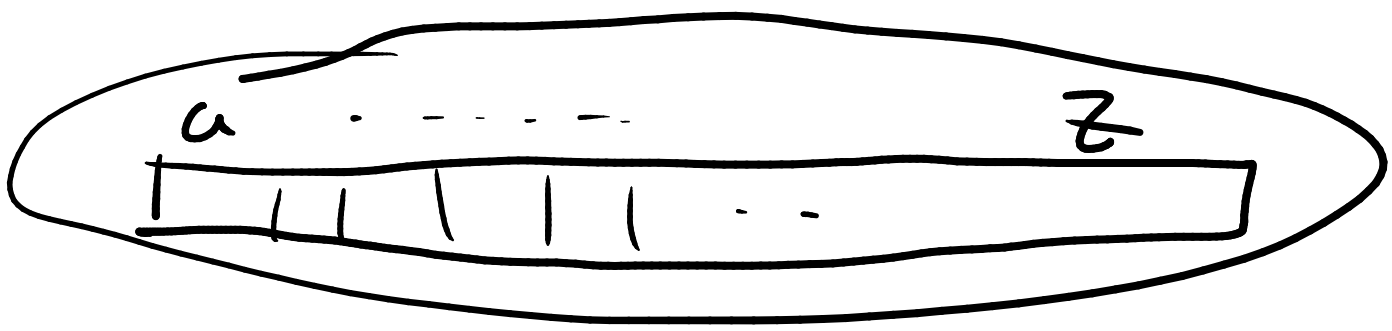
Tries — retrieval

Priority queues and heaps

→ Map/Set ADT
- hash tables
- binary search tree
- tries

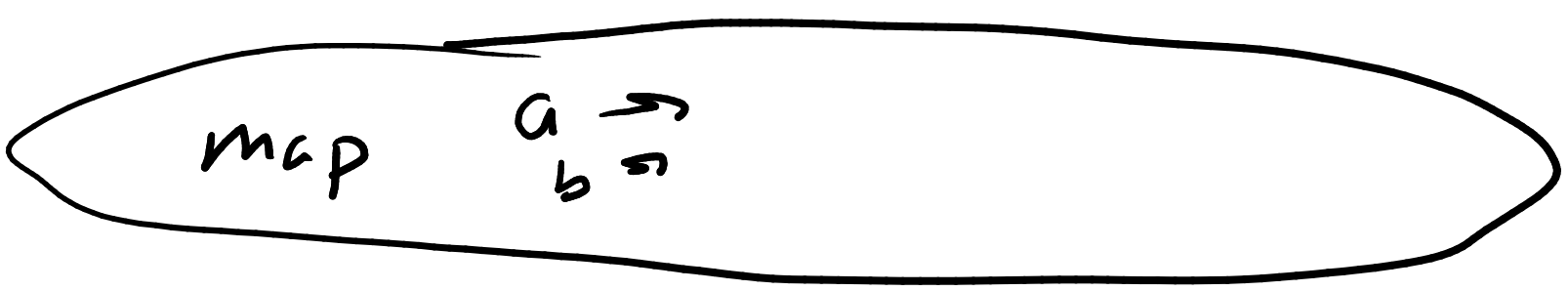
often used for strings
- "bat"
- "bats"

How do you connect each letter to children?



list

OR



class Node {

val children = mutableMapOf<Char, Node>()

}

---

val s = "bat"

root = Node()

root.children[s[0]] = Node

Char

key value

BST: lookup, insert, etc

O(n) really worst case

O(log n) balanced tree

n = # of items in tree

Tries: O(K)

K = length of the key

-really good for prefix searches

"Find all words stating w/ 'ba'"

ADT	Impl Strategies
Stack	arrays, linked lists
Queues	arrays, linked lists
Map/Set	tries, BSTs, hash tables
Priority Queue	heap

Priority Queue ADT

- insert(item)
- delete() ⇒ returns item with highest priority

⇒ typically return
* biggest or smallest

PQ is defined for one or the other

How do you implement a PQ?

use a list:

insert at end O(1)

delete: search for highest priority O(n)

OR

insert into sorted position O(n)

delete: remove from end O(1)

