CS180 - Binary Trees & BSTs

Note Title 11/18/2011

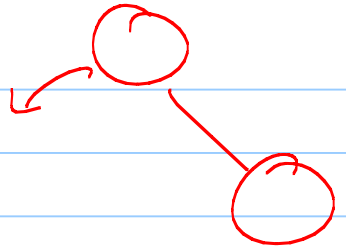
Announcements

- HW4 graded
- Midterm grades submitted
- HW6 due tonight
- HW7 is up - due Monday

Today:

- Finish code for BinaryTree.h
- Start BST.h

Find in a BST – find(x)

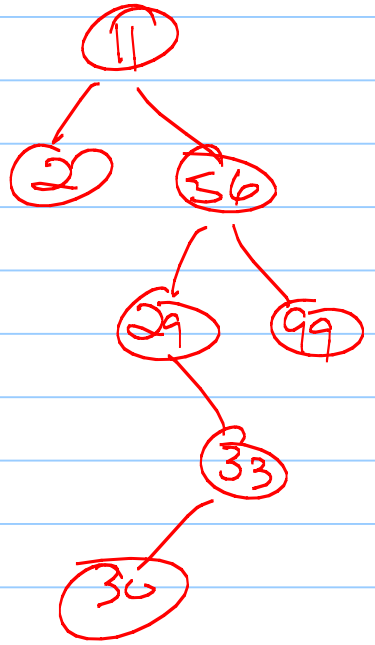


```
Start at root;
repeat
    it = root
    if x == (*it)
        return it
    else
        if x > *root
            it = it.right
        else
            it = it.left
```

Insert in a BST

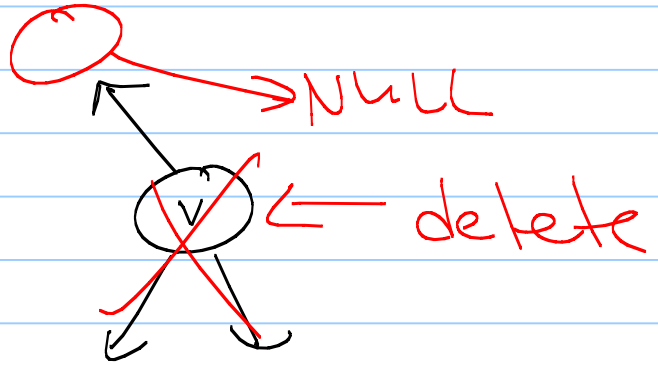
Find

when hit a "leaf"
put new value
as a child



Remove in a BST

Several cases: Let v be our target node to delete

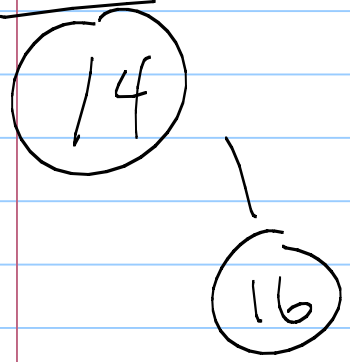


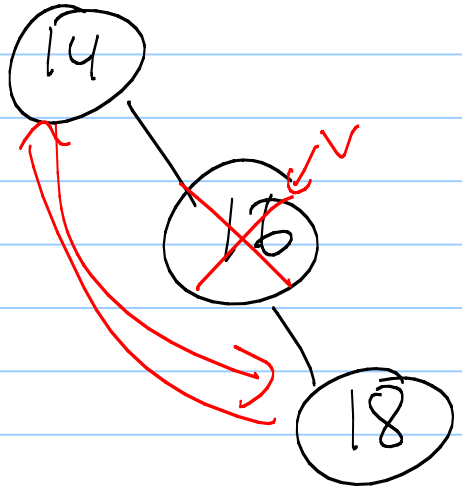
When is it easy?

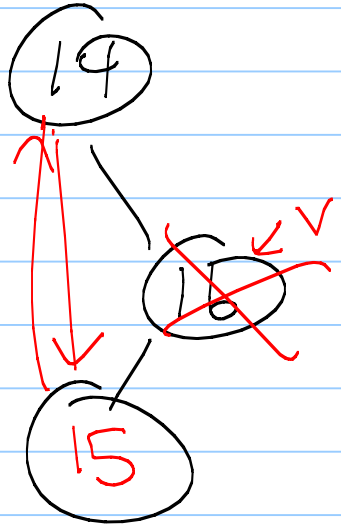
leaf - easy

Case 1: v is a leaf or v has only 1 child.

