Week 13 - Wednesday

COMP 2100

Last time

- What did we talk about last time?
- Lower bound on sorting
- Counting sort
- Radix sort
- Started heaps

Questions?

Project 4

Assignment 7

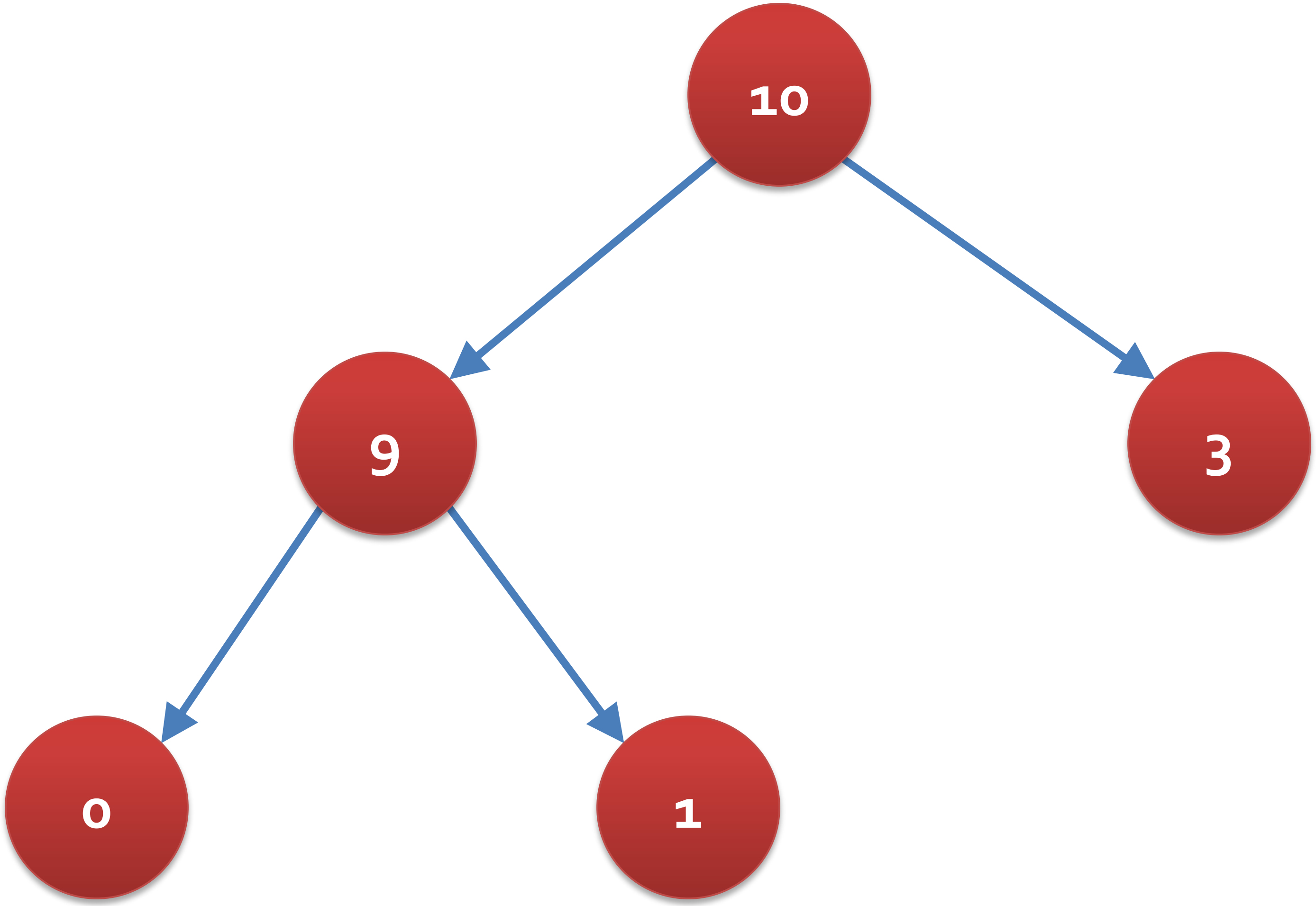
Heaps

Heaps

- A **maximum heap** is a **complete** binary tree where
  - The left and right children of the root have key values less than the root
  - The left and right subtrees are also maximum heaps