Special kind of tree: heap

O(log n), guaranteed for inserts and deletes

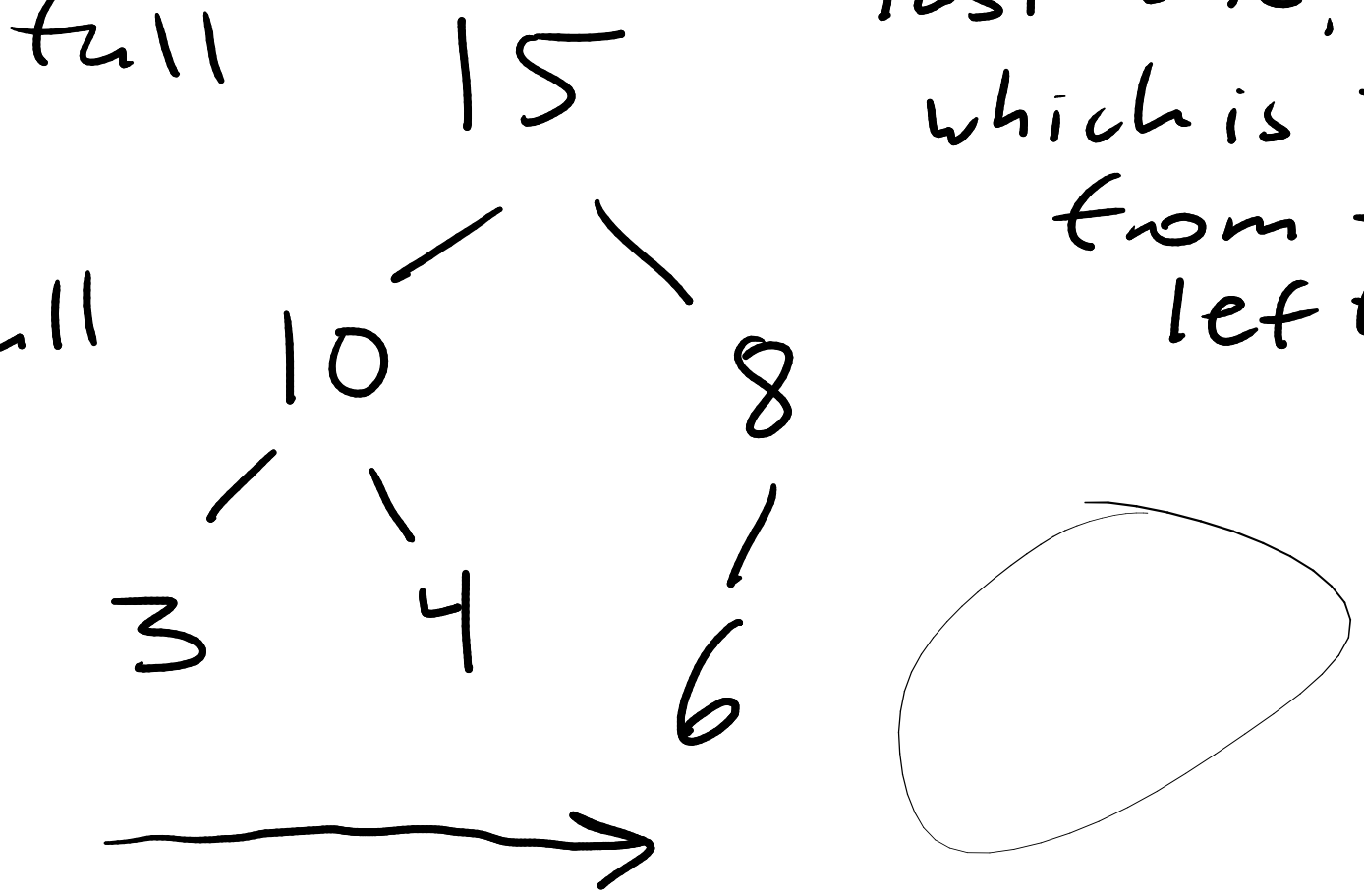
A heap is a binary tree (max) (min)

- where the value at every node is greater than its children (smaller)
- complete - every level is full except possibly last one, which is full from the left

full

full

full from left



How to insert

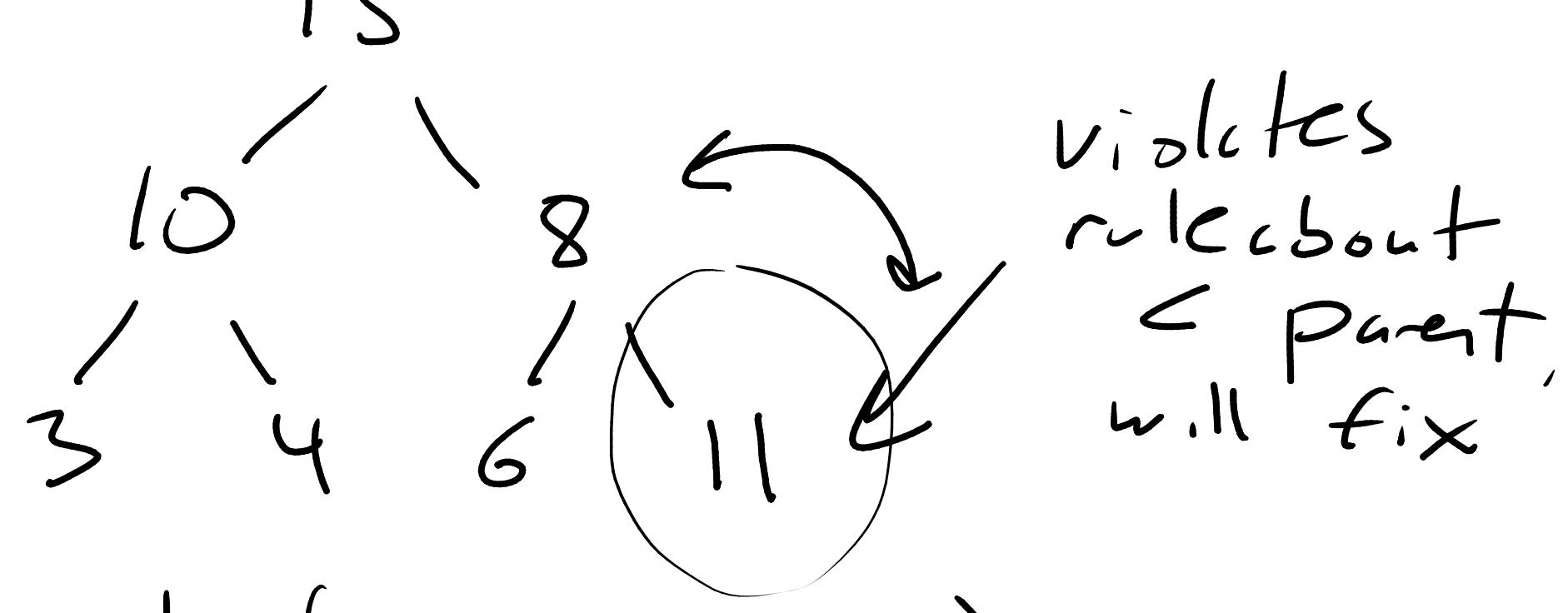
#1 RULE OF HEAPS:

must preserve full levels

rule at all times.

