CS180 - AVL trees

Note Title 10/31/2012

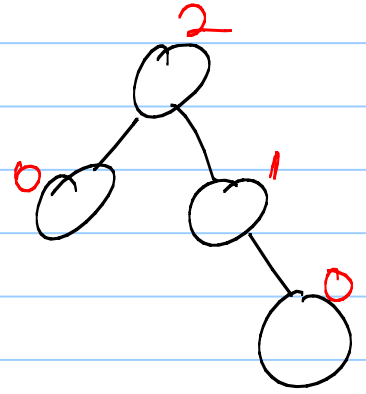
Announcements

- HW - due tomorrow
- Next HW is up, due in 1 week
  may work with partner
- Lab tomorrow
- Test in 2 weeks (on Monday)

AVL Trees : BST with:

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

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

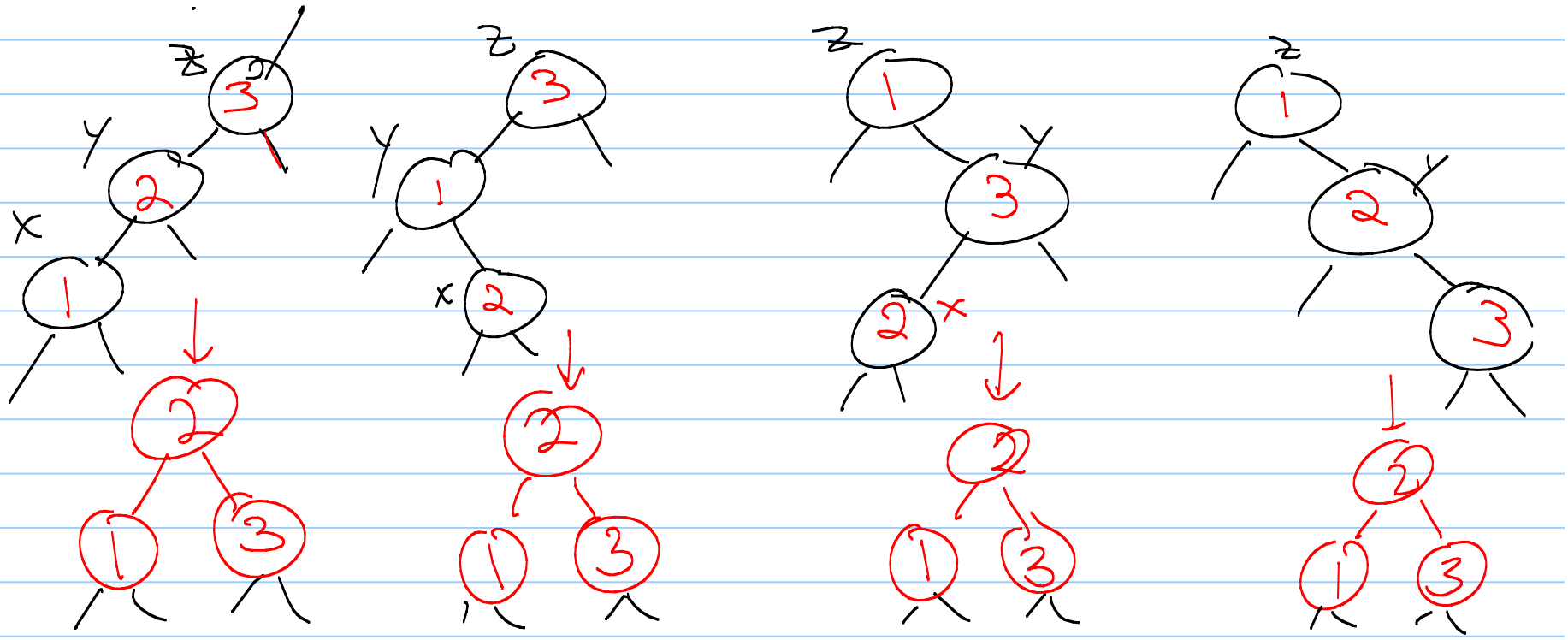


(How do we calculate height again?)

$h(v) = \max(h(\text{left}(v)), h(\text{right}(v)) + 1$

Insert: Do BST insert.

