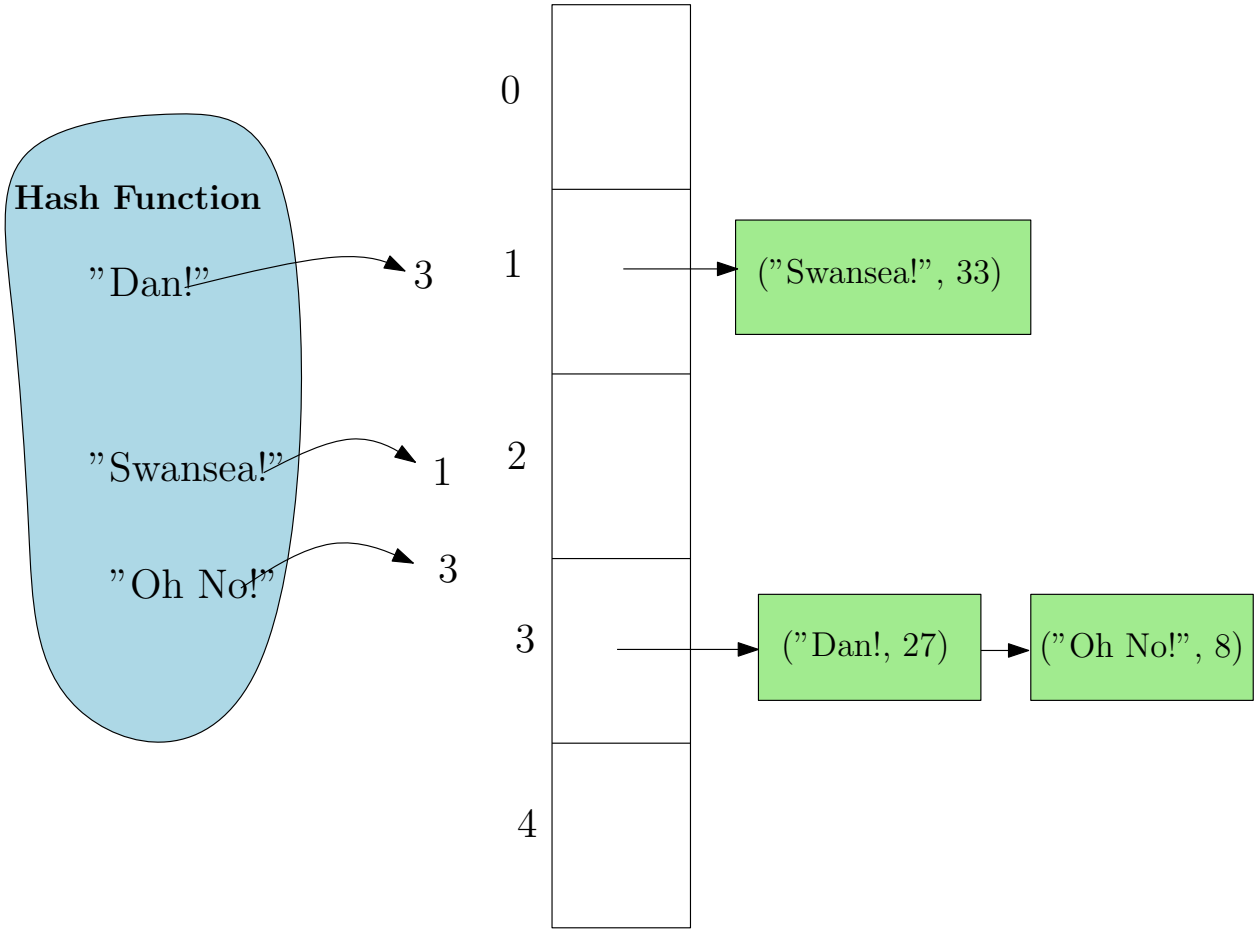
Heaps and Priority Queues

Daniel Archambault

Previously in CS-115



Hash it in there with non-integer keys.

Previously in CS-115

- What is a hash map?

Previously in CS-115

- What is a hash map?
- What are the advantages of a hash map?

Previously in CS-115

- What is a hash map?
- What are the advantages of a hash map?
- What is a hashing function?

Previously in CS-115

- What is a hash map?
- What are the advantages of a hash map?
- What is a hashing function?
- What is a collision? How do we deal with collisions?

Previously in CS-115

- Now it's time to get our priorities straight!