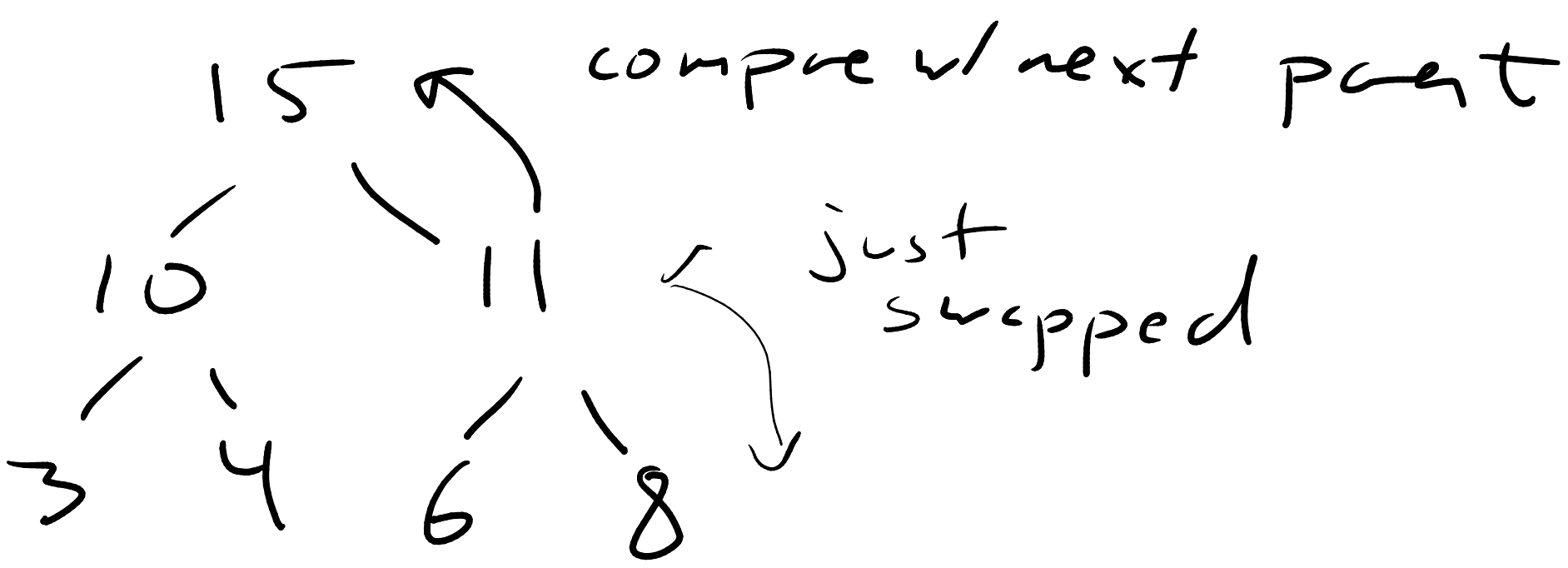
(Insert 11.)

Add new item at next legal location on bottom row



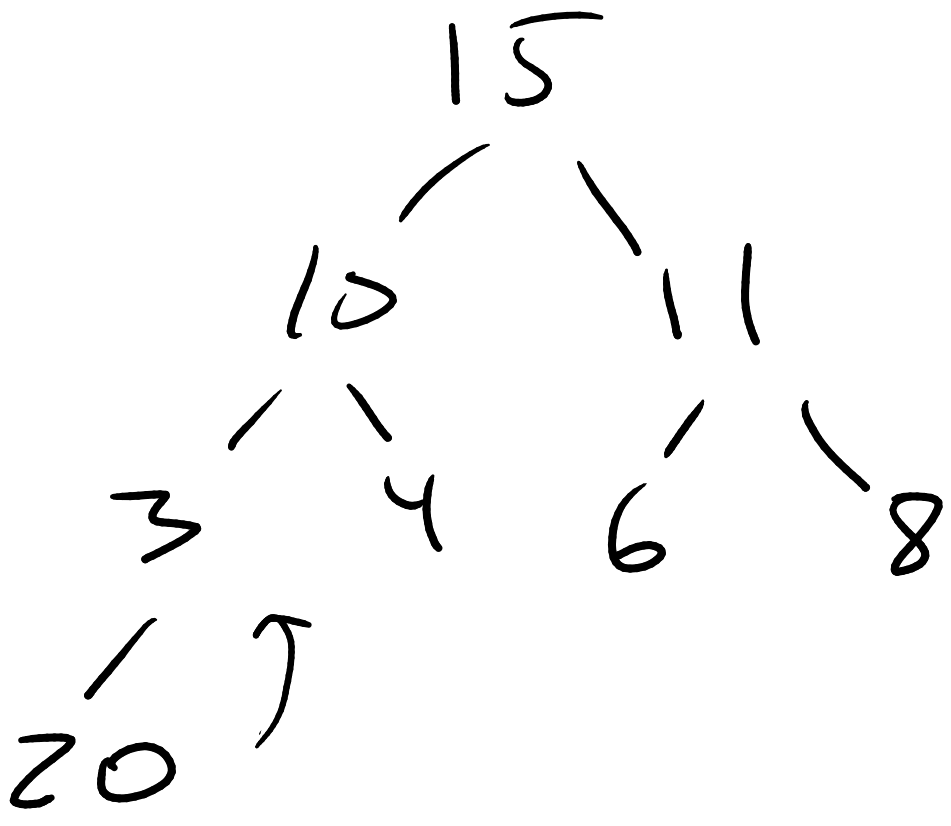
Fix it (percolate up)

