CS 180 - AVL trees

Note Title 11/13/2009

Announcements

- I have no voice today.
- SO - extra credit today!
  Take out paper & pencil/pen.
  Put your name on paper.
- Practice tests are up here - come & get one now!
- Review session Monday, exam on Wednesday.
- HW due Monday.

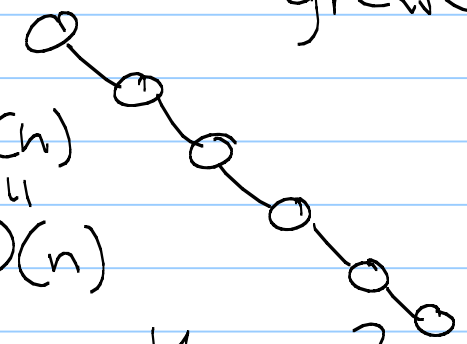
Recap:

Binary Search trees

What are they? a tree that is "sorted", so
- inorder traversal results in sorted list

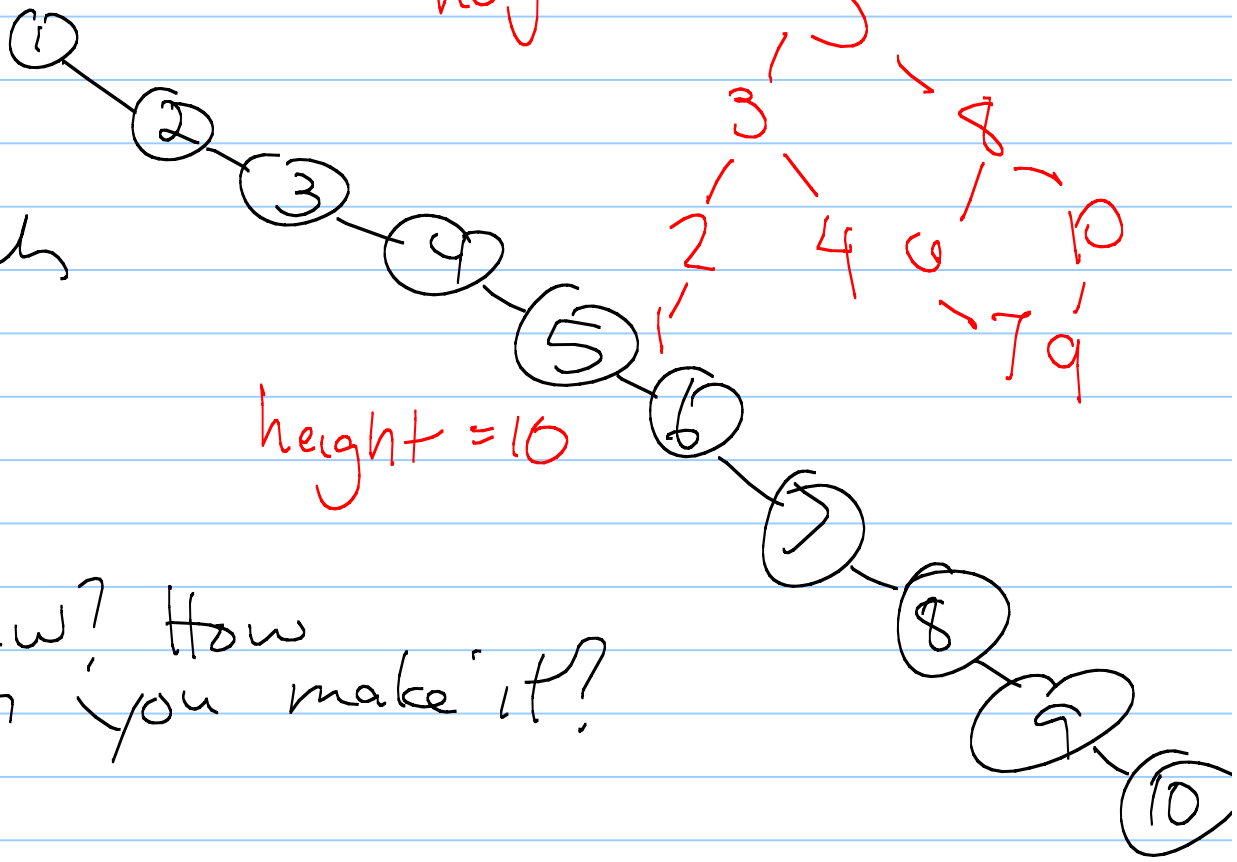
invariant: - For any node, left child is smaller + right is greater.