Heaps and Priority Queues

Priority Queue ADT

- Like a queue, but...
  - all items inserted into the queue have a priority
  - the front of the queue is always the item of highest priority
  - you can think of it as an emergency room
- Like queues, you have enqueue, dequeue, peek, and isEmpty.
- `https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html`

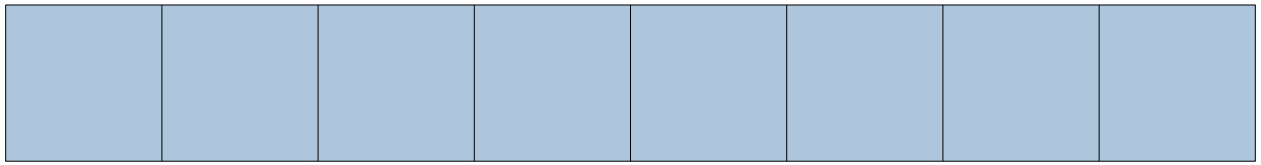
Priority Queue ADT

```
public interface PriorityQueue {
   public boolean isEmpty ();
   public void enqueue (Object newItem);
   public Object dequeue ();
   public Object peek ();
}
```

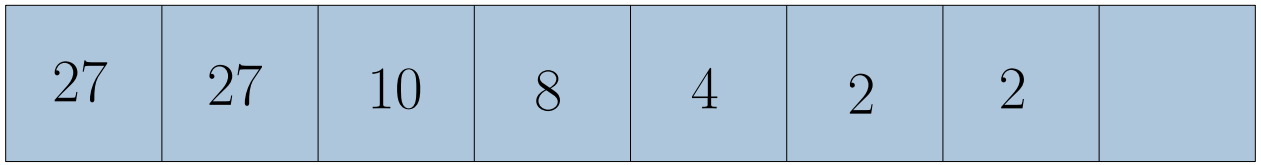
- Similar to Queue, but all must have priorities

isEmpty behaviour

- Returns true if there are no elements in the priority queue
- Otherwise, returns false



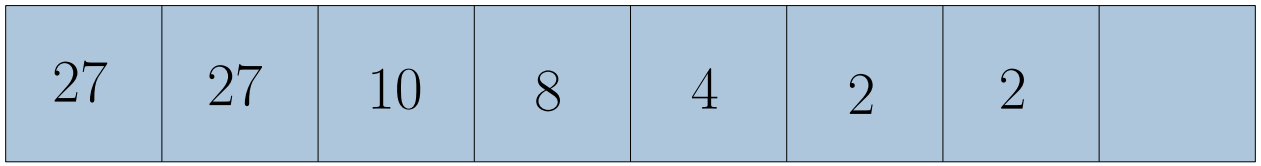
(a) true



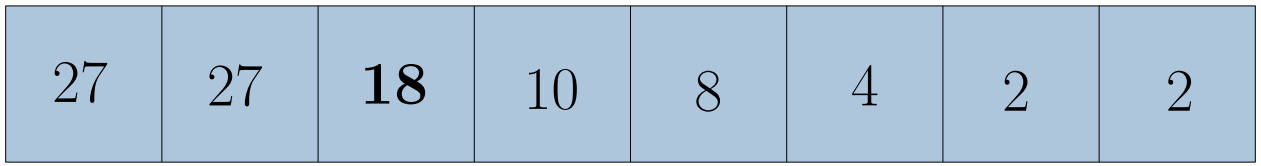
(b) false

enqueue behaviour

- Adds an item to the priority queue **in the right priority order**



(c) before enqueue of 18

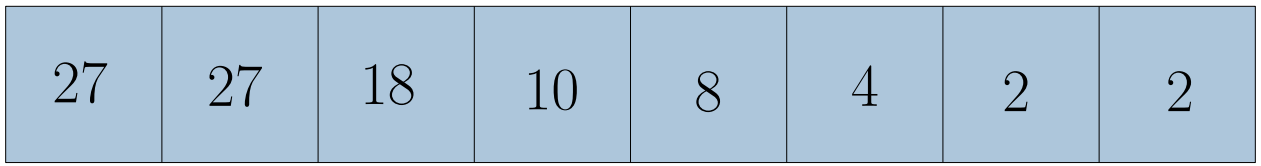


(d) after enqueue

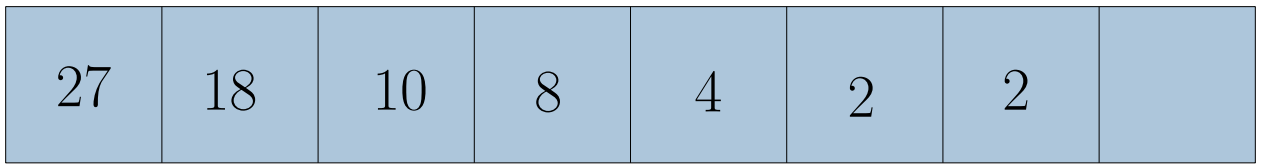
- This needs to be efficient.

dequeue behaviour

- Removes an item from the front of the priority queue
- The front is always the element of highest priority