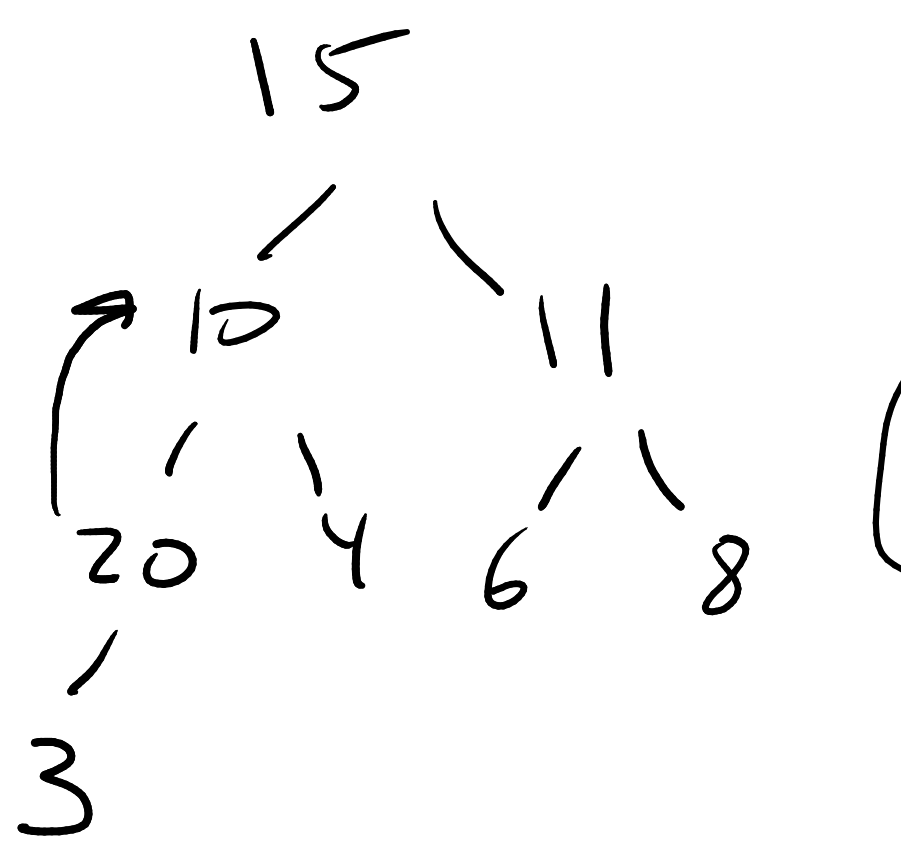
- compare w/ parent. If order is wrong, swap