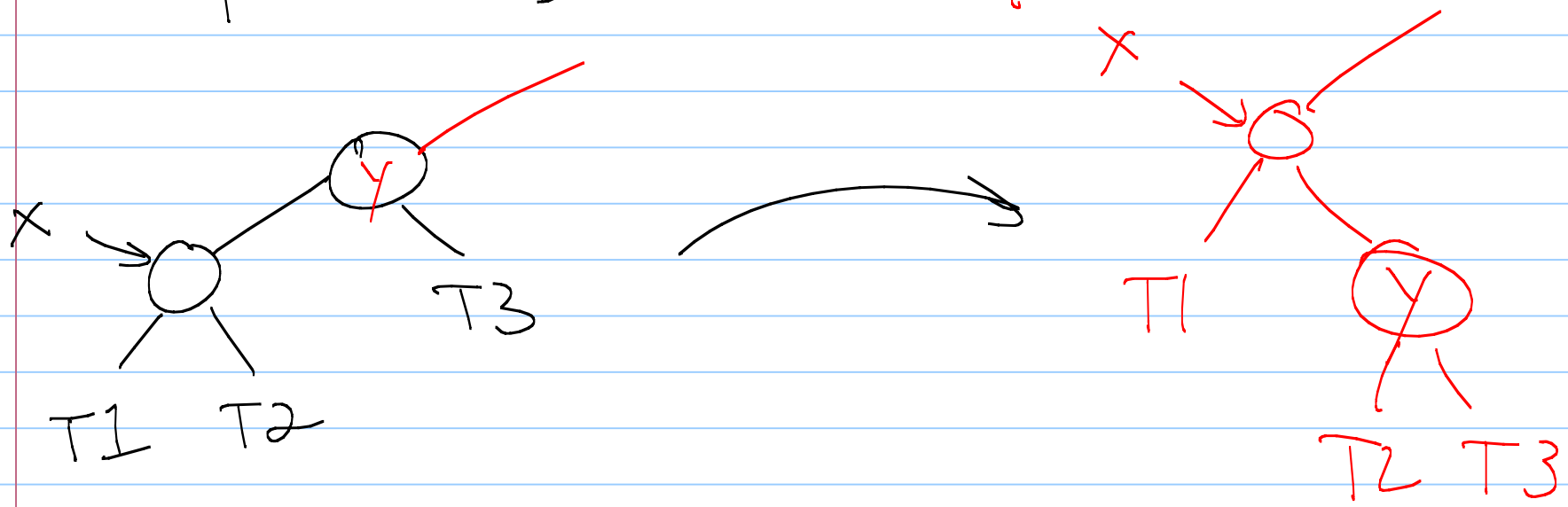
Then find lowest unbalanced node z:

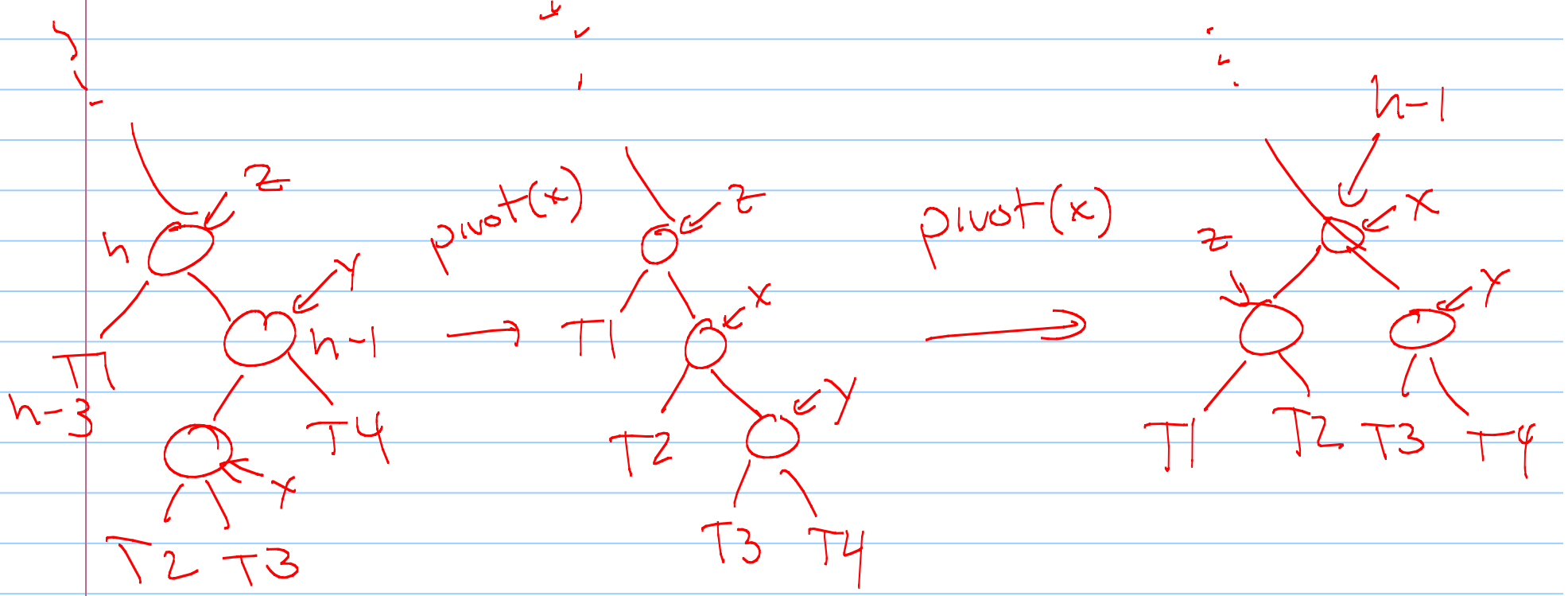


Key operation:

- pivot (x)

runtime? $O(1)$




z
h
y
h-1
T1
h-3
T4
x
T2 T3
pivot(x)
z
T1
x
y
T2
T3 T4
pivot(x)
h-1
x
z
y
T1 T2 T3 T4

Removing in AVL trees

Step 1: Remove - just like in BST

Step 2: Re-balance (if removal violated H-B property.)
- note: start just above node actually deleted

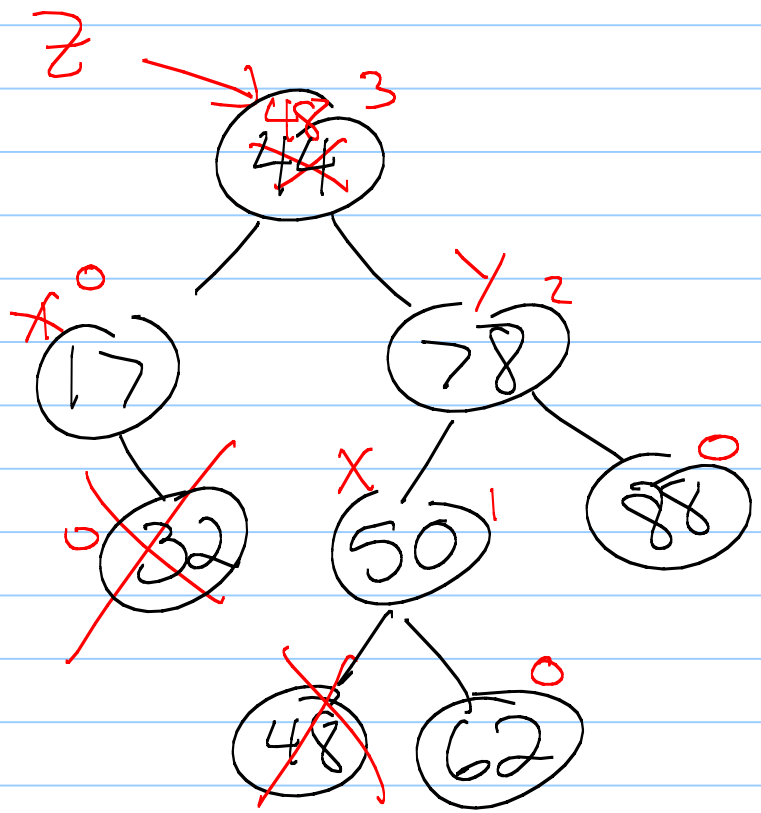
Note: Unlike insert, remove could actually unbalance all the way to the root.