(e) before dequeue d

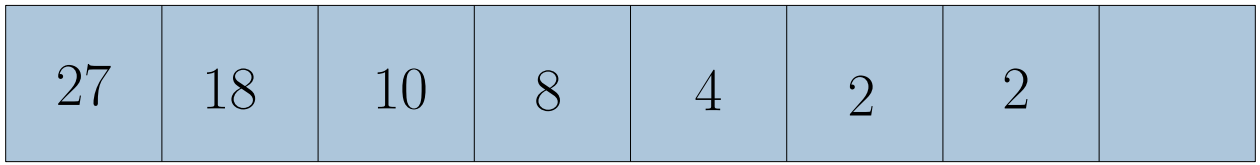


(f) after dequeue

- This needs to be efficient.

peek behaviour

- Returns the front of the priority queue

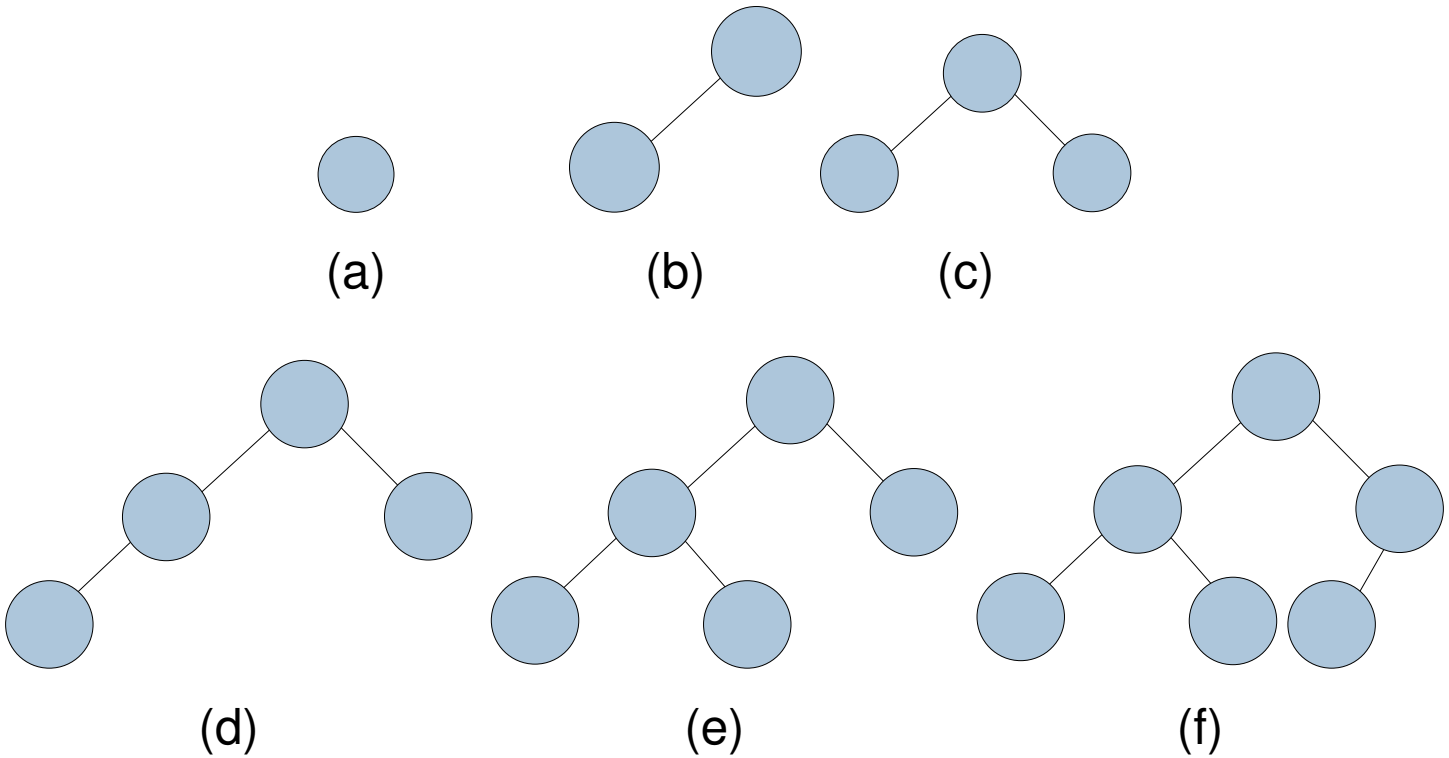


(g) returns 27

How do we do this?