How fast is: search?
insert?
delete?
] O(h) = O(n)



How could we improve these?
improve height = balance tree

Consider this tree:



- Make a search tree that is balanced.

ok-volunteer?

height = 4

height = 10

Can you redraw? How good can you make it?

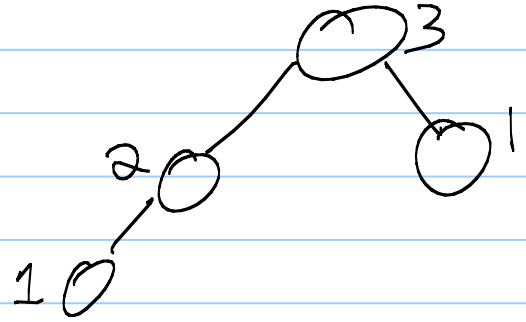
good = "small height"

AVL trees:

- Height-Balance property:
  For every (internal) node of T, the heights of the children differ by at most 1.

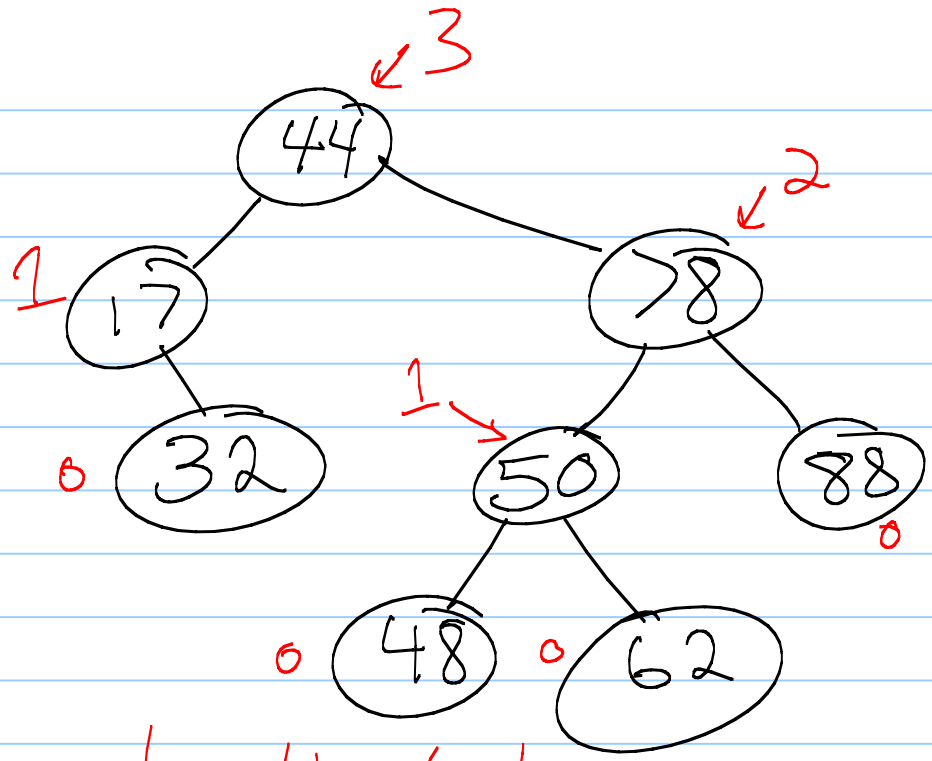
  ↳ height of tree $\leq 2 \cdot \log_2 n$

(Question: how do we calculate height?)



