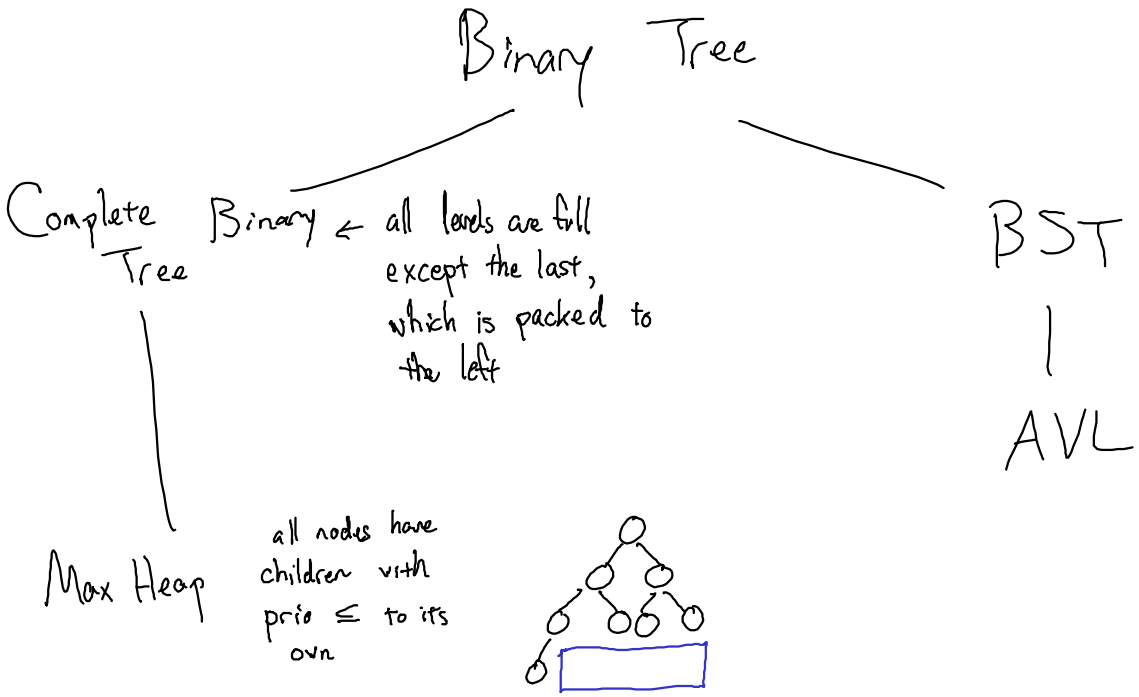
Last time on Priority Queues

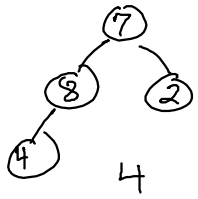
ADT: PriorityQueue — collection where adding element includes priority and remove (dequeue) gives highest priority element

```
void   enqueue(P prio, V value)
V      dequeue()
V      peek()
P      peekPriority()
int    getSize()
```



Method enqueue(prio, element)
    add (prio, element) to end of tree
    bubble up the new node
End Method

Since BT is complete, there is only one shape per size.

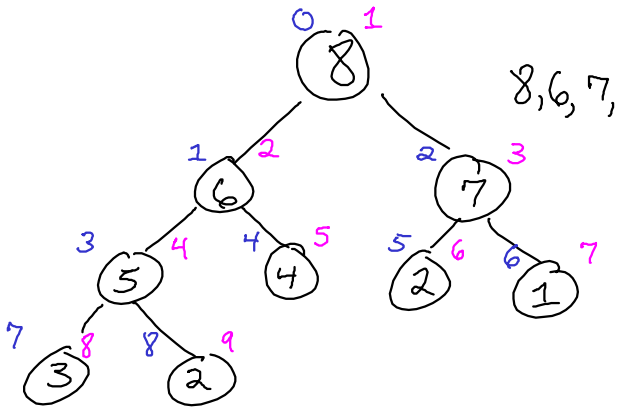


7, 8, 2, 4

4

- add to "end" of CBT
- remove from "end"
- swap nodes
- find a node's parent and children

ArrayList stores a Complete Binary Tree



8,6,7,5,4,2,1,3,2

(1-based)
Function getParentIndex(idx)
    Return $\lfloor \frac{idx}{2} \rfloor$
End Function

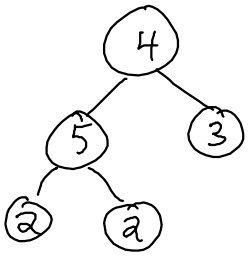
(0-based)
Function getParentIndex(idx)
    Return $\left\lfloor \frac{idx+1}{2} \right\rfloor - 1$
End Function

O(1)

Function getLeftChildIndex(idx)
    Return idx*2+1

Function getRightChildIndex(idx)
    Return idx*2+2

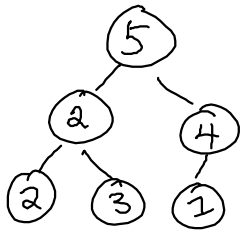
```
Method enqueue(prio, value):
    contents.insertLast((prio, value))
    bubbleUp(contents.getSize()-1)
End Method
Method bubbleUp(idx):
    If idx==0: Return
    pidx ← getParentIndex(idx)
    If contents.get(idx) > contents.get(pidx):
        Swap arraylist elements at idx and pidx
        bubbleUp(pidx)
    End If
End Method
```



4, 5, 3, 2, 2

blue red green

```
Method dequeue():
    Swap first & last in contents
    answer ← contents.removeLast()
    bubbleDown(0)
End Method
```

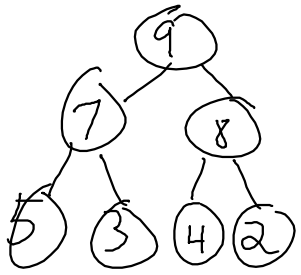


2, 5, 4, 2, 3, 1

```
Method bubbleDown(idx):
    idxR ← getRightChildIndex(idx)
    idxL ← getLeftChildIndex(idx)
    If idxR < contents.getSize():
        If contents.get(idxL) < contents.get(idxR) And contents.get(idxR) > contents.get(idx):
            Swap elements at idx with idxR
            bubbleDown(idxR)
        Else If contents.get(idxL) ≥ contents.get(idxR) And contents.get(idxL) > contents.get(idx)
            Swap elements at idx with idxL
            bubbleDown(idxL)
        End If
    Else idxL < contents.getSize()
        (one child)
    End If
End Method
```

not based on ↓ ↑ [4, 7, 8, 5, 3, 9, 2]

Top-k algorithm takes O(n) time



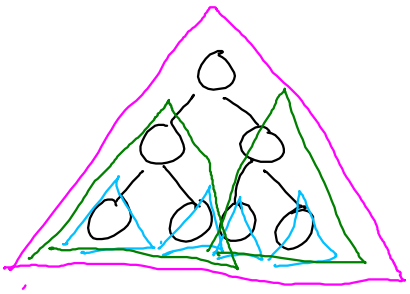
Method heapify():

For each index from contents.getSize()-1 down to 0:

bubbleDown(index)

EndFor

EndMethod



Most of these bubbleDowns are small.