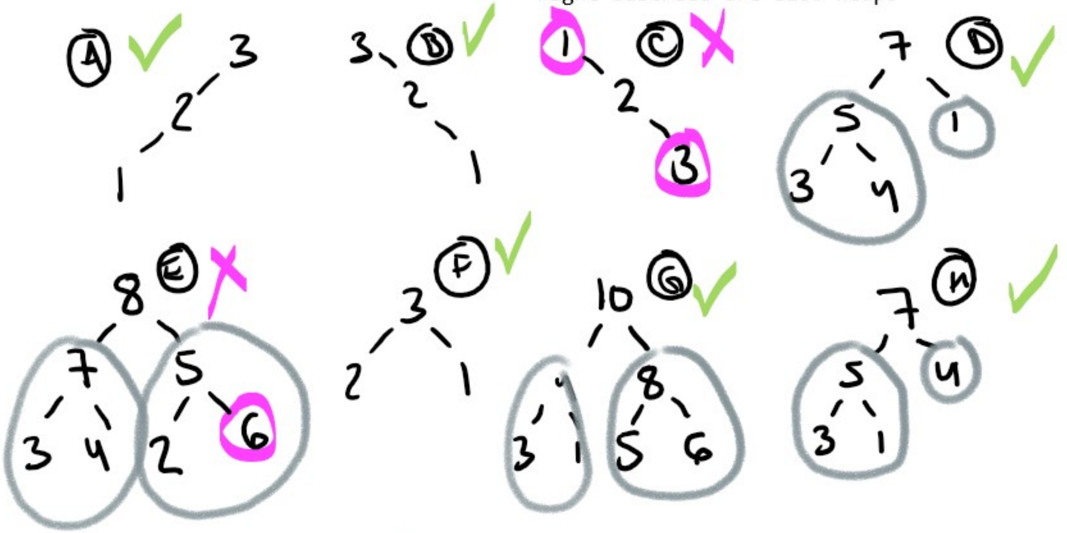
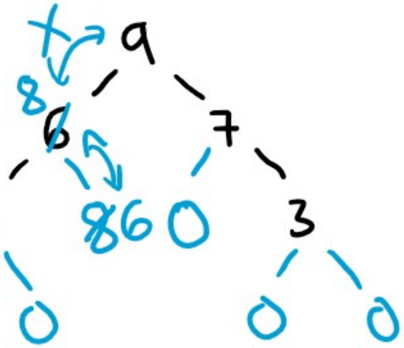
Which of these are max heaps?

Wednesday, November 16, 2022 1:19 PM

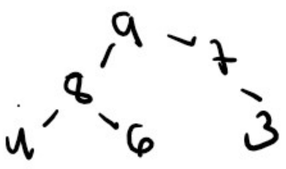
Max heap: binary tree where the maximum element is the root, and the left and right subtrees are also heaps



review of insertion:
insert 8



result:

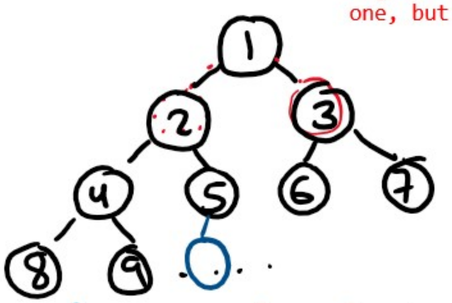


- a balanced heap reduces runtime
  want height to be log(# elts)
- what if we inserted systematically?
  "row-by-row", "left-to-right"

Challenge: if we use our BinTree class, how do we find the next empty spot to insert in? The possible solution has a downside (note that this only considers inserting into an empty spot, NOT swapping upwards):

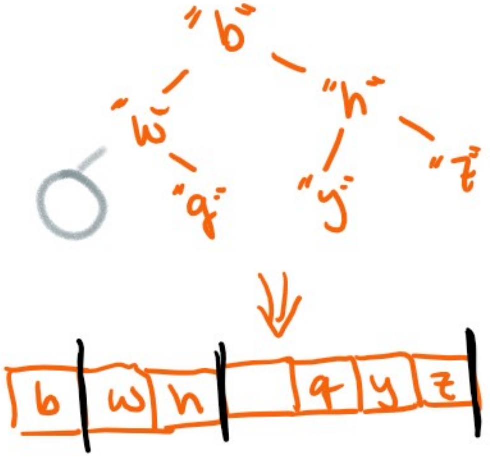
```
def insert_anywhere(self, new_elt):
    ''' inserts into an empty spot in the tree '''
    if (not self.left):    # found empty spot in left subtree
        self.left = BinTree(new_elt) # still need to swap up
    elif (not self.right): # found empty spot in right subtree
        self.right = BinTree(new_elt) # still need to swap up
    else:
        self.left.insert_anywhere(new_elt) # find a spot in the left subtree
    # could have also done: self.right.insert_anywhere(new_elt)
```

This "biases" the tree to the left (aka leads to an imbalanced tree). We could try to compare the heights of the left and right subtrees and insert into the smaller one, but that leads to complicated code.

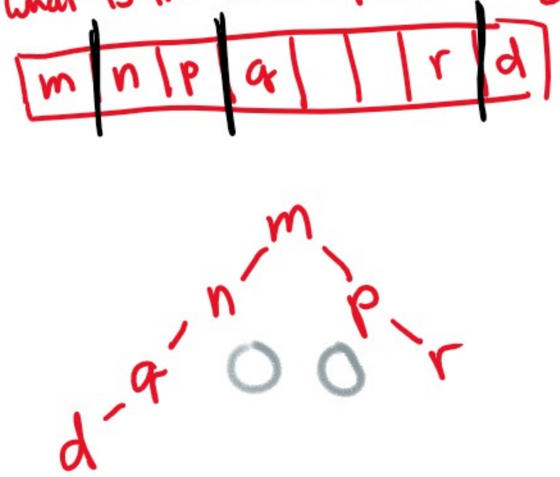


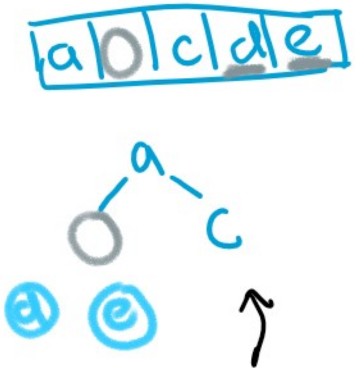
↑ hard to do w/ our recursive BinTree class :(

array representation of a binary tree

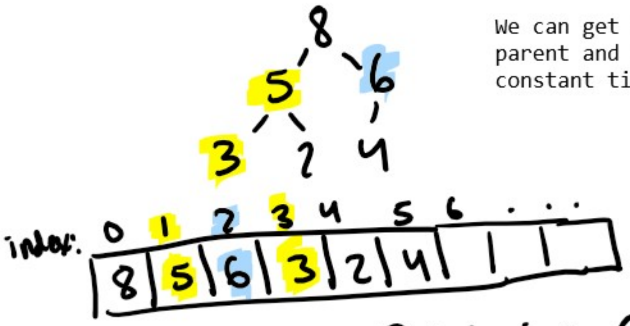


what is the tree represented by:





not all arrays can represent valid trees!



We can get an element's parent and children in constant time:

left child of elt. @ ind. i is @ ind $2*i+1$

right child is @ ind $2*i+2$

parent@ floor$\left(\frac{i-1}{2}\right)$

If we want to enforce heap structure in the way we said, this array representation will have no "holes" as long as we always insert in the first available slot of the array and "sift up"! This means this representation leads to a **balanced** and **easy-to-insert-into** heap!!