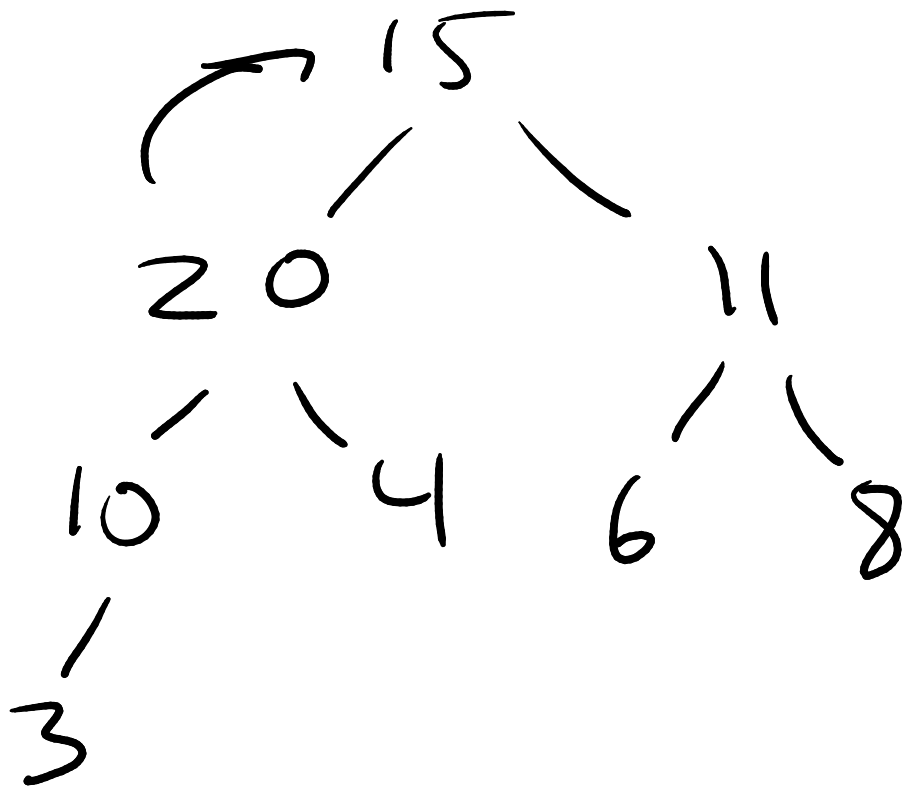


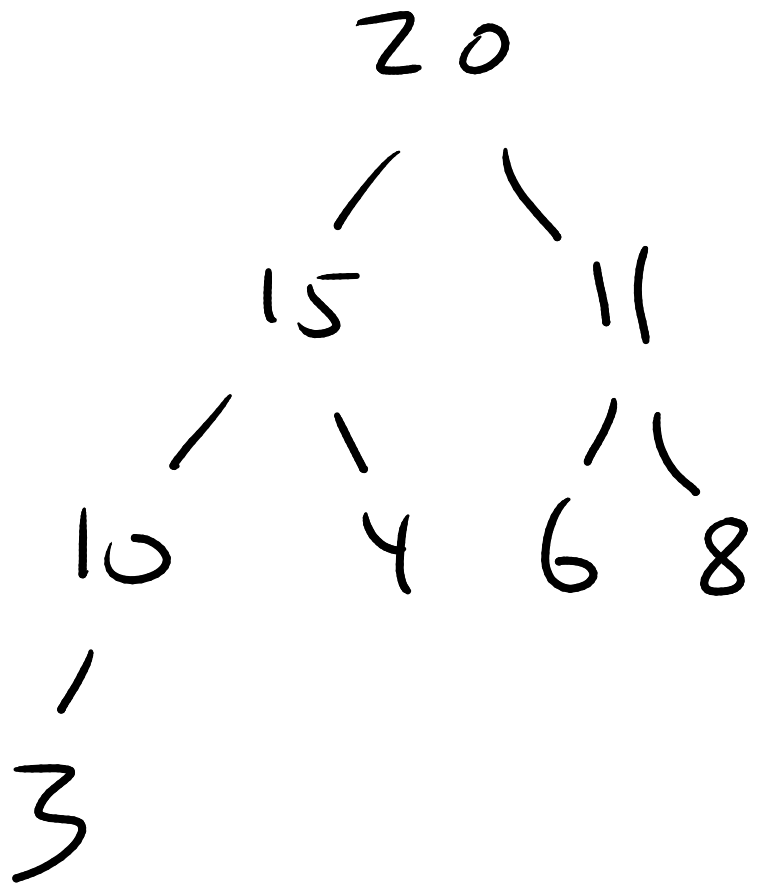
Repeat process until order is correct, or pass the root

(Insert 20)