- We can define **minimum heaps** similarly

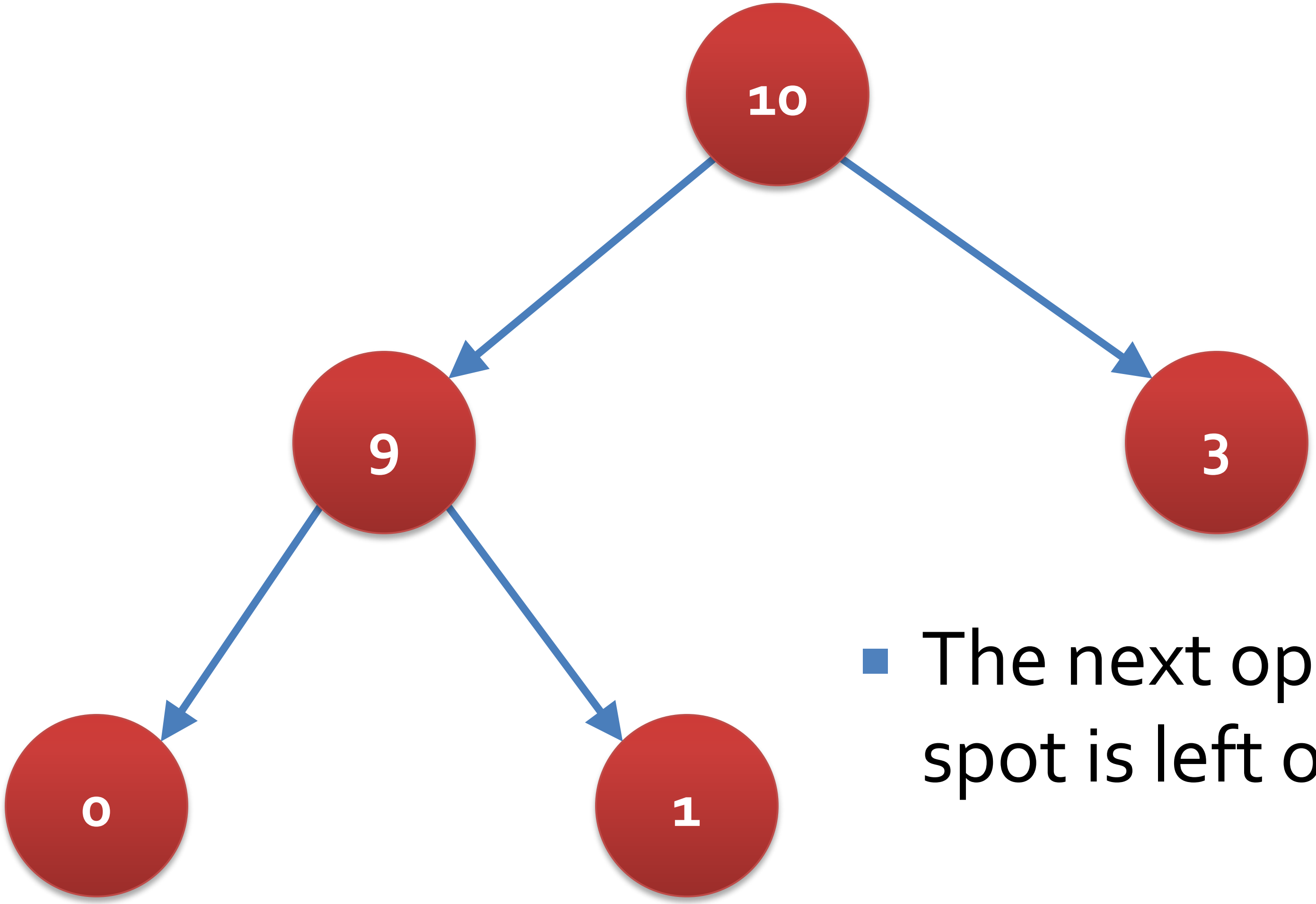
Heap example



How do you know where to add?

- We have to keep the tree complete
  - Recall that a complete binary tree is one where every level is filled, except possibly the last one, which is filled in from the left
- We always add to the next open spot in the current level
  - Or make a new level if the current level is full

