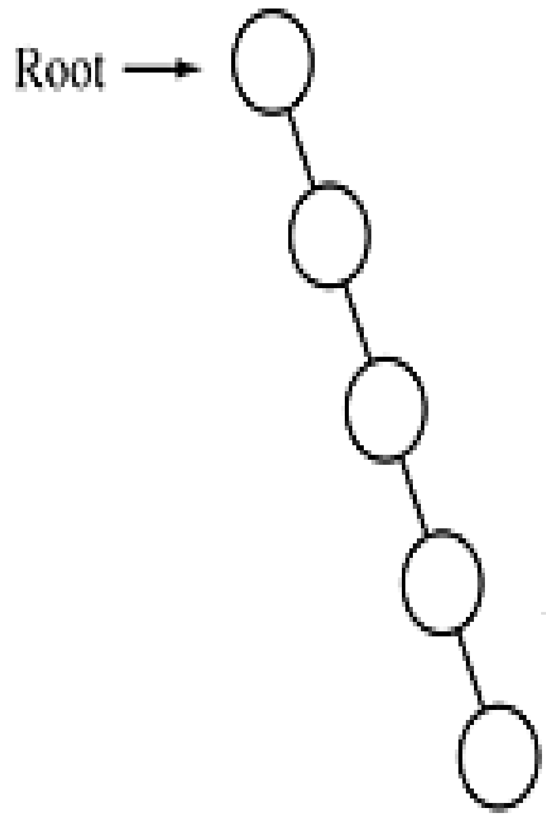
AVL Tree

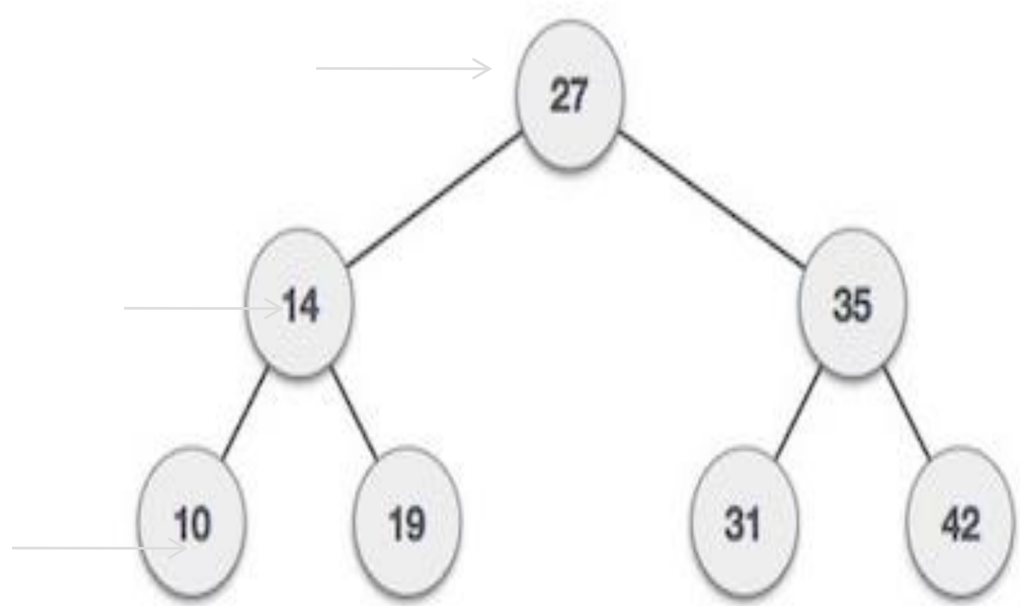
Problem with Binary search tree

- The disadvantage of a binary search tree is that its height can be as large as N-1
- This means that the time needed to perform insertion and deletion and many other operations can be N times in the worst case.
- We want a tree with small height.
- A binary tree with N node has height at least $\log_2 (N+1)-1$
- Thus, our goal is to keep the height of a binary search tree $\log_2 (N)$ times
- Such trees are called balanced binary search trees. Examples are AVL tree, red-black tree.

Binary search tree



Unbalanced Tree



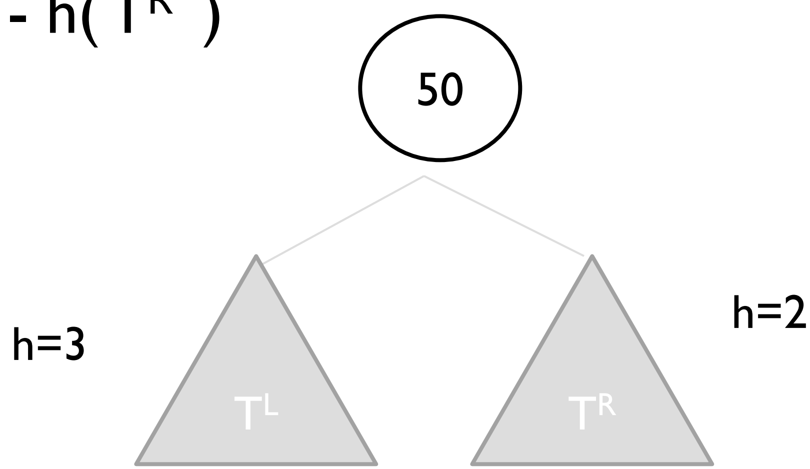
balanced Tree

AVL

- There is a need to maintain the binary search tree to be of balanced height, so that it is possible to obtain for the search option of $\log_2 (N)$ time in the worst case.
- One of the most popular balanced tree was introduced by Adelson velskii and landis (AVL)

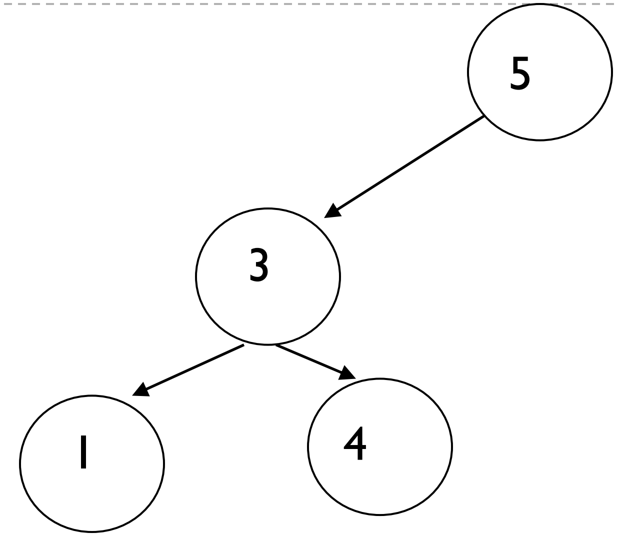
AVL Tree

- An empty binary search tree is an AVL tree
- A non empty binary tree T is an AVL tree iff T^L (left subtree) and T^R (right subtree) of T and $h(T^L)$ (height of left subtree) and $h(T^R)$ (height of right subtree) where $| h(T^L) - h(T^R) | <= 1$
- Balance factor BF is difference between $h(T^L)$ and $h(T^R)$ and the value will be -1, 0 or +1.
- $BF = h(T^L) - h(T^R)$

