CS200: Priority Queues, Heaps

Prichard Ch. 12

Priority Queues

- Characteristics
  - Items are associated with a Comparable value: priority
  - Provide access to one element at a time - the one with the highest priority

- Uses
  - Operating systems
  - Network management
    - Real time traffic usually gets highest priority when bandwidth is limited

Priority Queue ADT Operations

1. Create an empty priority queue

   `createPQueue()`

2. Determine whether empty

   `pqIsEmpty():boolean`

3. Insert new item

   ```
   pqInsert(in newItem:PQItemType) throws
       PQueueException
   ```

4. Retrieve and delete the item with the highest priority

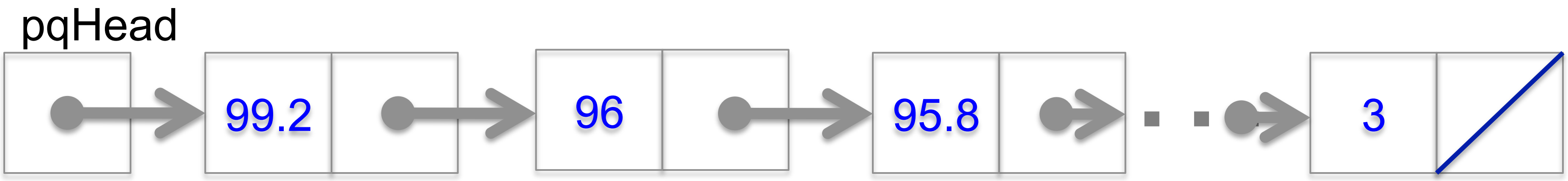
   `pqDelete():PQItemType`

PQ – ArrayList Implementation

30		3	20	…	95	95	96	99
size		0	1					29

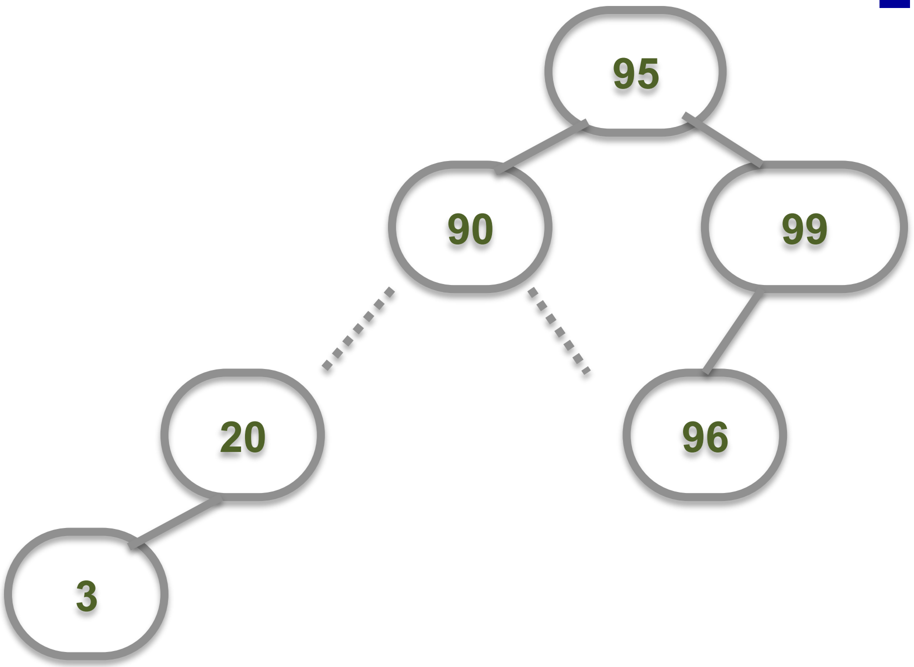
- ArrayList ordered by priority
  - pqInsert: find the correct position for add at that position, the ArrayList.add(i,item) method will shift the array elements to make room for the new item
  - pqDelete: remove last item (at size()-1)
  - **Why did we organize it in increasing order?**

PQ – Reference-based Implementation



- Reference-based implementation
  - Sorted in descending order
    - Highest priority value is at the beginning of the linked list
    - `pqDelete` returns the item that `pqHead` references and changes `pqHead` to reference the next item.
    - `pqInsert` must traverse the list to find the correct position for insertion.

