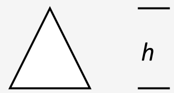
AVL Trees

COMP 2210 – Dr. Hendrix

AUBURN UNIVERSITY

SAMUEL GINN COLLEGE OF ENGINEERING

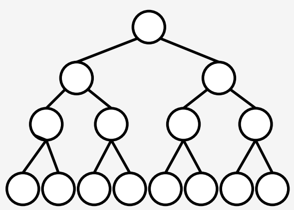
Shapes and height

height 

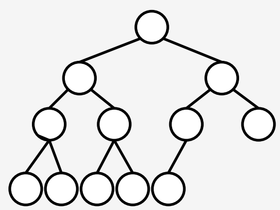
Many tree algorithms are dependent to some extent on the tree's height.

best-case BST

full

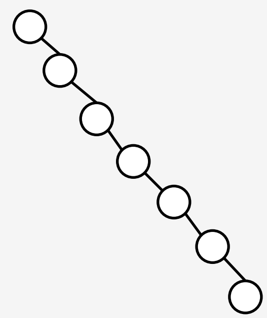


complete



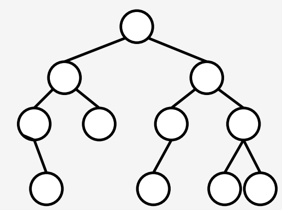
$h(t) = floor(\log_2 N) + 1$

worst-case BST



$h(t) = N$

balanced BST

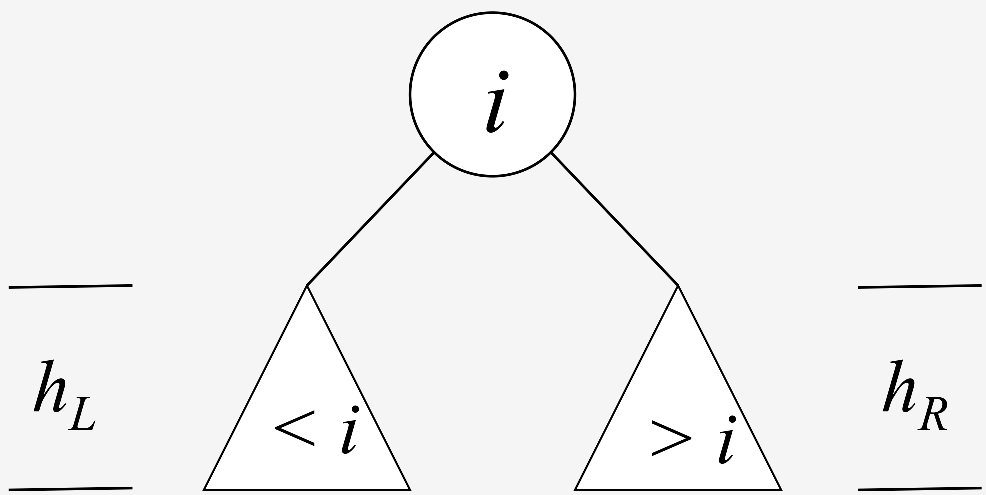


$h(t) = O(\log N)$

AVL Trees

An AVL tree is a **binary search tree**

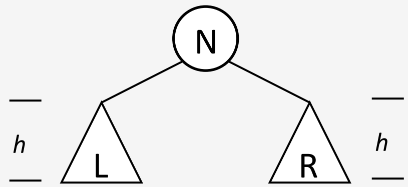
in which the heights of the left and right subtree of *every* node differ by at most 1.

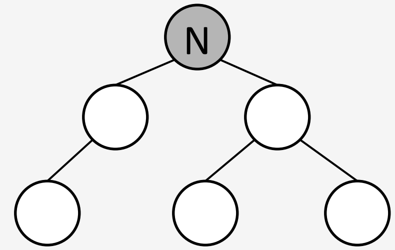


$$|h_R - h_L| \leq 1$$

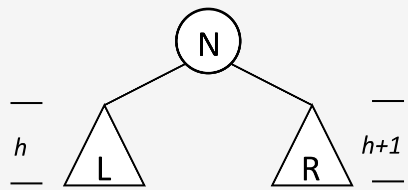
Structural possibilities

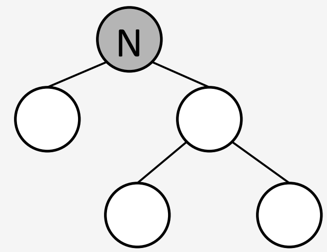
Equal heights



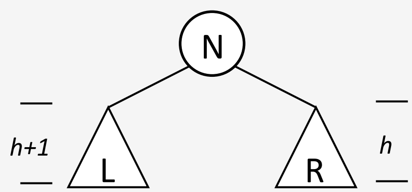


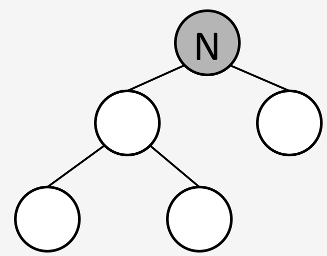
Right is 1 level taller





Left is 1 level taller

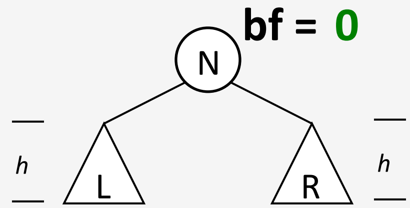


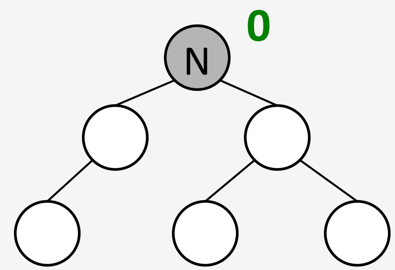


Balance factors

Every node in an AVL tree has a **balance factor**.

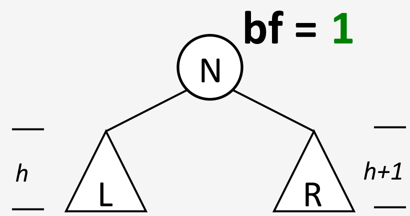
$$bf_N = h_R - h_L$$

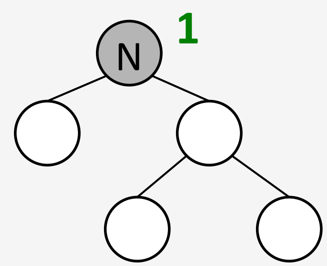




! 

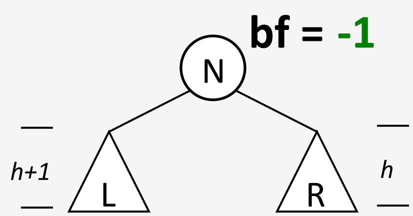
Remember to subtract heights, not balance factors.

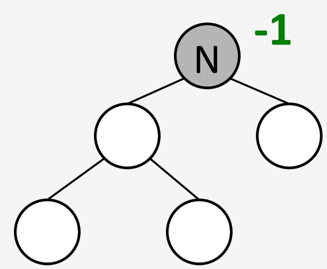




! 

The text counts path lengths differently from me.



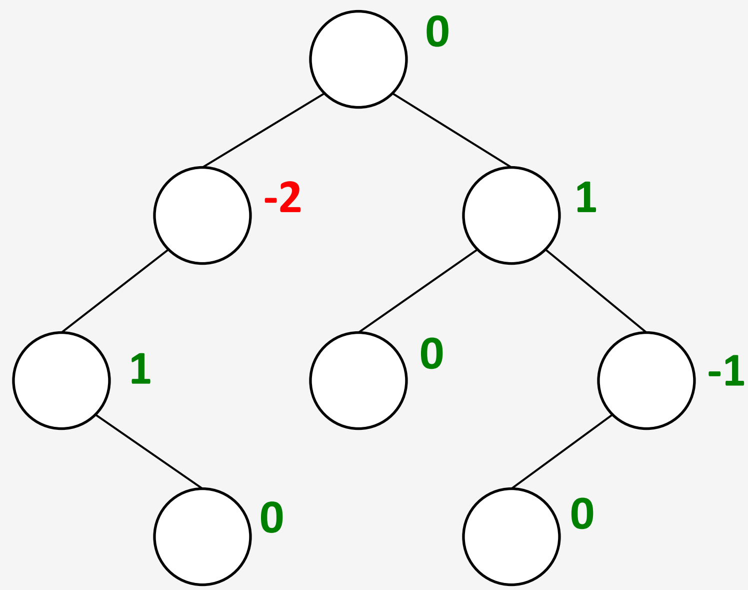


! 