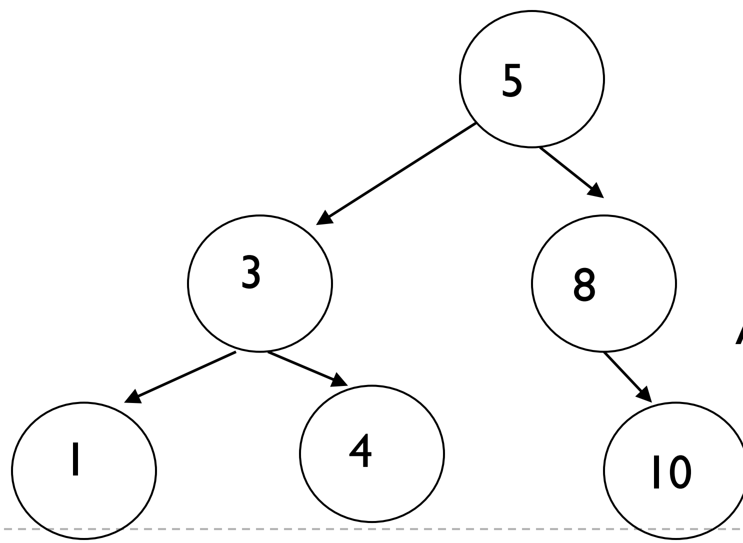


BF = 3-2 =1

AVL Tree



Not AVL Tree

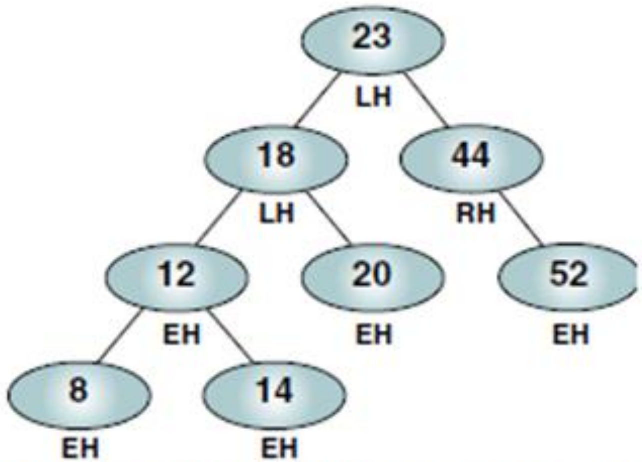


AVL Tree

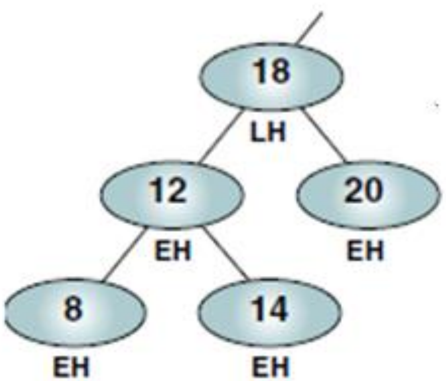
AVL Tree- Balance Factor

- BF = $h(T^L) - h(T^R)$
- LH for left high (+1) to indicate that the left subtree is higher than the right subtree
- EH for even high (0) to indicate that the subtrees are the same height
- RH for right high (–1) to indicate that the left subtree is shorter than the right subtree.

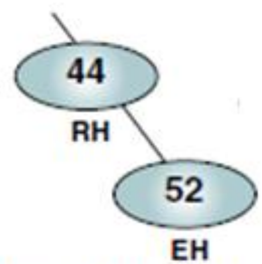
AVL Tree- Balance Factor



(a) Tree 23 appears balanced: $H_L - H_R = 1$



(b) Subtree 18 appears balanced: $H_L - H_R = 1$

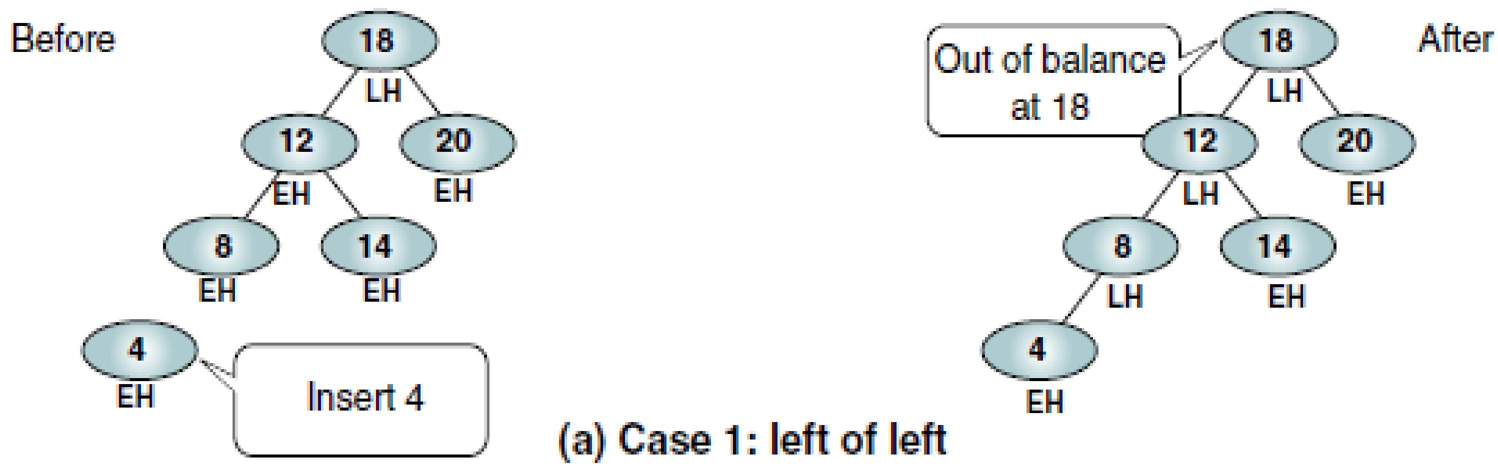


(c) Subtree 44 is balanced: $| H_L - H_R | = 1$

Balancing Trees

- Whenever we insert or delete a node from a tree, the resulting tree may be unbalanced.
- We must rebalance it.
- AVL trees are balanced by rotating nodes either to the left or to the right.
- All unbalanced trees fall into one of these four cases:
- **Left of left**: Inserted node is in the left subtree of left subtree of node A
- **Right of rig**ht: Inserted node is in the right subtree of right subtree of node A
- **Right of left**: Inserted node is in the right subtree of left subtree of node A
- **Left of right**: Inserted node is in the left subtree of right subtree of node A