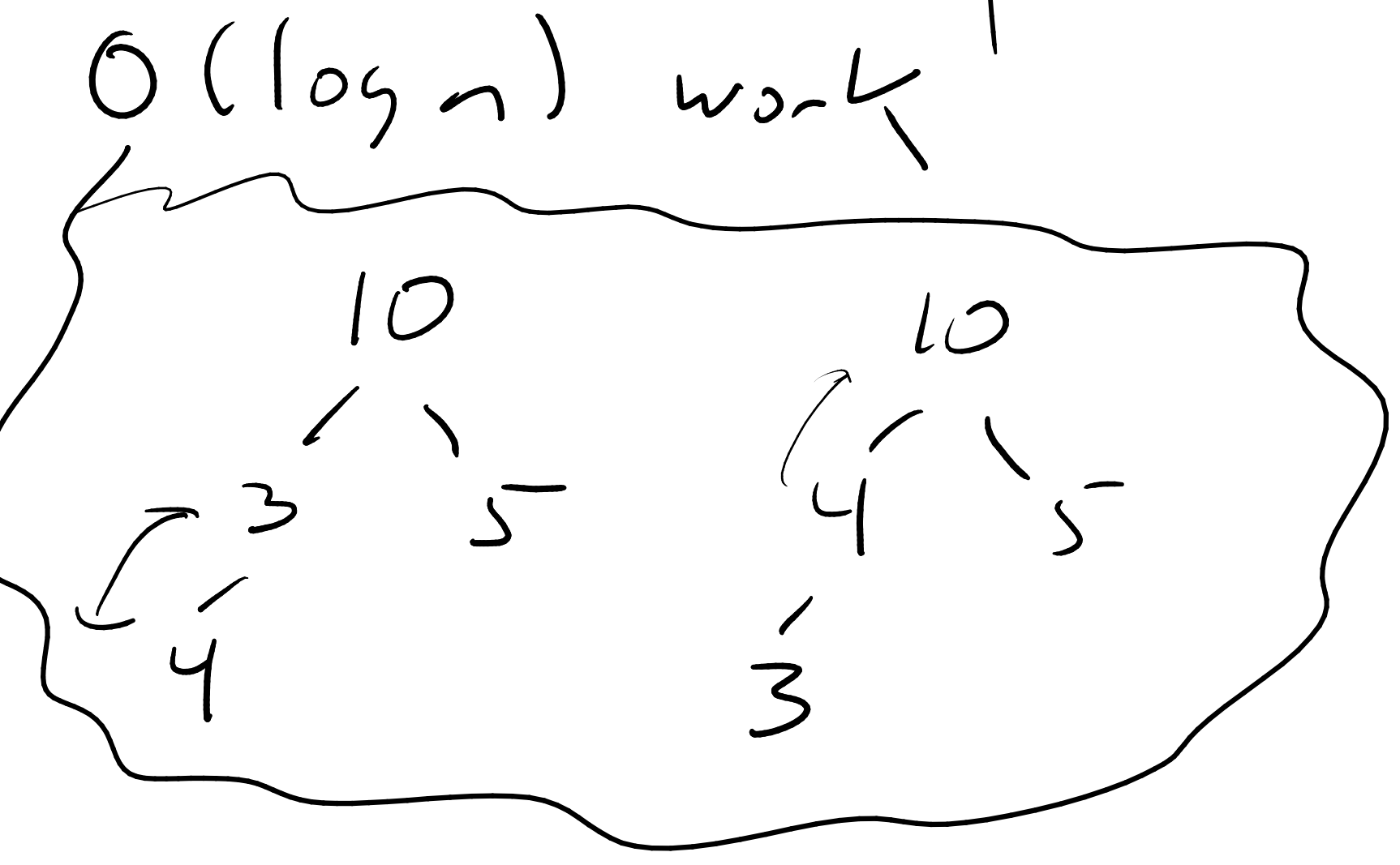








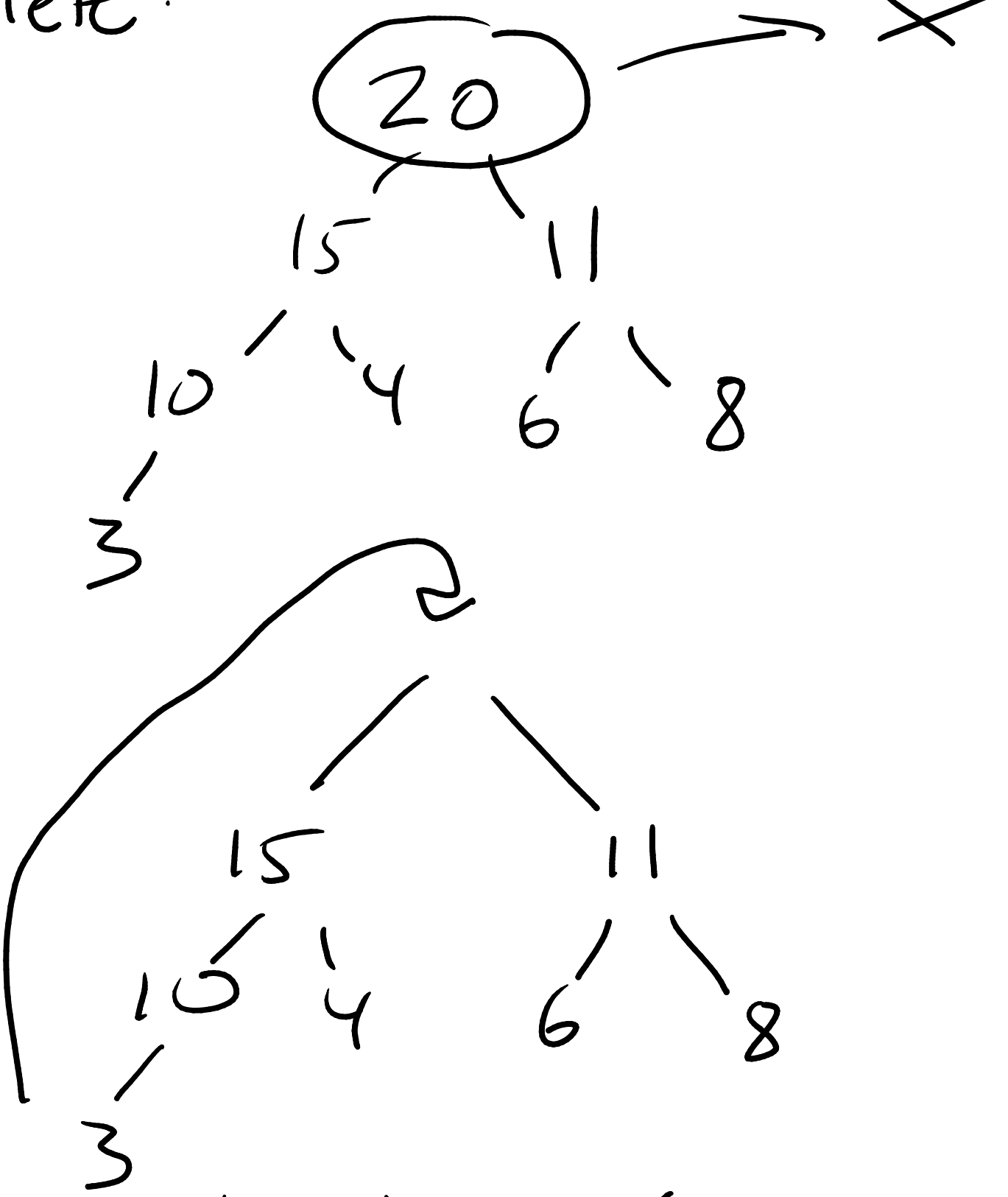
O(log n) work



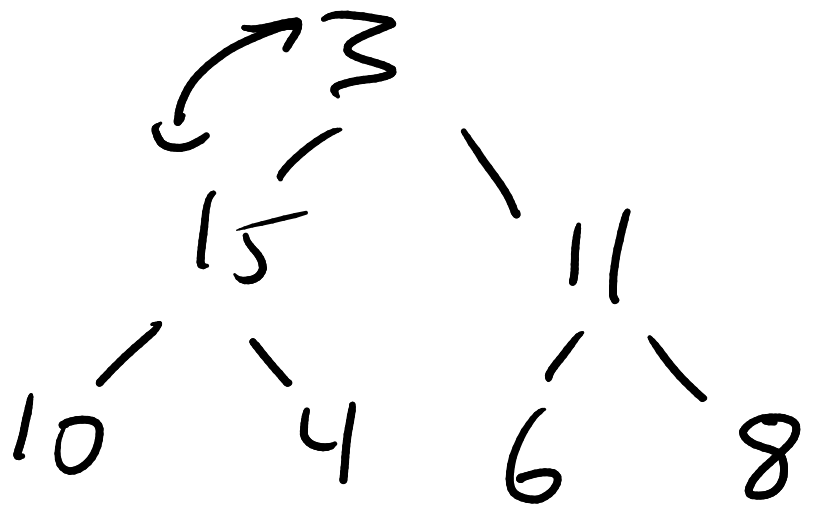
Question example

Peek: look at root O(1)

delete:

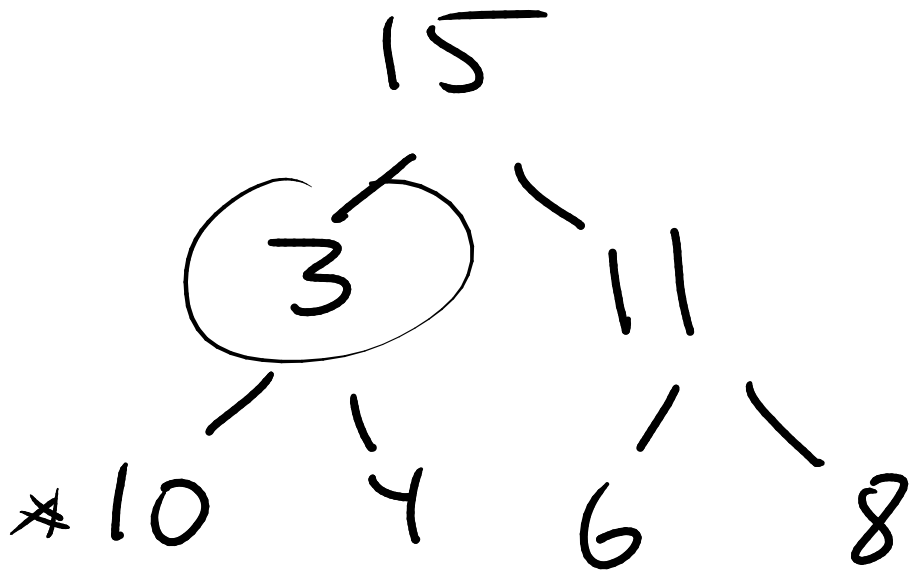


move last item to root

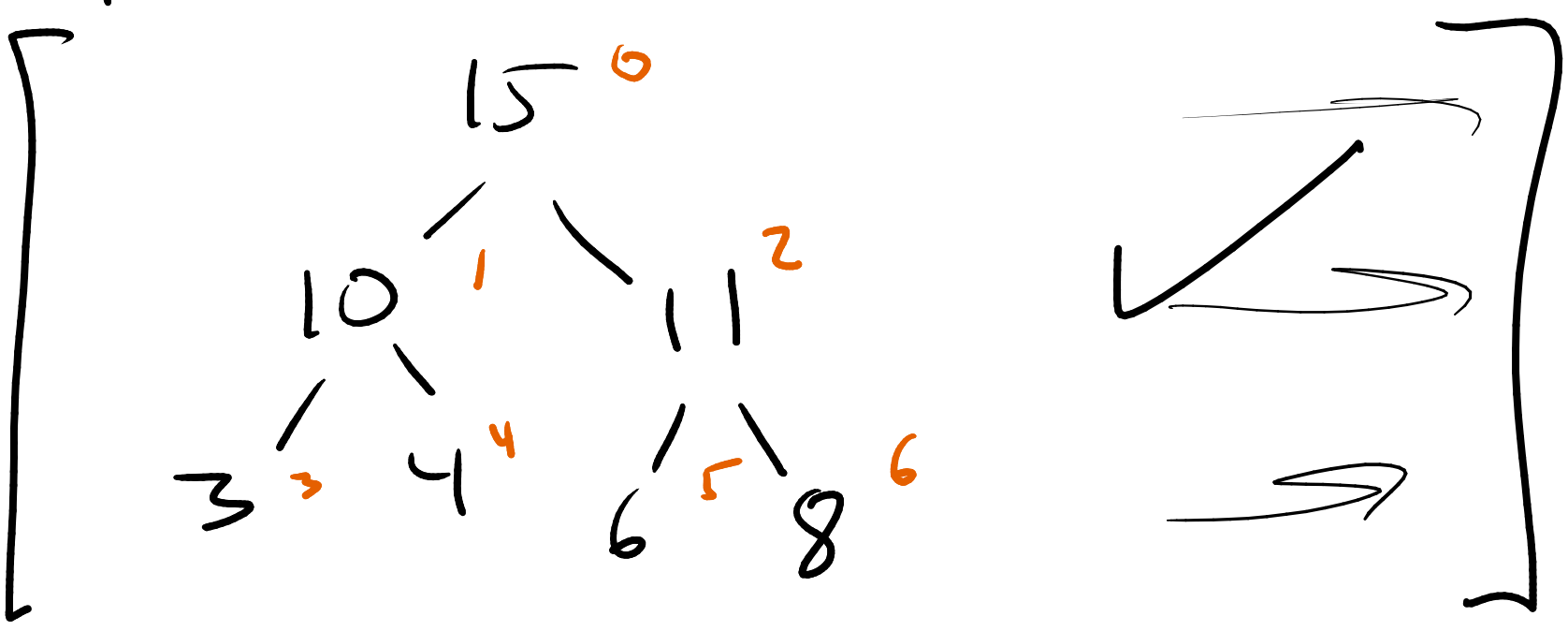


Fix ordering (percolate down)

Stating at root, check childen
if at lest one is greater,
swap it.
(if both are greater, swap up
large one)



