CSE 373: Data Structures and Algorithms

AVL Trees

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Outline

So far

- List
- Dictionaries
- Add and remove operations on dictionaries implemented with arrays or lists are O(n)
- Trees, BSTs in particular, offer speed up because of their branching factors
- BSTs are in the average case, but not in the worse case

	Insert()	Find()	Remove()
Average case	O(log n)	O(log n)	O(log n)
Worst case	O(n)	O(n)	O(n)

Today

- Can we do better? Can we adapt our BST so we never get the worst case

Review: Worksheets

Balanced BST observation

BST: the shallower the better!

For a BST with n nodes inserted in arbitrary order

– Average height is $O(\log n)$

– Worst case height is $O(n)$

Solution: Require a **Balance Condition** that

Simple cases such as inserting in key order lead to the worst-case scenario

1. ensures depth is always $O(\log n)$ – strong enough!
2. is easy to maintain – not strong enough!

AVL trees: Balanced BSTs

AVL Trees must satisfy the following properties:

- binary trees: every node must have between 0 and 2 children
- binary search tree (BST property): for every node, all keys in the left subtree must be smaller and all keys in the right subtree must be larger than the root node
- Balanced (AVL property): for every node, there can be no more than a difference of 1 in the height of the left subtree from the right. Math.abs(height(left subtree) – height(right subtree)) ≤ 1

AVL stands for **A**delson-**V**elsky and **L**andis (the inventors of the data structure)

The AVL property:

1. ensures depth is always $O(\log n)$ – Yes!
2. is easy to maintain – Yes! (using single and double rotations)

Potential balance conditions (1)

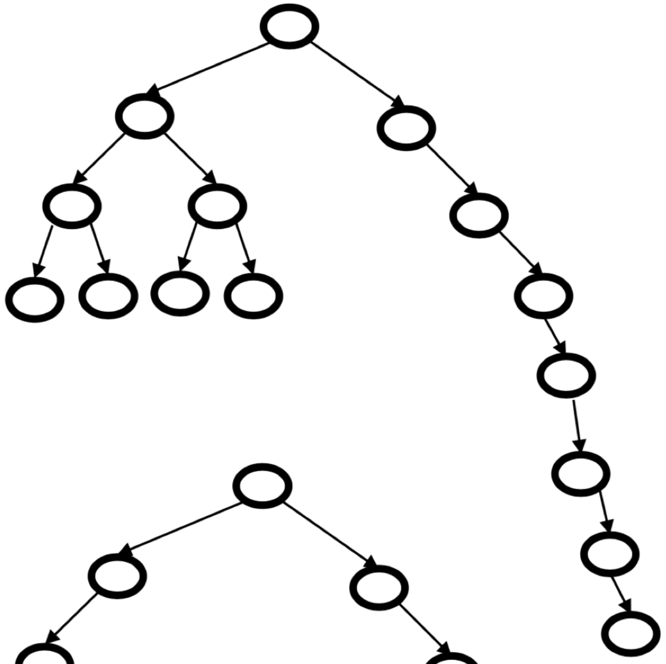
1. Left and right subtrees of the *root* have equal number of nodes

2. Left and right subtrees of the *root* have equal *height*

Potential balance conditions (1)

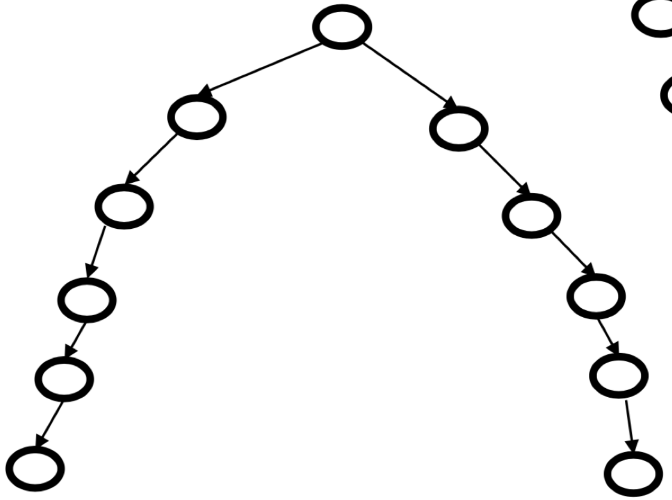
1. Left and right subtrees of the *root* have equal number of nodes

   Too weak!
   Height mismatch example:



2. Left and right subtrees of the *root* have equal *height*

   Too weak!
   Double chain example:



Potential balance conditions (2)

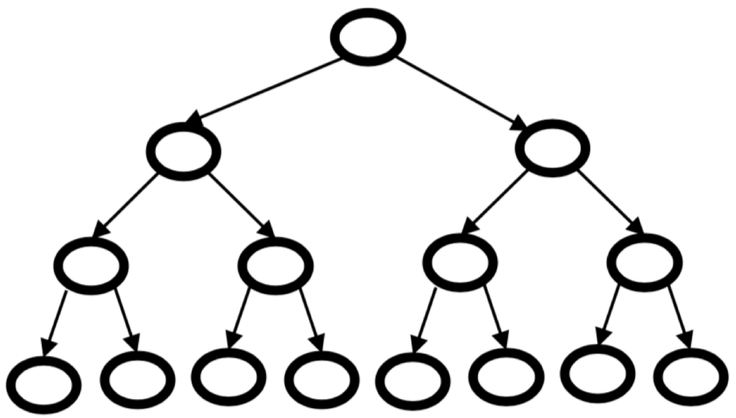
3. Left and right subtrees of every node have equal number of nodes

4. Left and right subtrees of every node have equal *height*

Potential balance conditions (2)

3. Left and right subtrees of every node have equal number of nodes

Too strong!
Only perfect trees (2^n – 1 nodes)



4. Left and right subtrees of every node have equal *height*

Too strong!
Only perfect trees (2^n – 1 nodes)

AVL balance condition

AVL condition: For every node, the height of its left subtree and right subtree differ by at most 1.