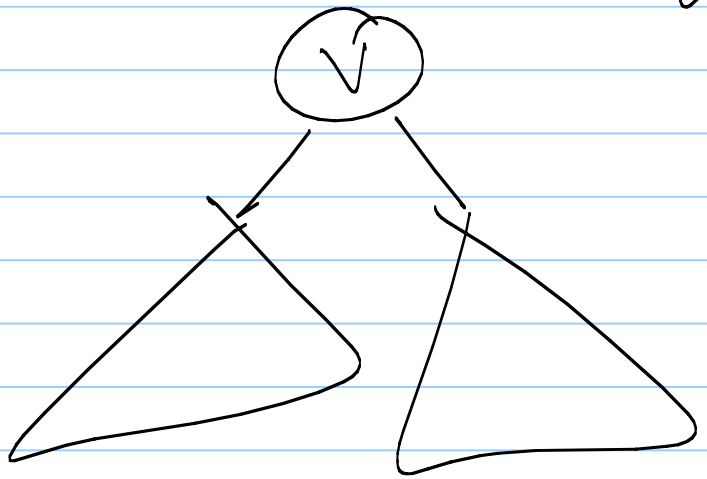
Ex:





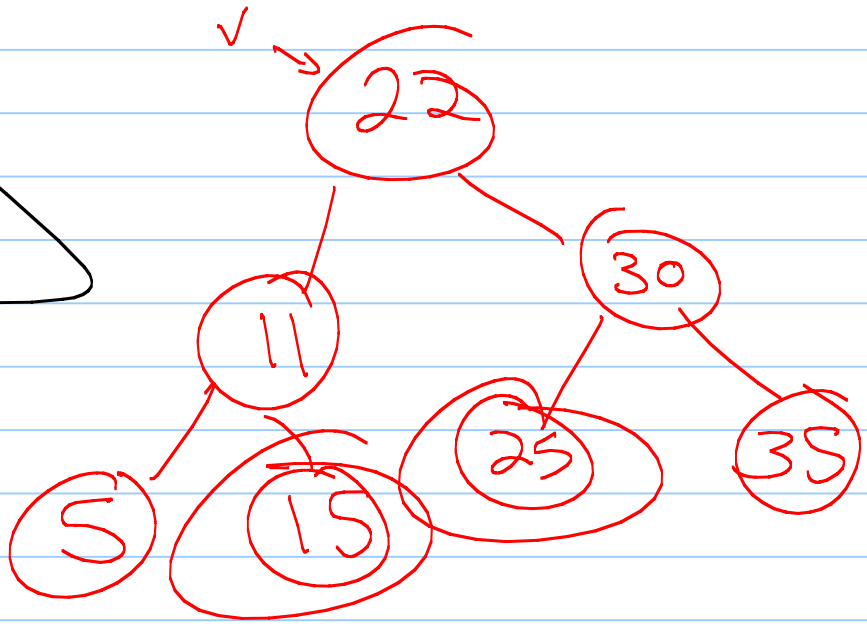


Case 2: v has two children

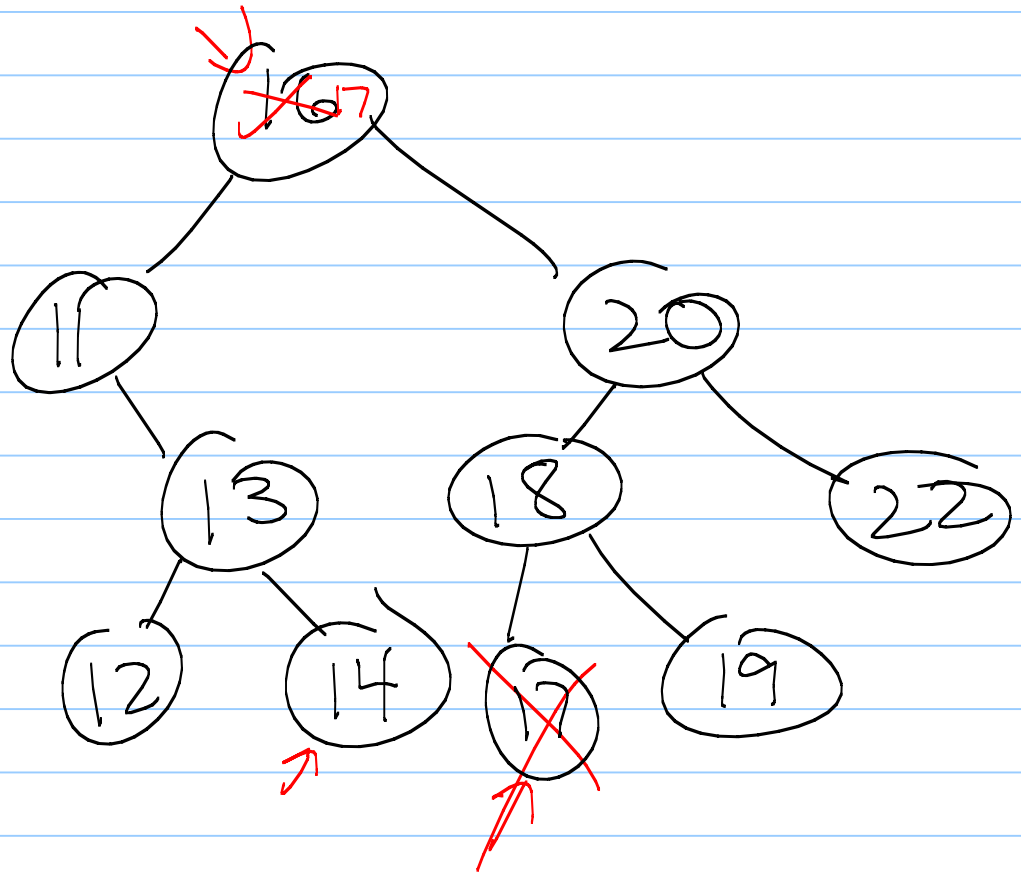


One of the neighbors in an inorder traversal.