height = max{height of child} + 1

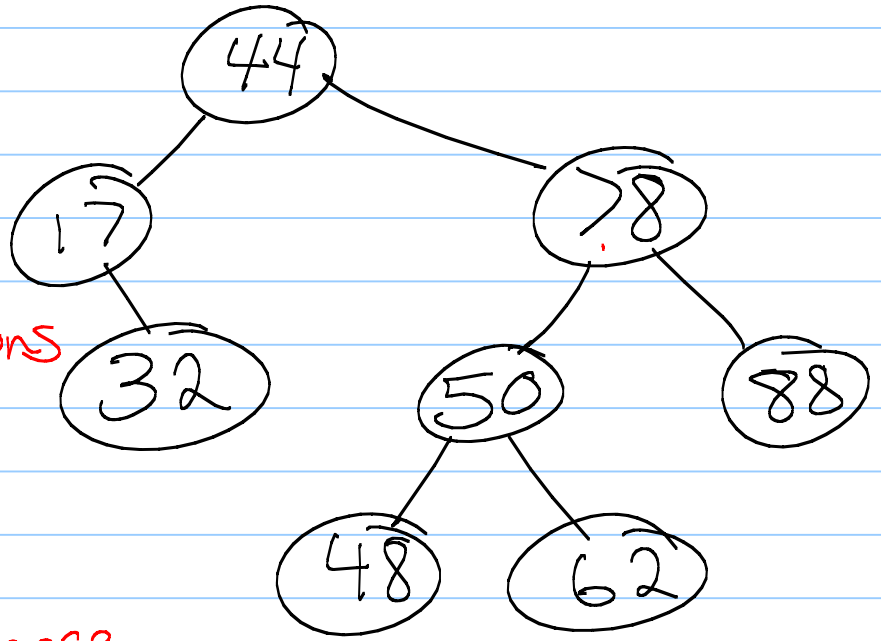
Ex:



satisfies height-balance property

Now how can we mess this up?

(In other words how can the height change?)



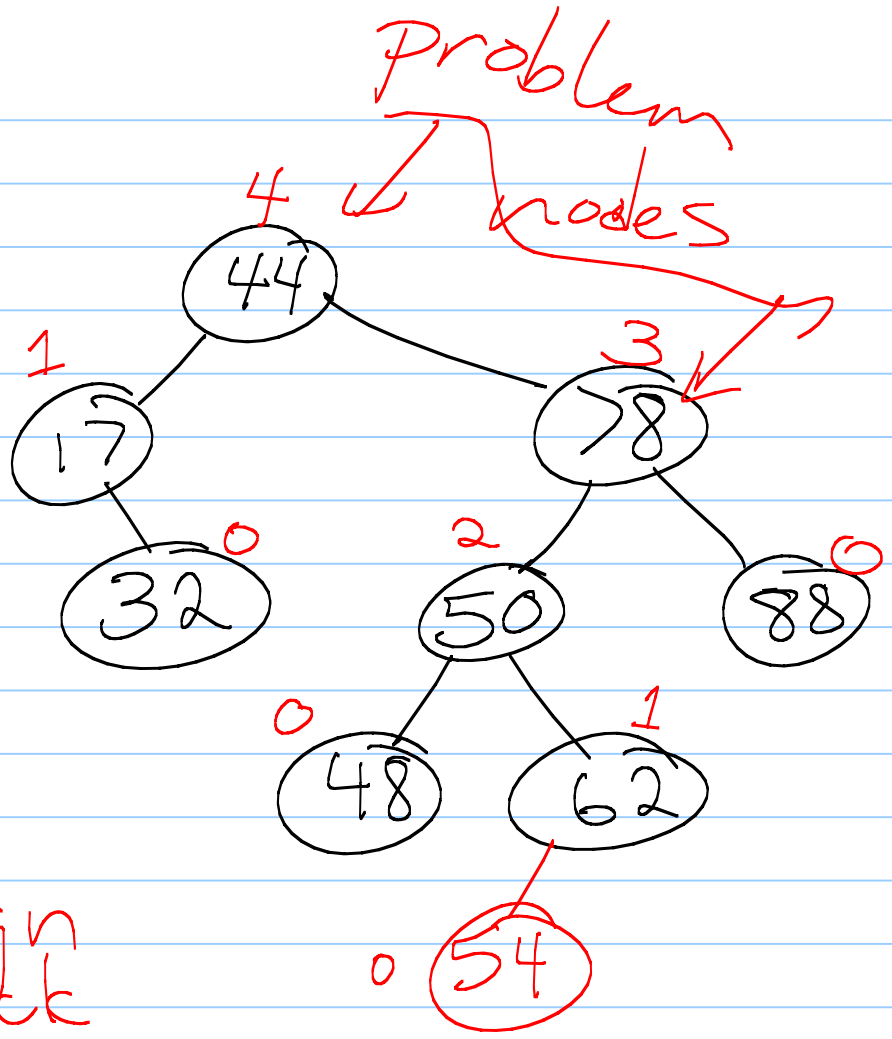
insert! (or remove) } 2 functions

So - find doesn't change

Insert:

insert(54)

new heights:



problem nodes are all on path from root to new node

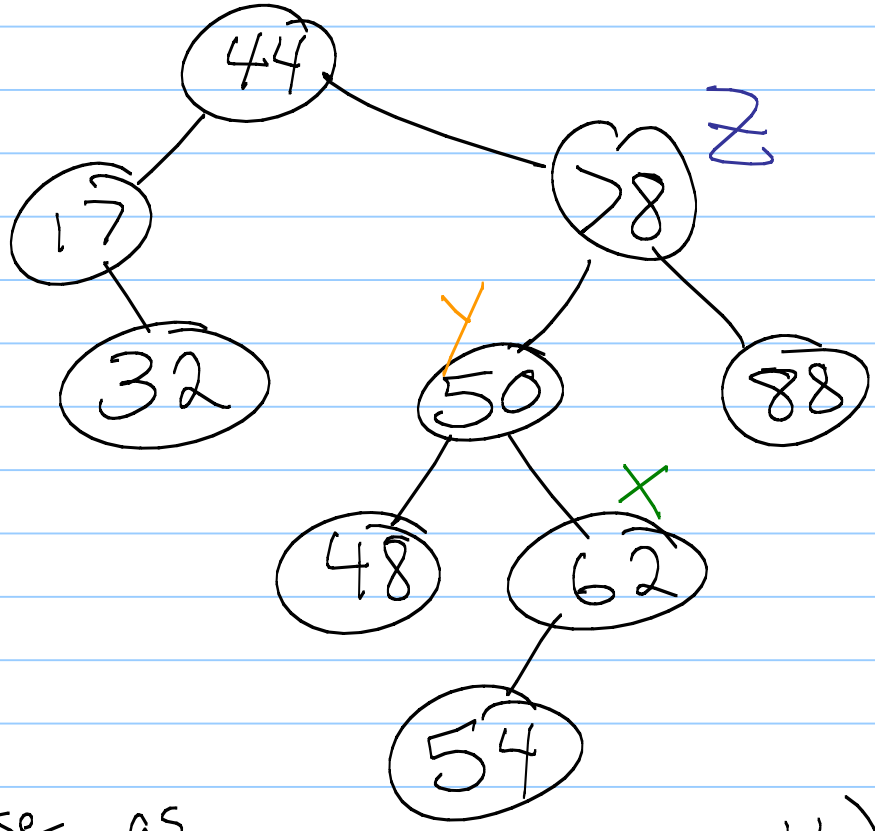
$\Rightarrow$ only have $\leq 2 \cdot \log n$ nodes to check

Consider the lowest node which does not satisfy height-balance property.
↳ call this z

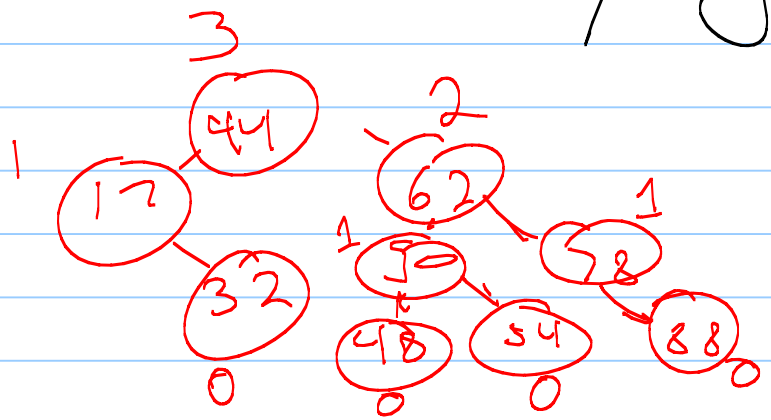
Let y be z's child with higher height.

Let x be y's child with higher height.

Now- Fix it! (Use as few pointer changes as possible)
(using only z, y + x)