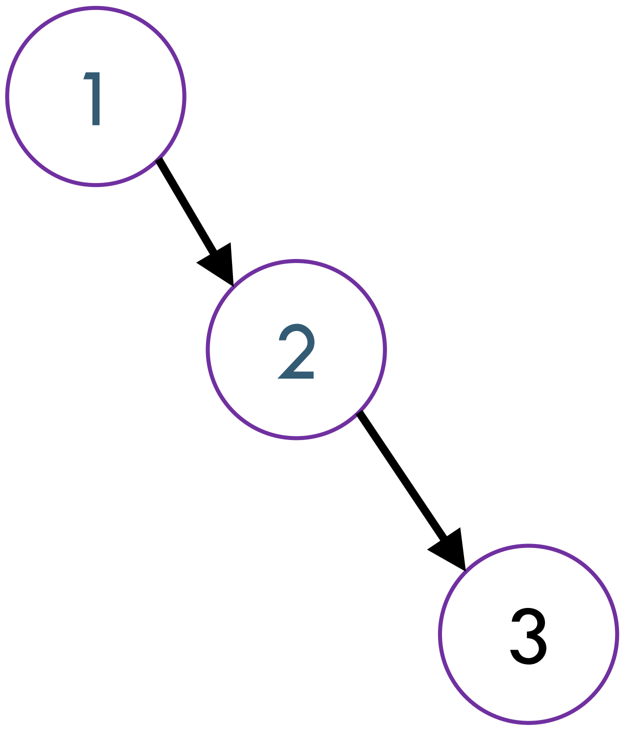
```
balance(node) = Math.abs( height(node.left) – height(node.right) )
```

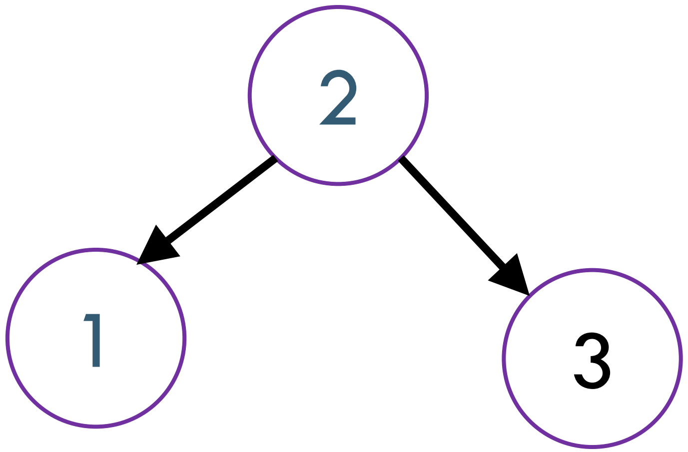
AVL condition: for every node, balance(node) ≤ 1

Worksheet (Q9)