Example:

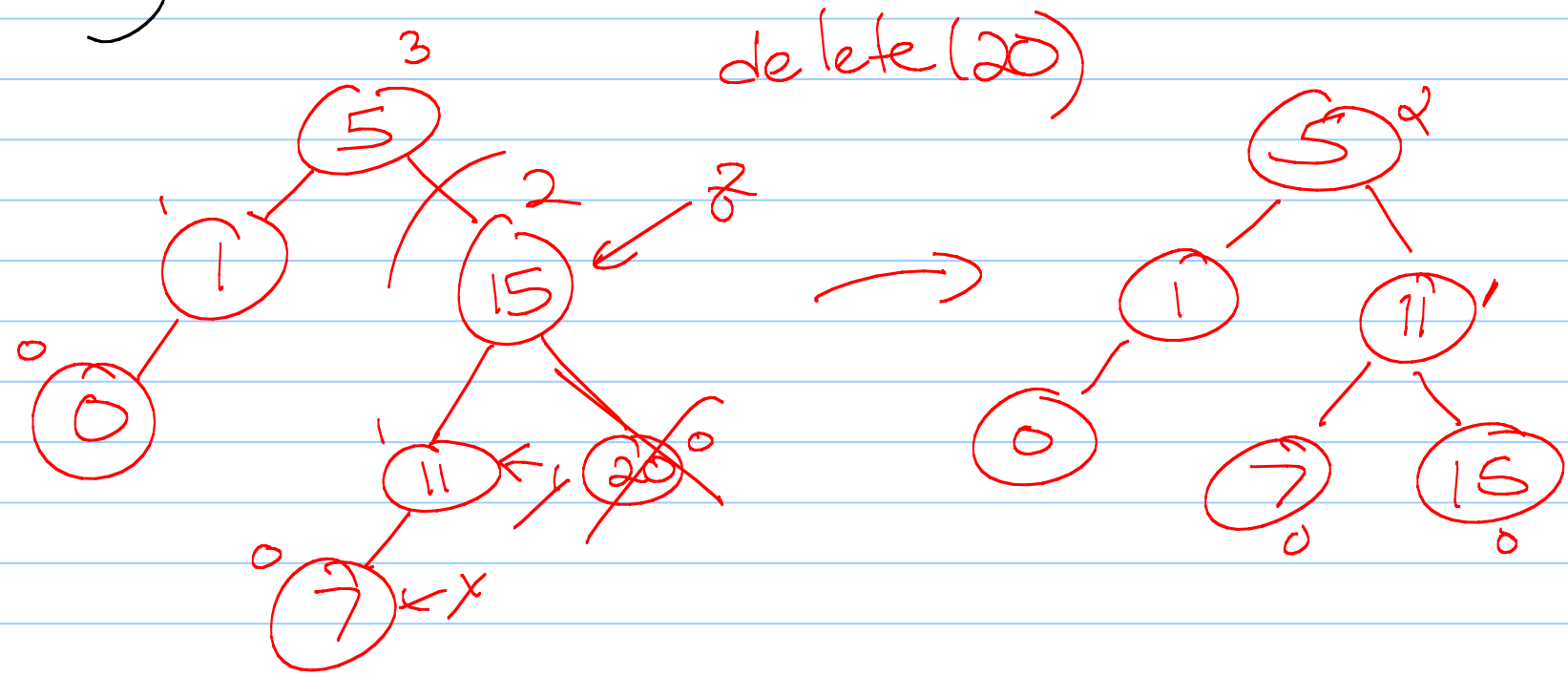
remove(44)

remove(32)

↳ pivot x twice



Fixing the tree



Algorithm to remove

- Remove as in BST
- Track lower node that was removed.
  - Travel up tree, searching for unbalanced nodes (& fixing) until you reach the root.

Performance

For insert & delete, follow root to leaf path at most 3 times:
- find
- next in inorder (for remove)
- travel back up tree balancing

At each node: $O(1)$ time: ≤2 pivots & bunch of checks/updates

How large is root to leaf path? $\hookrightarrow 2 \cdot \log_2 n$

Total time: $O(\log n)$