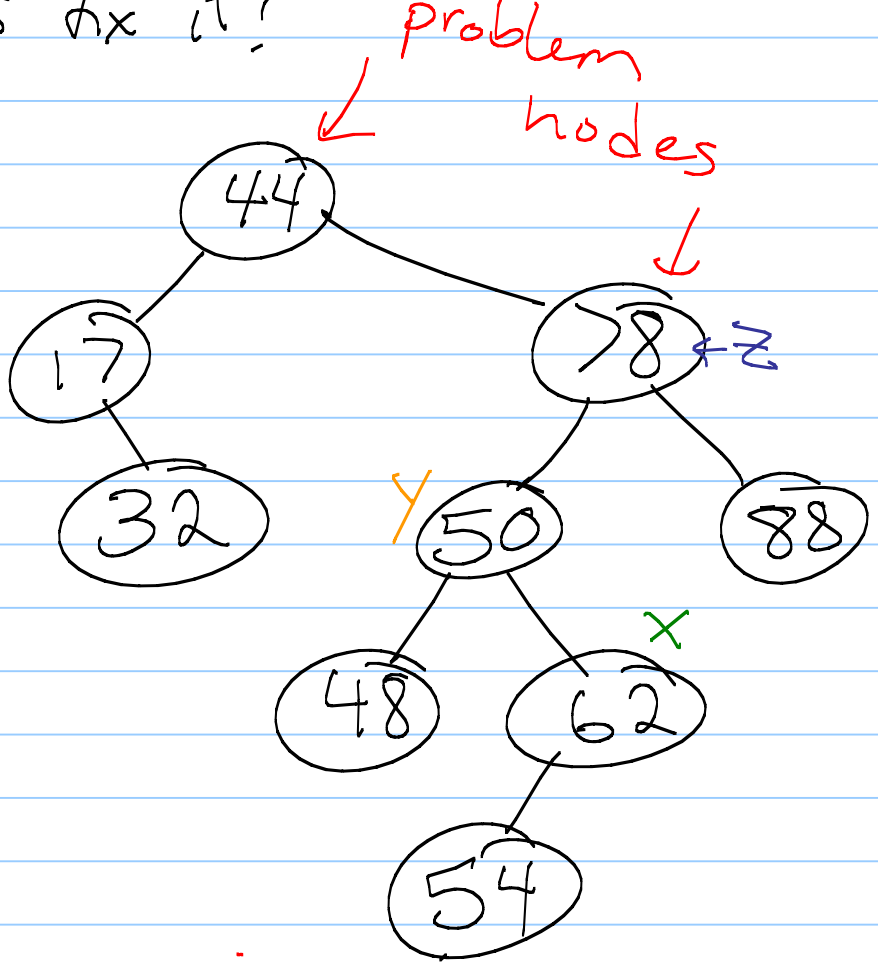


OK - how did you guys fix it?



→ "promoted" X

reattached things

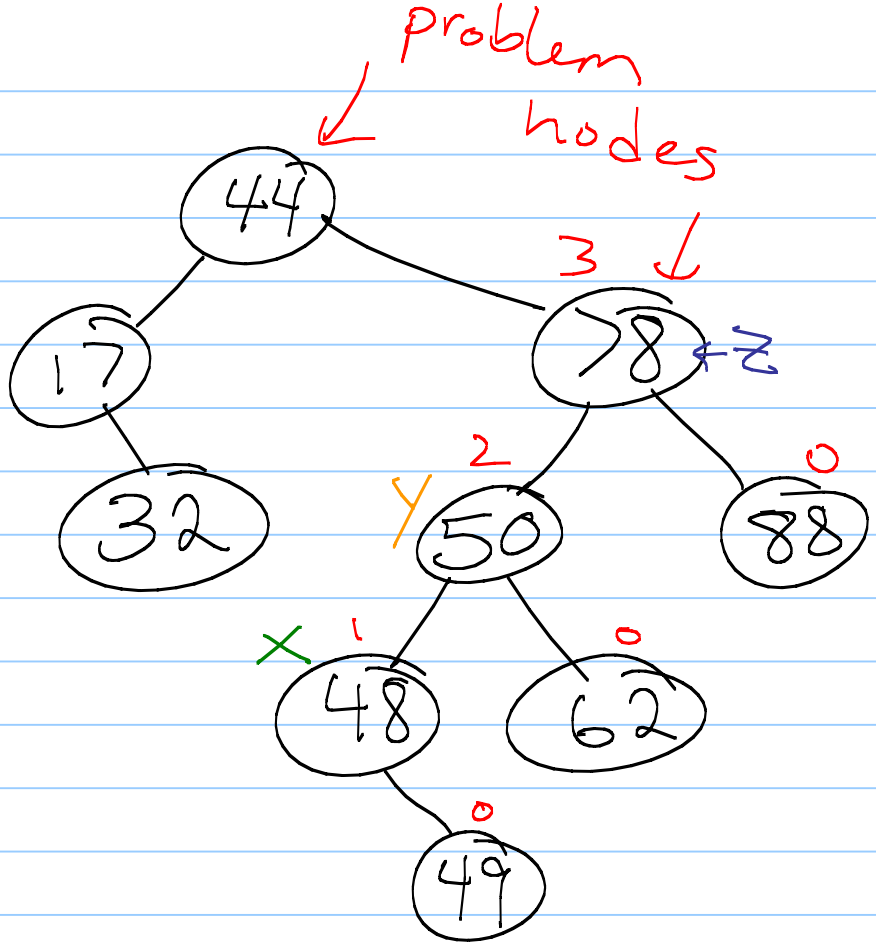


Another one: insert(49)