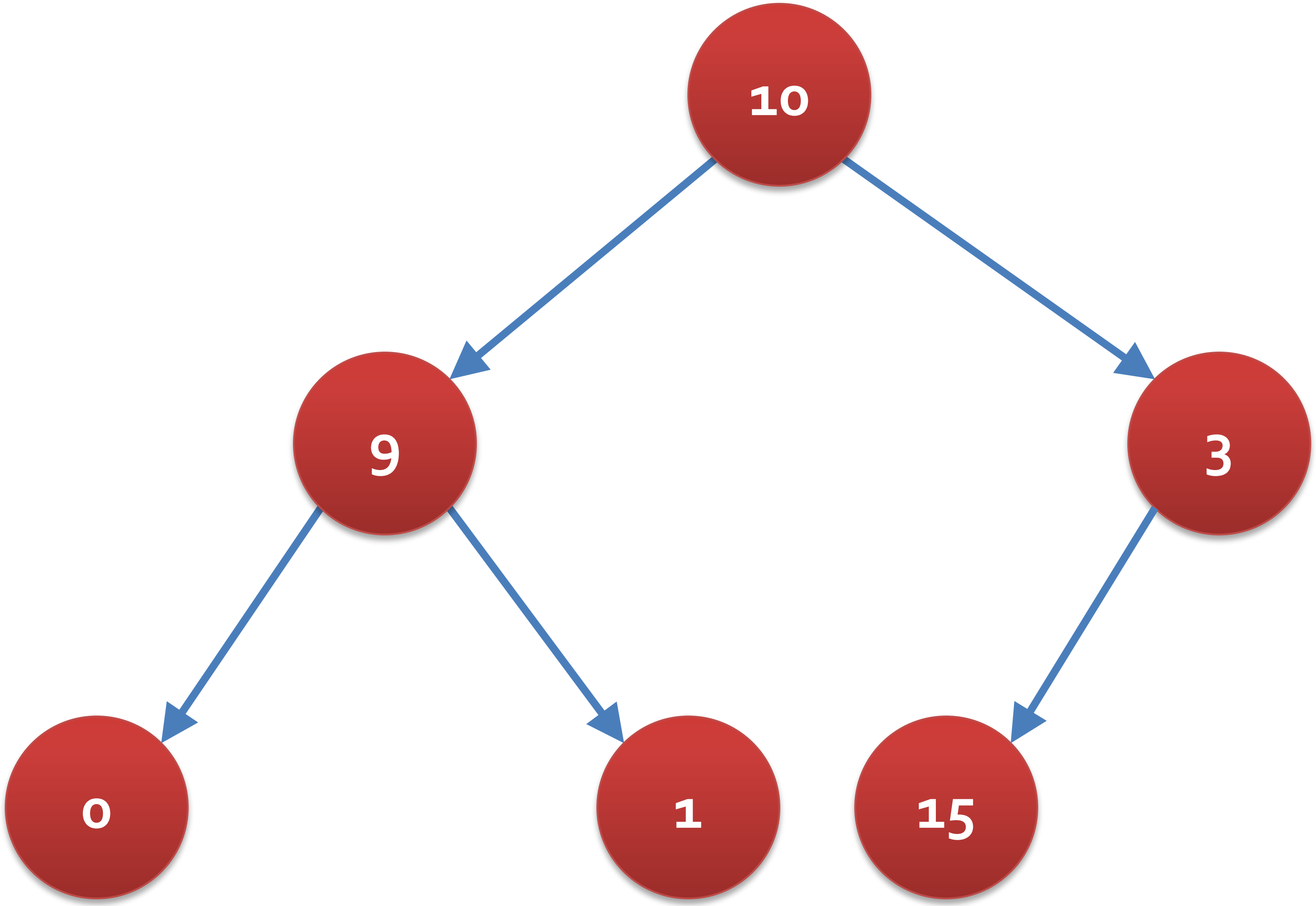
New node



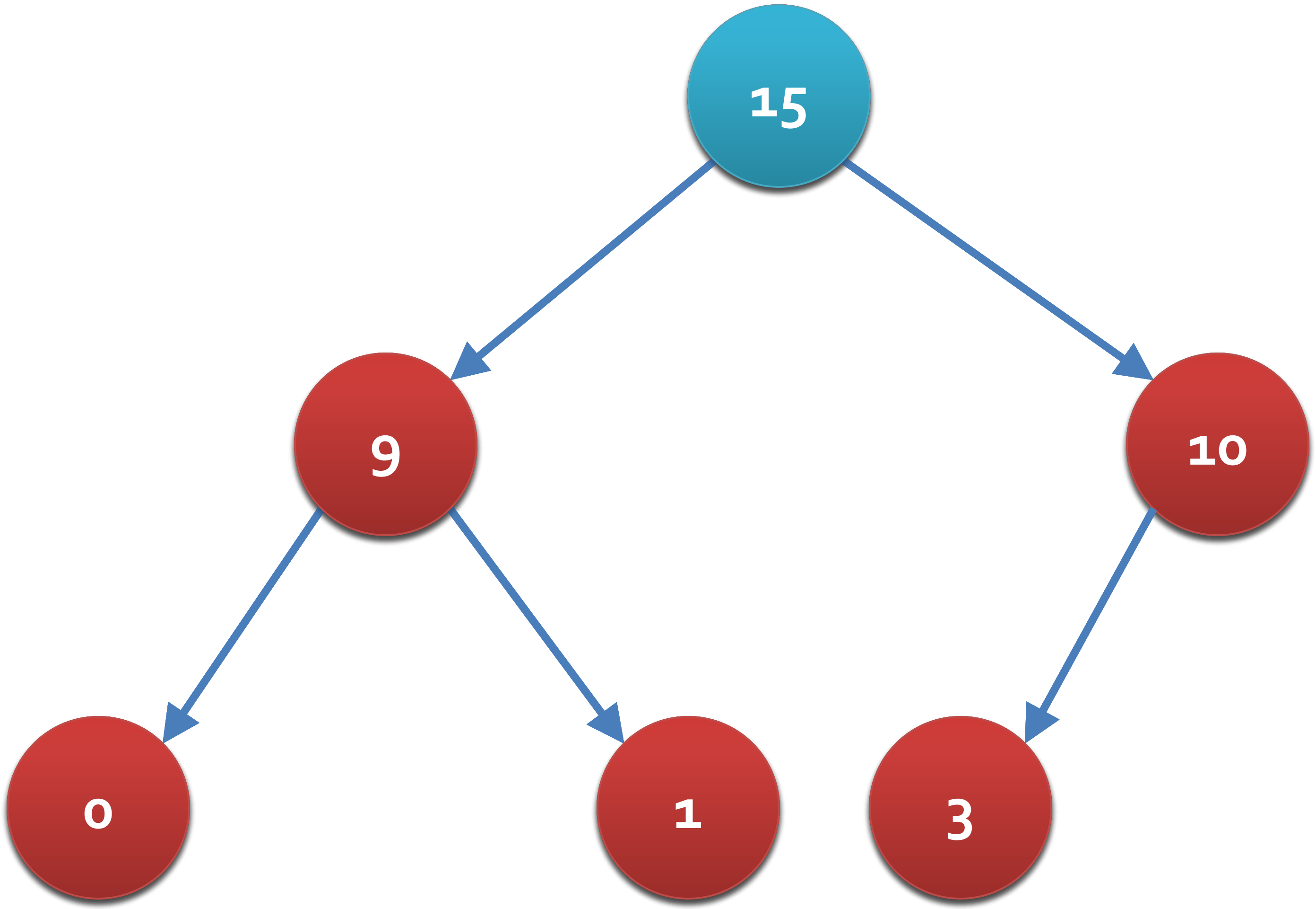