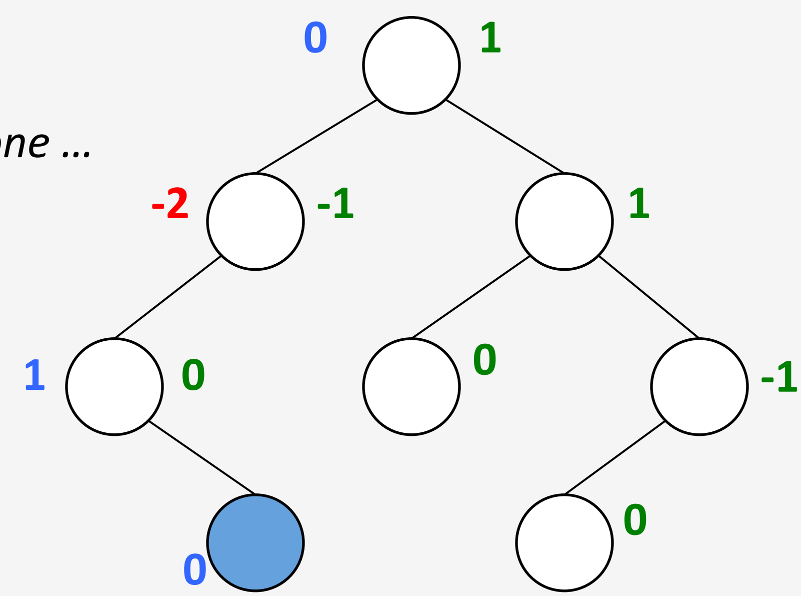
Balance factors are sometimes computed as $h_L - h_R$.