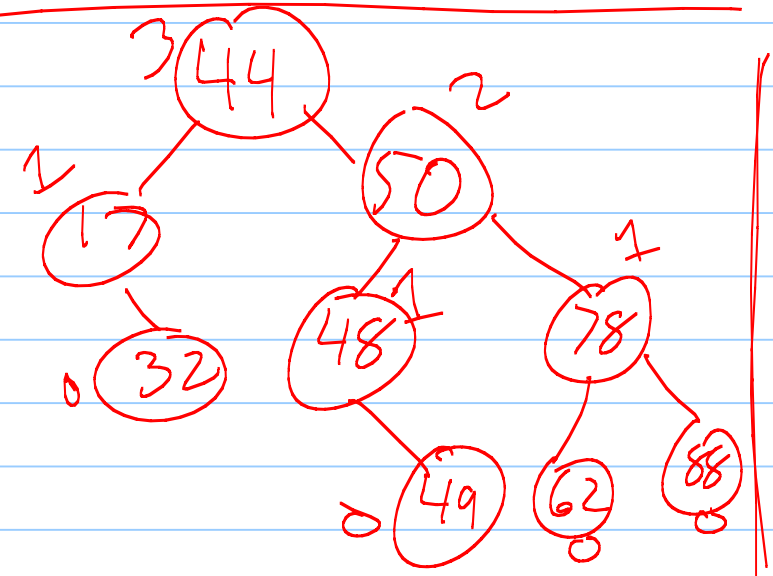
Consider the lowest node which does not satisfy height-balance property.
↳ call this z

Let y be z's child with higher height.

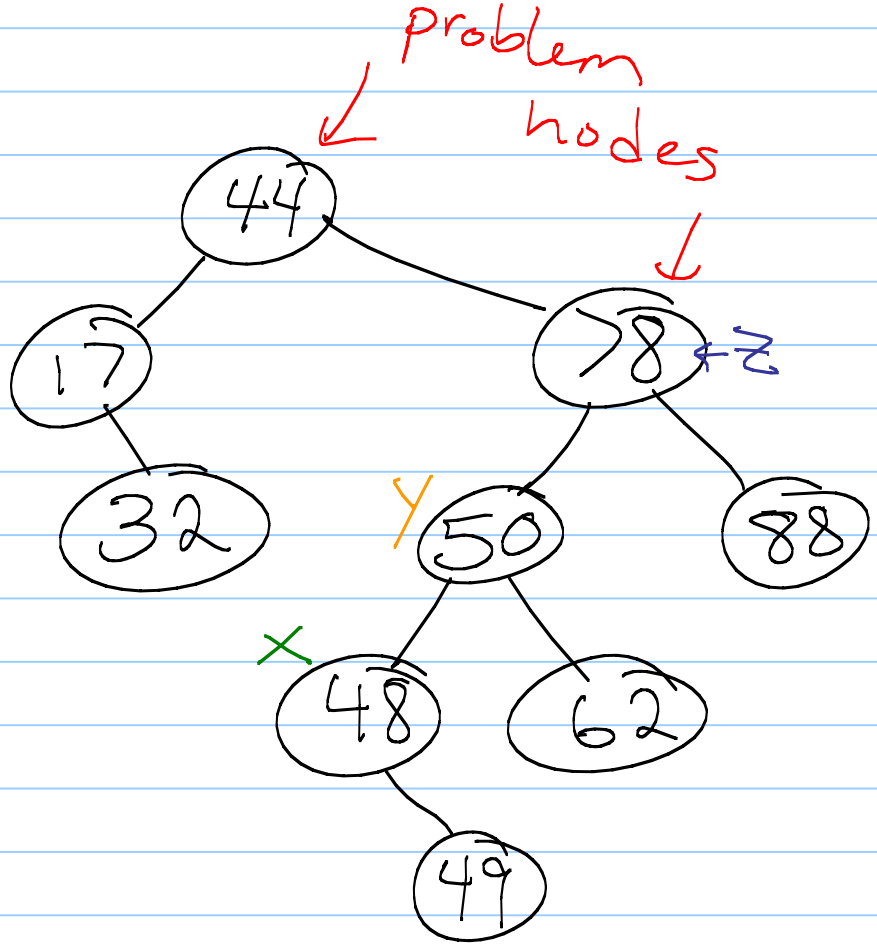
Let x be y's child with higher height.