- The next open spot is left of 3

Add 15

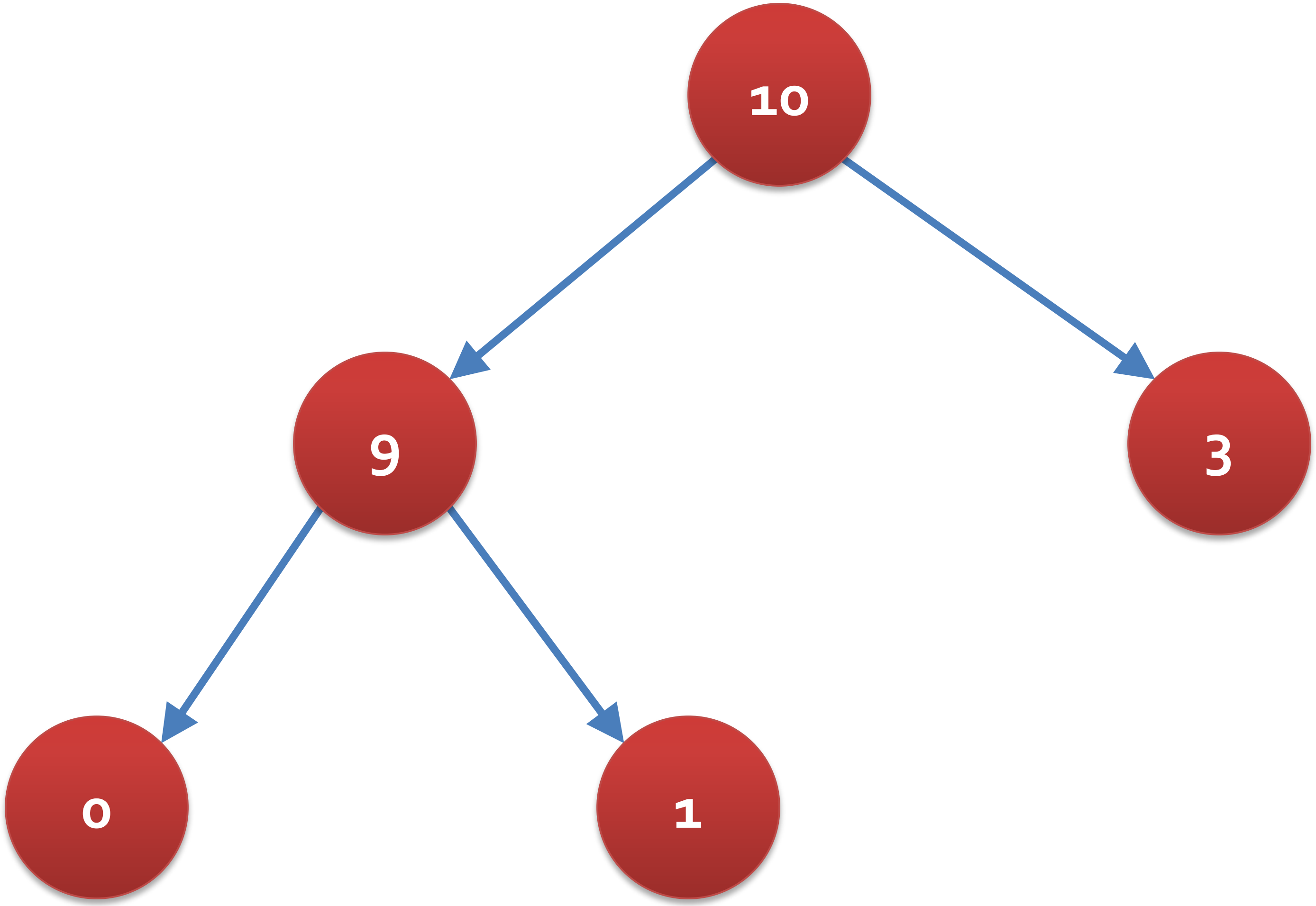
- Oh no!



