Priority queue (PQ): needs 3 operations

1. Insert a new item
2. Remove max-priority item
3. Get max-priority item (without removing) (peek)

How do we build one from scratch?

Some data structure: arrays, lists, trees, hash maps/dictionary, hash sets/set, graphs

How might you use these? Which ones might be best?

HashSets/set

=> Not really a way to specify priority

HashMap/dict

1. Key = priority, Value = item
2. Key = item, Value = priority

→ We *could* implement a heap using one of these options, but we would need to search the whole map O(N)

Linked list
Array list

keep a sorted list

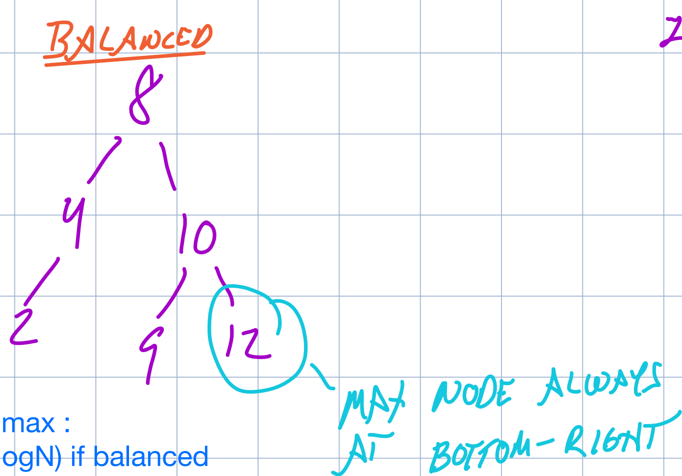
- get-max: O(1) (pick first element)
- remove-max: O(1) for linked list, for array list would need to shift elements to keep sorted order => O(N)
- insert: O(N) to find position in sorted list

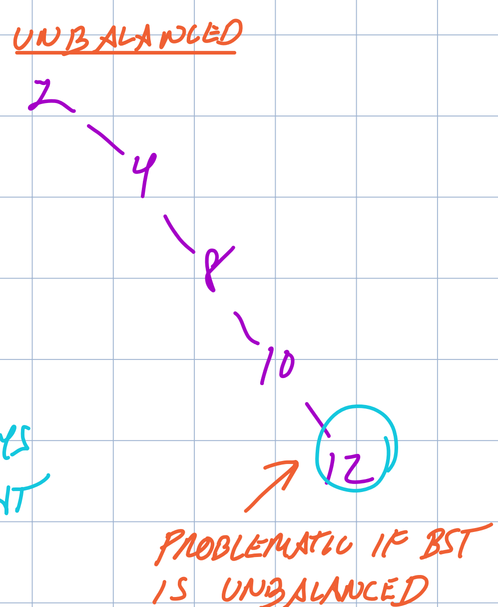
Trees

We know one way to do an ordered representation with trees…

=> BST (Binary search tree): for any node, every smaller node is on the left, any larger node is on the right

What if we used this as a priority queue?





insert, get-max :

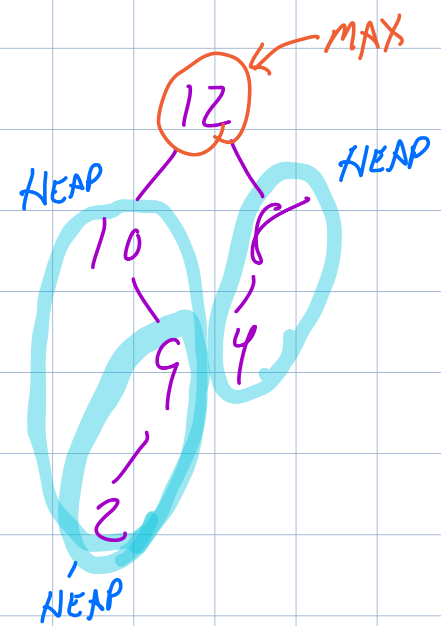
O(logN) if balanced

O(N) if unbalanced

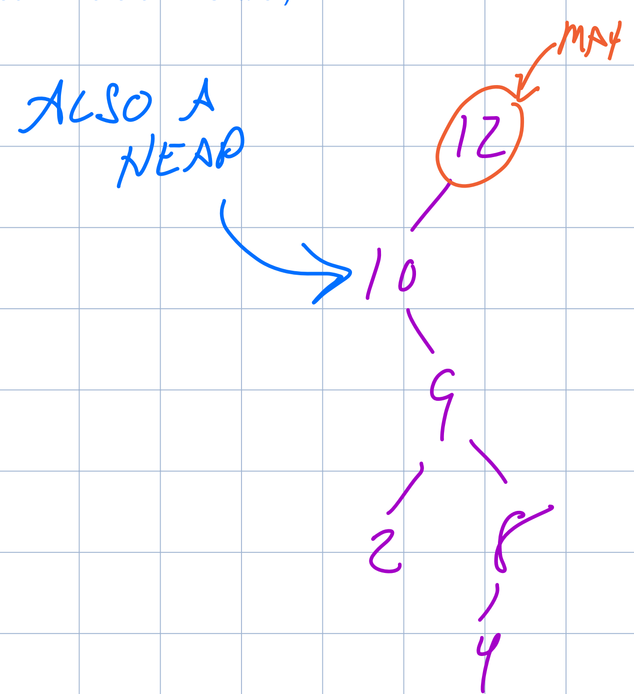
What if we relax the rules a bit?
=> For a priority queue, we don't need a total order like a BST.
What if we just keep the max item at the top??