Balance factor example

NOT an AVL Tree



But it could have been one …

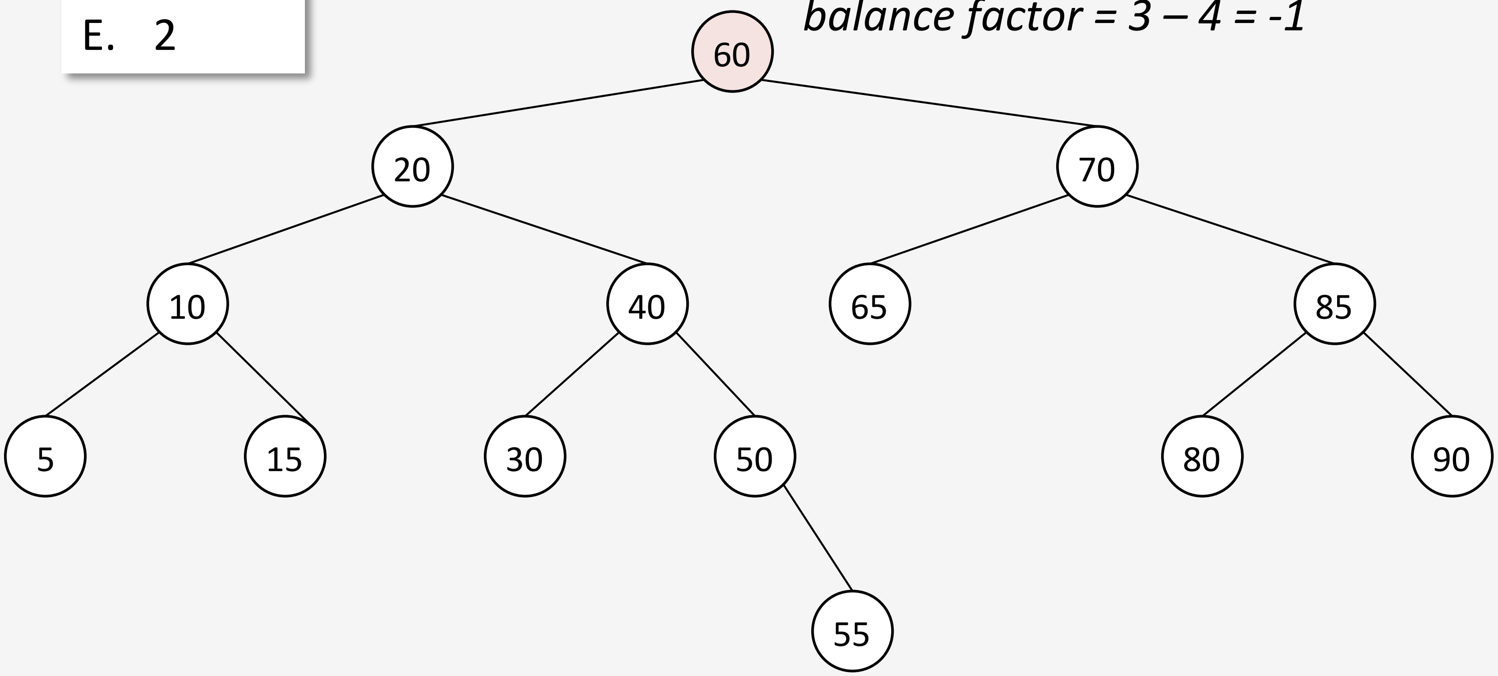


Participation question

Q. In the AVL tree below, what is the balance factor of the shaded node?

A. -2
B. -1
C. 0
D. 1
E. 2

balance factor = 3 – 4 = -1

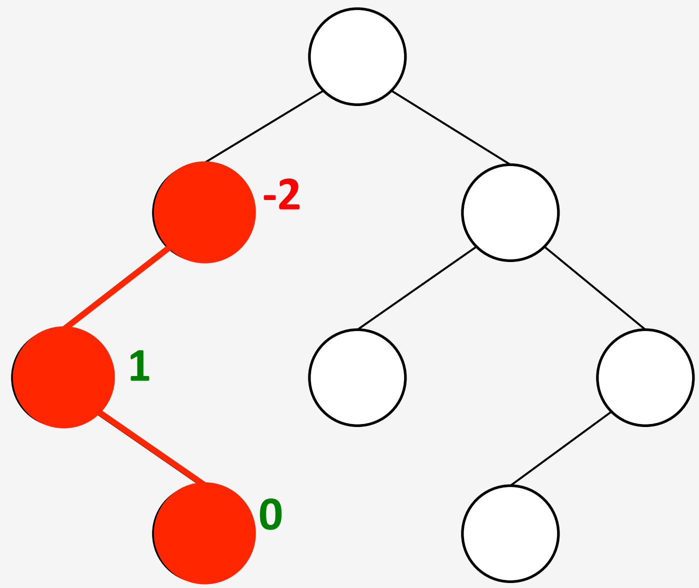


Rebalancing

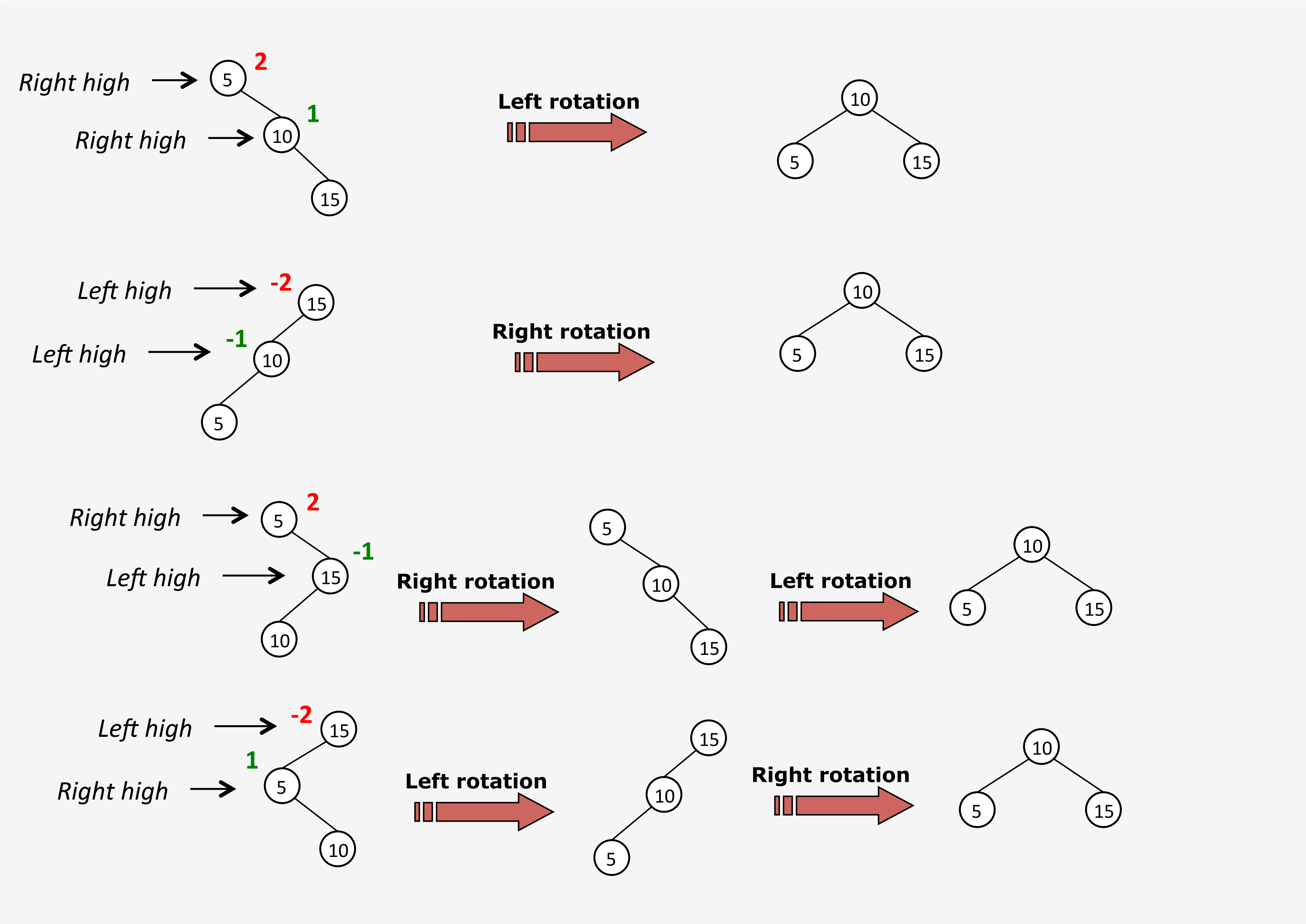
A bf of ±2 means that the subtree rooted at that node is out of balance.

Balance will be restored by subtree rotations.

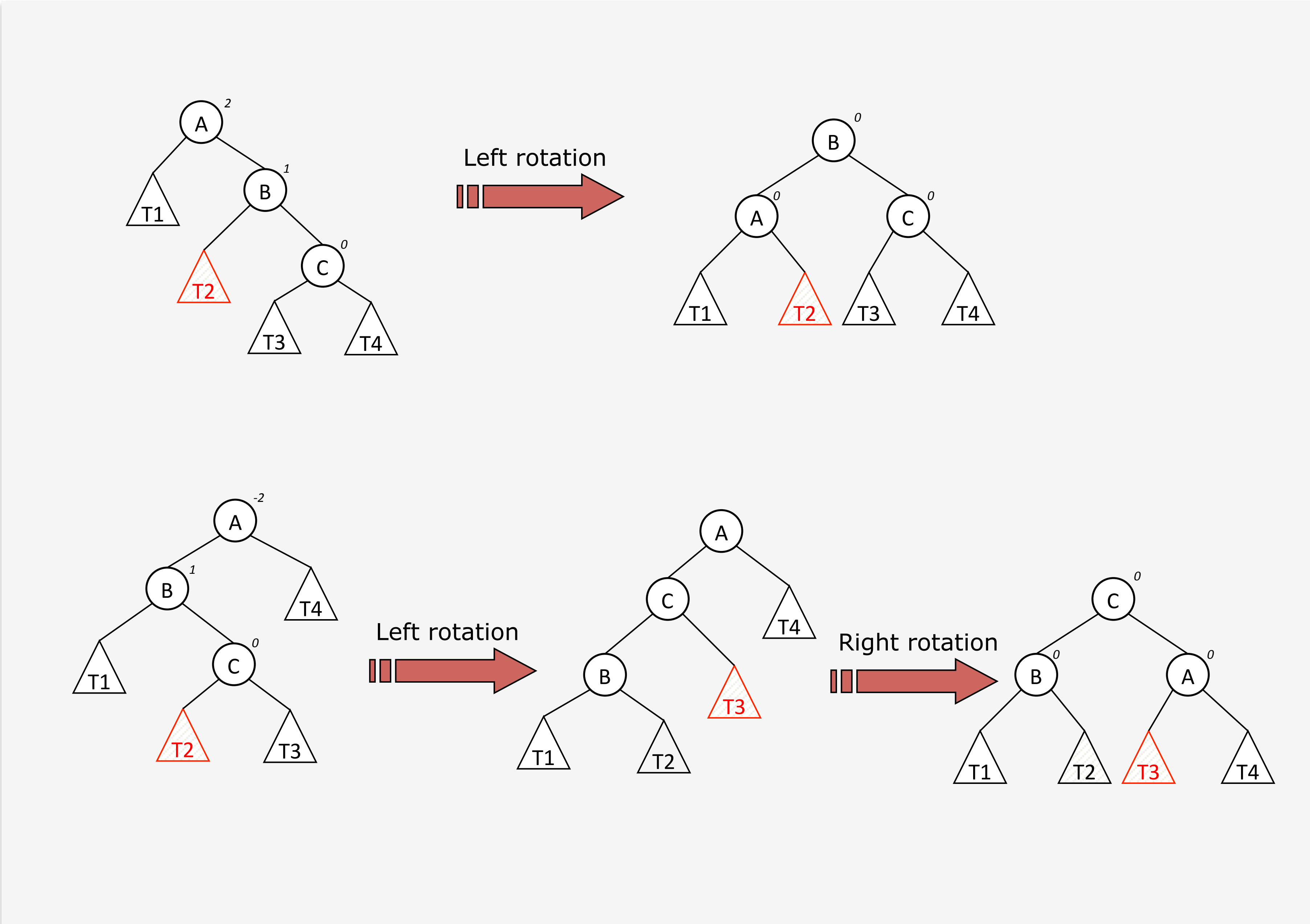
All rotations will occur in the context of a 3-node neighborhood.