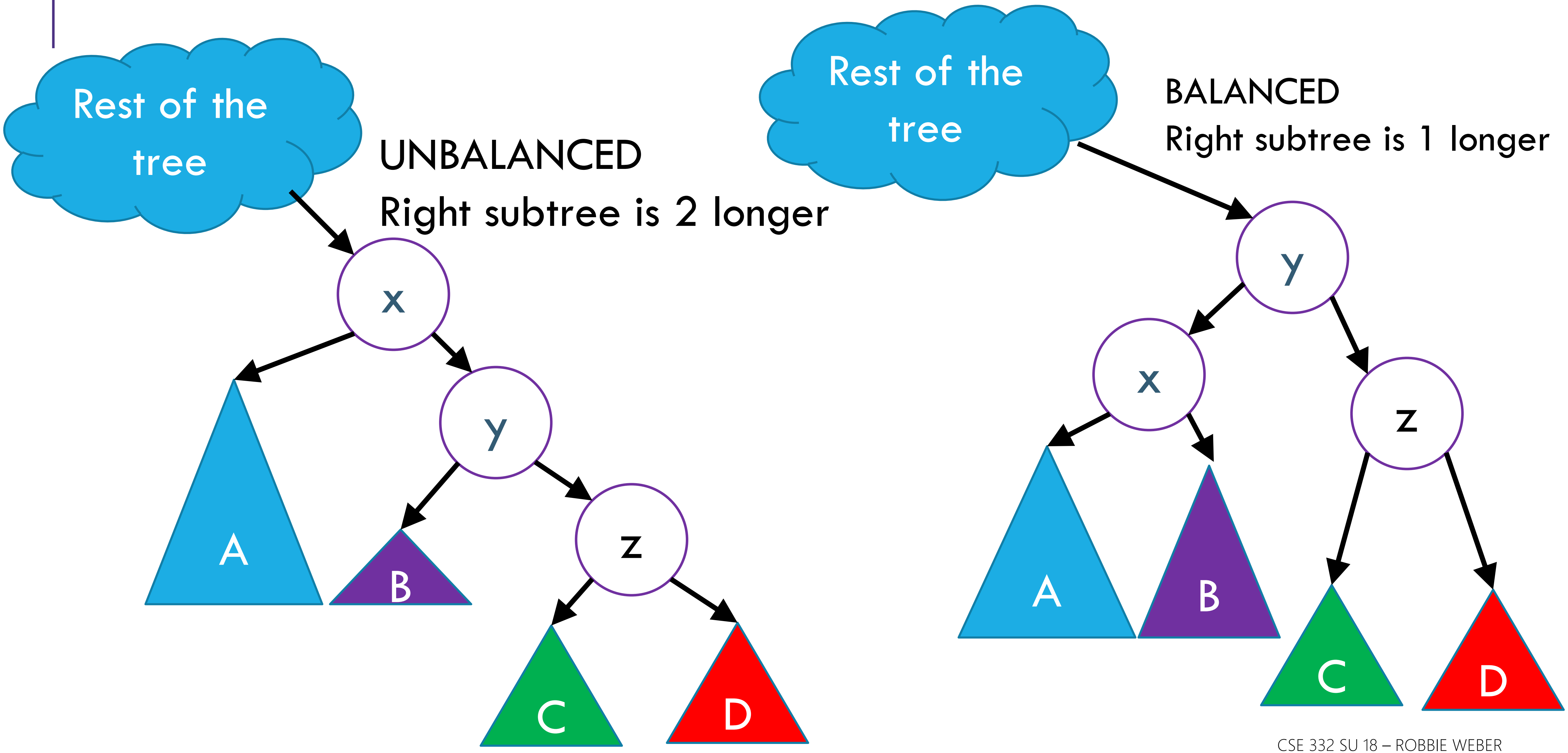
Insertion

What happens if when we do an insert(3), we break the AVL condition?