What can go in v's spot?



Ex:

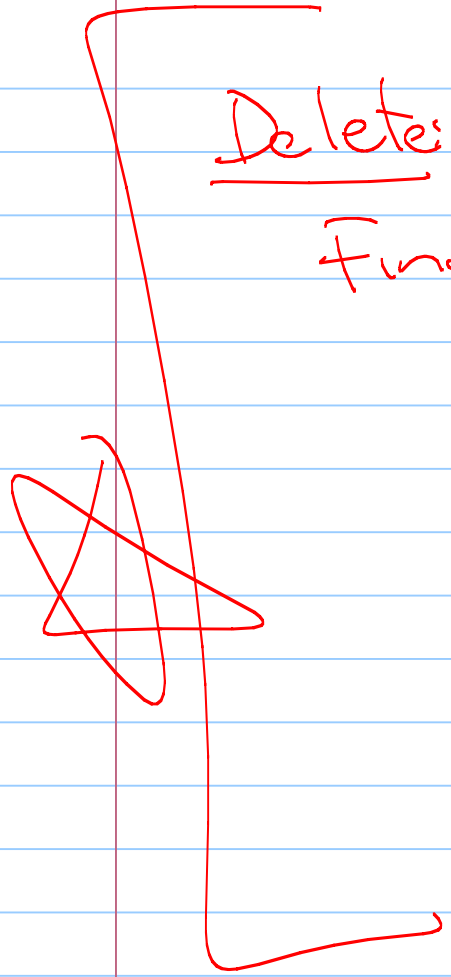


Key: Next node in an inorder traversal has valid value and can have at most one child.

Why? It can't have a left child.

(why?)

If it had a left child, that comes first in an inorder traversal.



Delete:

Find node: (v):

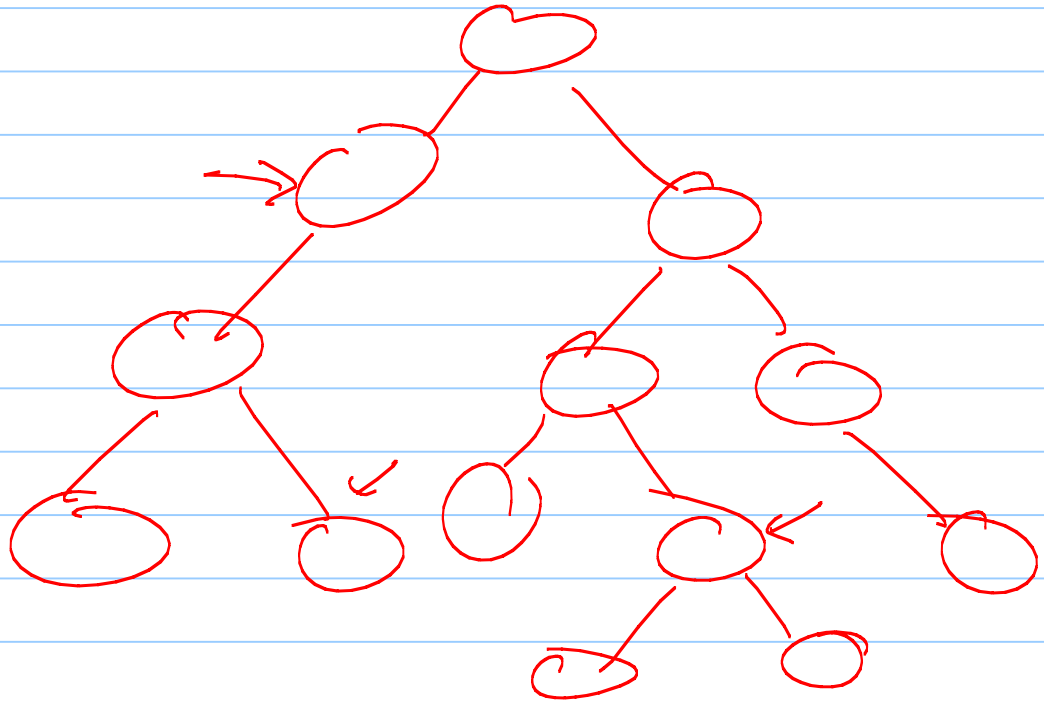
if only 0 or 1 child
deleteAndPromote (x) child