Rebalancing operations



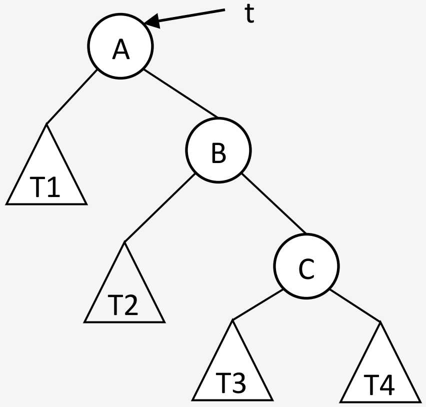
Subtree displacement



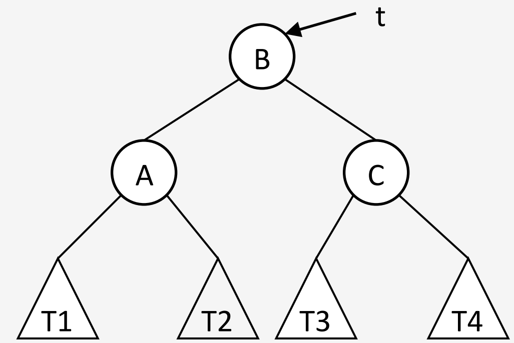
Coding rotations

```
t = rotateLeft(t);
```

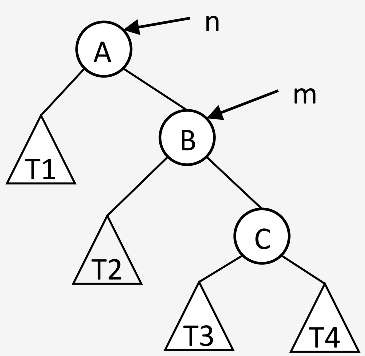
Left rotation around t



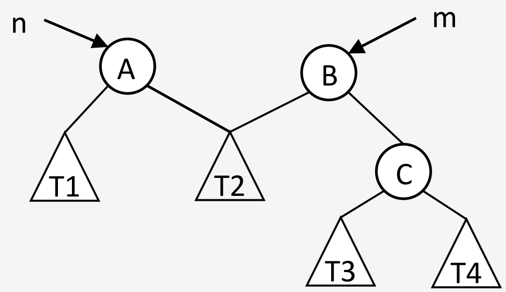

```
public BTN rotateLeft(BTN n)
{
   BTN m = n.right;
   n.right = m.left;
   m.left = n;
   return m;
}
```



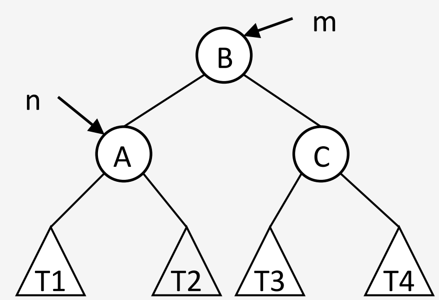

```
BTN m = n.right;
```




```
n.right = m.left;
```

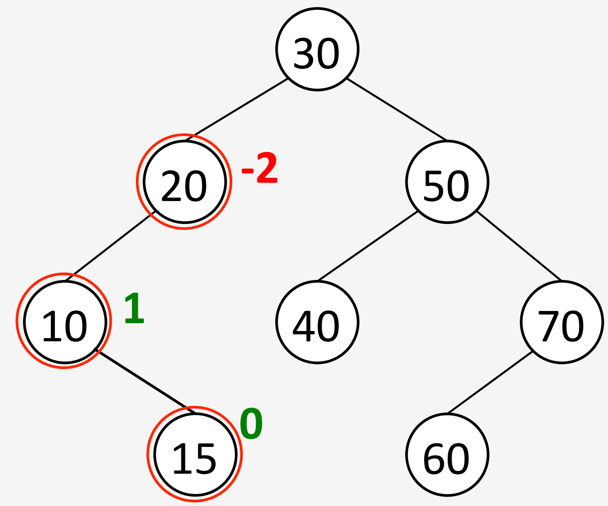



```
m.left = n;
```



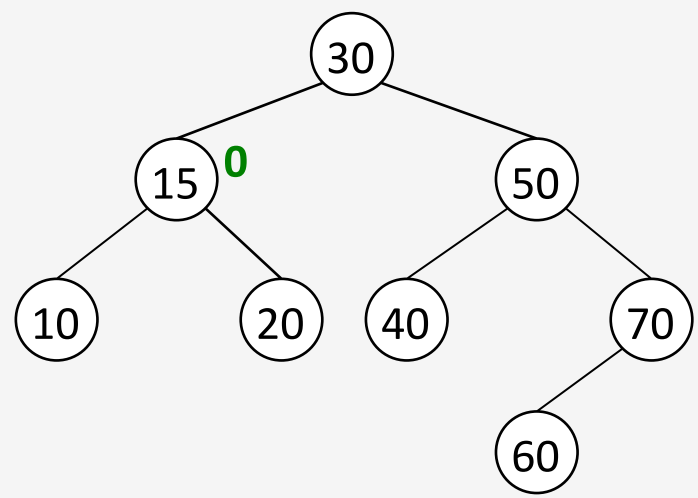
Inserting a new element

Use the standard BST insertion algorithm to insert the new node. (Ex: 15)



Beginning with the node just inserted, walk the reverse path back toward the root, recalculating balance factors.

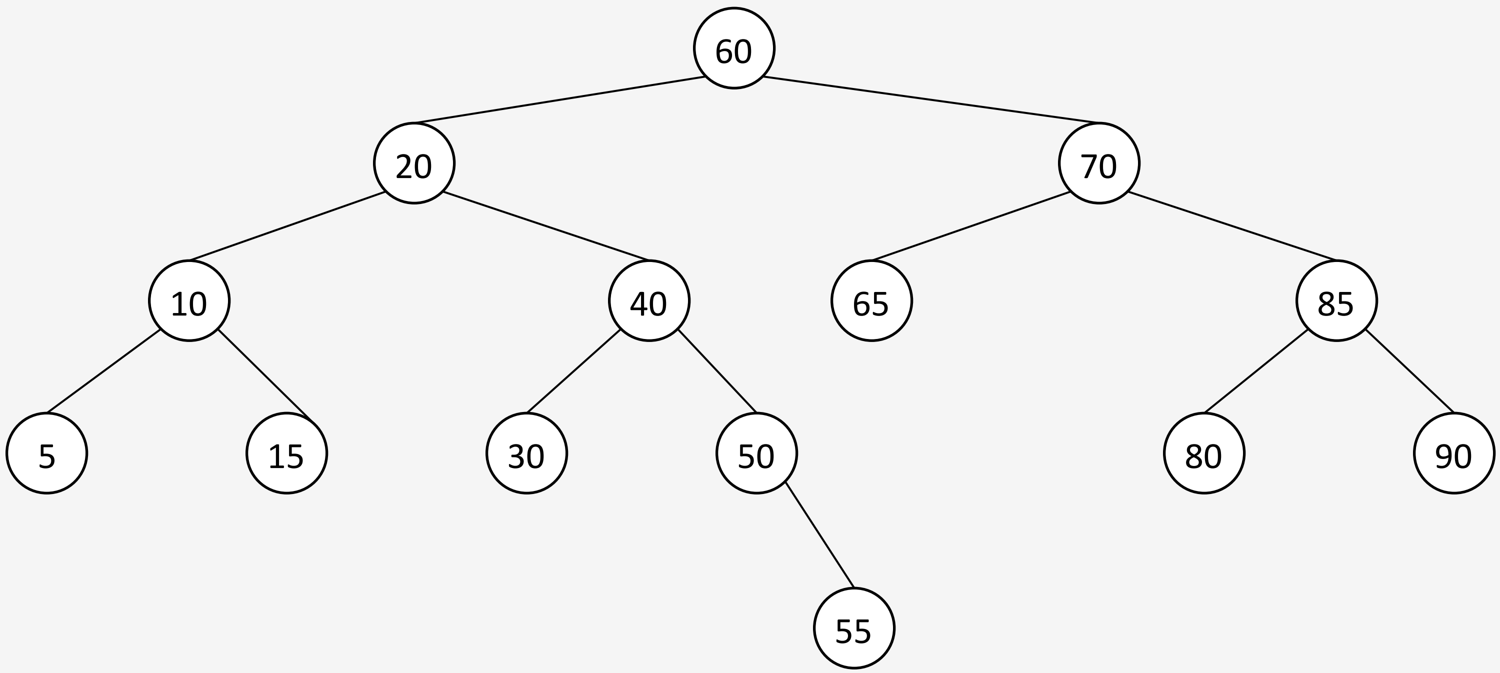
Stop at the first (lowest) node that has a balance factor of ±2. This node roots the 3-node neighborhood that will be rotated.



At most one rebalancing operation will be required per insertion.