PQ – BST Implementation



- Binary search tree
  - Where is the highest value of the nodes?
  - pqInsert is easy, why?
    - at a new leaf, e.g.30
  - pqDelete?
    - need to remove the max
    - also easy, why?
      - max has at most one child

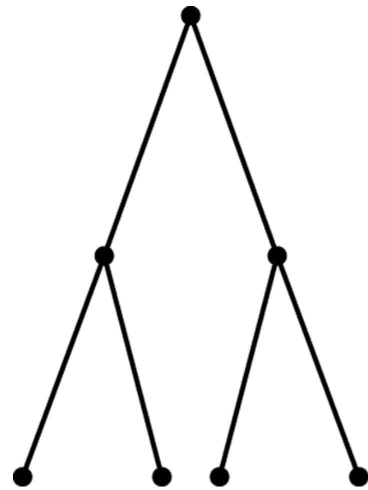
The problem with BST

- BST can get unbalanced so in the worst case pqInsert and pqDelete can get O(n)
- A more balanced tree structure would be better.
- What is a **balanced** binary tree structure?
  - Height of any node's right sub-tree differs from left sub-tree by 0 or 1
- A complete binary tree is balanced, and it is easy to put the nodes in an array. WHY?

Question: is a balanced binary tree complete?

Recap tree definitions

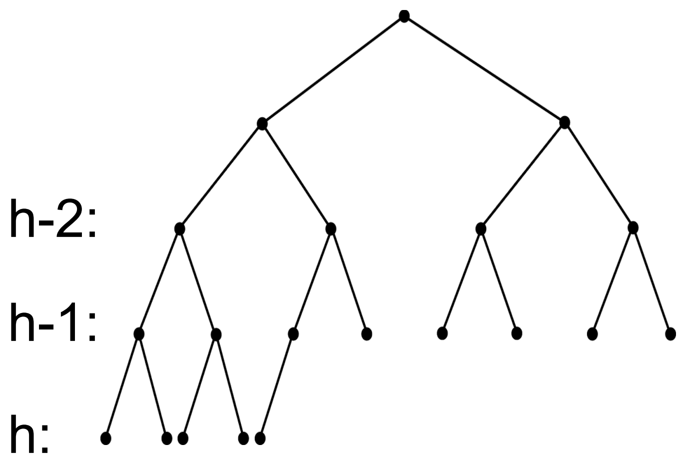
- m-ary tree
  - Every internal vertex has no more than m children.
  - Our main focus will be binary trees
- Full m-ary tree
  - all interior nodes have m children
- Perfect m-ary tree
  - Full m-ary tree where all leaves are at the same level



- Perfect binary tree
  - number of leaf nodes: 2^{h-1}
  - total number of nodes: $2^h - 1$

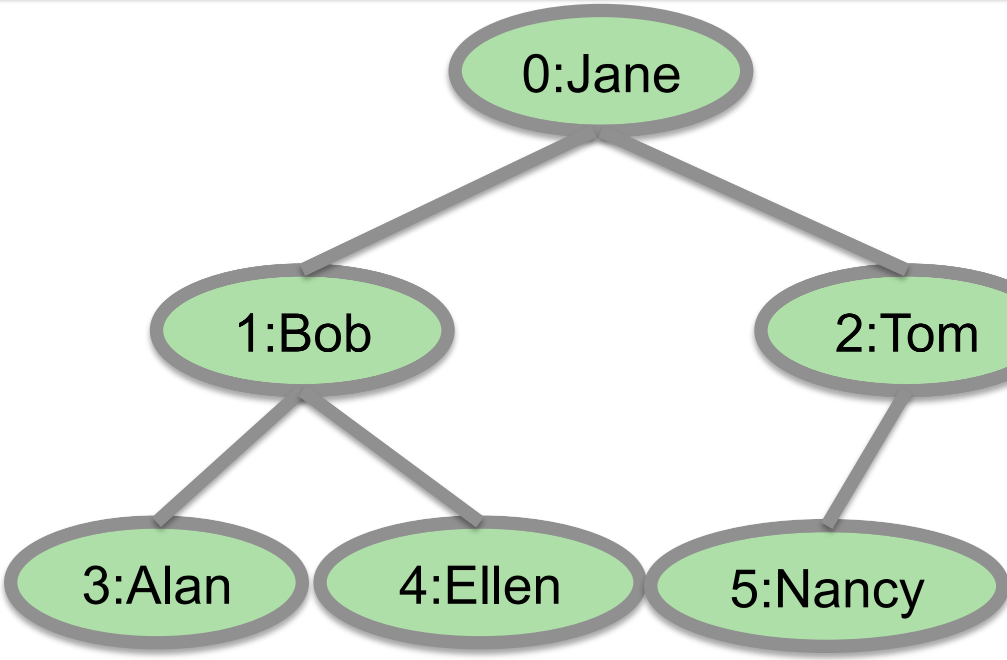
More tree definitions

- Complete binary tree of height h
  - zero or more rightmost leaves not present at level h
- A binary tree T of height h (Prichard) is complete if
  - All nodes at level h – 2 and above have two children each, and
  - When a node at level h – 1 has children, all nodes to its left at the same level have two children each, and
  - When a node at level h - 1 has one child, it is a left child
  - So the leaves at level h go from left to right



Complete Binary Tree

Level-by-level numbering of a complete binary tree, NOTE 0 based!