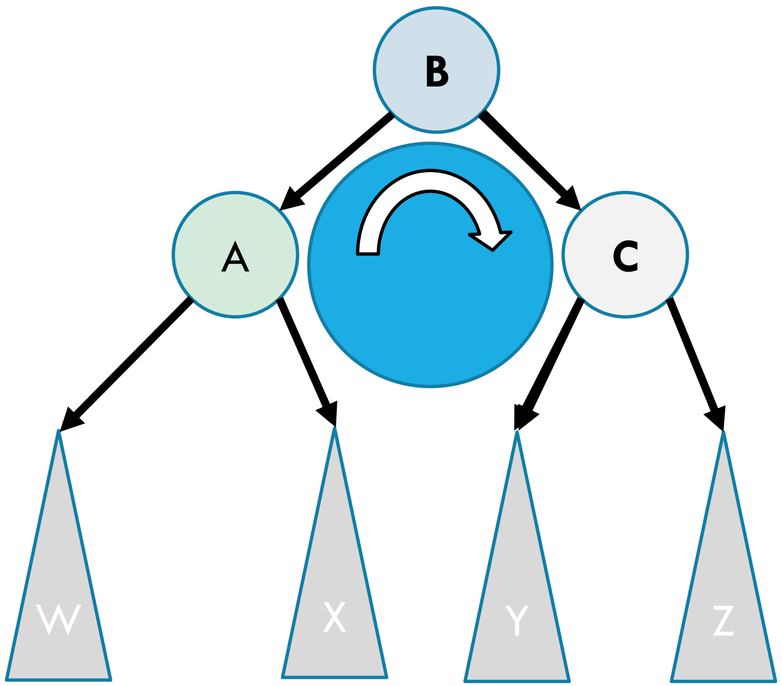




Left Rotation

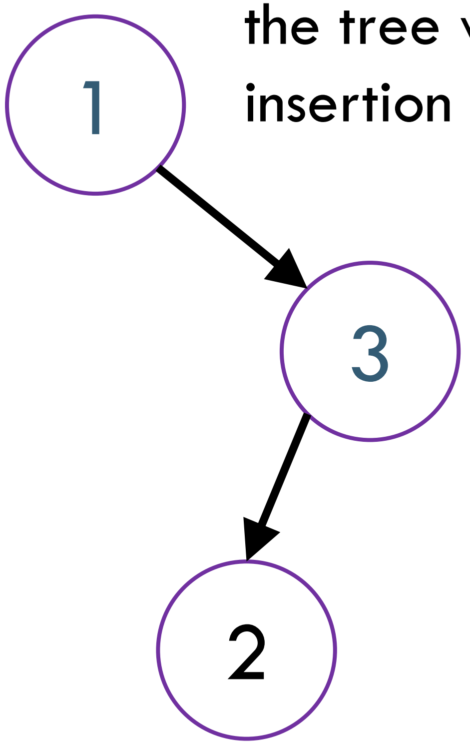


Tree Rotations: Right rotation



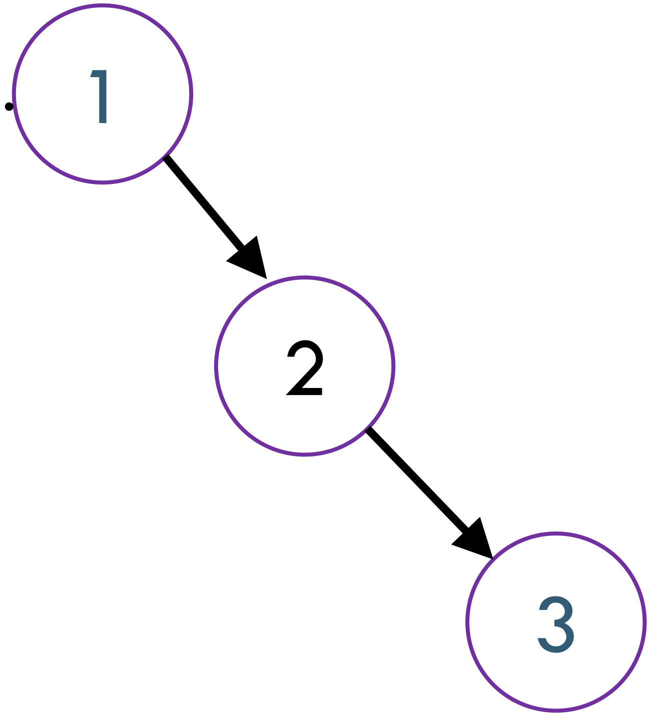
It Gets More Complicated

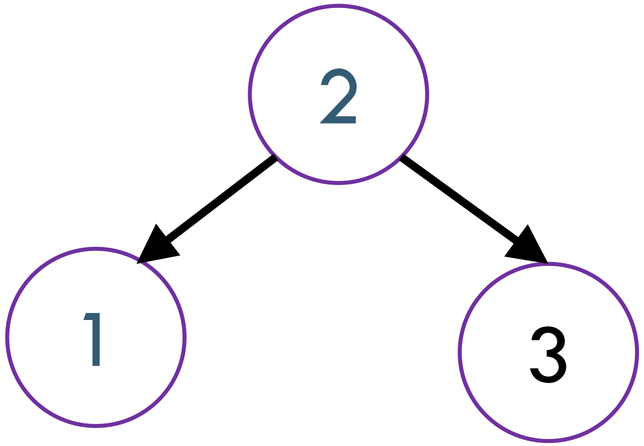
There's a "kink" in the tree where the insertion happened.



Can't do a left rotation

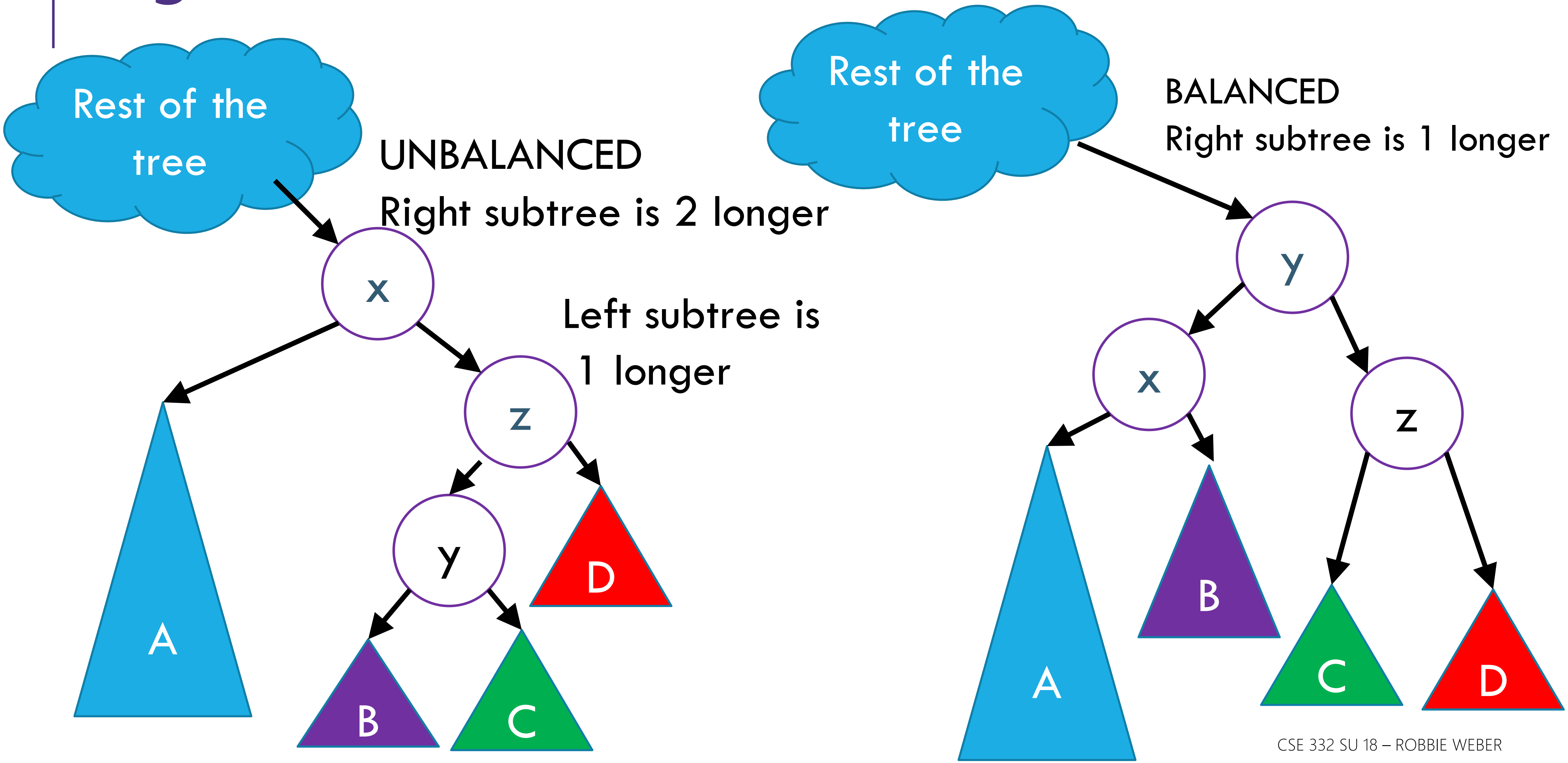
Do a "right" rotation around 3 first.