Building an AVL tree

Insert: 10, 85, 15, 70, 20, 60, 30, 50, 65, 80, 90, 40, 5, 55

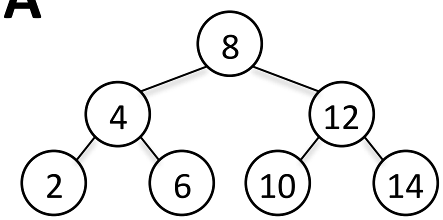


Participation question

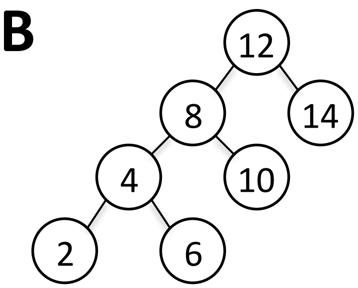
Q. Which AVL tree would result from inserting the following values in the order they are written?

```
14, 12, 10, 8, 6, 4, 2
```

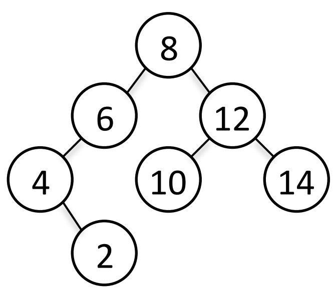
A



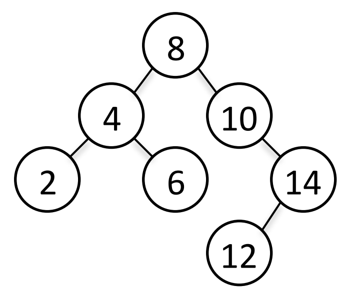
B



C

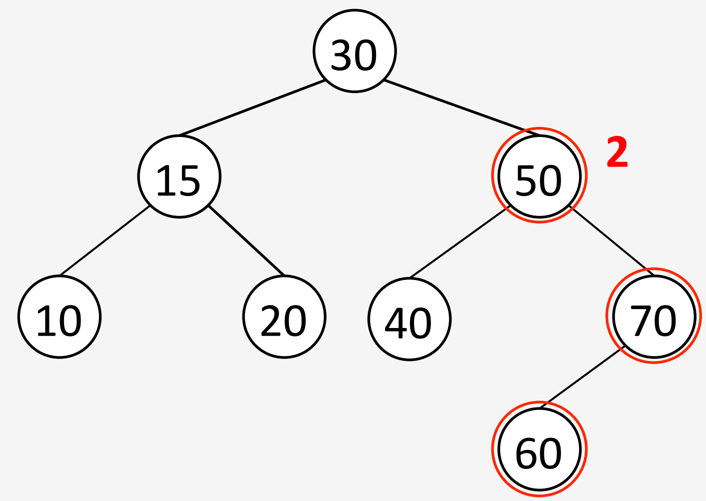


D



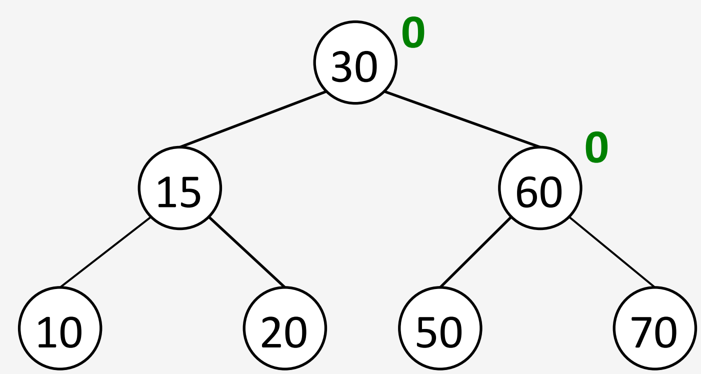
Deleting an element

Use the standard BST deletion algorithm to delete the element. Ex: 40



Beginning at the *point of deletion*, walk the reverse path back toward the root, recalculating balance factors.

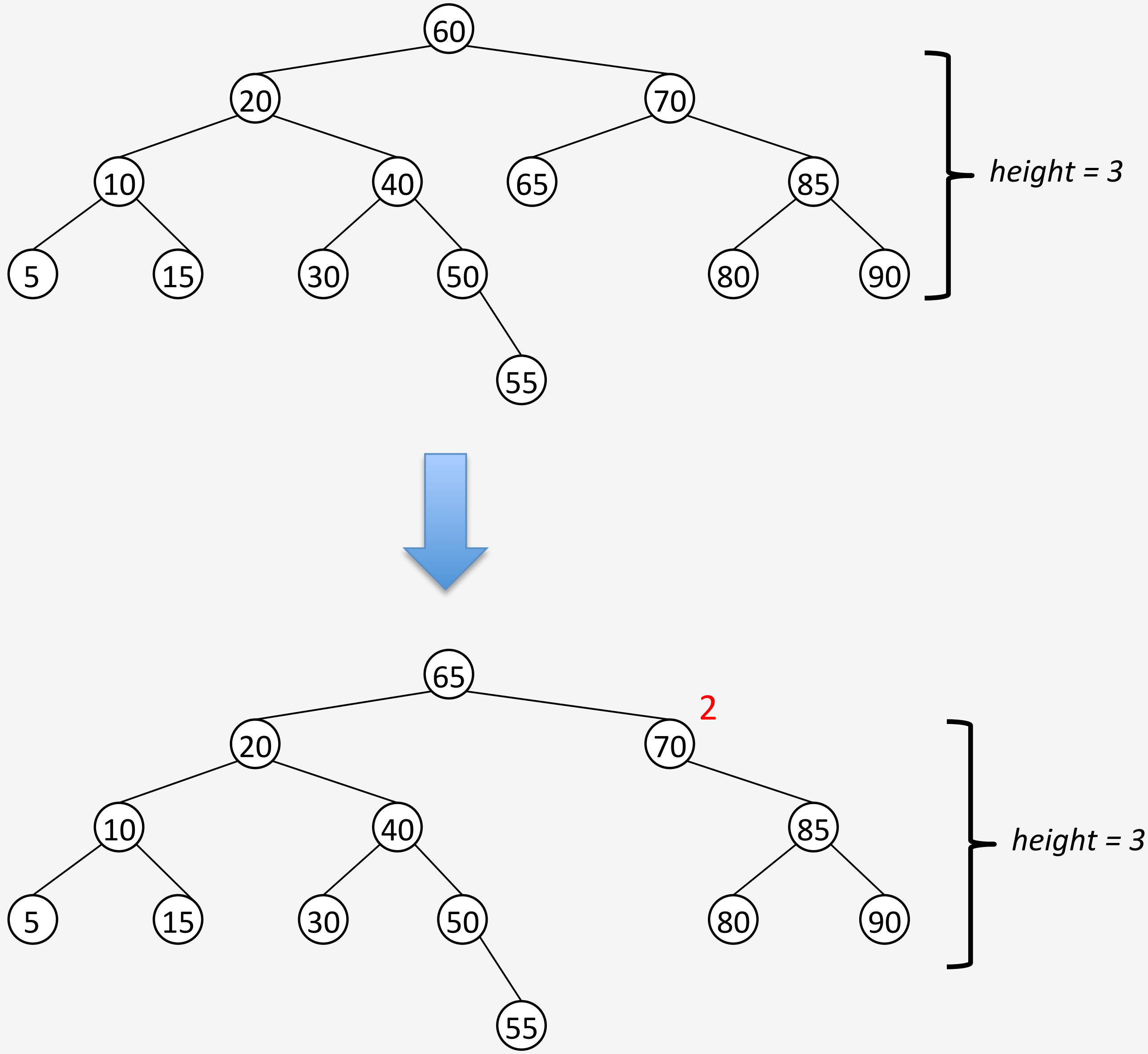
Stop at the first (lowest) node that has a balance factor of ±2. This node roots the 3-node neighborhood that will be rotated.