What is the parent child index relationship?

*left child i: at 2*i+1*

right child i: at 2(i+1)*

lparent i: at (i-1)/2

So we can store a complete binary tree in an array!!

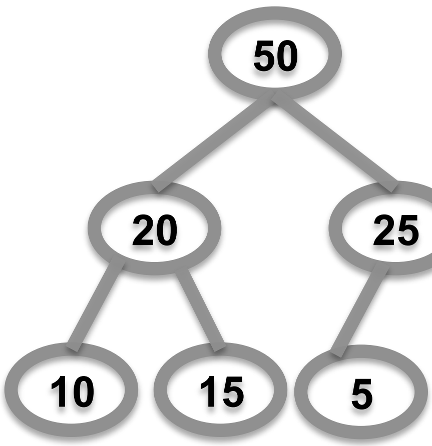
Heap - Definition

- A maximum heap (maxheap) is a ***complete*** **binary tree** that satisfies the following:
  - It is a leaf, or it has the heap property:
    - Its root contains a key greater or equal to the keys of its children
    - Its left and right sub-trees are also maxheaps
  - A minheap has the root less or equal children, and left and right sub trees are also minheaps

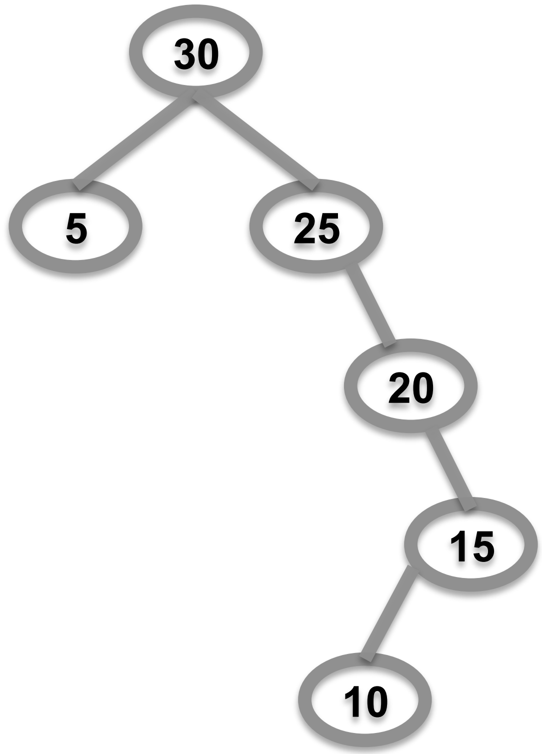
maxHeap Property Implications

- Implications of the heap property:
  - The root holds the maximum value (global property)
  - Values in descending order on every path from root to leaf
- A Heap is NOT a binary search tree, as in a BST the nodes in the right sub tree of the root are larger than the root

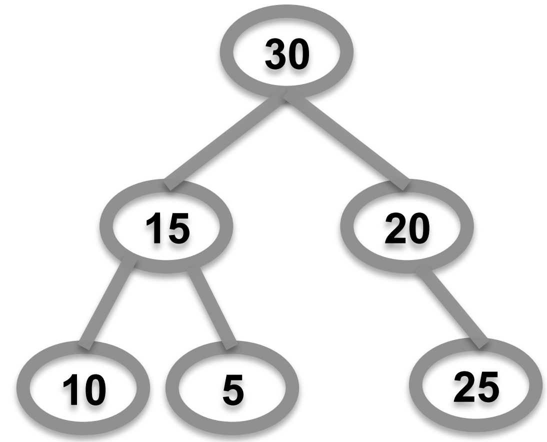
Examples



Satisfies heap property AND Complete



Satisfies heap property BUT Not complete



Does not satisfy heap property AND Not complete

Questions

- Is the root of a max heap the max of the tree?
- Is there a traversal (pre, in, post, level) that sorts a max heap?
- Is the path from a leaf to the root sorted?