If 2:
find next node w, in inorder traversal
copy it to v
delete w (& promote child)

Operator ++

Inoder traversal:
go left
self
right

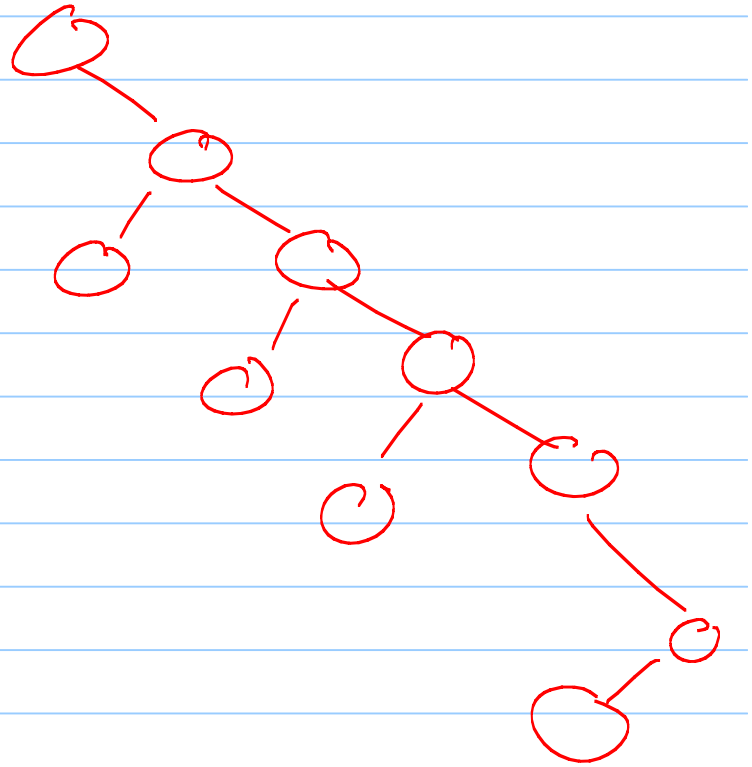
in tree:



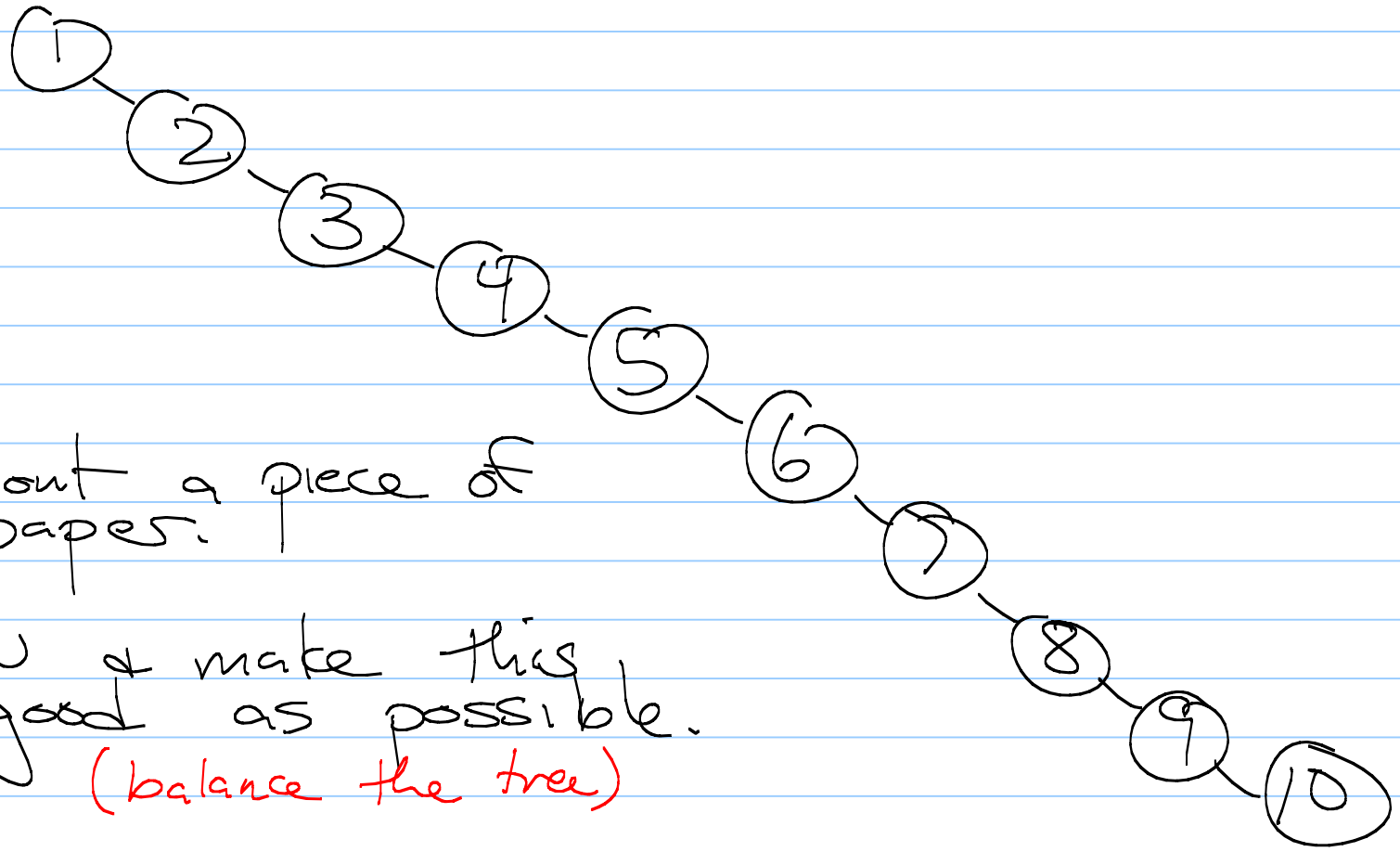
Runtimes

Worst case: $O(n)$

In fact, worse than list or vector implementation → lost $O(1)$.



Consider this tree:

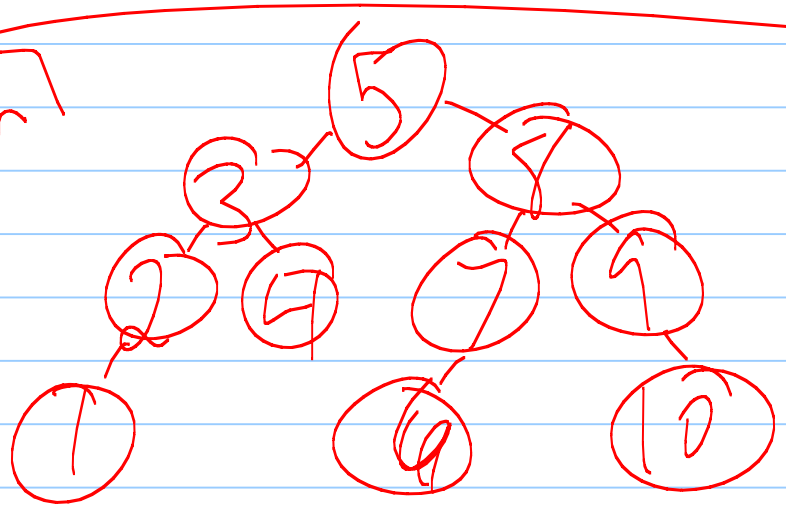


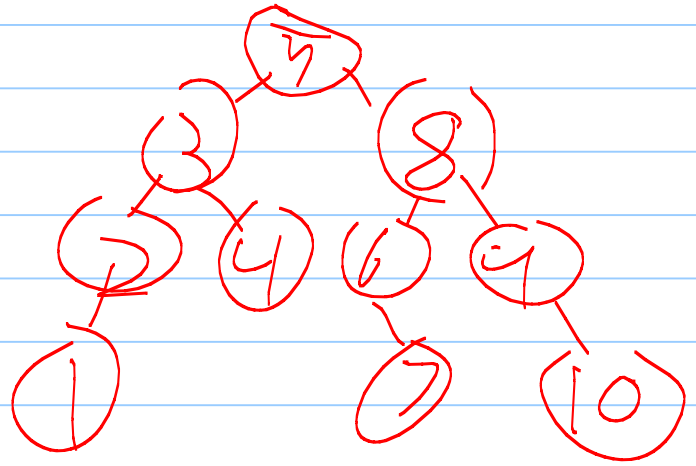
Take out a piece of paper.

Redraw & make this as good as possible.
(balance the tree)