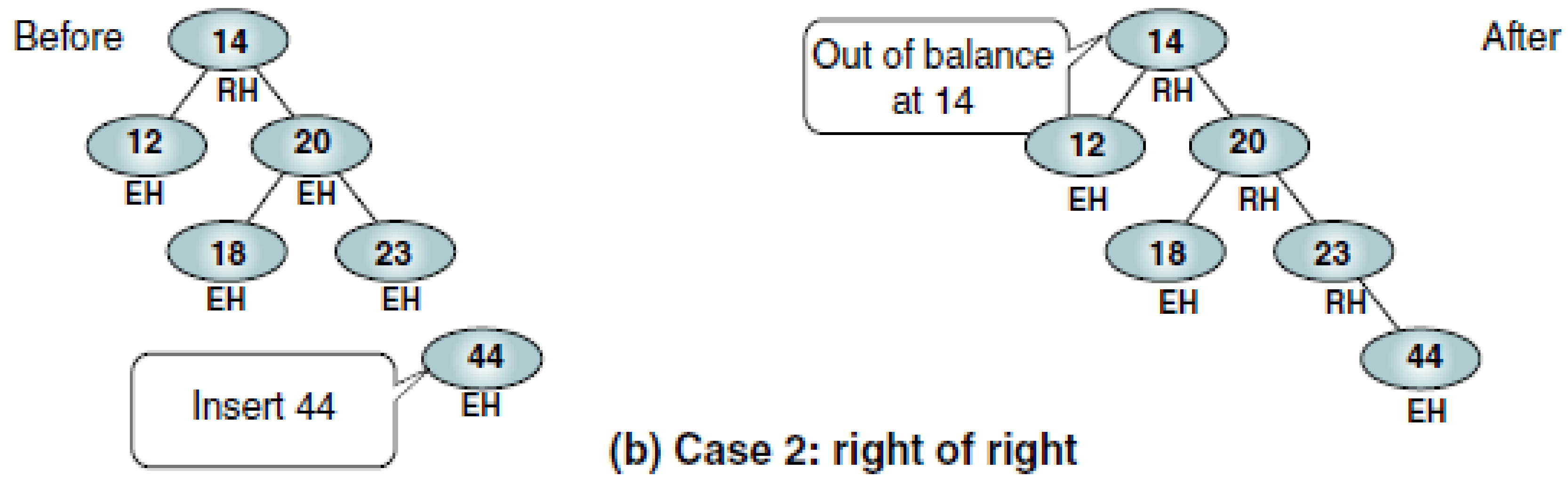
Balancing Trees



(a) Case 1: left of left

Out of balance AVL Tree

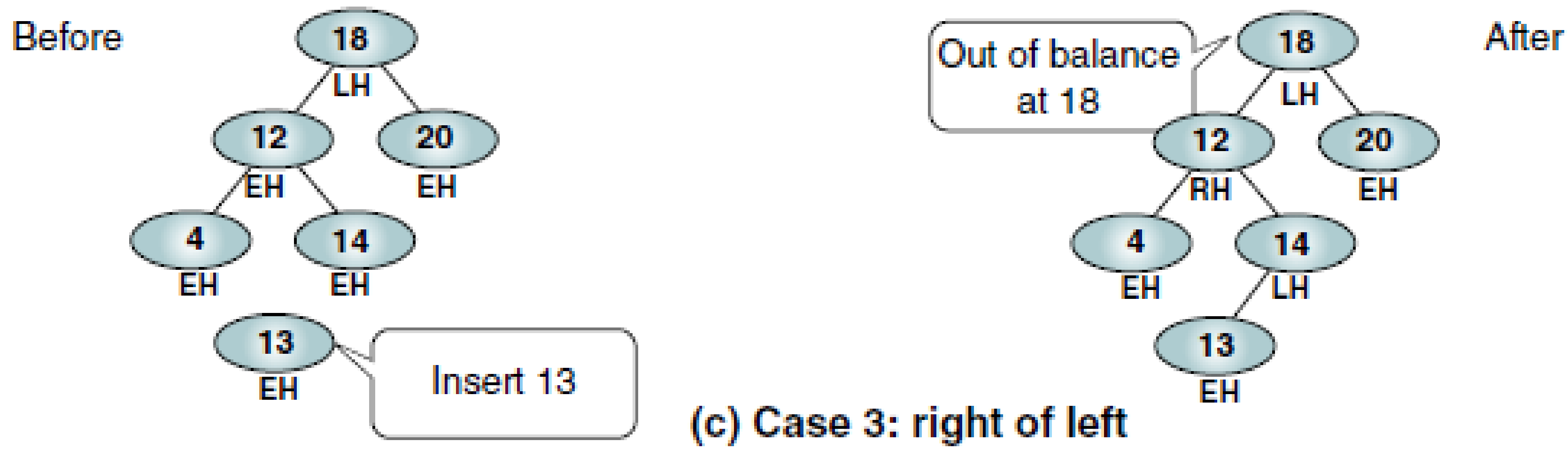
Balancing Trees



(b) Case 2: right of right

Out of balance AVL Tree

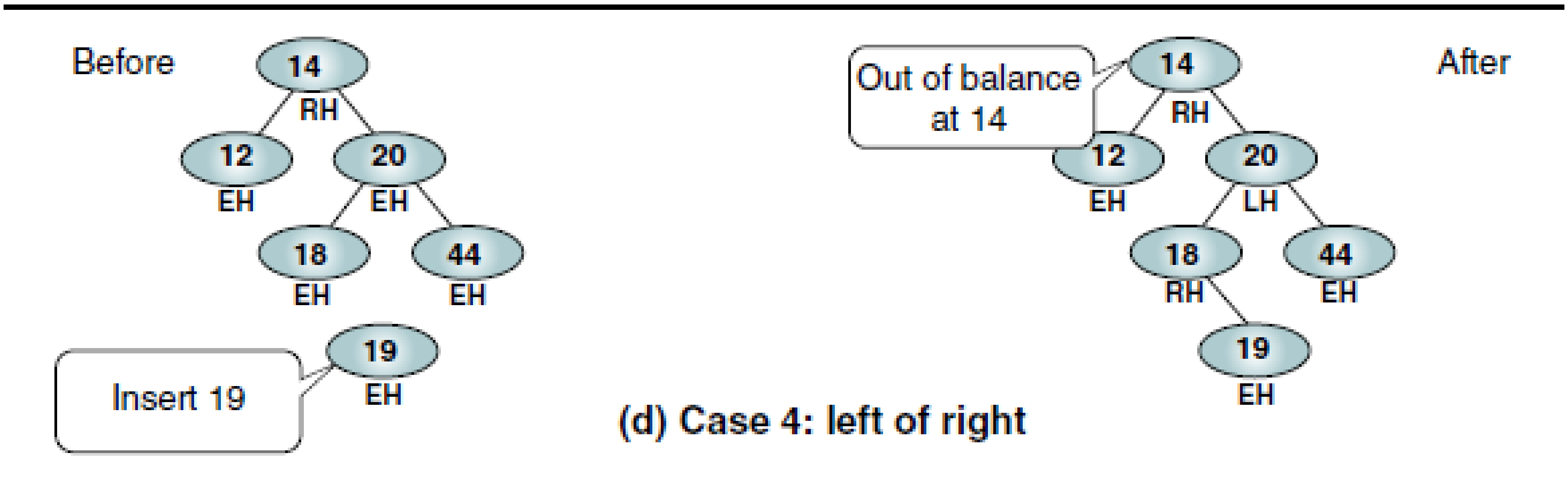
Balancing Trees



(c) Case 3: right of left

Out of balance AVL Tree

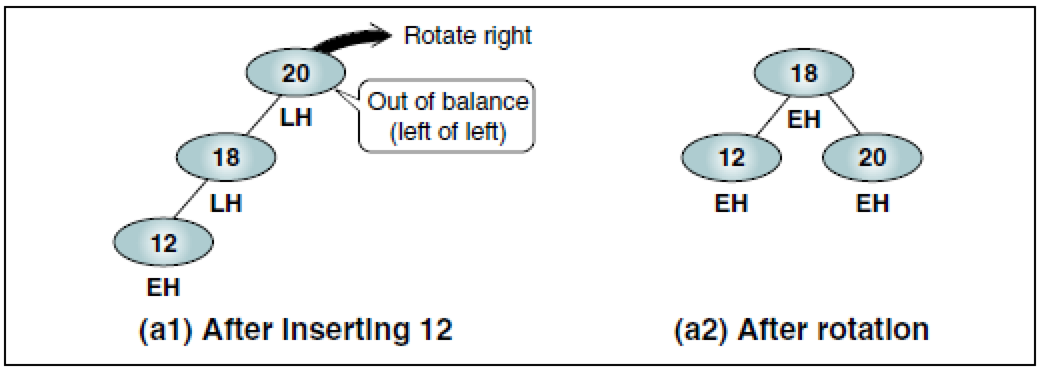
Balancing Trees



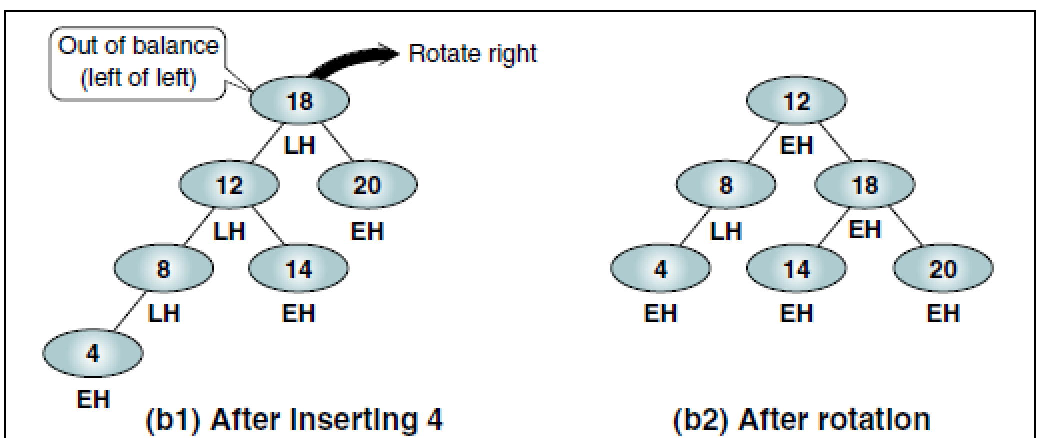
Out of balance AVL Tree

Balancing Trees- Left of Left