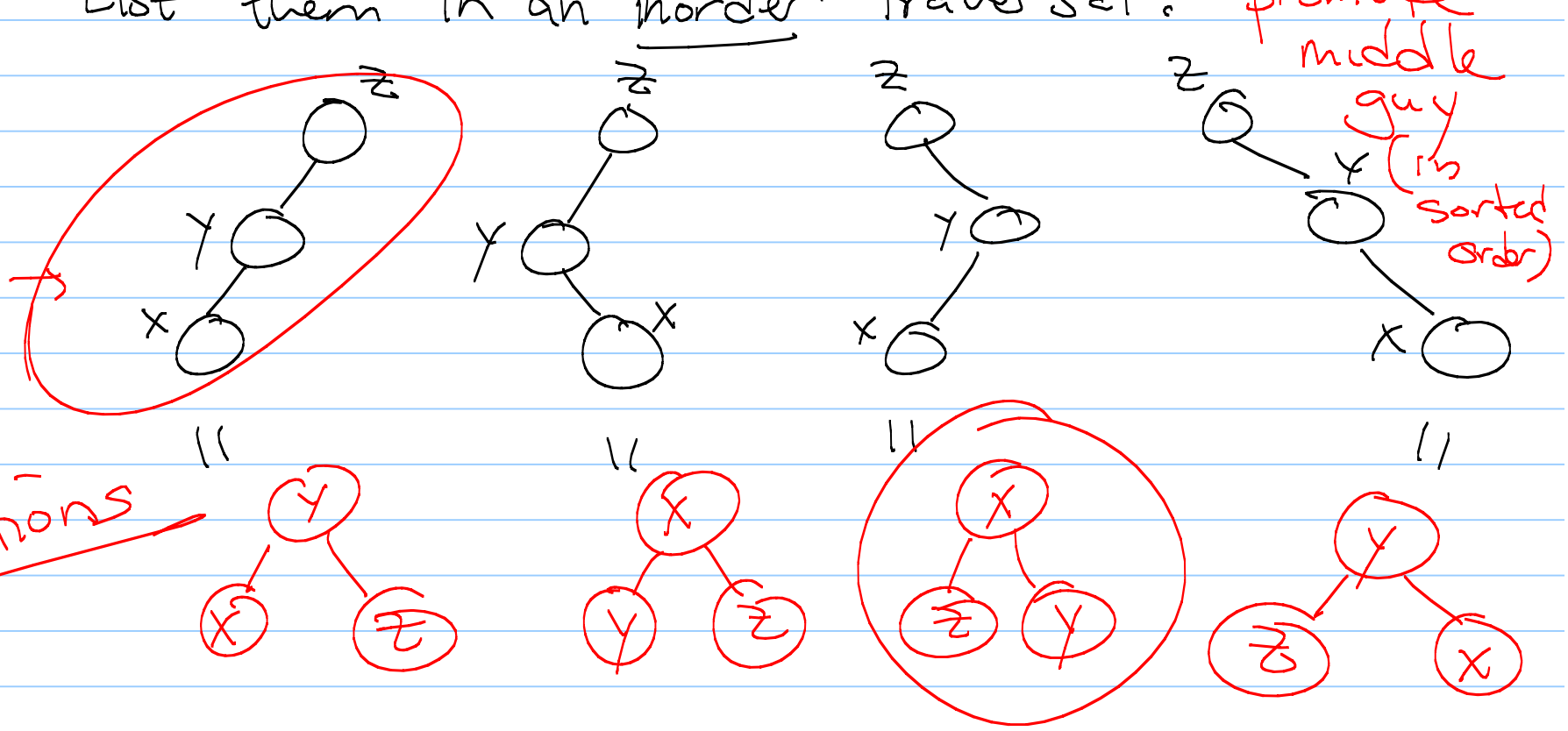
How to fix this one?