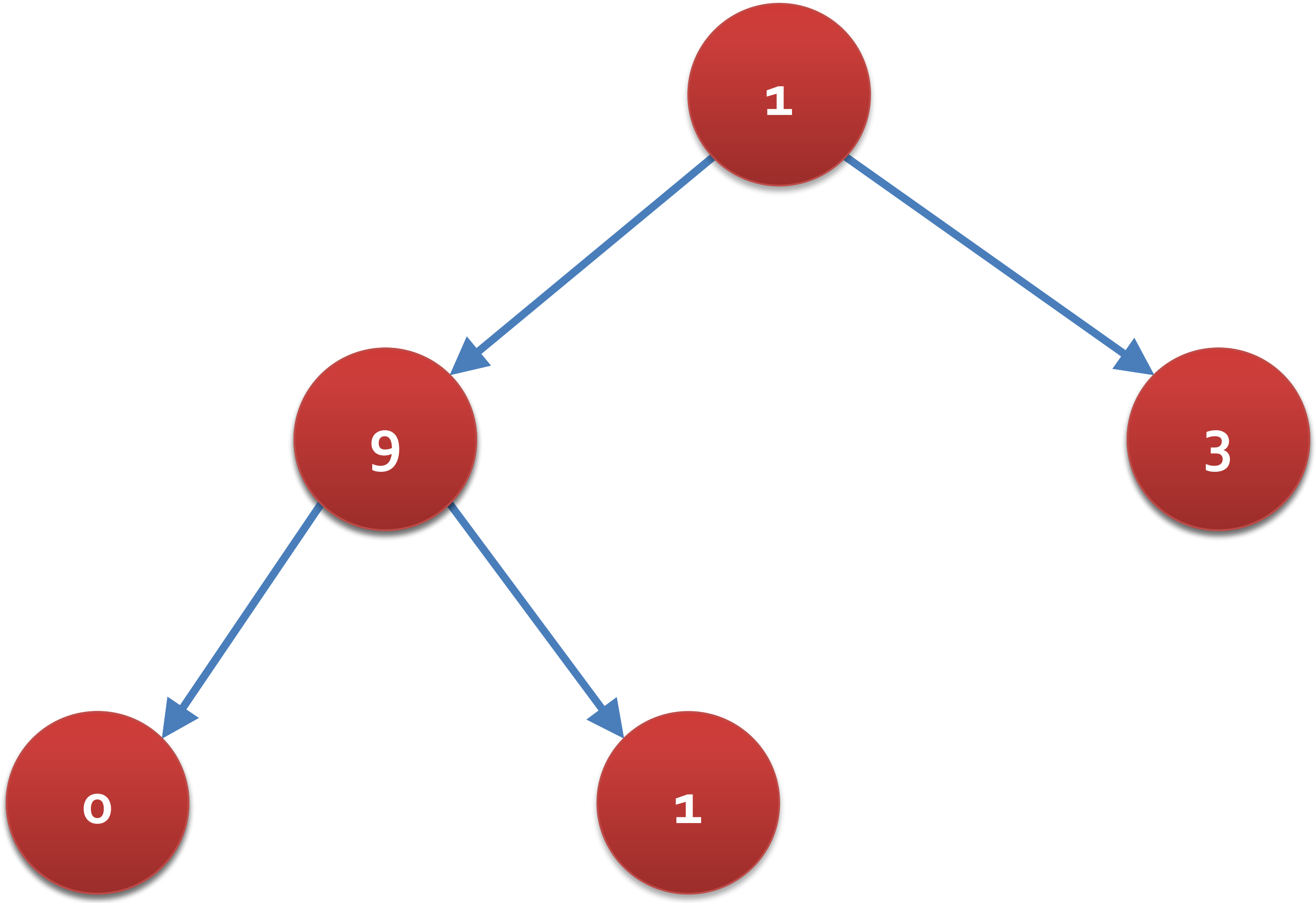
After an add, bubble up



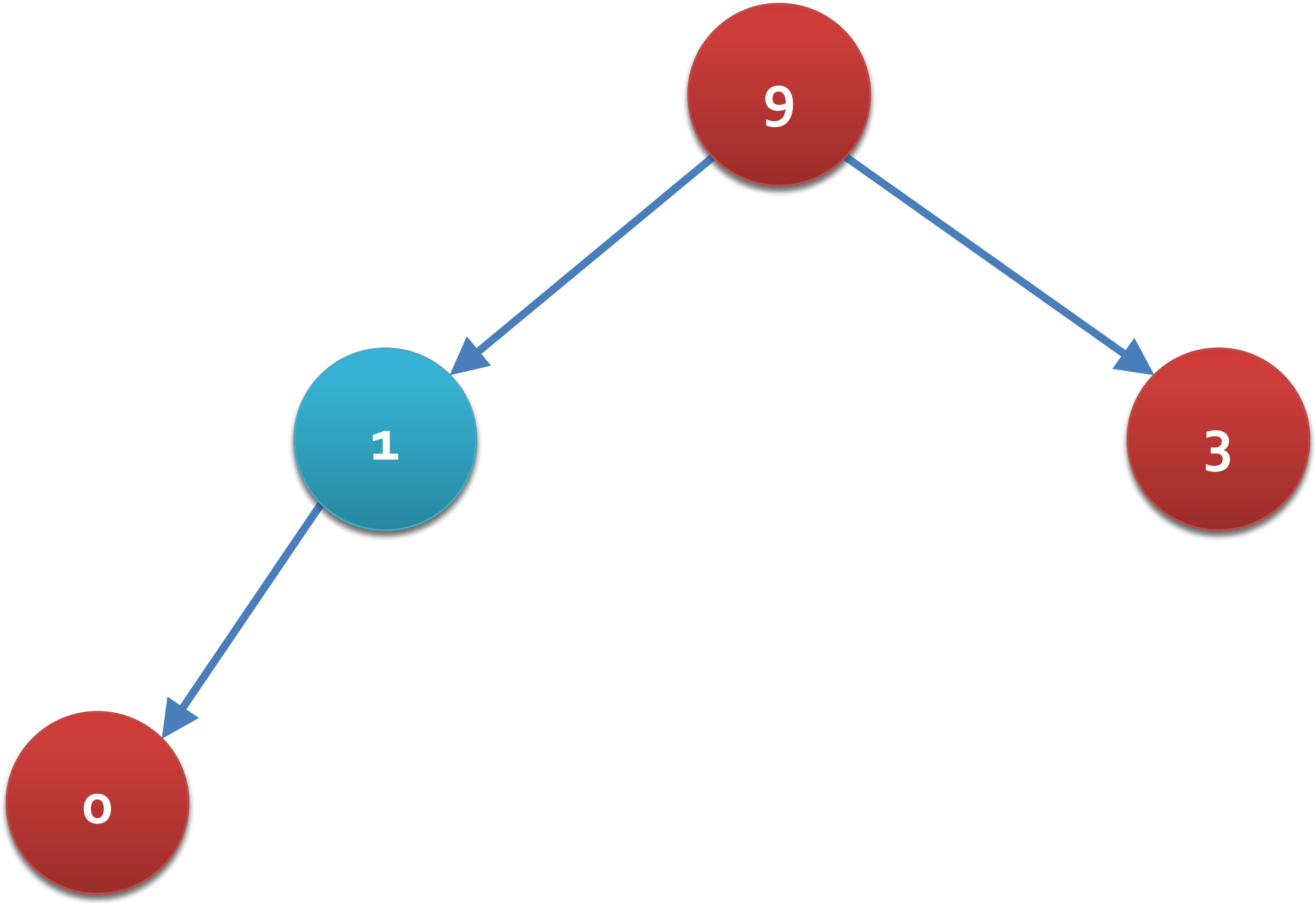
Only the root can be deleted



Replace it with the "last" node



Then, bubble down



Operations

- Heaps only have:
  - Add
  - Remove Largest
  - Get Largest
- Which cost:
  - Add: O(log ***n***)
  - Remove Largest: O(log ***n***)
  - Get Largest: O(1)
- Heaps are a perfect data structure for a priority queue