- When the out-of-balance condition has been created by a left high subtree of a left high tree, we must balance the tree by rotating the out-of-balance node to the right.



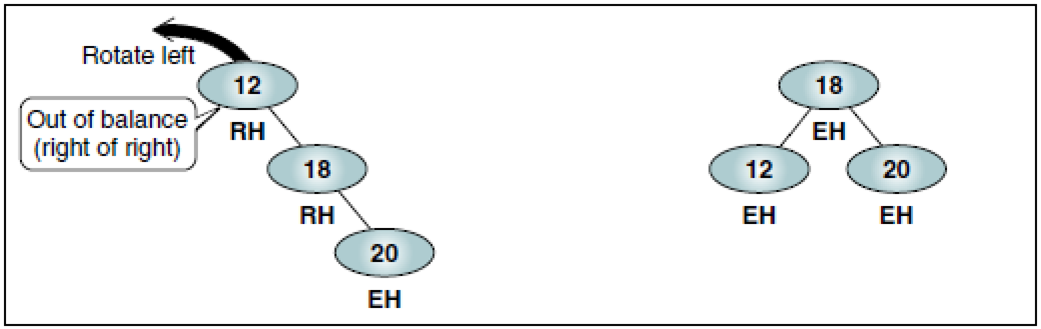
(a) Simple right rotation



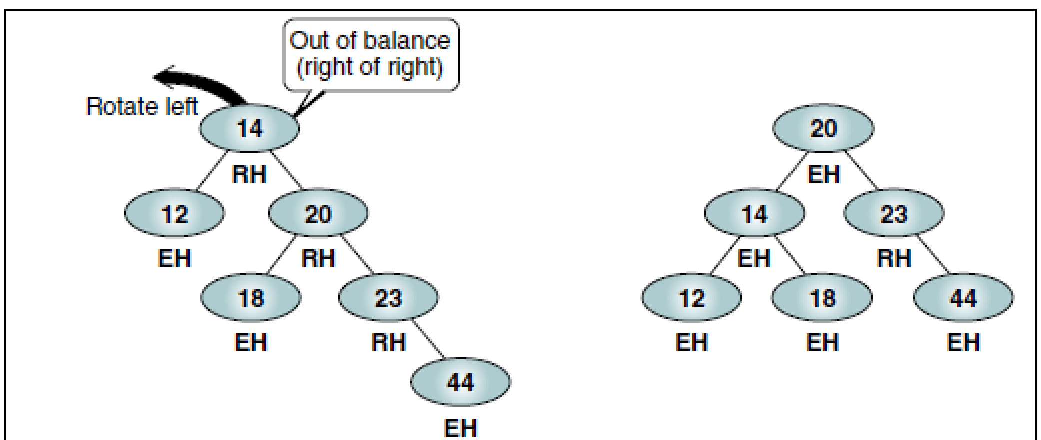
(b) Complex right rotation

Balancing Trees- Right of Right

- When the out-of-balance condition has been created by a right high subtree of a right high tree, we must balance the tree by rotating the out-of-balance node to the left.



