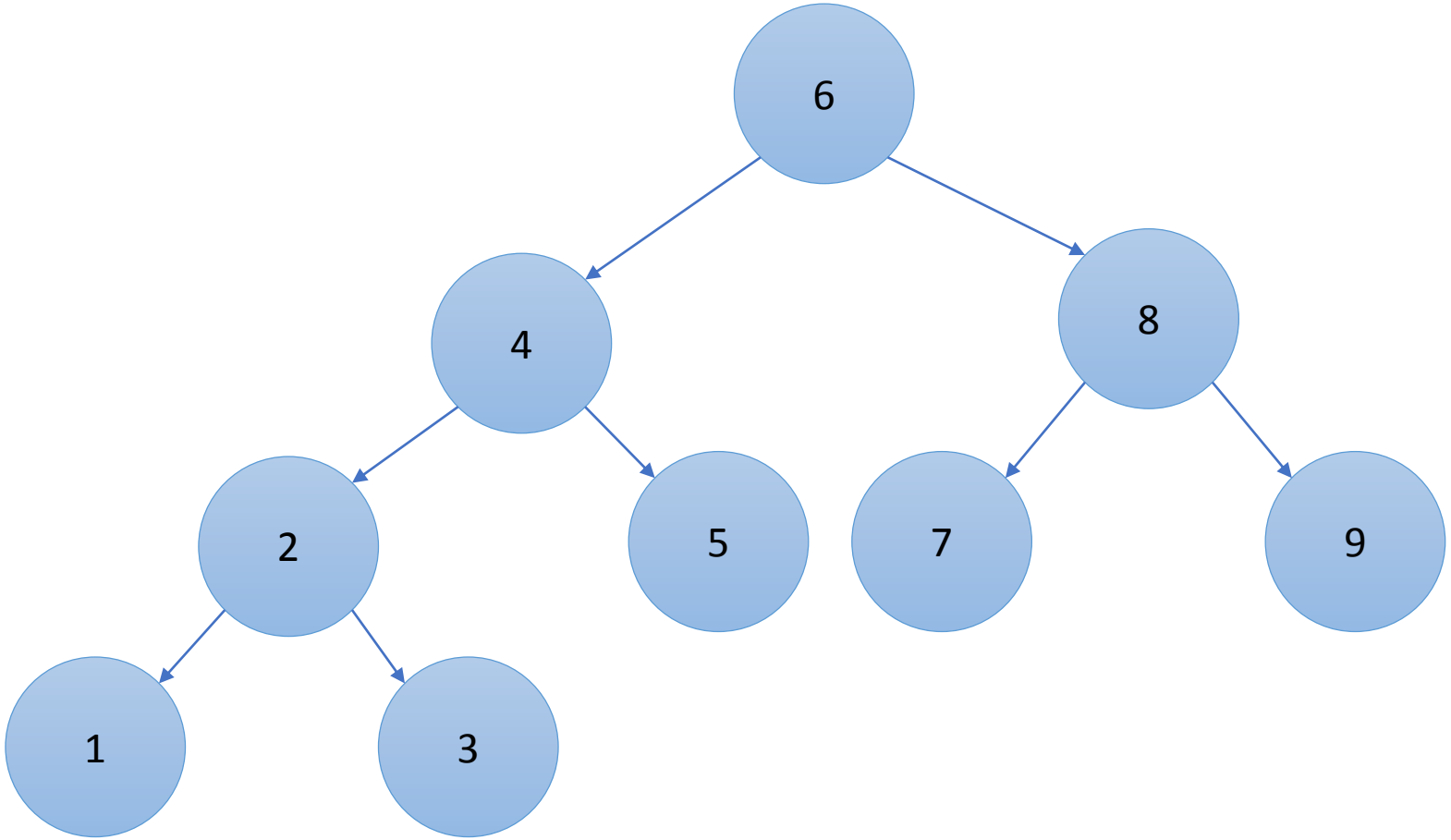
0	1	2	3	4	5	6	7	8
1	2	7	3	5	8	9	4	6

Heapify

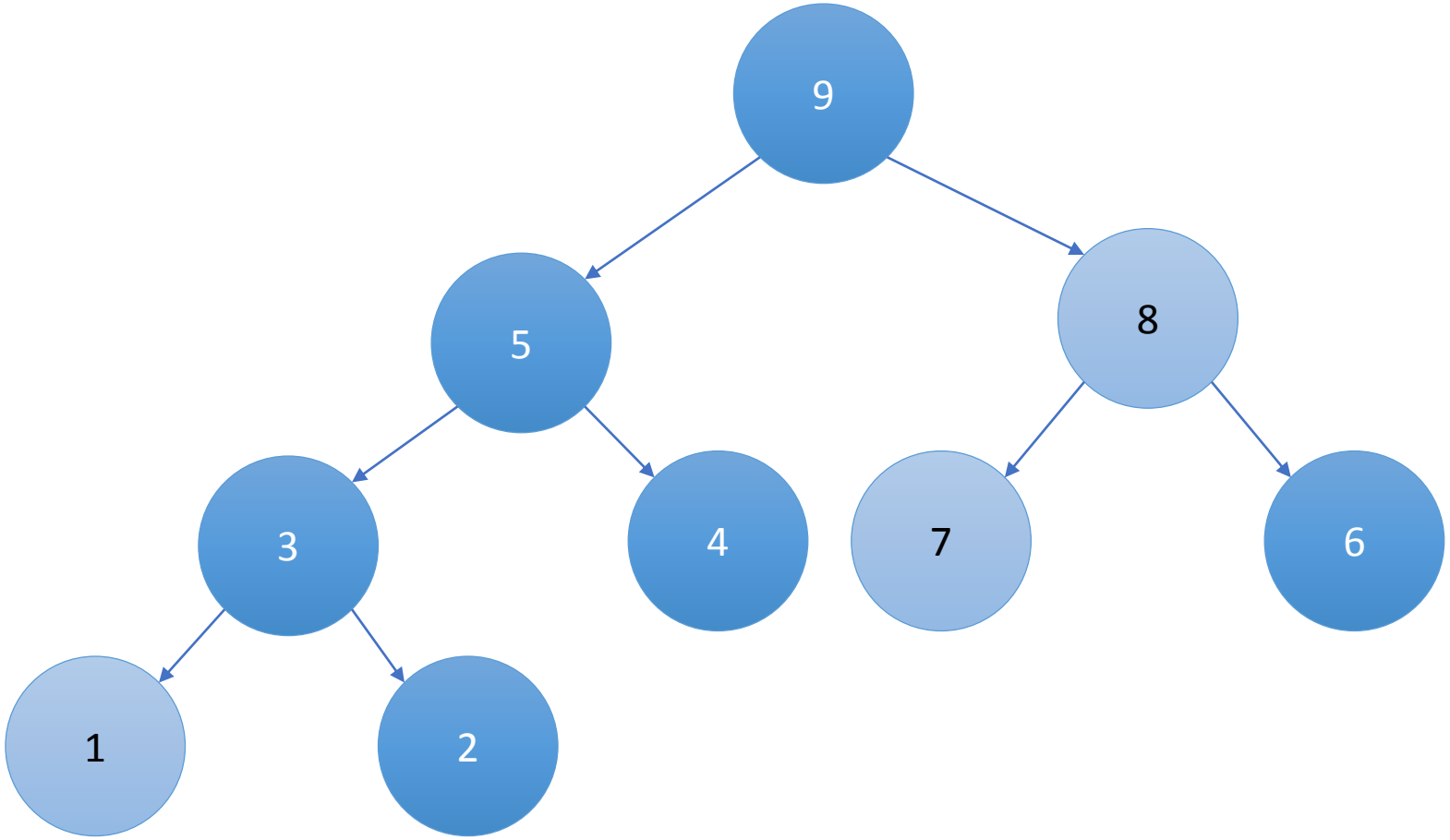
For each element in reverse-array order

… while the heap property is not being met, swap that element with one of its children

Activity 4: heapify this tree into a Max-Heap



Activity 4: heapify this tree into a Max-Heap



Summary

- Heap
  - Complete Binary Tree w/ Heap Property
  - Min-Heap & Max-Heap
  - Root is always the smallest/biggest value
  - Remove method always removes the root node

- Add method
- Remove Method
- Heapify Method

Todo

- Worksheet
- Lab 30

Quiz Next Class!

Test is class after that!

- Binary Trees
  - Pre/in/post order
  - BNodes vs List
- Binary Search Trees
  - add
  - search
  - remove
- Heap
  - Add
  - Remove
  - heapify