Heap ADT

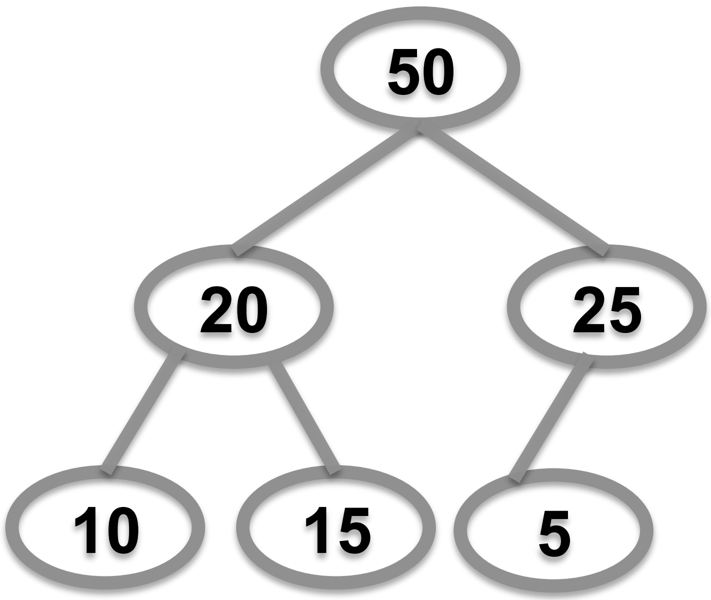
```
createHeap()  // create empty heap

heapIsEmpty():boolean
  // determines if empty

heapInsert(in newItem:HeapItemType)
  throws HeapException
  /* inserts newItem based on its search key.
    Throws exception if heap full
    This may not happen if e.g.implemented
    with an ArrayList */

heapDelete():HeapItemType
  // retrieves and then deletes heap's root
  // item which has largest search key
```

Array(List) Implementation



0	50
1	20
2	25
3	10
4	15
5	5

Array(List) Implementation

- Traversal:
  - Root at position 0
  - Left child of position i at position 2*i+1
  - Right child of position i at position 2*(i+1)
  - Parent of position i at position (i-1)/2

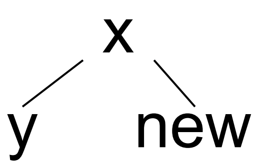
    (int arithmetic **truncates**)

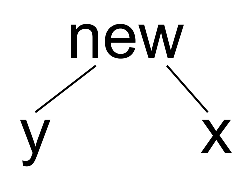
Heap Operations - `heapInsert`

- **Step 1**: put a new value into first open position (maintaining completeness), i.e. at the end
- but now we potentially violated the heap property, so:
- **Step 2**: bubble values up
  - **Re-enforcing the heap property**
  - Swap with parent, if new value > parent, until in the right place.
  - The heap property holds for the tree below the new value, when swapping up

Swapping up

- Swapping up enforces heap property for sub tree below the new, inserted value:



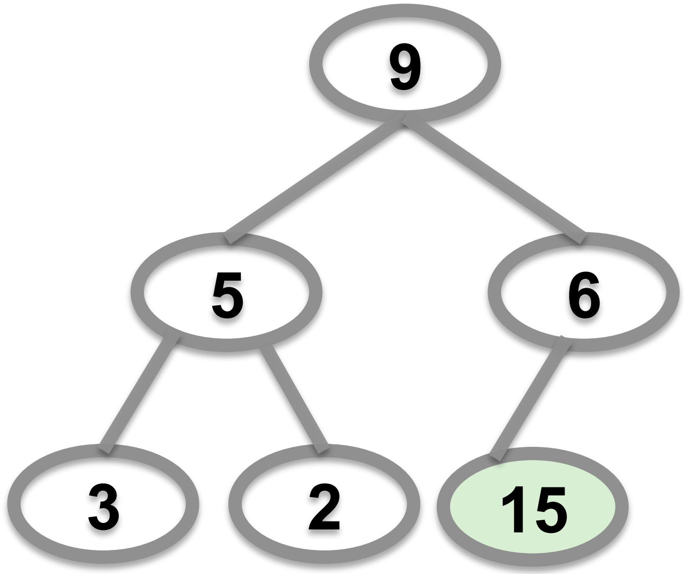


- if (new > x) swap(x,new)