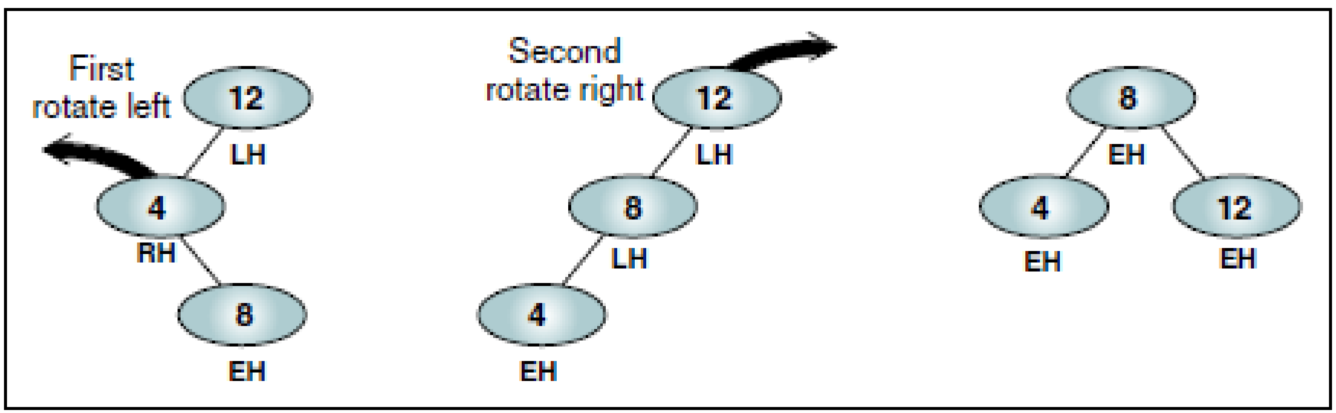
(a) Simple left rotation



(b) Complex left rotation

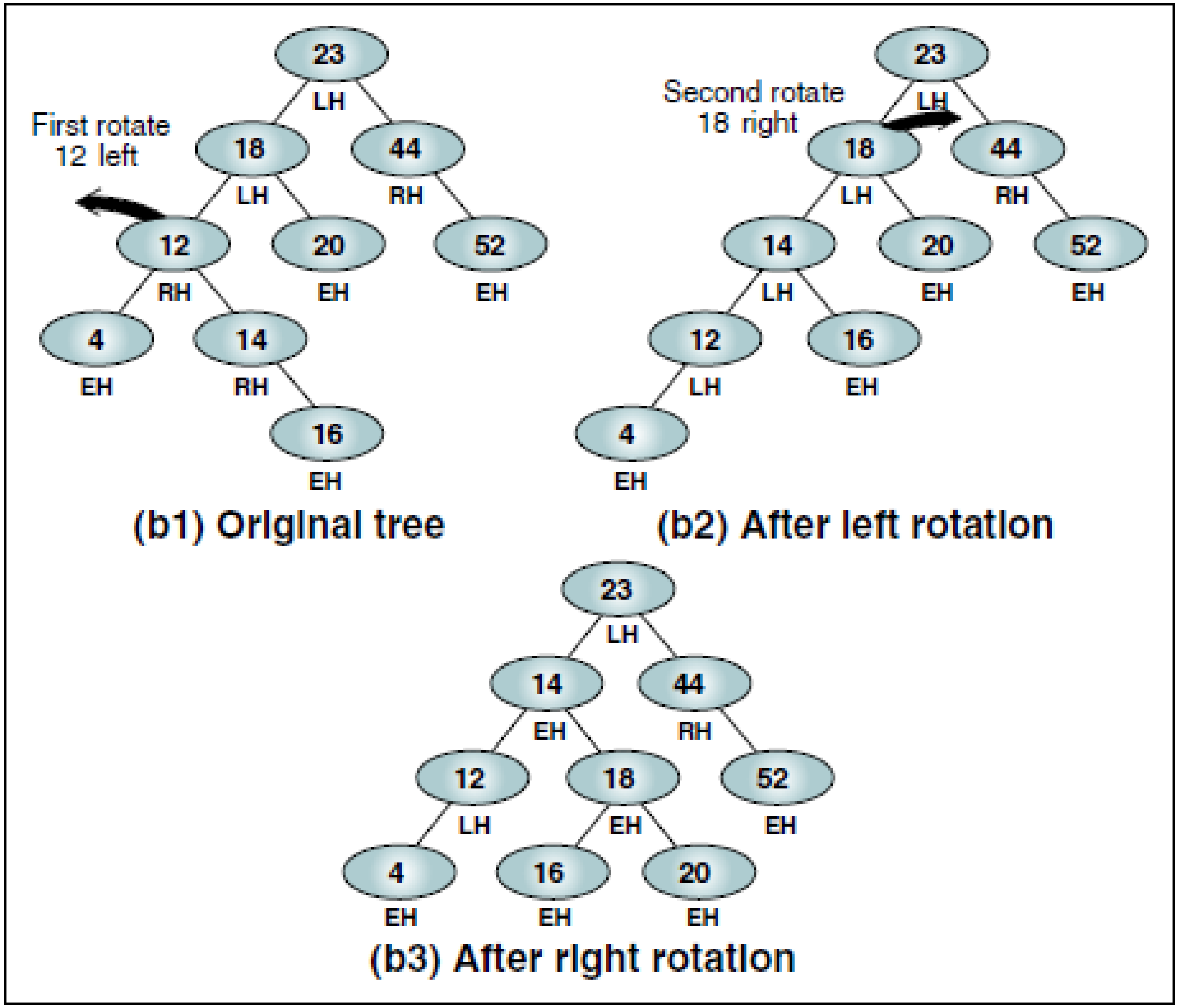
Balancing Trees- Right of Left

- When the out-of-balance condition has been created by a right high subtree of a left high tree, we must balance the tree by rotating two nodes, one to the left and one to the right, to balance the tree.