- We do this using another ADT call a Heap
- Usually, heaps are implemented with trees
- There are max heaps and min heaps
  - Max heaps keep the maximum at the top
  - Min heaps keep the minimum at the top
- We will implement heaps with linked structures

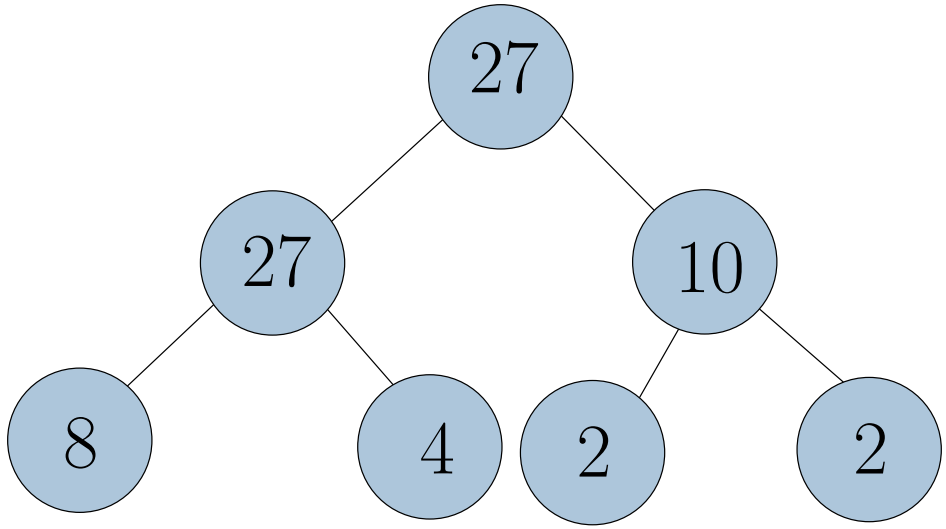
Insertion Order for Max Heaps



- Heaps are trees that grow using **level order** of the tree

Heap ADT Implemented with links

- Every node of the tree is greater than both its children

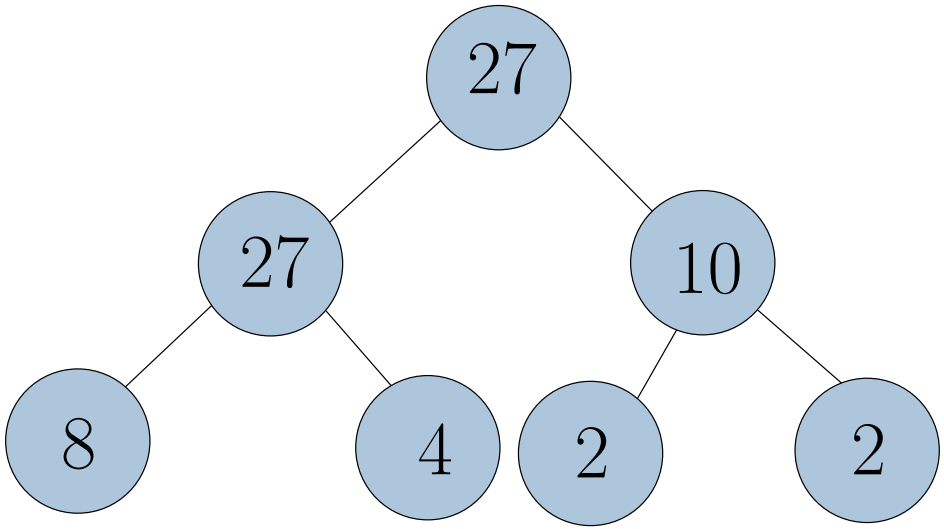


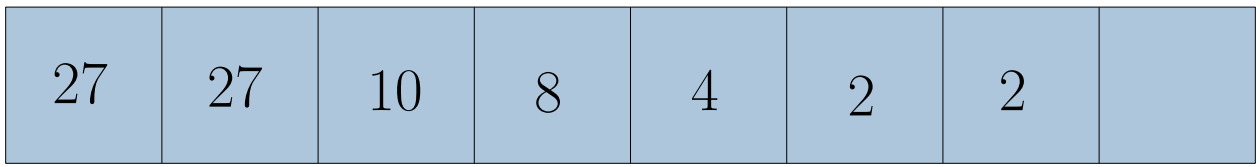
27	27	10	8	4	2	2	

- We don't know the absolute order of priorities, but we always know the maximum priority!
  - Why? Root will always be the maximum by definition

