

# CSE 332

## Data Abstractions

## Priority Queues & Heaps



### Outline

#### 1 PriorityQueues

#### 2 Heaps

### Queue FIFOQueue vs. PriorityQueue

1

The Queue we've seen thus far is a FIFO (First-In-First-Out) Queue:

#### Queue (FIFOQueue) ADT

enqueue( <b>val</b> )	Adds <b>val</b> to the queue.
dequeue()	Returns the <b>least-recent</b> item not already returned by a dequeue. (Errors if empty.)
peek()	Returns the <b>least-recent</b> item not already returned by a dequeue. (Errors if empty.)
isEmpty()	Returns true if all inserted elements have been returned by a dequeue.

But sometimes we're interested in a PriorityQueue instead:

That is, a Queue that prioritizes certain elements (e.g. a hospital ER). Examples, in practice, include...

- OS Process Scheduling
- Sorting
- Compression (You did this already!)
- Greedy Algorithms (e.g. "shortest path")
- Discrete Event Simulation (priority = time step the event happens)

### PriorityQueues!

2

#### Queue (FIFOQueue) ADT

insert( <b>val</b> )	Adds <b>val</b> to the queue.
deleteMin()	Returns the <b>highest priority</b> item not already returned by a deleteMin. (Errors if empty.)
findMin()	Returns the <b>highest priority</b> item not already returned by a deleteMin. (Errors if empty.)
isEmpty()	Returns true if all inserted elements have been returned by a deleteMin.

- Data in PriorityQueues **must be comparable** (by priority)!
- Highest Priority = Lowest Priority Value
- The ADT **does not specify how to deal with ties!**



### Implementing A Priority Queue

3

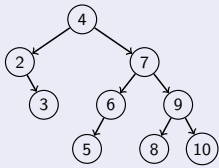
For each of the following potential implementations, what is the worst case runtime for insert and deleteMin? Assume all arrays do not need to resize.

- Unsorted Array
  - Insert by inserting at the end which is  $\mathcal{O}(1)$
  - deleteMin by linear search which is  $\mathcal{O}(n)$
- Unsorted Linked List
  - Insert by inserting at the front which is  $\mathcal{O}(1)$
  - deleteMin by linear search which is  $\mathcal{O}(n)$
- Sorted Circular Array List
  - Insert by binary search; shifting elements which is  $\mathcal{O}(n)$
  - deleteMin by moving front which is  $\mathcal{O}(1)$
- Sorted Linked List
  - Insert by linear search which is  $\mathcal{O}(n)$
  - deleteMin by remove at front which is  $\mathcal{O}(1)$
- Binary Search Tree
  - Insert by search which is  $\mathcal{O}(n)$
  - deleteMin by findMin which is  $\mathcal{O}(n)$

## A New Data Structure: Heap

4

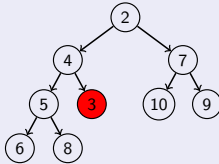
### Recall BSTs



**BST Property:**  
Left Children are smaller  
Right Children are larger

For a PriorityQueue, how could we store the items in a tree?

### And Now, Heaps



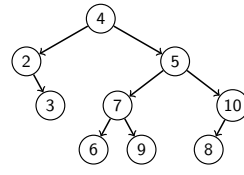
**Heap Property:**  
All Children are larger

**Structure Property:**  
Insist the tree has no "gaps"

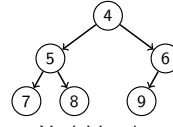
## Is It A Heap?

5

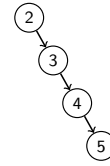
For each of the following, is it a heap?



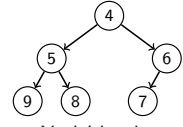
No, it fails the structure property. But if we replace the 6 with anything larger and take the piece from 5 down, it is.



Yup! It's a heap.



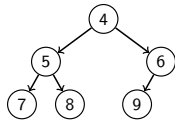
No, it fails the structure property. But 5 is.



Yup! It's a heap.

## Heap Properties?

6

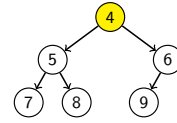


- Where is the minimum item in a heap?  
It's at the top!
- What is the height of a heap with  $n$  items?  
Note that all but the last row is full. So,  $n \leq \sum_{i=0}^{k-1} 2^i = 2^k - 1$ . So,  $\lg n \leq k$ .  
So,  $k = \lceil \lg n \rceil$ .
- How do we implement a PriorityQueue as a Heap?  
findMin is easy, but ... deleteMin? insert?