(a) Simple double rotation right

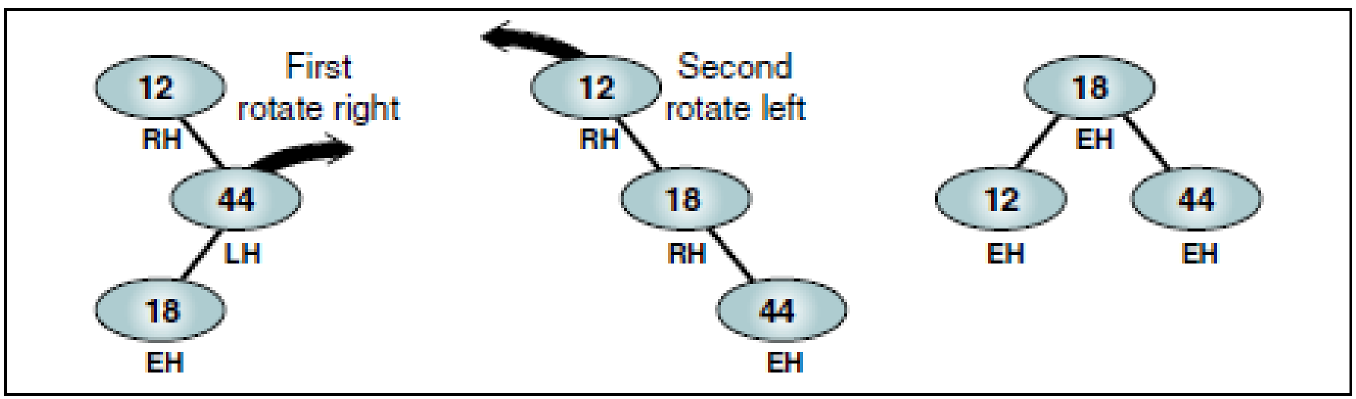
Balancing Trees- Right of Left



(b) Complex double rotation right

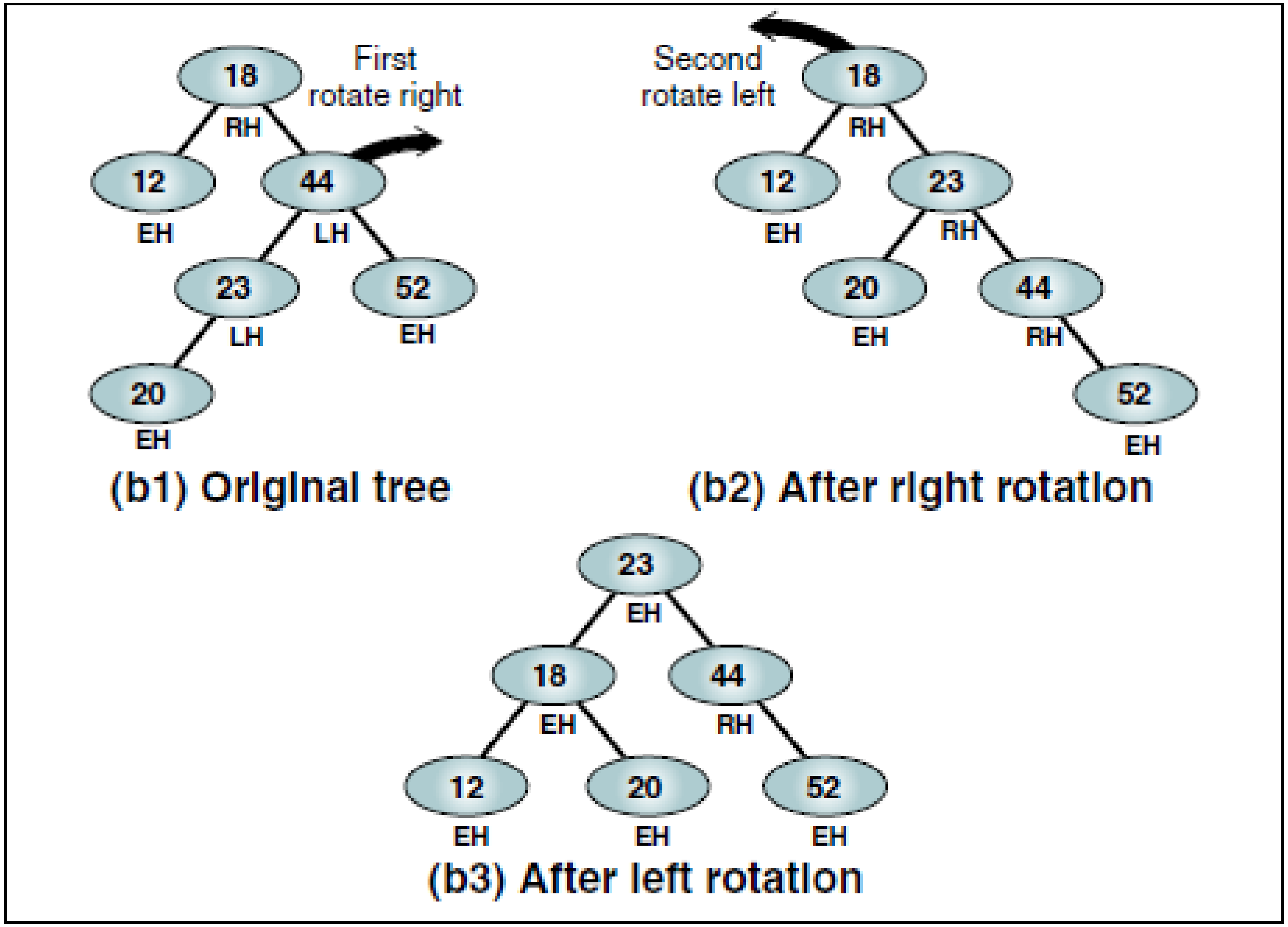
Balancing Trees- Left of Right

- When the out-of-balance condition has been created by a lefthigh subtree of a right high tree, we must balance the tree by rotating two nodes, one to the right and one to the left.