isEmpty Implementation

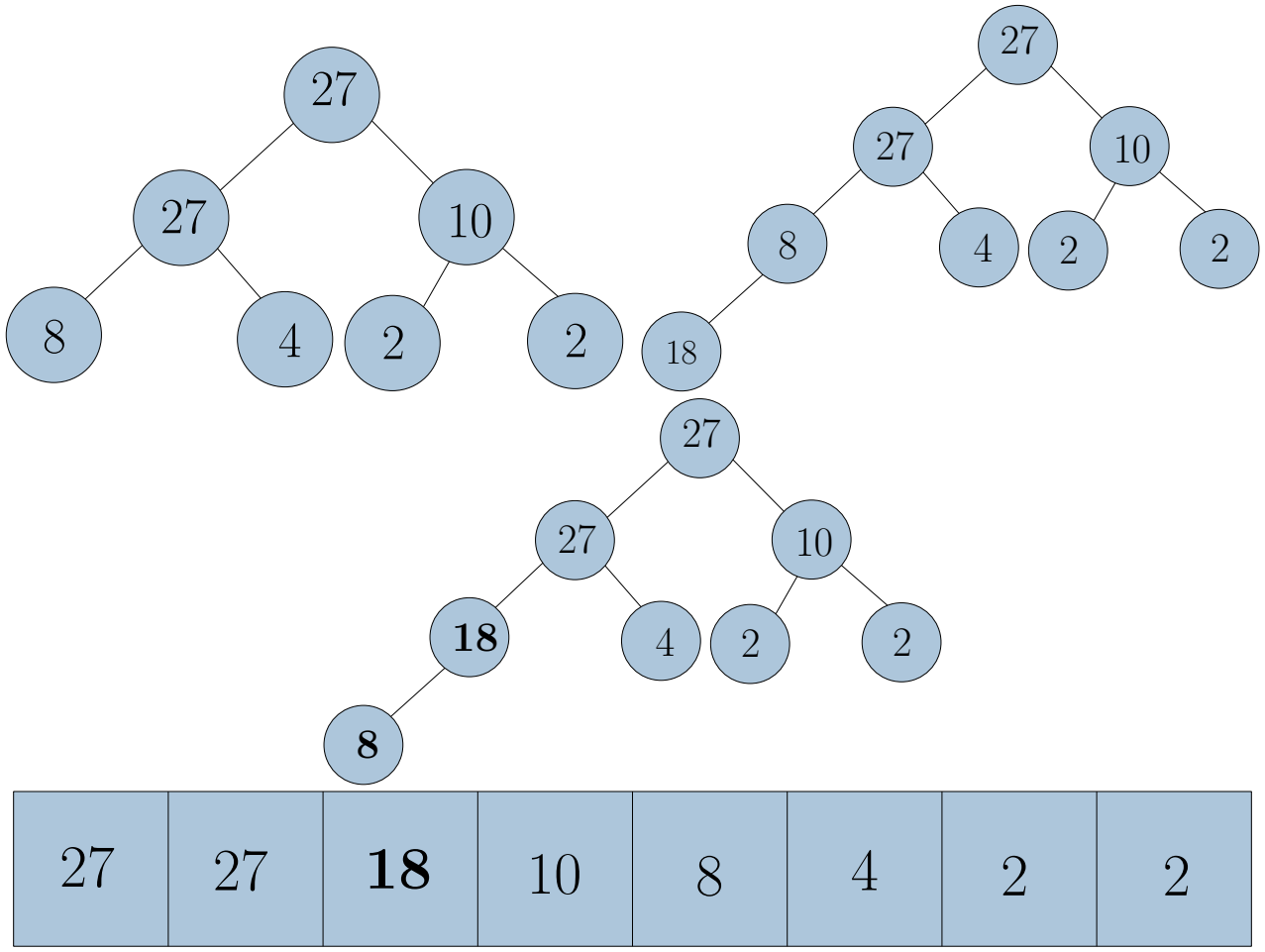
- Check the tree. If `root == null` return true.