Heap (binary max heap): a binary tree (*NOT a BST*) with two constraints:
- max item is at the root
- left and right subtrees are also heaps

DATA: 2, 4, 8, 9, 10, 12

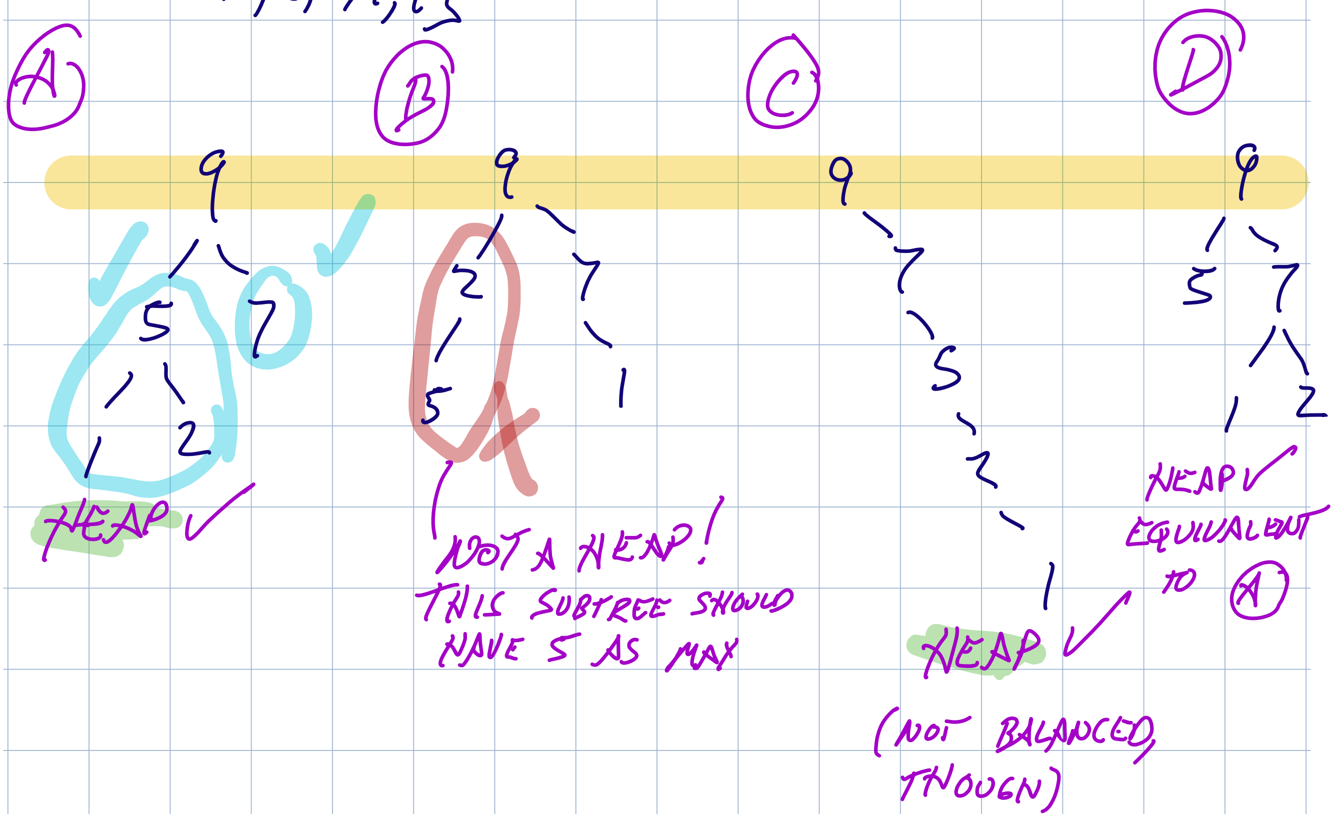


Note: can have different valid representations for the same heap
(may be more or less-balanced… more on this later)



Example: which of these are heaps?