(a) Simple double rotation right

Balancing Trees- Left of Right



(b) Complex double rotation right

Extended Example

Insert 3,2,1,4,5,6,7, 16,15,14

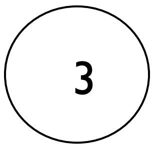


Fig 1

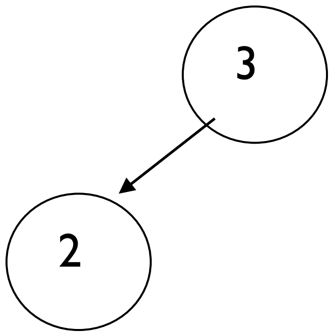


Fig 2

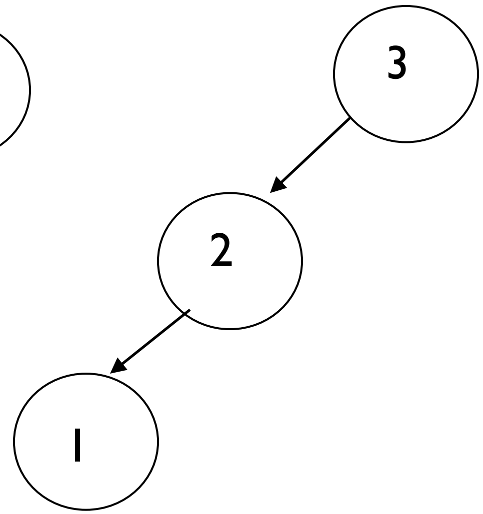


Fig 3

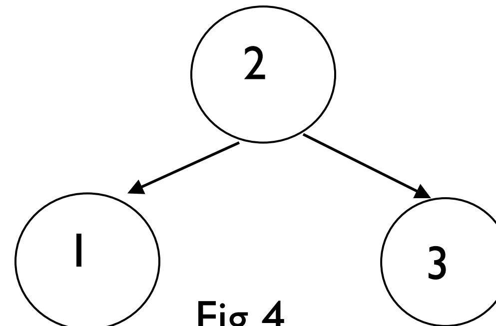


Fig 4

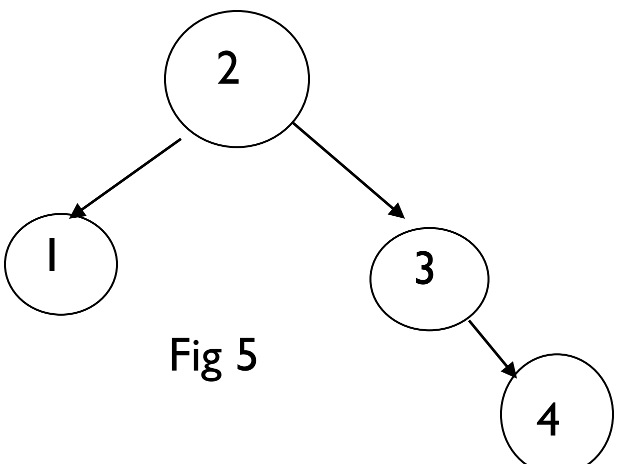


Fig 5

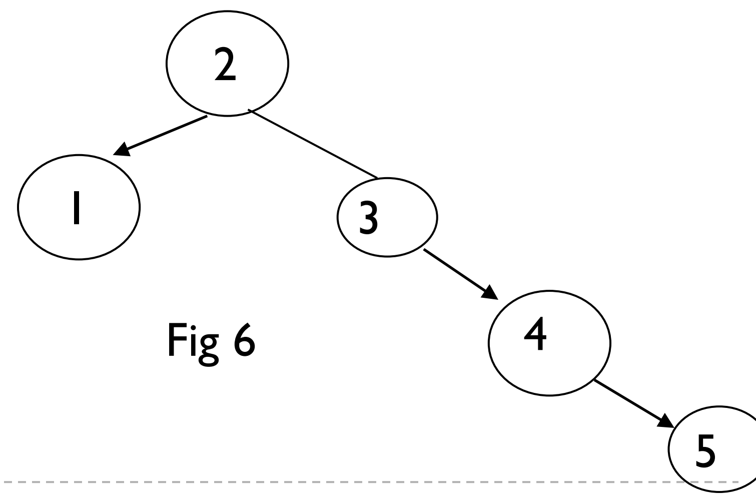