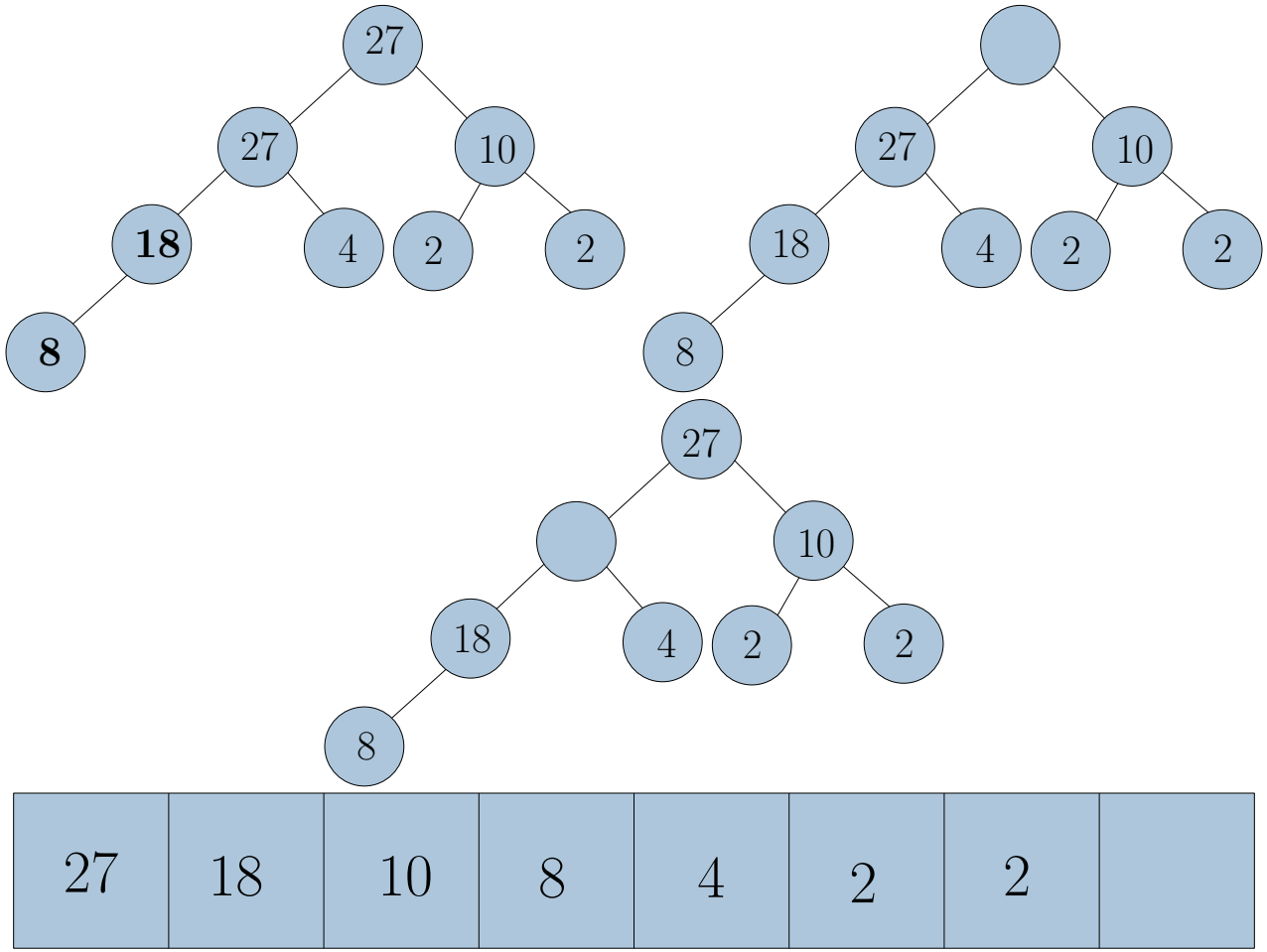
Enqueue Implementation

- Insert item at leaves. If out of order, swap to correct order



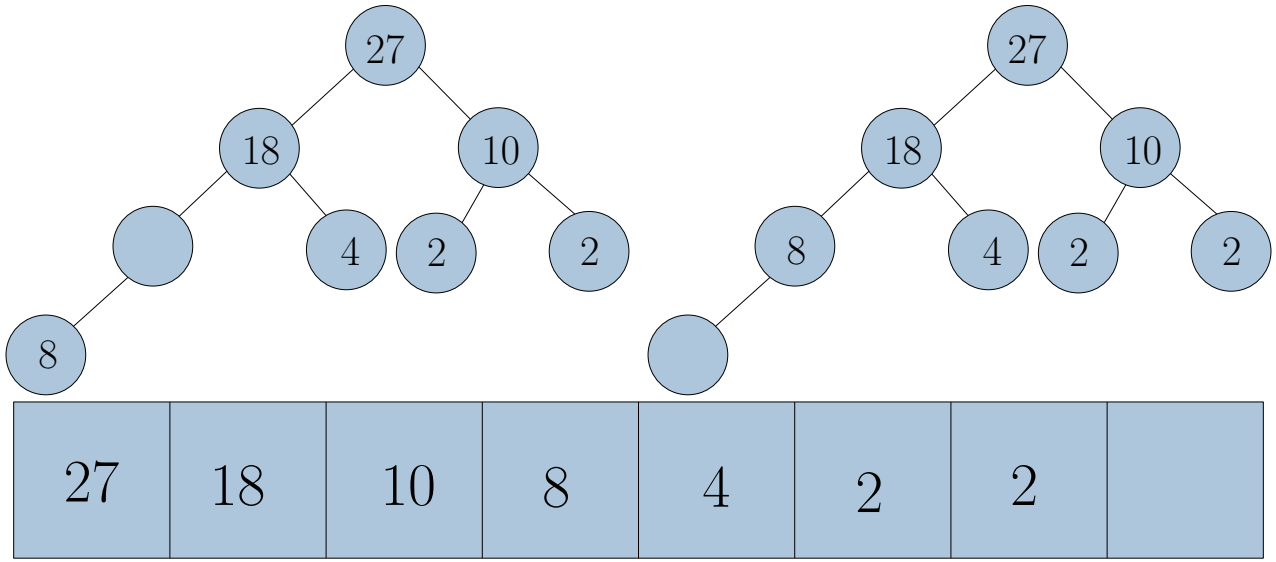
Dequeue Implementation

- Insert item at leaves. If out of order, swap to correct order