Possible answers

$\lceil \log_2 n \rceil$





Balanced tree: many possibilities

Goal: $\log_2 n$

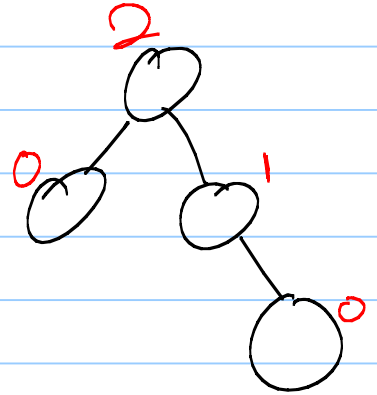
Balanced binary tree

- AVL trees
- Red-Black trees
- Splay tree
  - keep $O(\log n)$ time

AVL Trees

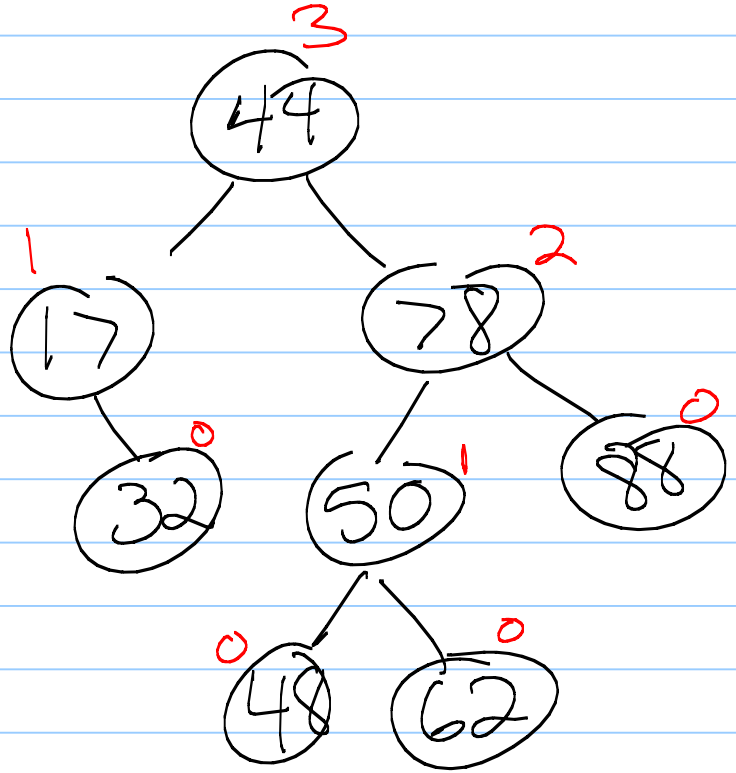
Height - Balance Property:
For every node of T, the heights of the children differ by at most 1.

$\Rightarrow$ max height $\leq 2 \cdot \lceil \log_2 n \rceil$



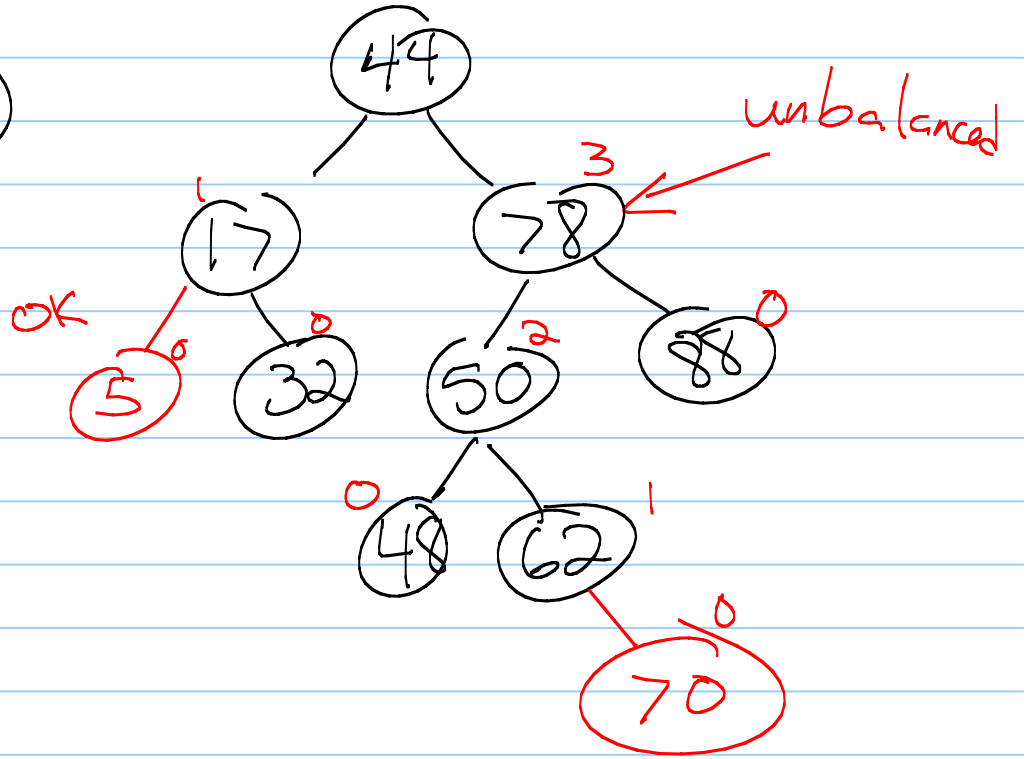
(How do we calculate height again?)

Ex:



Now: How can we mess this up?

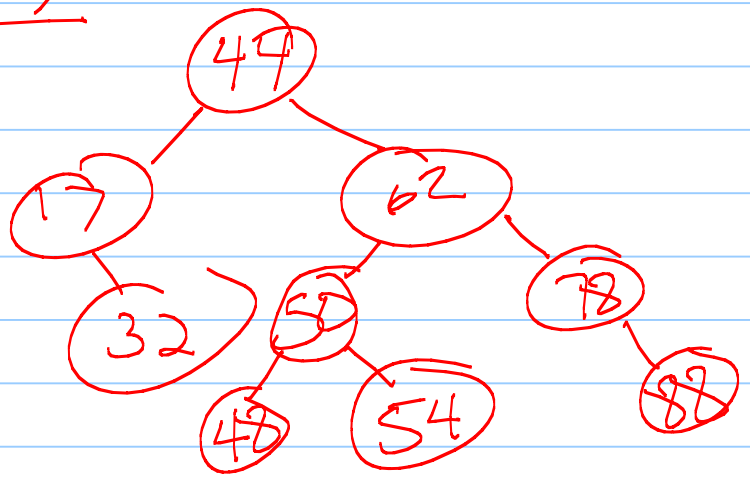
(In other words, how can the height change?)