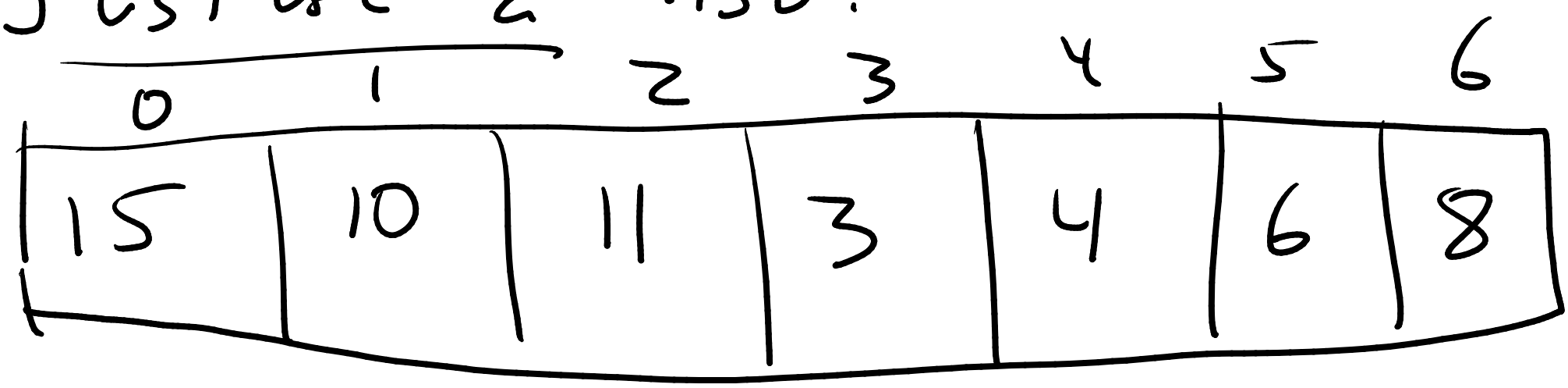
Repect downward



How do you code/store this in memory?

Typically don't use node classes, pointers, etc.

Just use a list.



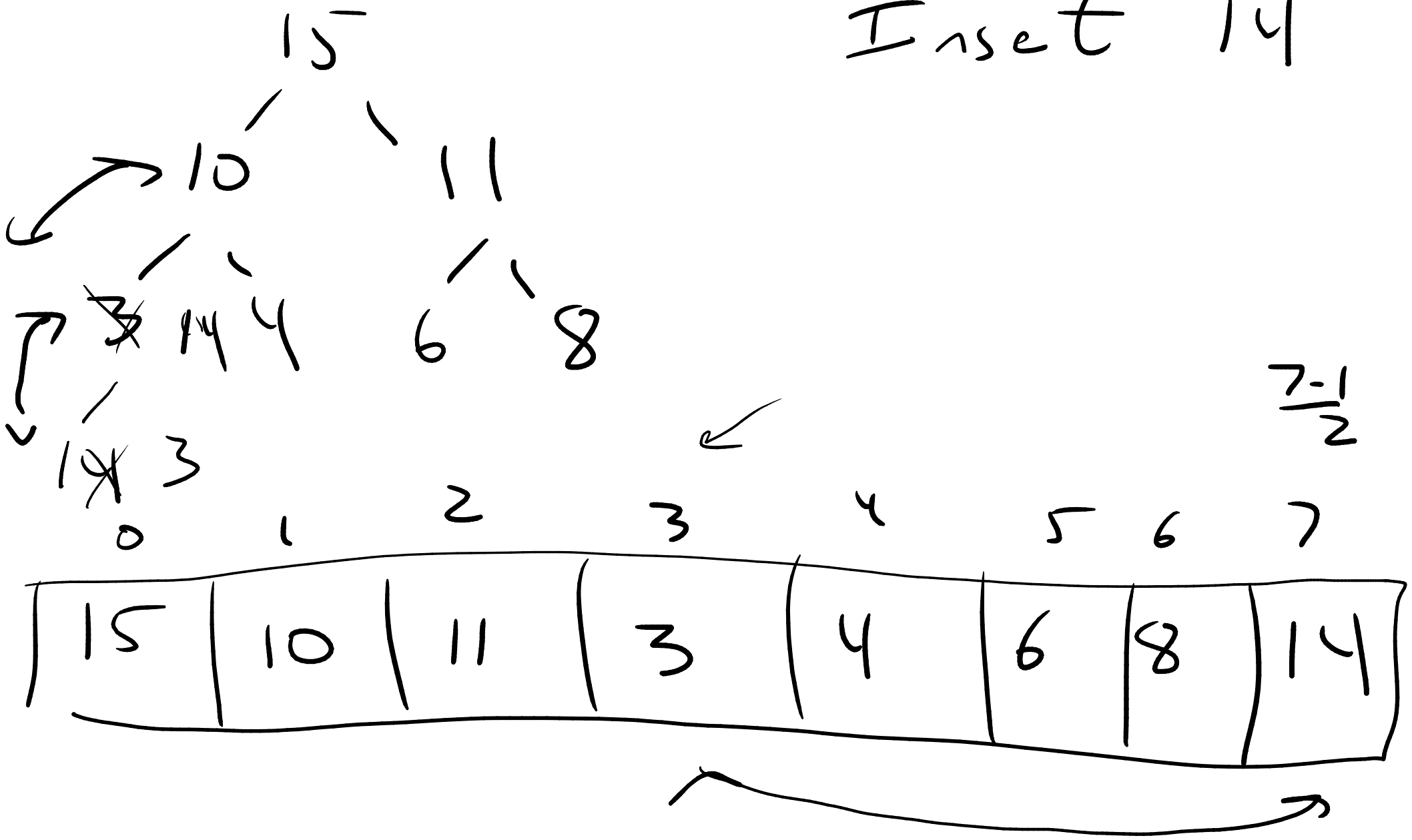
Where is root? location 0.

For any node at location i, its children are at

$2i+1$ and $2i+2$

Likewise, for a node j, its parent is $\frac{j-1}{2}$ (round down)

Inset 14


15
10
11
3
14
4
6
8
14
3
7-1/2
0
1
2
3
4
5
6
7
15
10
11
3
4
6
8
14
15
10
11
14
4
6
8
3
15
14
11
10
4
6
8
3