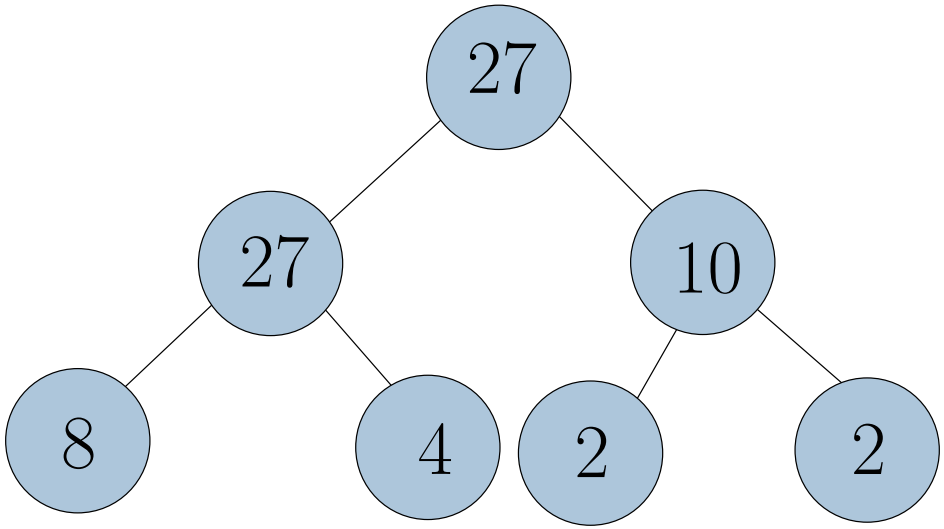
Dequeue Implementation

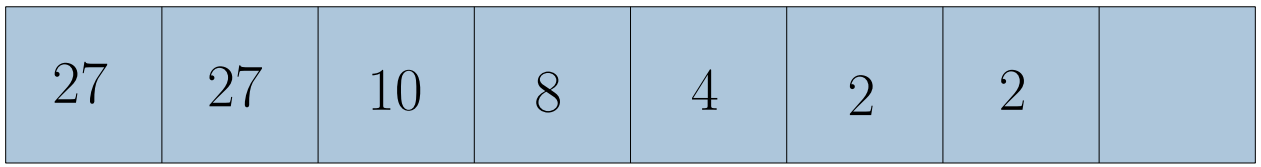
- Insert item at leaves. If out of order, swap to correct order



Peek Implementation

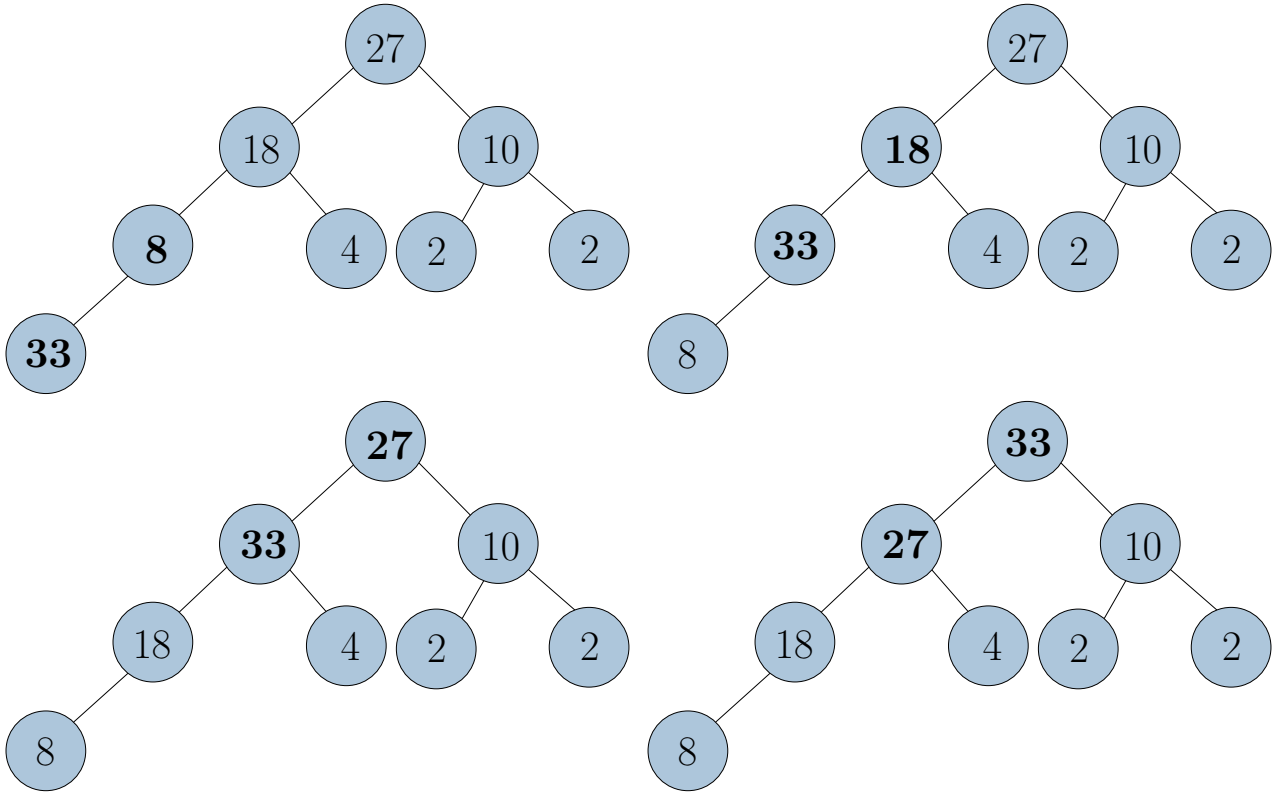
- Return the contents of the root if not null