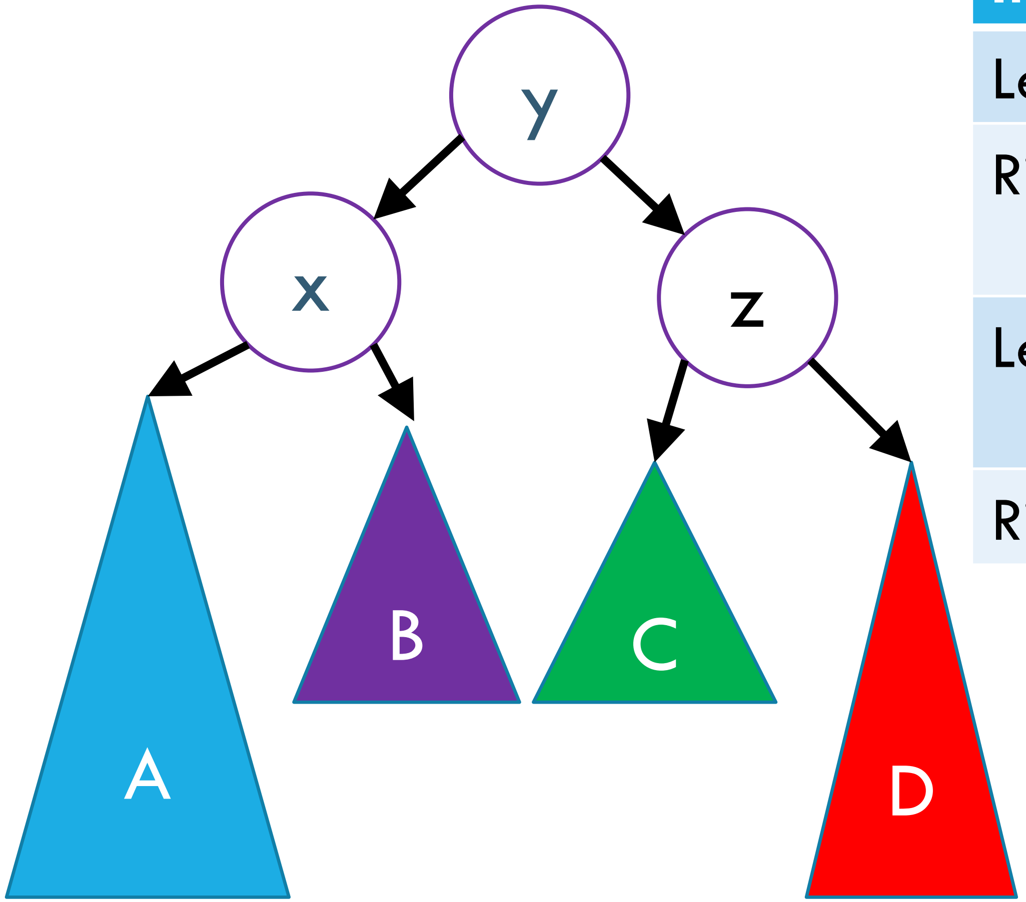


Now do a left rotation.