Priority queue implementation

Priority queue

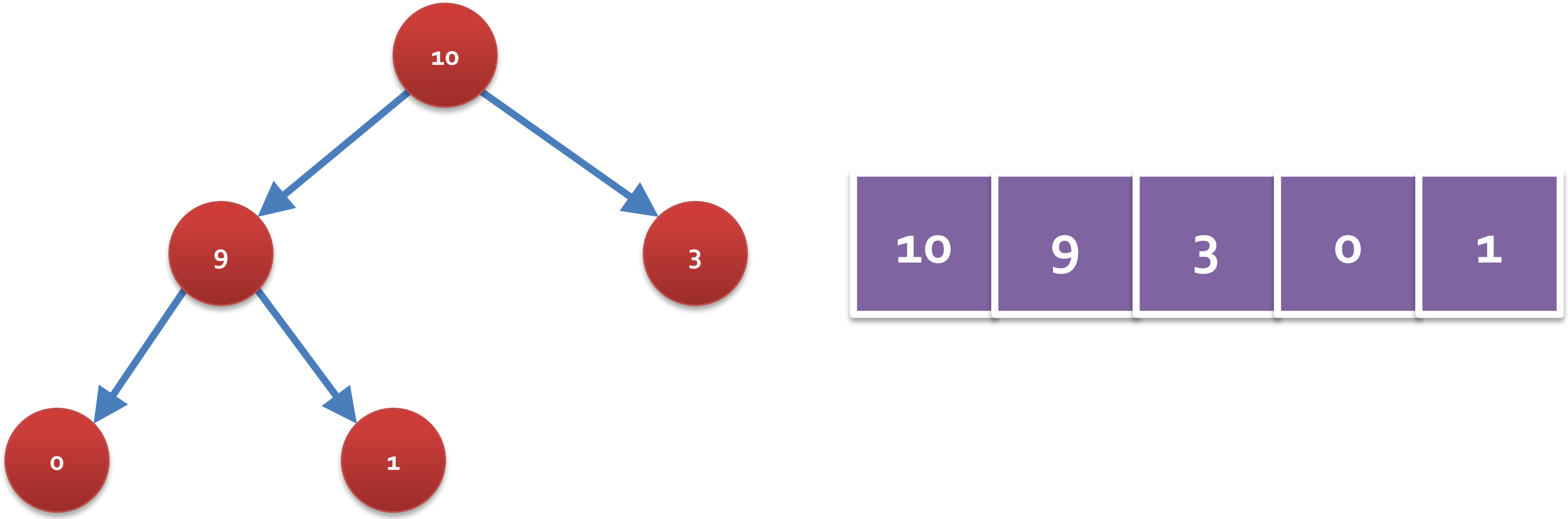
- A priority queue is an ADT that allows us to insert key values and efficiently retrieve the highest priority one
- It has these operations:
  - **Insert(key)** Put the key into the priority queue
  - **Max()** Get the highest value key
  - **Remove Max()** Remove the highest value key