Insertions and Many Swaps

- An insert could involve many swaps

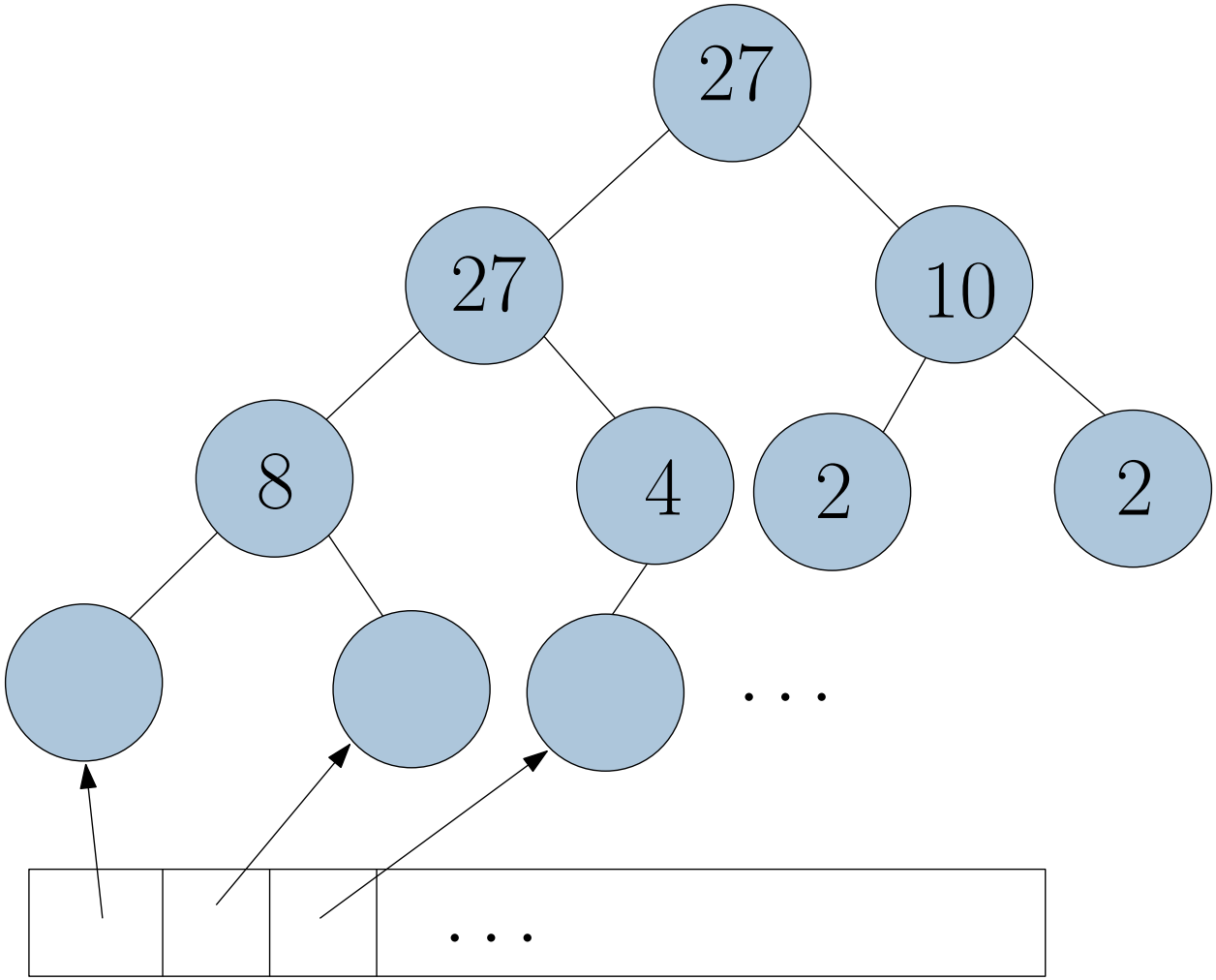


Dan! But How!

- How can the swap implementation be implemented?
- Tree node has a reference to an element
  - to swap, compare the two keys
  - if child is greater than parent, swap the references of the elements only

Dan! But How! (2)

- How do we do a level order traversal of a tree?
  - You can use a queue!



Summary

- Priority Queues are one of the most complete data structures we can look at.
  - Involves a tree and a queue
  - Introduces a new ADT called a heap
- It is just like a queue, but...
- High priorities percolate up to the top of the queue.