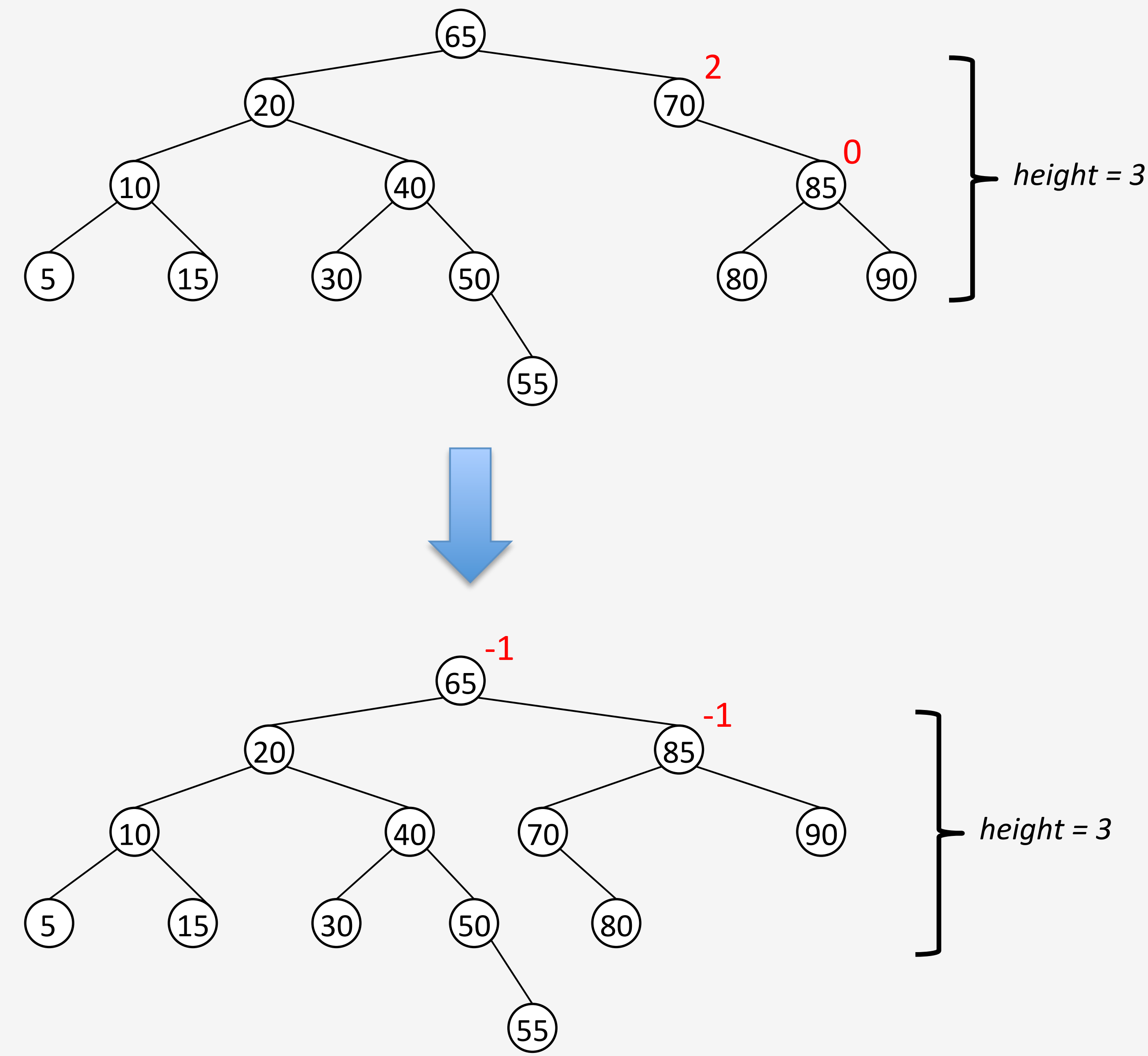
Multiple rebalancing operations may be required per deletion, so the reverse walk must go to the root each time.

Example deletion

Delete 60:
(use successor)