x>y, therefore

new > y

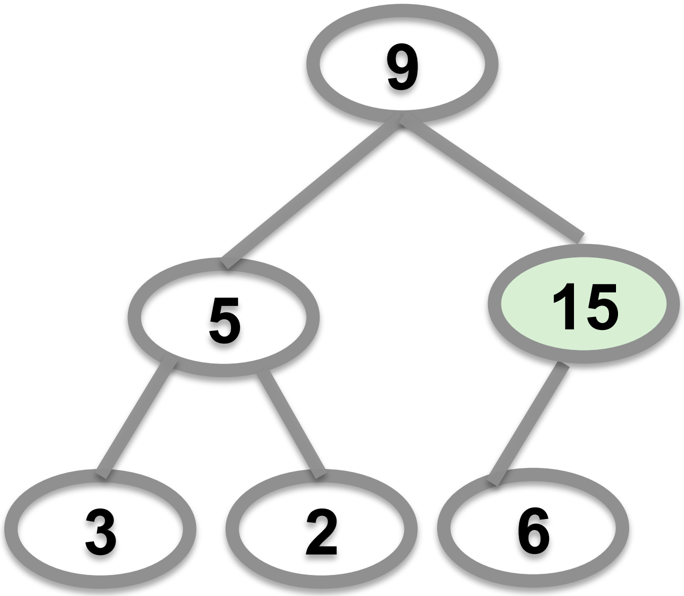
Insertion into a heap (Insert 15)



Insert 15

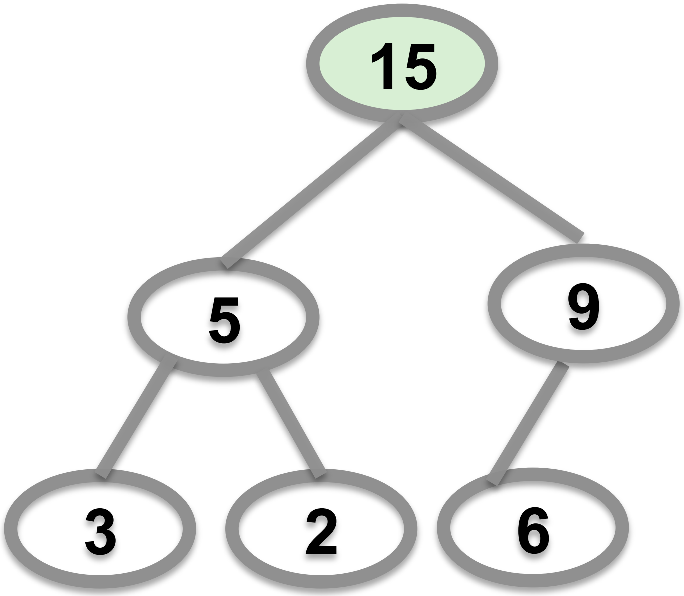
bubble up

Insertion into a heap (Insert 15)



bubble up

Insertion into a heap (Insert 15)

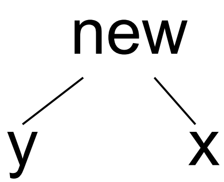


Heap operations – `heapDelete`

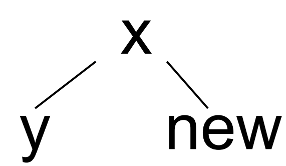
- **Step 1**: remove value at root (Why?)
- **Step 2**: substitute with rightmost leaf of bottom level (Why?)
- **Step 3**: percolate / bubble down
  - Swap with **maximum** child as necessary, until in place
  - each bubble down restores the heap property for the max child
  - this is called **HEAPIFY**

Swapping down

- Swapping down enforces heap property at the swap location:

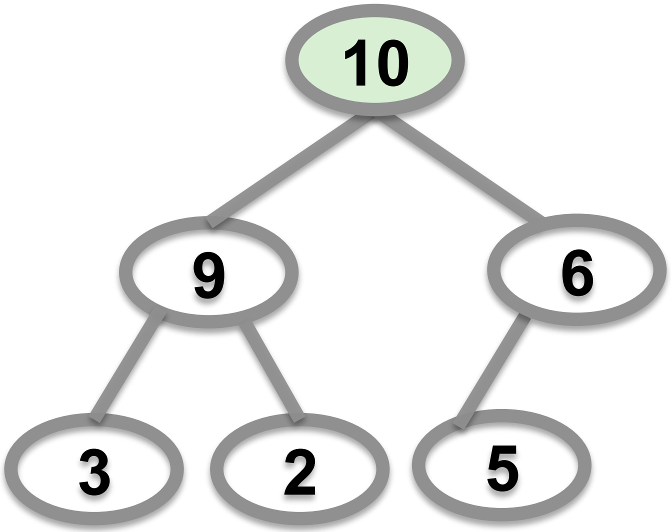


- new<x and y<x:
    swap(x,new)