Implementation

- It turns out that a heap is a great way to implement a priority queue
- Although it's useful to *think* of a heap as a complete binary tree, almost no one implements them that way
- Instead, we can view the heap as an **array**
- Because it's a **complete** binary tree, there will be no empty spots in the array

Array view

- Illustrated:



10	9	3	0	1

- The left child of element ***i*** is at 2*i* + 1
- The right child of element ***i*** is at 2*i* + 2

Array implementation of priority queue

```
public class PriorityQueue {

  private int[] keys = new int[10];
  private int size = 0;
  …
}
```

Insert

```
public void insert(int key)
```

- Always put the key at the end of the array (resizing if needed)
- The value will often need to be bubbled up, using the following helper method

```
private void bubbleUp(int index)
```

Max

```
public int max()
```

- Find the maximum value in the priority queue
- Hint: this method is really easy

Remove Max

```
public int removeMax()
```

- Store the value at the top of the heap (array index `0`)
- Replace it with the last legal value in the array
- This value will generally need to be bubbled down, using the following helper method
  - Bubbling down is harder than bubbling up, because you might have two legal children!

```
private void bubbleDown(int index)
```

Heap Sort

Heap sort

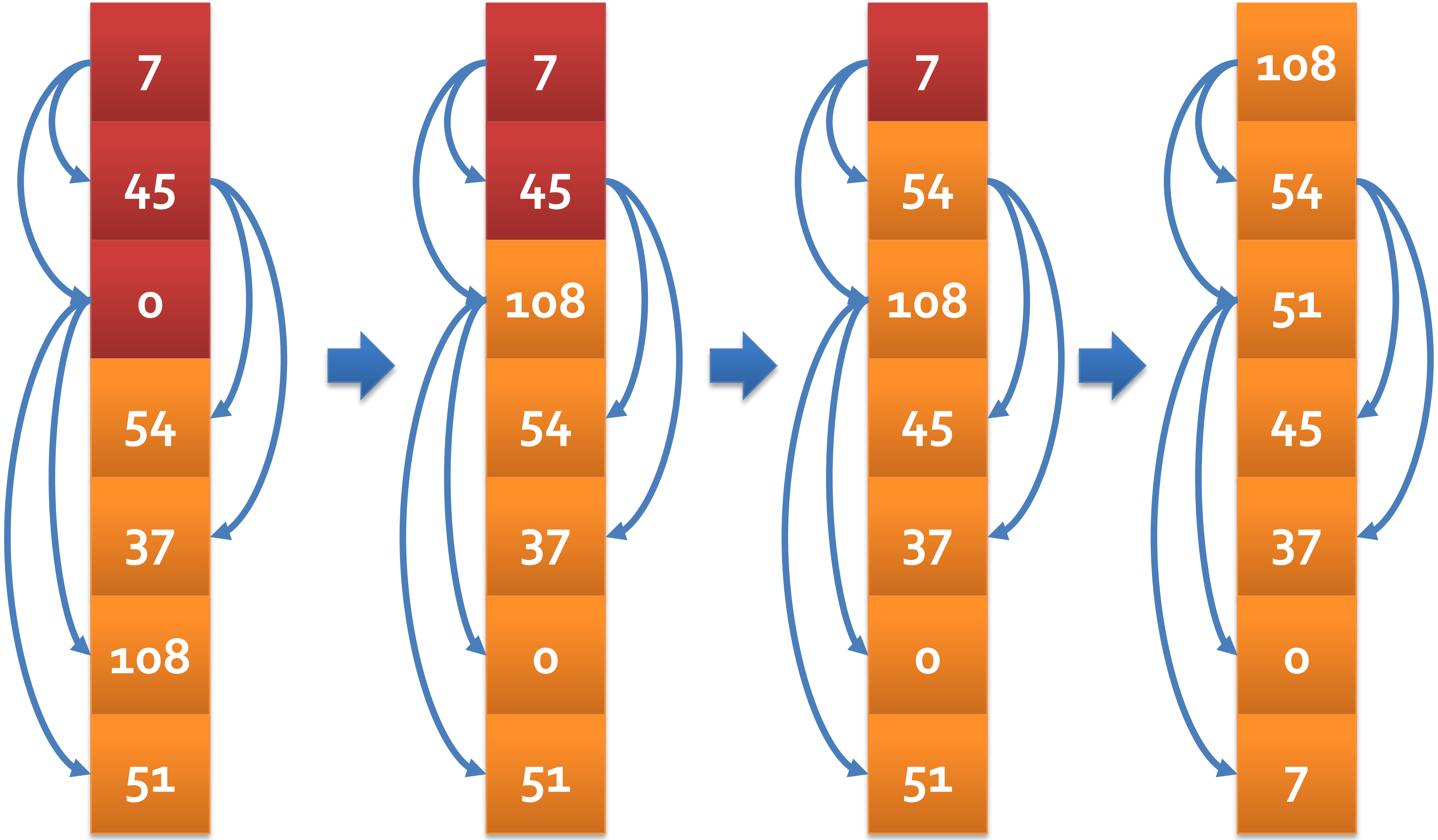
- Pros:
  - **Best**, **worst**, and **average** case running time of O(***n*** log ***n***)
  - In-place
  - Good for arrays
- Cons:
  - Not adaptive
  - Not stable