Right Left Rotation

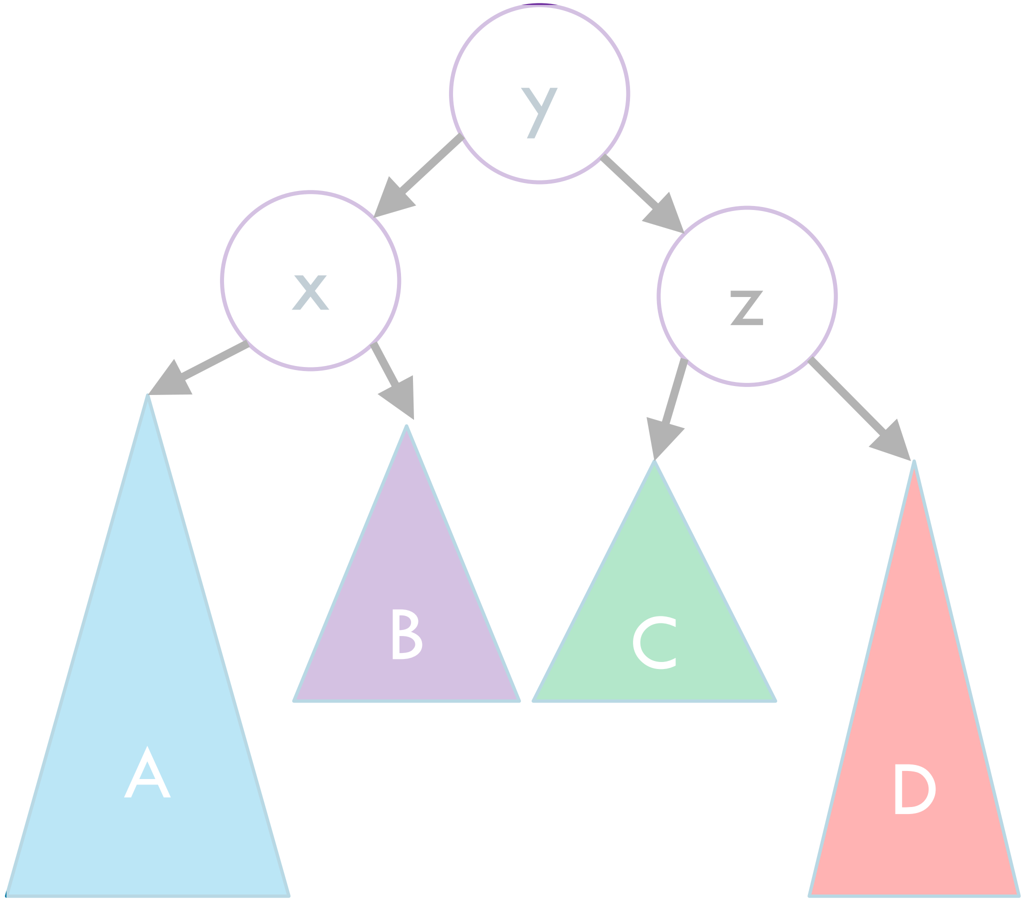


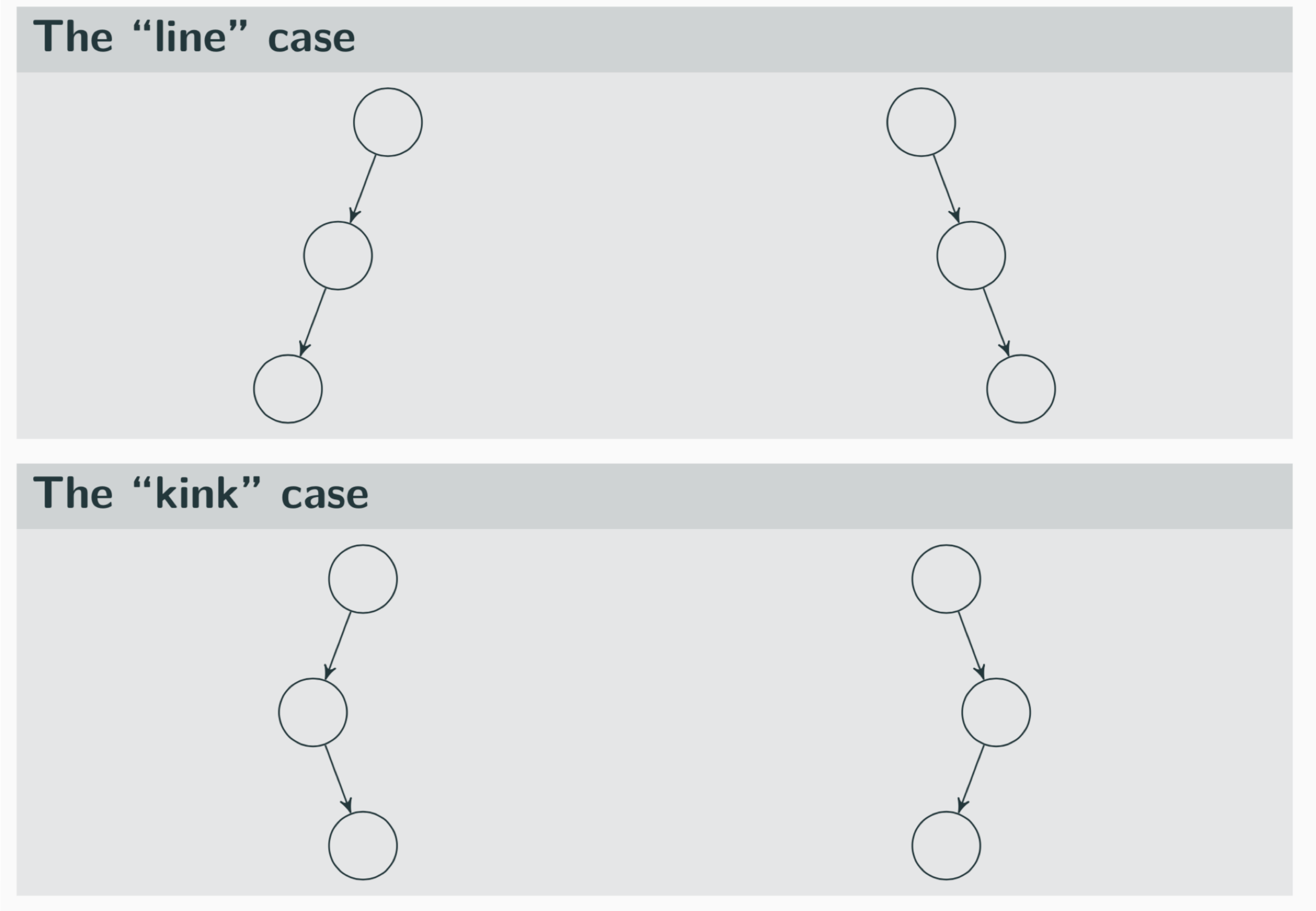
Four cases to consider



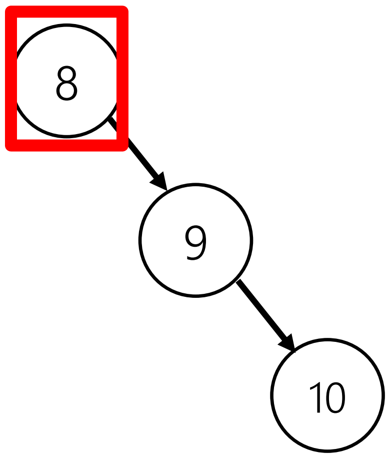
Insert location	Solution
Left subtree of left child of y	Single right rotation
Right subtree of left child of y	Double (left-right) rotation
Left subtree of right child of y	Double (right-left) rotation
Right subtree of right child of y	Single left rotation