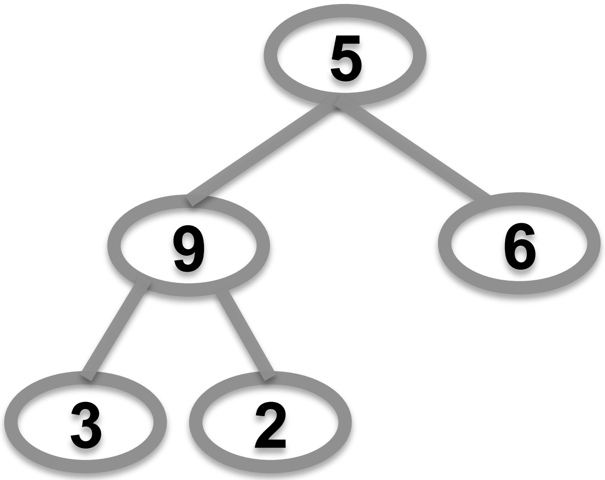
x>y and x>new

Deletion from a heap

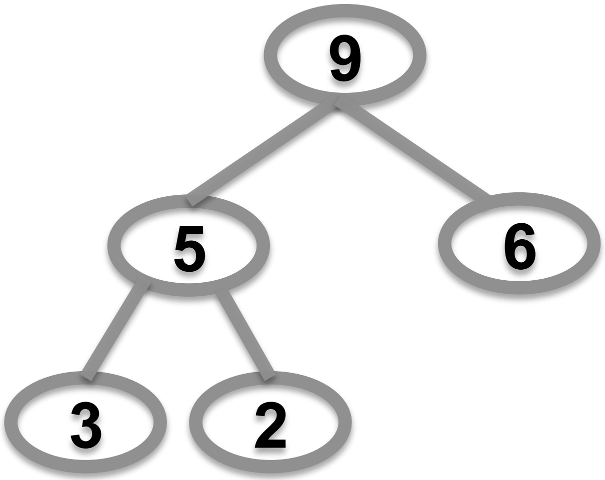


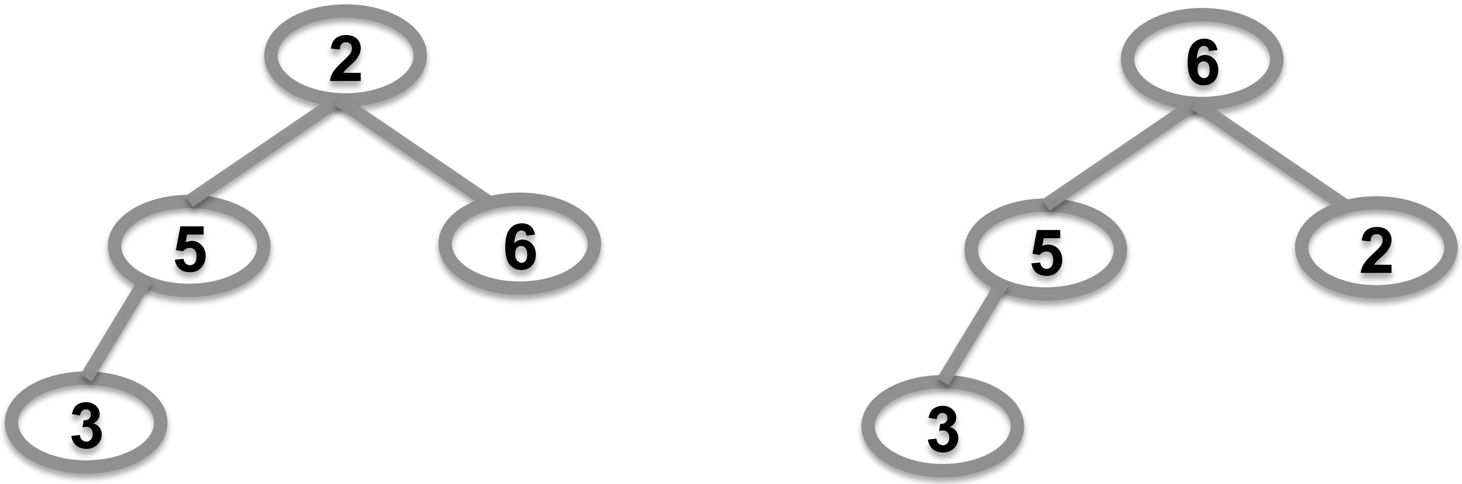
Delete 10
Place last node in root

bubble down
heapify
draw the heap



delete again
draw the heap




2
5
6
3
6
5
2
3

Array-based Heaps: Complexity

	Average	Worst Case
insert	O(log n)	O(log n)
delete	O(log n)	O(log n)

Heap versus BST for PriorityQueue

- BST can also be used to implement a priority queue
- How does worst case complexity compare?
- How does average case complexity compare?
  - what does it assume?

Small number of priorities

- A heap of queues with a queue for each priority value.