DATA: {1, 2, 5, 7, 9}



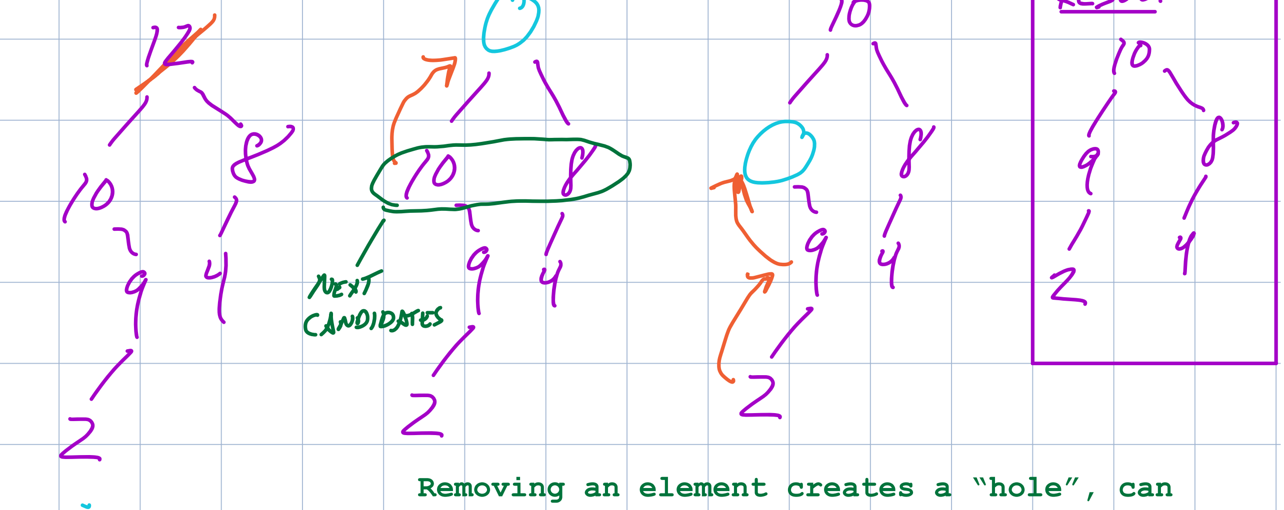
Checking in on our priority queue goals: what can we infer about the runtime of using a heap for a PQ?

```
Priority queue (PQ):  needs 3 operations
 1. Insert a new item           => ???
 2. Remove max-priority item    => ???
 3. Get max-priority item (without removing)
    => O(1) => can just look at top of heap!
```

What about add and remove?

"HOLE" FROM REMOVING 12

EX. REMOVE 12

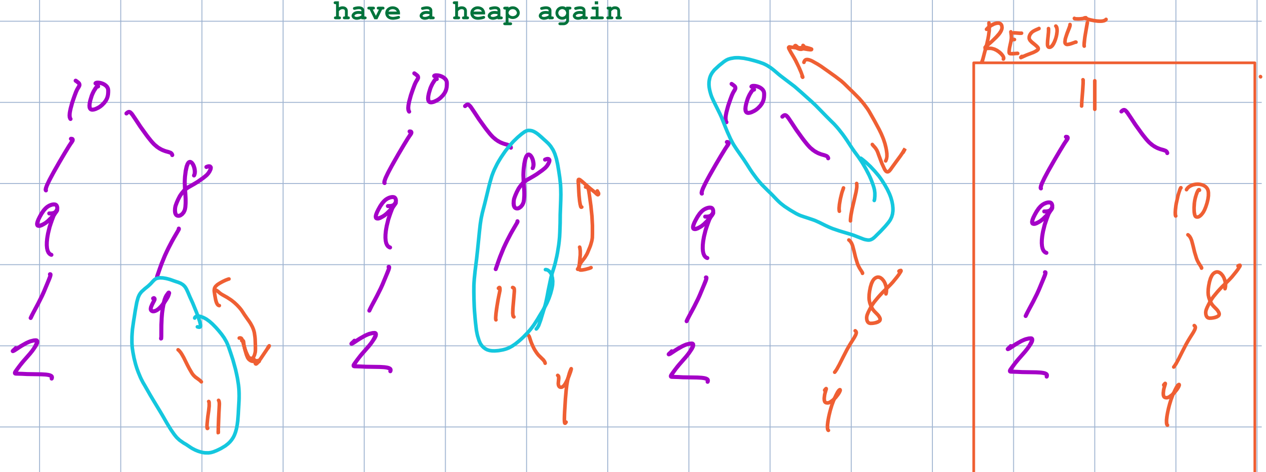


Removing an element creates a "hole", can reorder subtree to fill it

To reorder, we only need to consider one "branch" of the heap => If heap is balanced, this takes O(logN)

WHAT IF WE WANT TO INSERT 11?

Strategy: add to bottom, reorder until we have a heap again



Again, we only need to reorder one "branch" of the heap => O(logN)

So to SUMMARIZE...

```
Priority queue (PQ):  needs 3 operations

if heap is balanced:
 1. Insert a new item          => O(logN)
 2. Remove max-priority item   => O(logN)
 3. Get max-priority item (without removing)
     => O(1)


   If heap is unbalanced, the insert/delete steps are harder:
     - insert:  O(N)
     - remove_max:  O(N)
     - get_max:  O(1)

   Open questions (for next time):
     - How to find an empty spot to insert?
     - How to keep the heap balanced to ensure logN runtime?
     - What does "balanced" even mean, anyway?
```