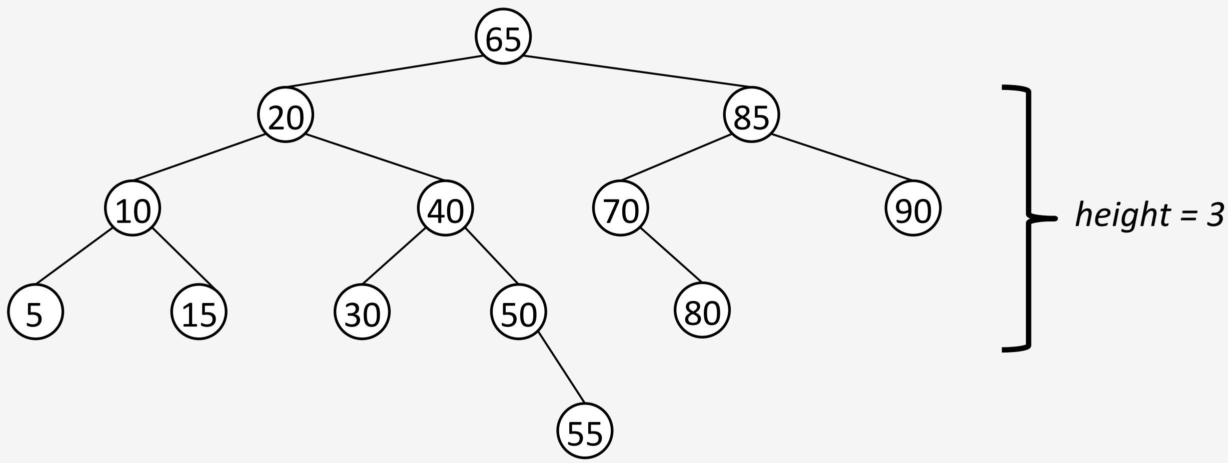


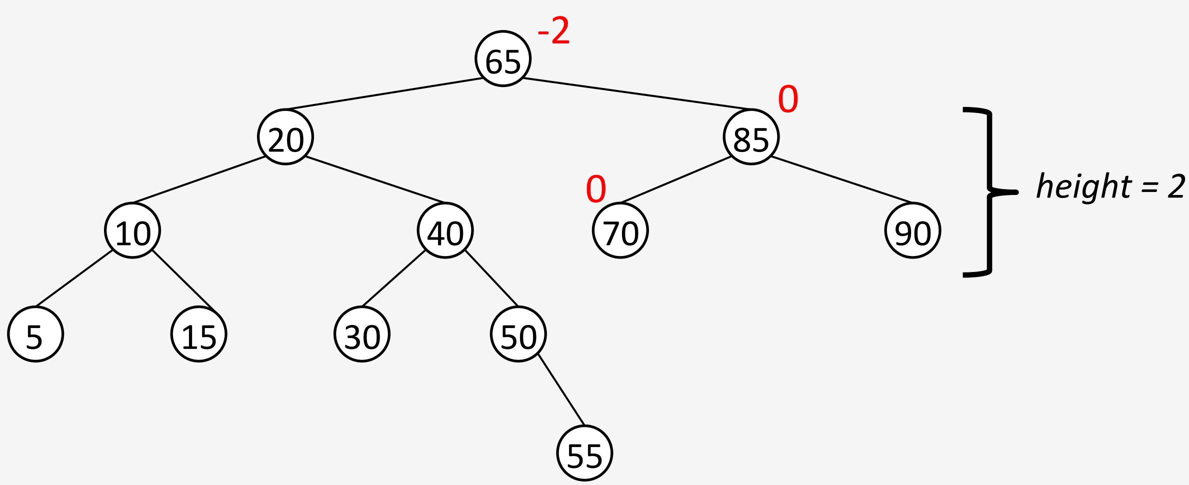
Example deletion



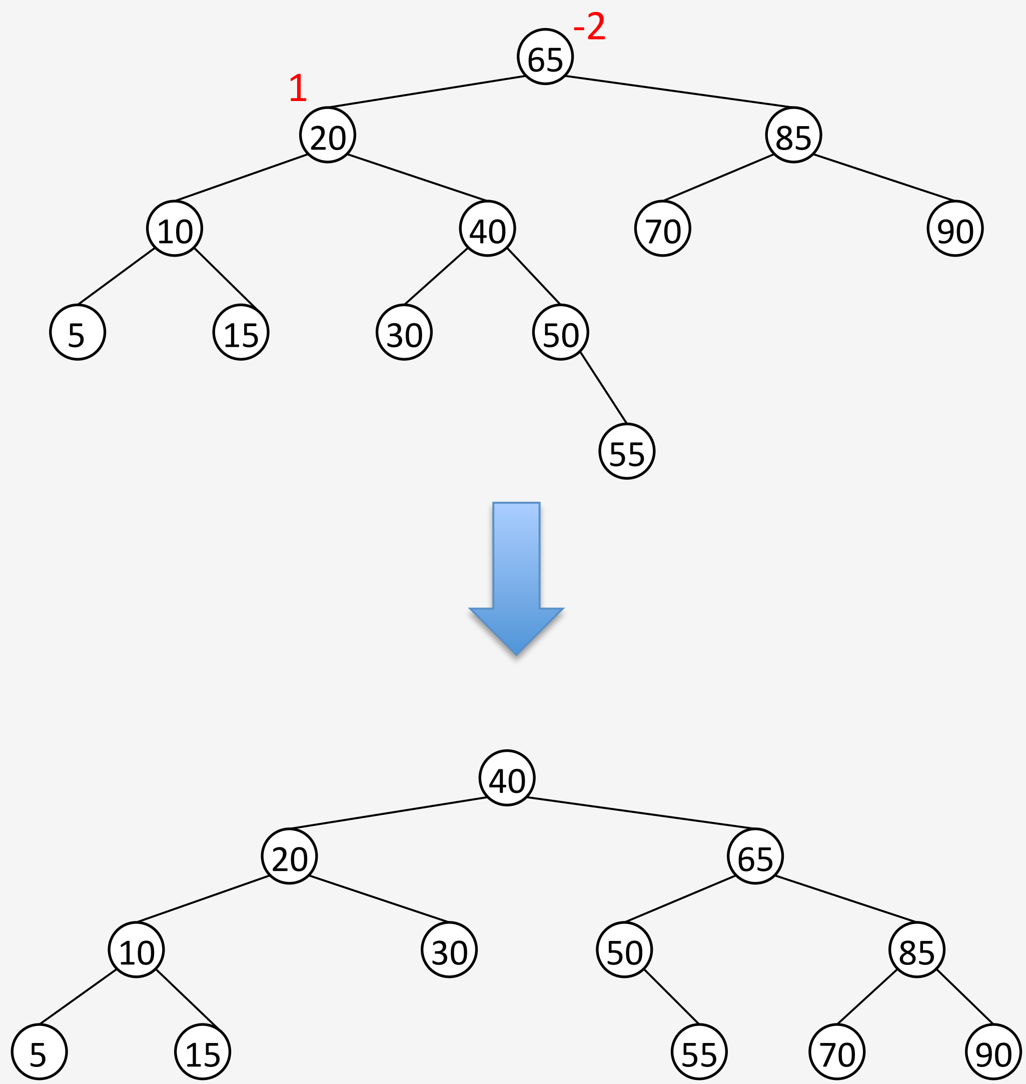
Example deletion

Delete 80:



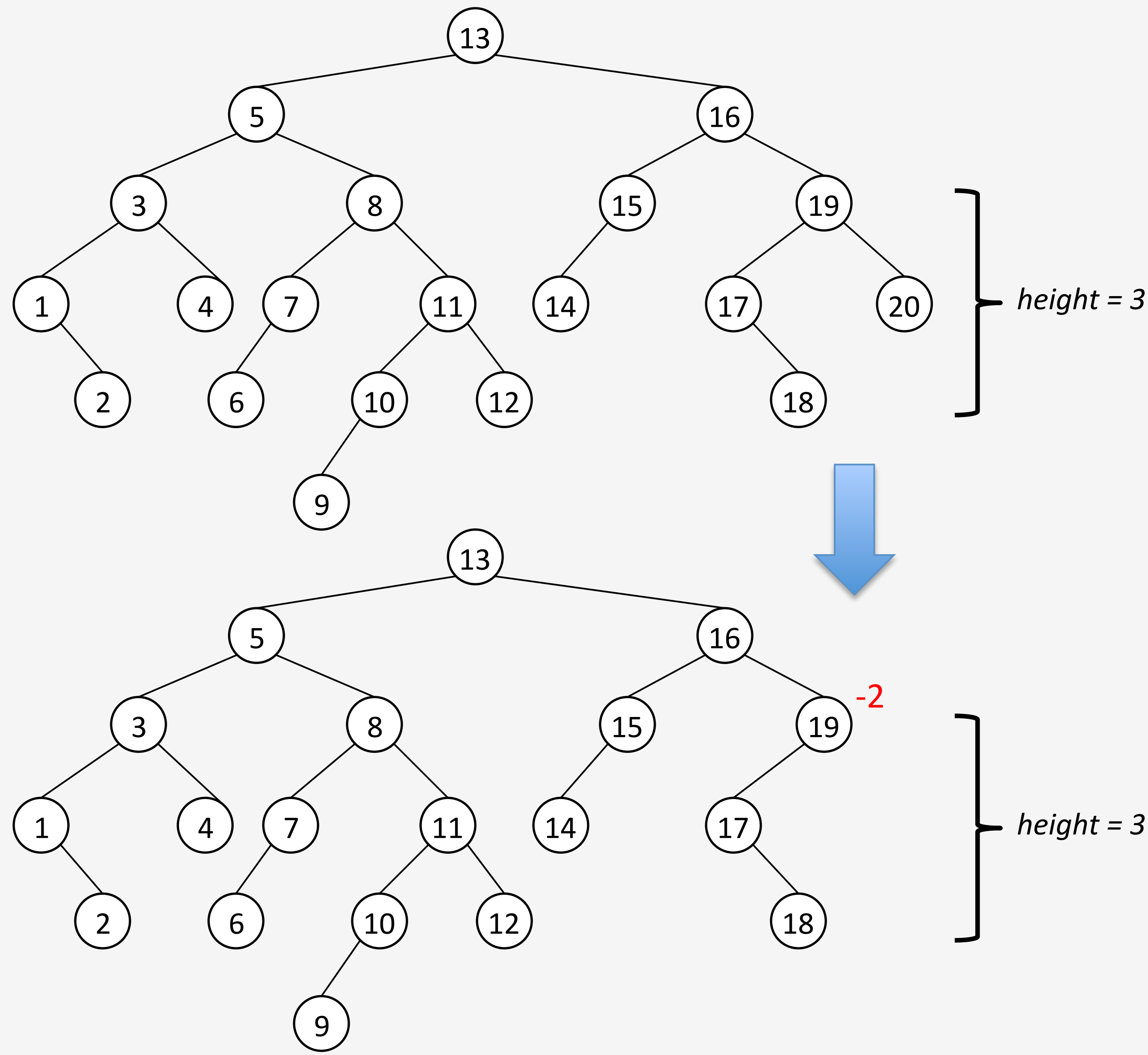


Example deletion

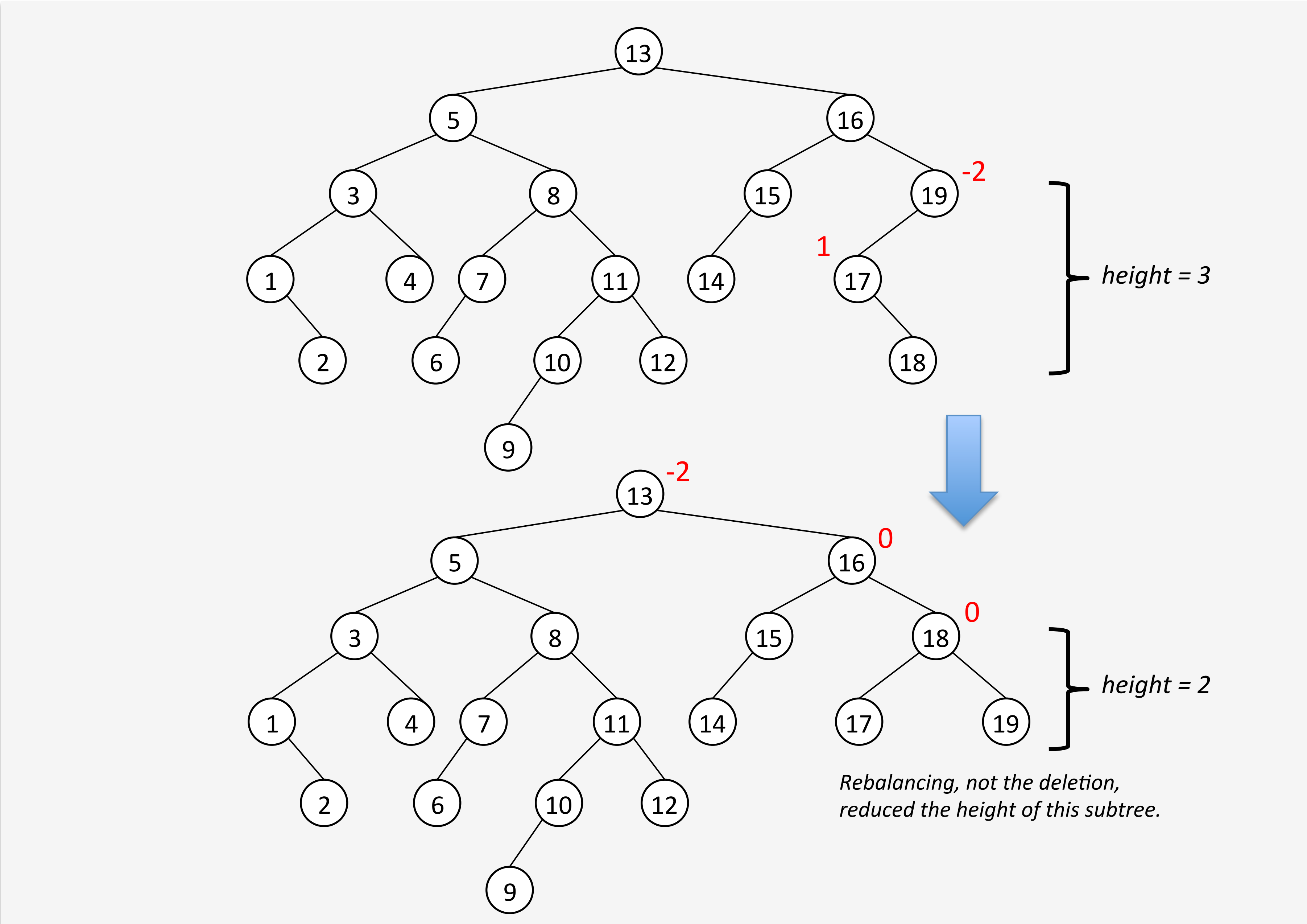


Example deletion

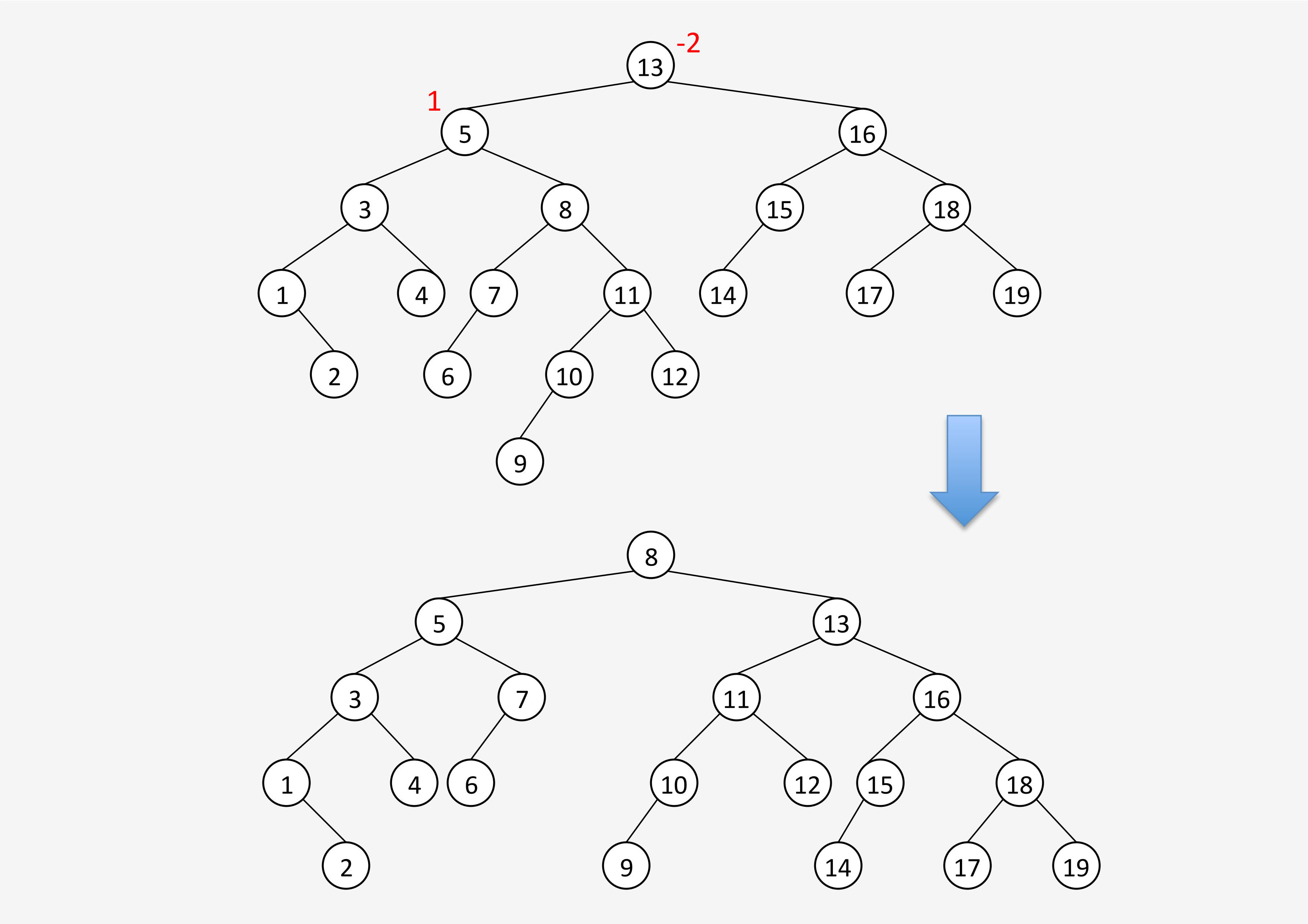
Delete 20:



Example deletion



Example deletion



Remind me: what's the point of all this?

Performance analysis of lists …

Performance analysis

method	Indexed List		Non-indexed List		Self-ordered List	
	Array	Nodes	Array	Nodes	Array	Nodes
remove(element)	O(N)	O(N)	O(N)	O(N)	O(N)	O(N)
addAfter(element, target)	•	•	O(N)	O(N)	•	•
add(element)	O(1)	O(1)	•	•	O(N)	O(N)
add(index, element)	O(N)	O(N)	•	•	•	•
get(index)	O(1)	O(N)	•	•	•	•
indexOf(element)	O(N)	O(N)	•	•	•	•

Tell me why … **Tell me how…**

If we could use binary search on a node-based structure, then add() and remove() would be O(log N) – a huge improvement!

Stay tuned … this is exactly where we're headed.

COMP 2210 • Dr. Hendrix • 22

Balanced binary search trees are like a structural implementation of the binary search algorithm.

So, now we can use binary search on a structure built with linked nodes.

AVL trees offer guaranteed O(log N) performance on all three major collection operations: add, remove, and search.

	Self-Ordered Lists		
	Array	Linked List	AVL Tree
add(element)	O(N)	O(N)	O(log N)
remove(element)	O(N)	O(N)	O(log N)
search(element)	O(log N)	O(N)	O(log N)