Four cases to consider

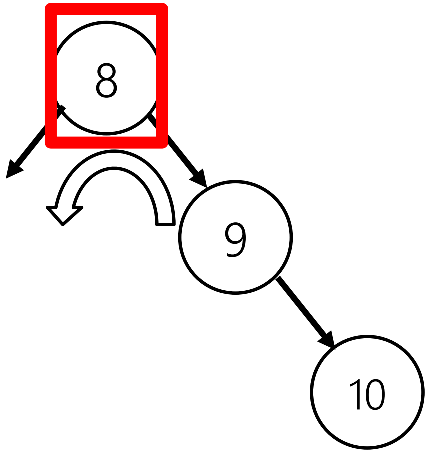




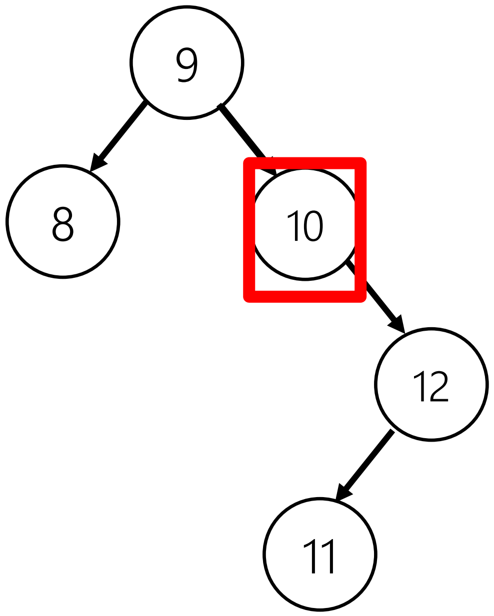
AVL Example: 8,9,10,12,11



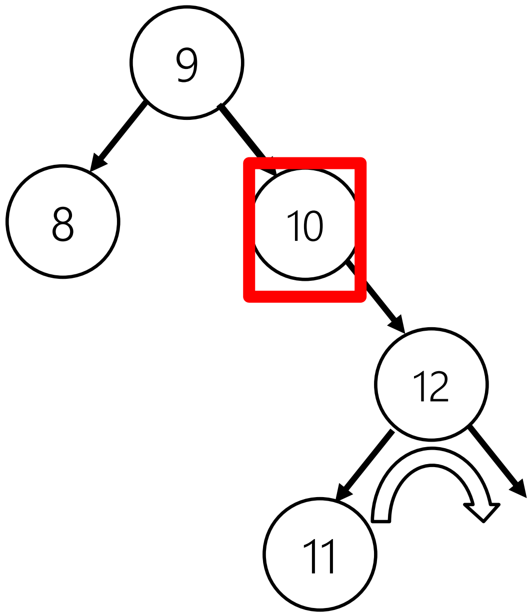
AVL Example: 8,9,10,12,11



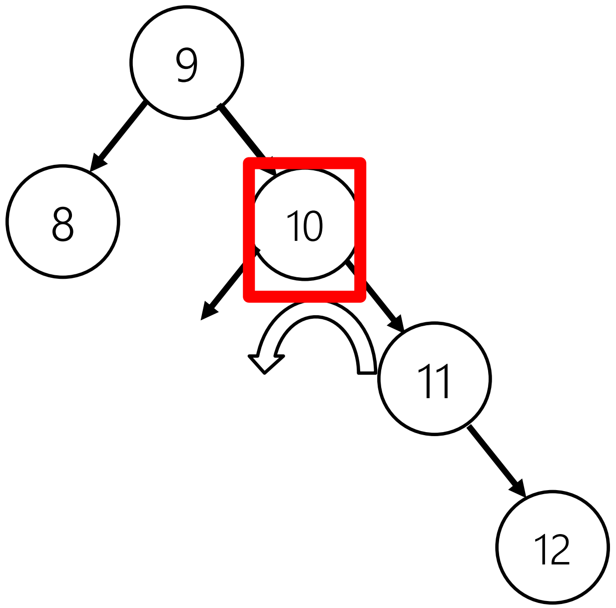
AVL Example: 8,9,10,12,11



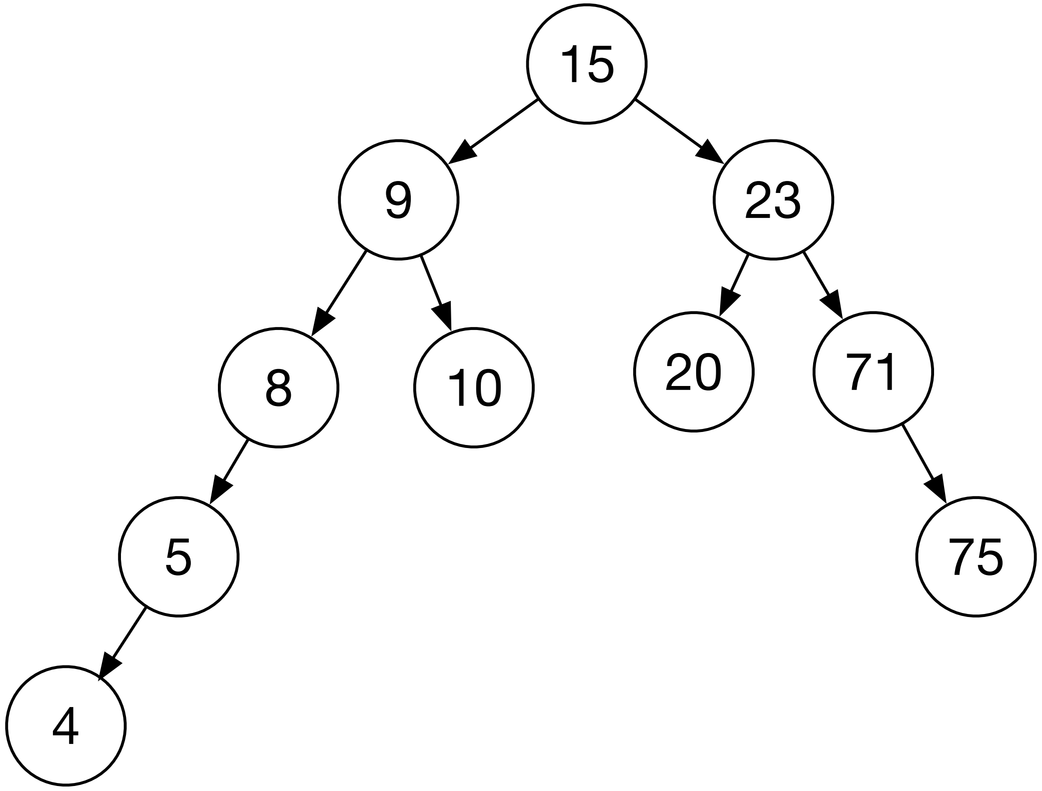
AVL Example: 8,9,10,12,11



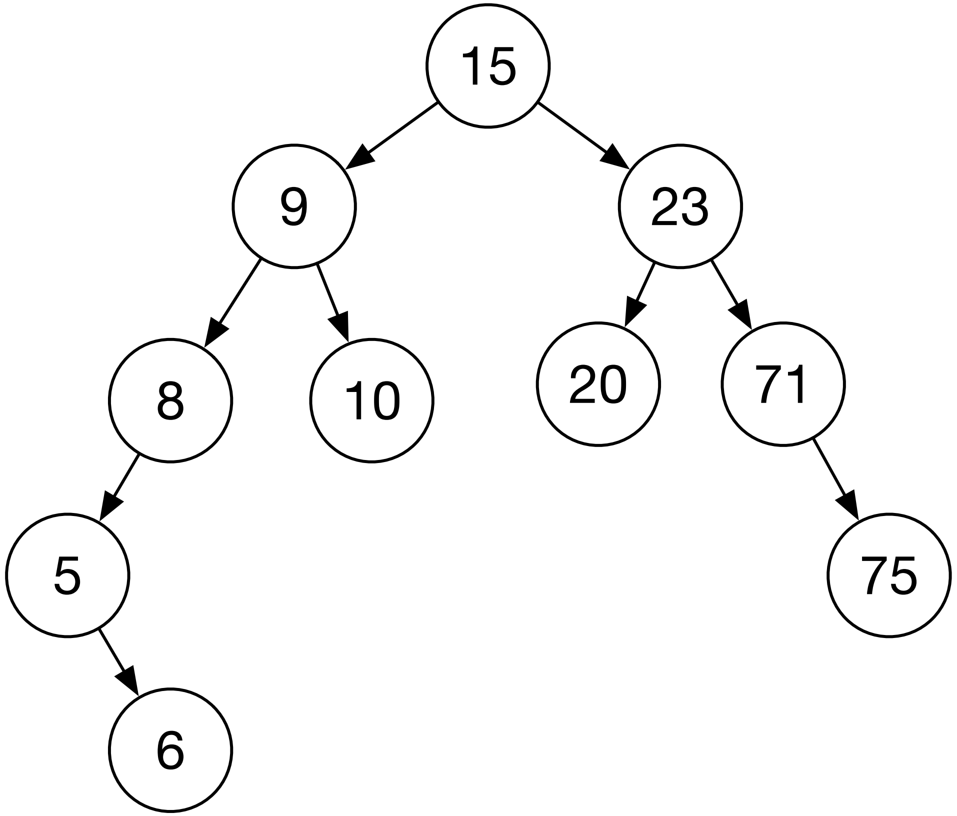
AVL Example: 8,9,10,12,11



Worksheet (Q10A)



Worksheet (Q10B)



How Long Does Rebalancing Take?

Assume we store in each node the height of its subtree.

How do we find an unbalanced node?

How many rotations might we have to do?

How Long Does Rebalancing Take?

Assume we store in each node the height of its subtree.

How do we find an unbalanced node?

- Just go back up the tree from where we inserted.

How many rotations might we have to do?

- Just a single or double rotation on the lowest unbalanced node.
- A rotation will cause the subtree rooted where the rotation happens to have the same height it had before insertion.

Lots of cool Self-Balancing BSTs out there!

Popular self-balancing BSTs include:

- AVL tree
- Splay tree
- 2-3 tree
- AA tree
- Red-black tree
- Scapegoat tree
- Treap

(Not covered in this class, but several are in the textbook and all of them are online!)

(From https://en.wikipedia.org/wiki/Self-balancing_binary_search_tree#Implementations)