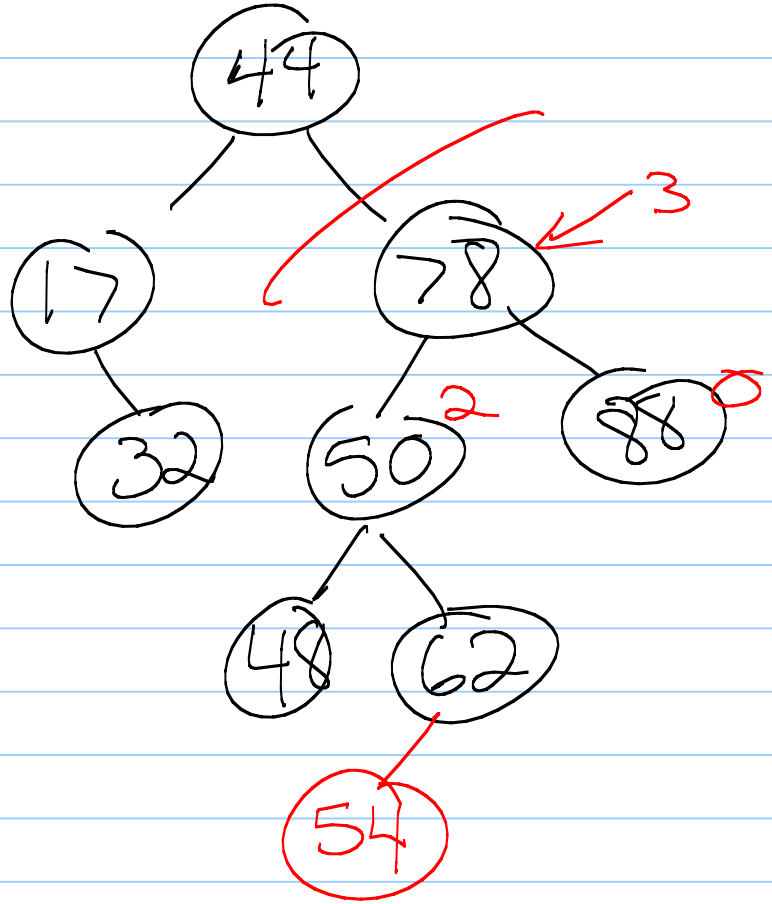


Insert:

insert(54)

Fix:



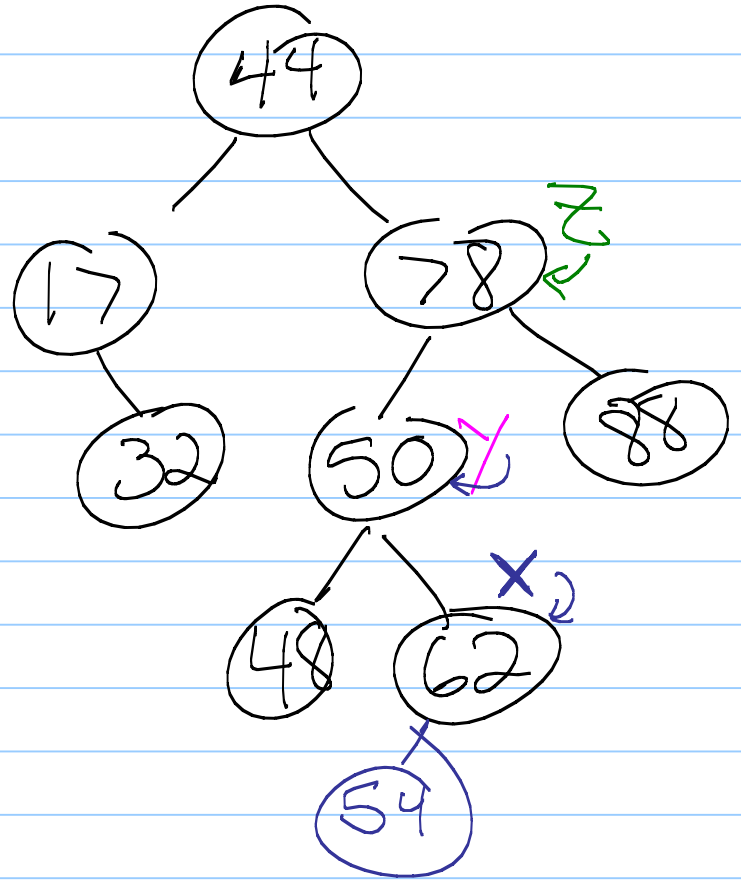


So: consider the lowest node which does not satisfy height-balance property -- call this z.