HeapSort

- Algorithm
  - Insert all elements (one at a time) to a heap
  - Iteratively delete them
    - Removes minimum/maximum value at each step
- Computational complexity?
- Let's check out the code

HeapSort

- Alternative method (in-place):
  - **buildHeap:** create a heap out of the input array:
    - Consider the input array as a complete binary tree
    - Create a heap by iteratively expanding the portion of the tree that is a heap
      - Leaves are already heaps
      - Start at last internal node
      - **Go backwards** calling **`heapify`** with each internal node
  - Iteratively swap the root item with last item in unsorted portion and rebuild

Building the heap

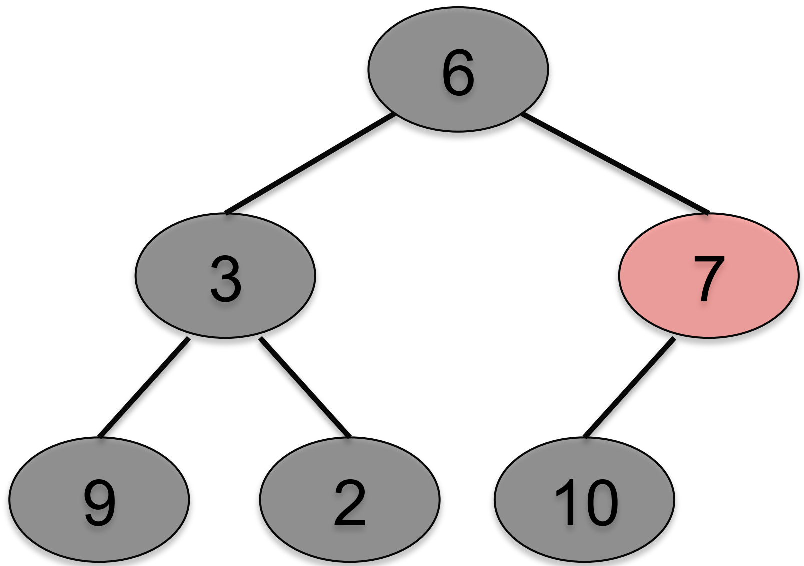
```
buildheap(n){
    for (i = (n-2)/2 down to 0)
    //pre: the tree rooted at index is a semiheap
    //i.e., the sub trees are heaps
    heapify(i); // bubble down
    //post: the tree rooted at index is a heap
}
```

- WHY start at (n-2)/2?
- WHY go backwards?

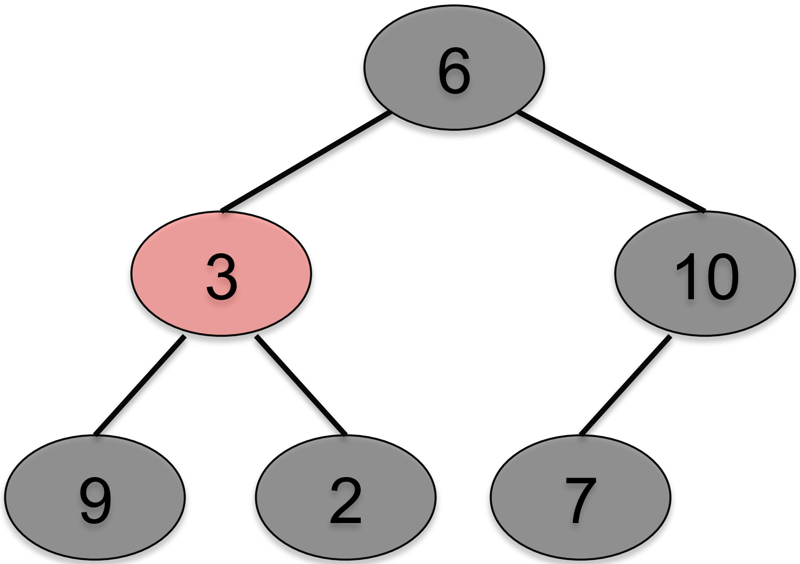
- The whole method is called **buildHeap**
- One bubble down is called **heapify**