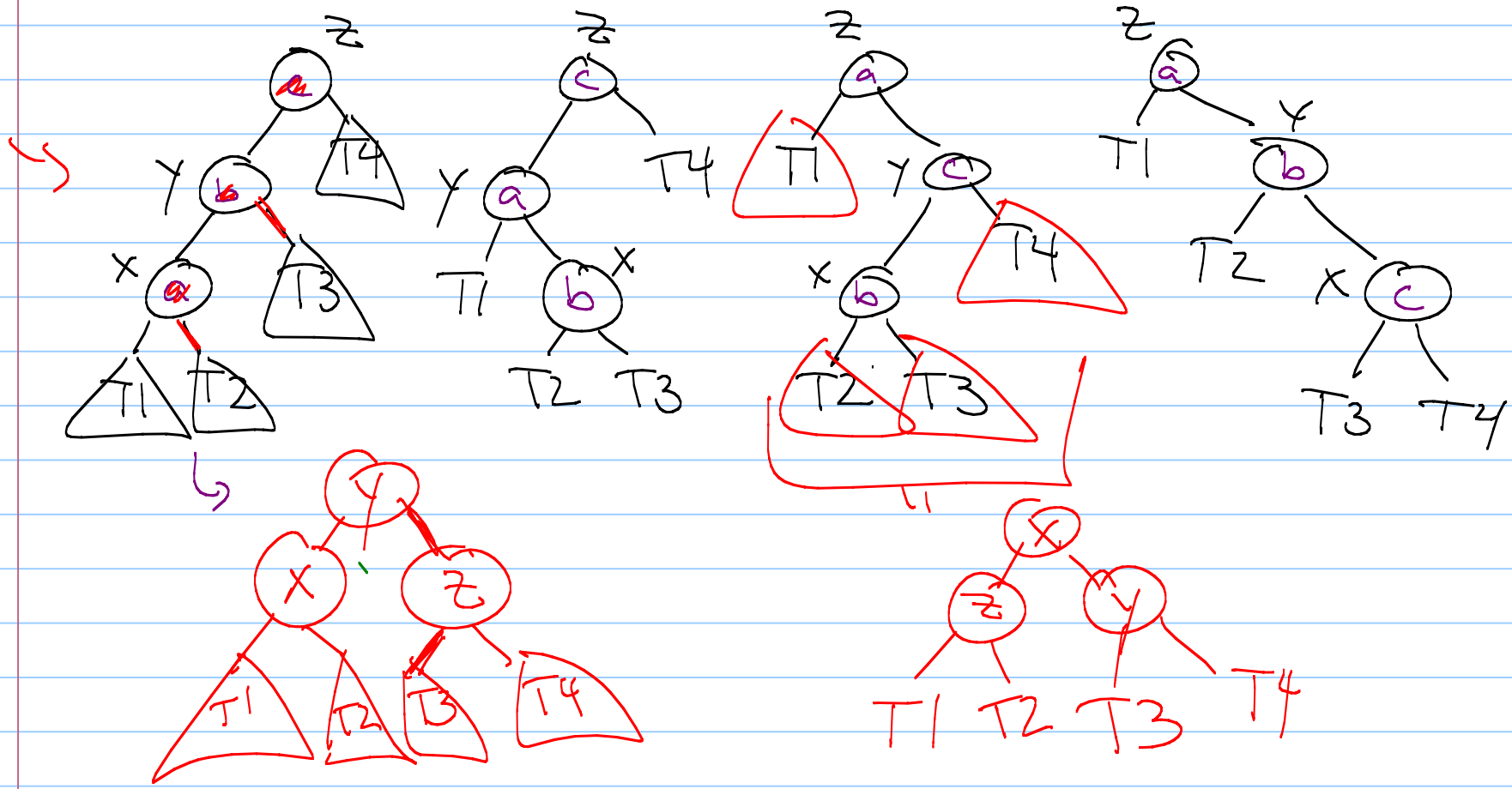
- promoted Y
  ↓ reattached



Generalize: Consider x, y, & z.
List them in an inorder traversal:



Restructure:



Any way you do it, b becomes the new root of this subtree!

↳ and a is left child, c is right child.

Any way you do it, a & c's children are T1, T2, T3, T4 (in that order)!

How long does this take?

14 pointer manipulations
(plus ~3 or 4 comparisons)

$O(1)$ time

After 14 pointer manipulations, I have fixed 1 broken node - 2.

How many nodes were broken?

$2 \log_2 n$

So - to fix tree, spend $14 \cdot 2 \log_2 n = O(\log n)$ time

Insert:

- find where new node goes