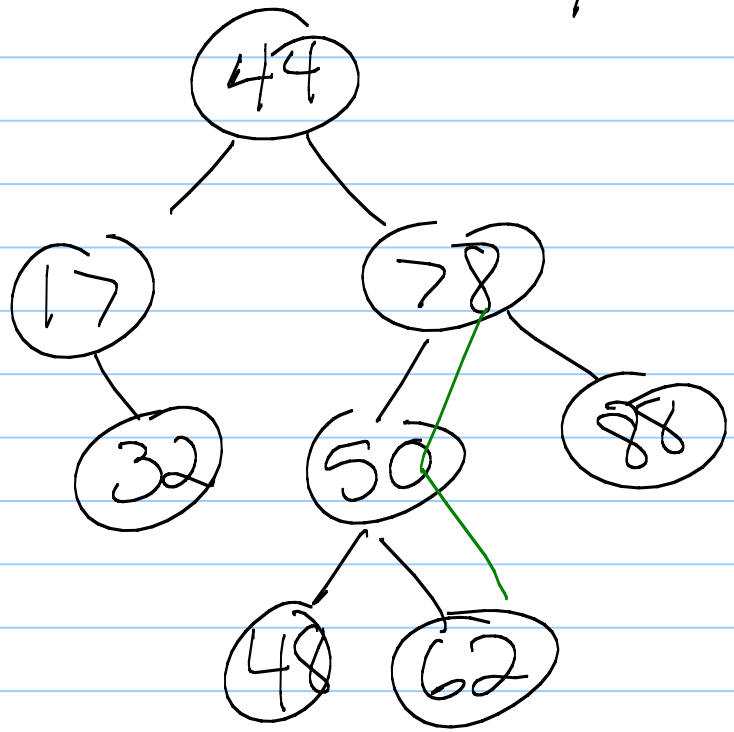
Let y be z's child with larger height.

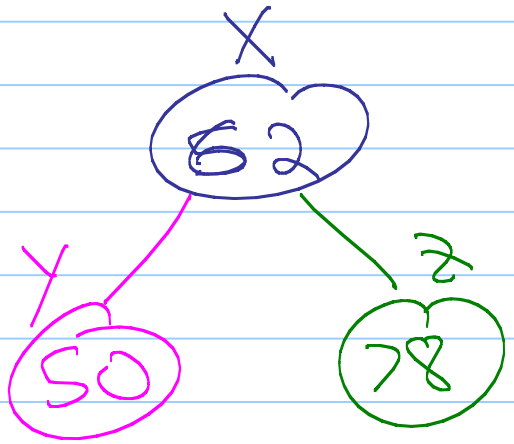
Let x be y's child with larger height.

Now - fix it!



What did you do?





tomorrow:

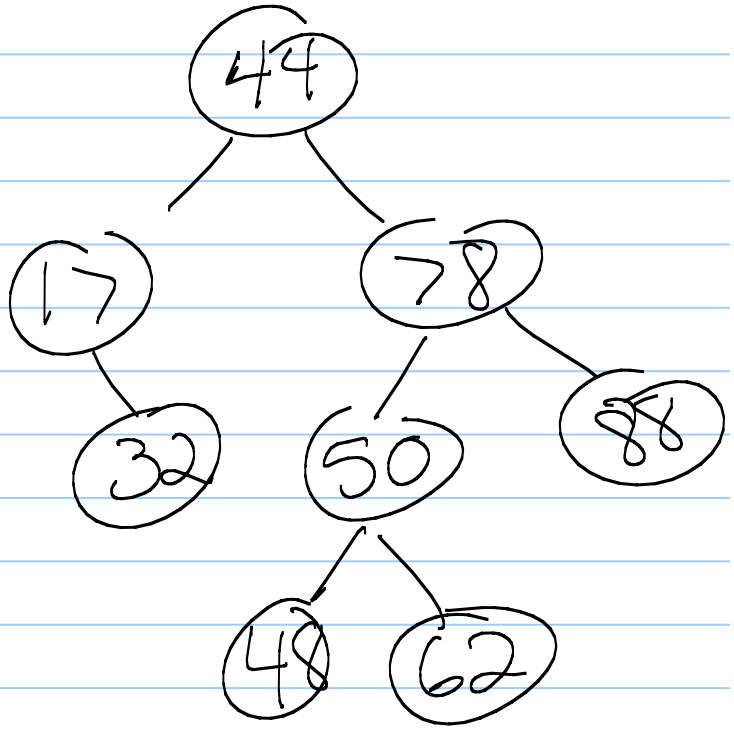
Another - insert (49)

So: consider the lowest node which does not satisfy height-balance property V - call this

Let be z's child with larger height.

Let be y's child with larger height.

Now - fix it!



What did you do?