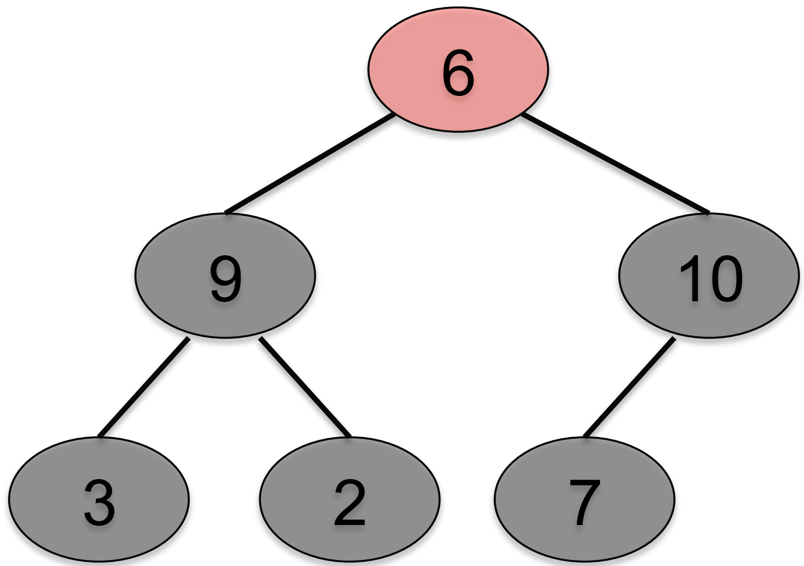
6	3	7	9	2	10

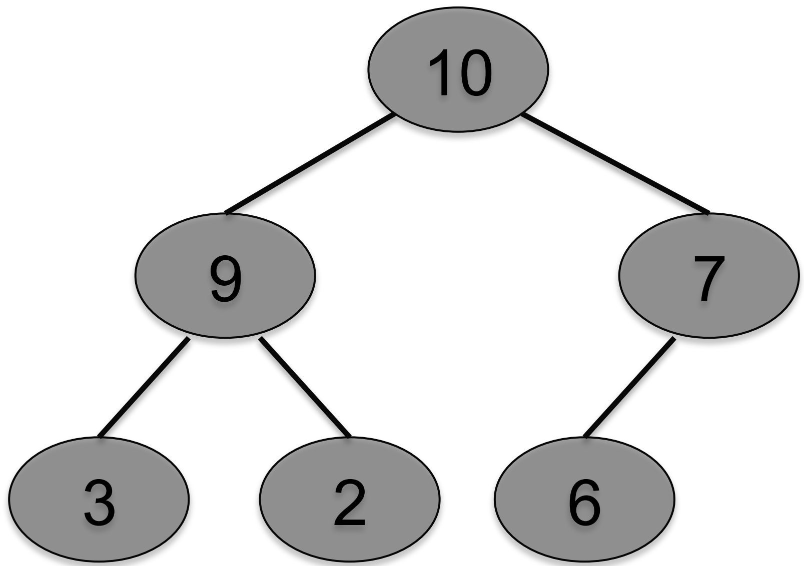
Draw as a Complete Binary Tree:



Repeatedly heapify, starting at last internal node, going backwards


6
3
10
9
2
7

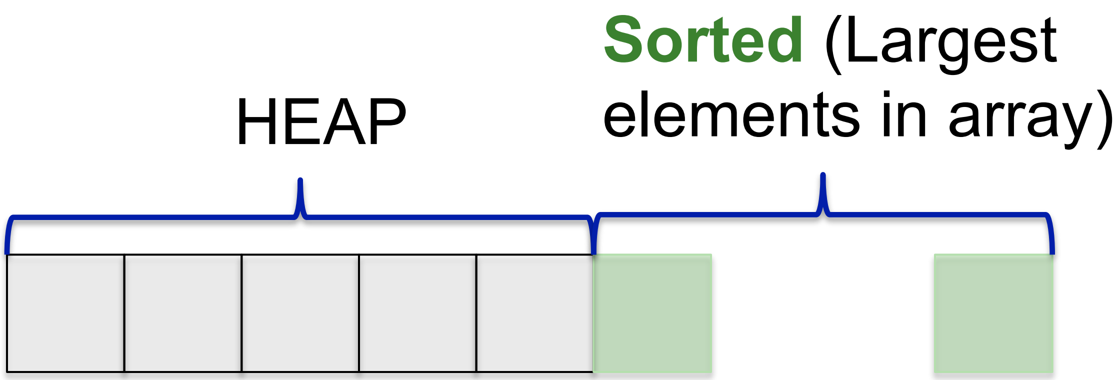

6
9
10
3
2
7



10	9	7	3	2	6

In place heapsort using an array

- First build a heap out of an input array using buildHeap(). See previous slides.
- Then partition the array into two regions; starting with the full heap and an empty sorted and stepwise growing sorted and shrinking heap.



Do it, do it

HEAP

10	9	6	3	2	5
9	5	6	3	2	10
6	5	2	3	9	10
5	3	2	6	9	10
3	2	5	6	9	10
2	3	5	6	9	10
2	3	5	6	9	10

SORTED