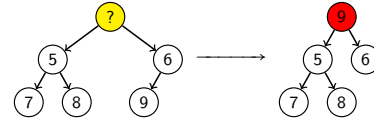
## Implementing deleteMin For a Heap

7

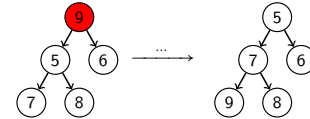
- Find the min:



- Remove the min and fill the hole with the last child



- "Percolate Down" to fix the invariant:



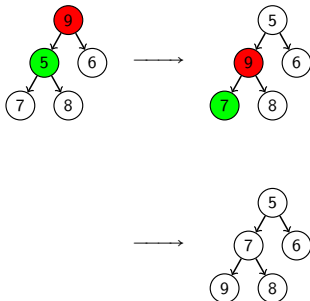
## "Percolate Down"?

8

```

1 percolateDown(node) {
2   while (node.data is greater than either child) {
3     swap data with smaller child
4   }
5 }

```



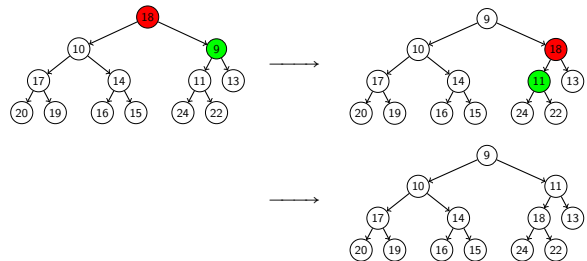
## "Percolate Down" (Another Example)

9

```

1 percolateDown(node) {
2   while (node.data is greater than either child) {
3     swap data with smaller child
4   }
5 }

```



### Runtime Analysis?

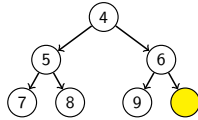
The height of the heap is  $\lceil \lg n \rceil$ . So, the runtime is  $O(\lg n)$ .

## Implementing insert For a Heap

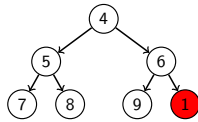
10

Let's try insert(1):

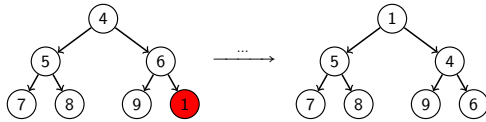
- Where do we put a new item?



- Fill our new hole with 1:



- "Percolate Up" to fix the invariant:

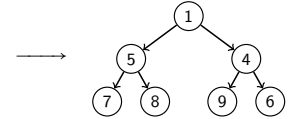
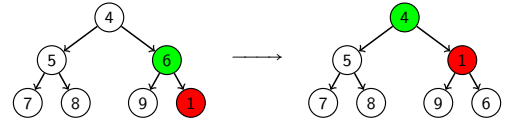


## "Percolate Up"?

11

```

1 percolateUp(node) {
2   while (node.data is smaller than parent) {
3     swap data with parent
4   }
5 }
    
```

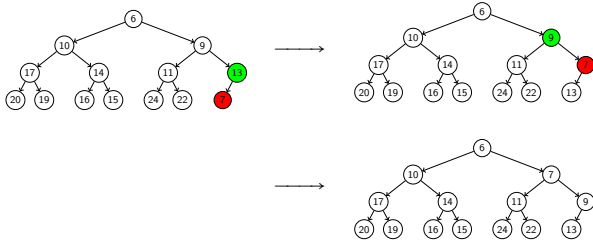


## "Percolate Up" (Another Example)

12

```

1 percolateUp(node) {
2   while (node.data is smaller than parent) {
3     swap data with parent
4   }
5 }
    
```



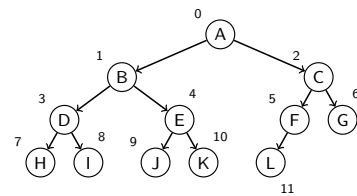
### Runtime Analysis?

The height of the heap is  $\lceil \lg n \rceil$ . So, the runtime is  $\mathcal{O}(\lg n)$ .

## And... how do we implement Heap?

13

We've insisted that the tree be complete to be a valid Heap. Why?



Fill in an array in **level-order** of the tree:

heap: 

A	B	C	D	E	F	G	H	I	J	K	L	0	0	0
<small>h(0)</small>	<small>h(1)</small>	<small>h(2)</small>	<small>h(3)</small>	<small>h(4)</small>	<small>h(5)</small>	<small>h(6)</small>	<small>h(7)</small>	<small>h(8)</small>	<small>h(9)</small>	<small>h(10)</small>	<small>h(11)</small>	<small>h(12)</small>	<small>h(13)</small>	<small>h(14)</small>

If I have the node at index  $i$ , how do I get its:

- Parent?  $3 \rightarrow 1, 4 \rightarrow 1, 10 \rightarrow 4, 9 \rightarrow 4, 1 \rightarrow 0$

This indicates that it's approximately  $n/2$ . In fact, it's  $\frac{n-1}{2}$ .

- Left Child?  $2(n+1) - 1$
- Right Child?  $2(n+1)$