Heap sort algorithm

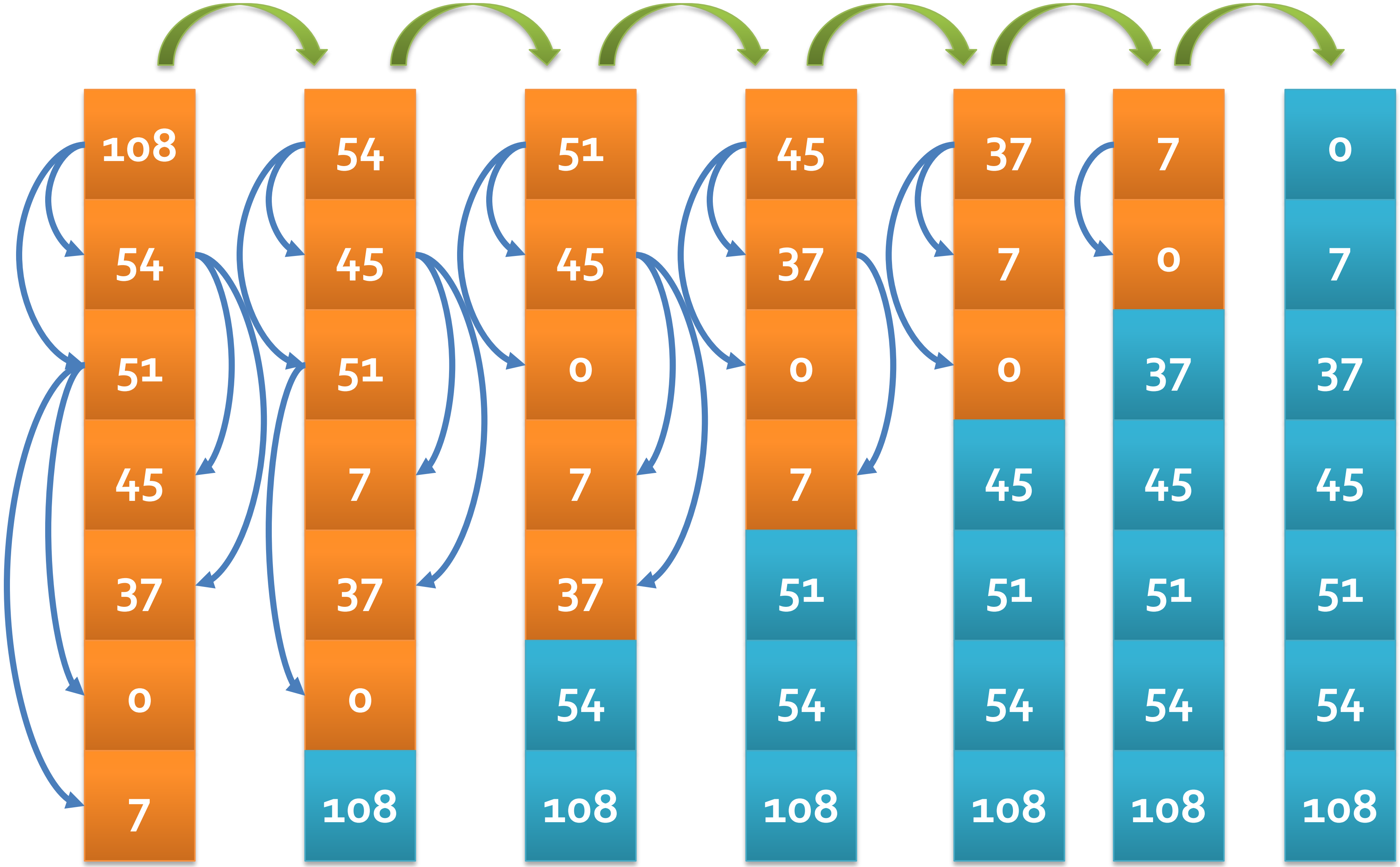
- Make the array have the heap property:
  1. Let ***i*** be the index of the parent of the last two nodes
  2. Bubble the value at index ***i*** down if needed
  3. Decrement ***i***
  4. If ***i*** is not less than zero, go to Step 2

1. Let ***pos*** be the index of the last element in the array
2. Swap index 0 with index ***pos***
3. Bubble down index 0
4. Decrement ***pos***
5. If ***pos*** is greater than zero, go to Step 2

Heap sort heapify example



Heap sort extraction example



108	54	51	45	37	7	0
54	45	45	37	7	0	7
51	51	0	0	0	37	37
45	7	7	7	45	45	45
37	37	37	51	51	51	51
0	0	54	54	54	54	54
7	108	108	108	108	108	108

Heap sort implementation

- Heap sort is a clever algorithm that uses part of the array to store the heap and the rest to store the (growing) sorted array
- Even though a priority queue uses both bubble up and bubble down methods to manage the heap, heap sort **only** needs bubble down
- You don't need bubble up because nothing is added to the heap, only removed

Upcoming

Next time…

- TimSort
- Tries
- Substring search

Reminders

- Work on Project 4
- **Finish Assignment 7**
  - **Due Friday!**
- Keep reading Section 2.4 and read Section 5.2