Fig 6

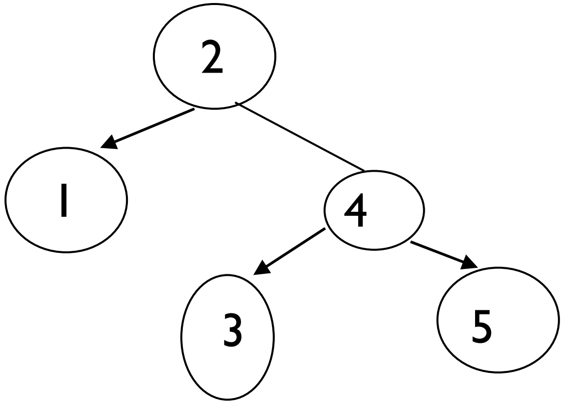


Fig 7

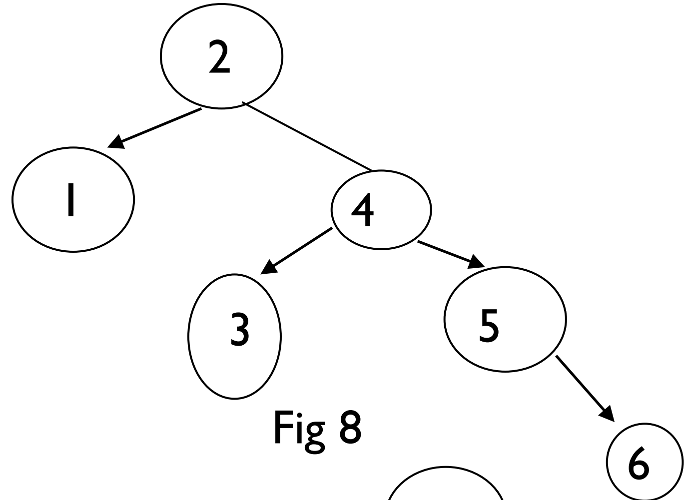


Fig 8

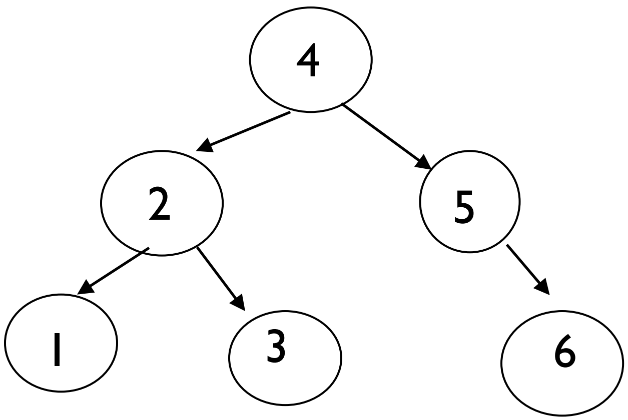


Fig 9

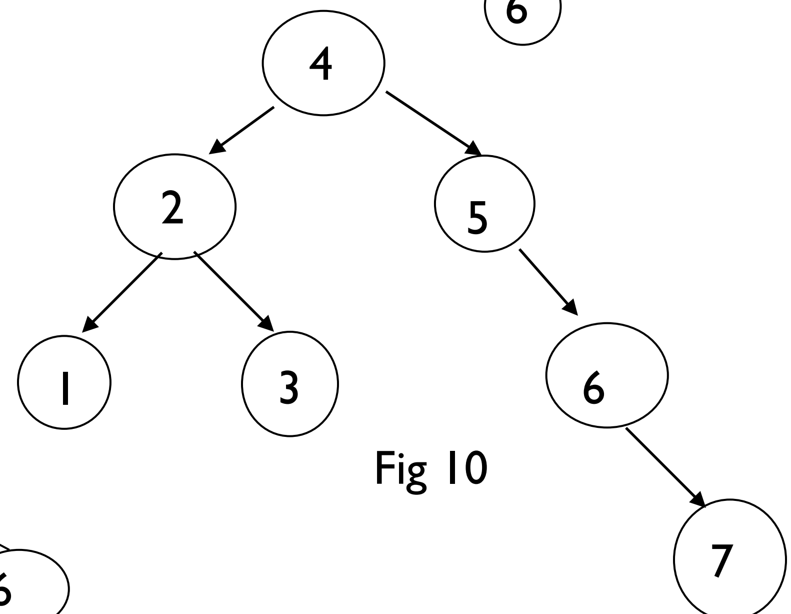


Fig 10

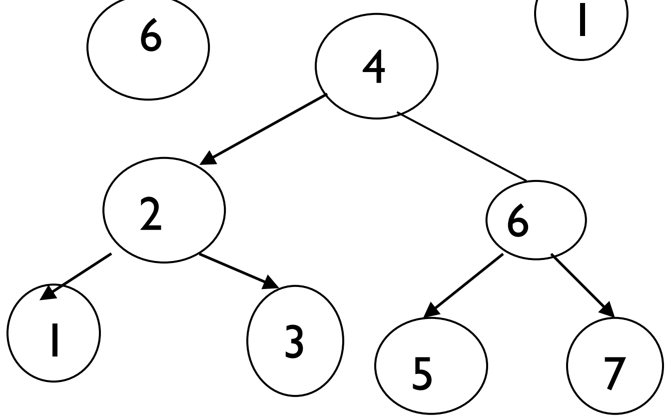


Fig 11

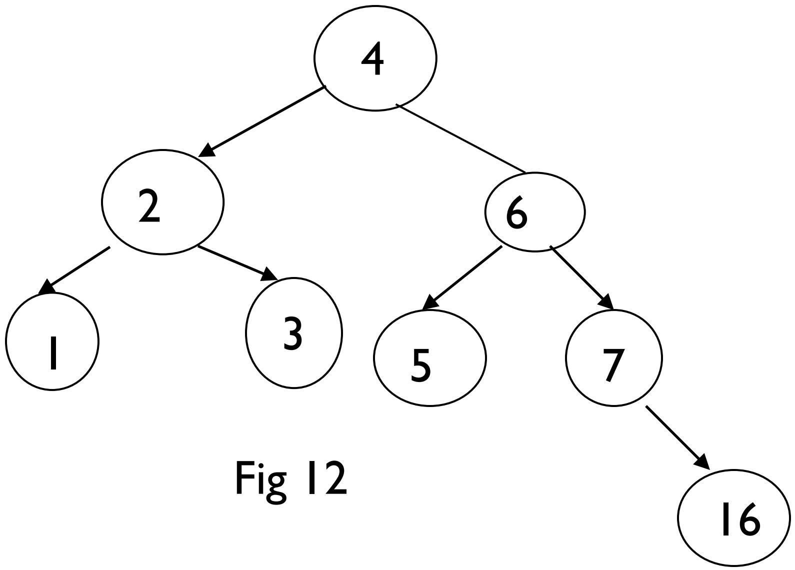


Fig 12

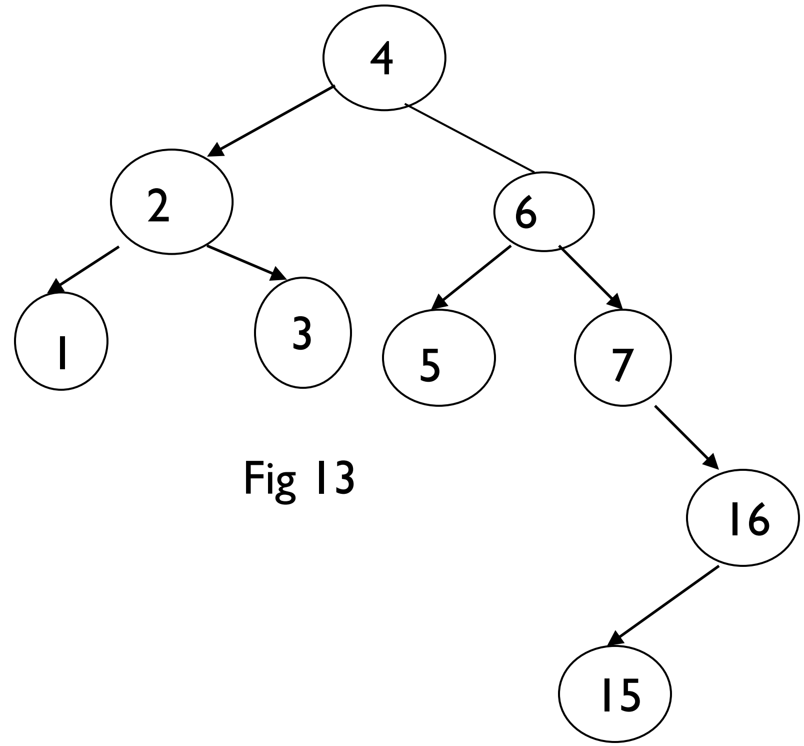


Fig 13

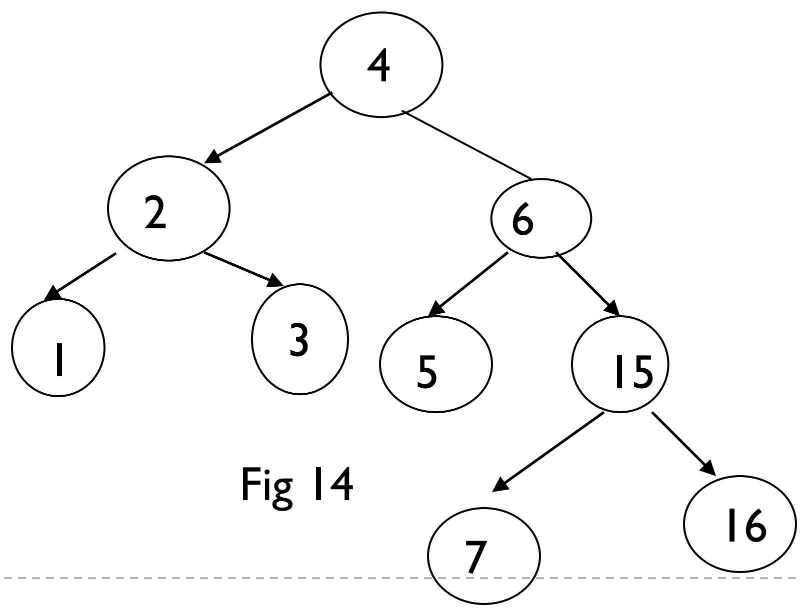


Fig 14

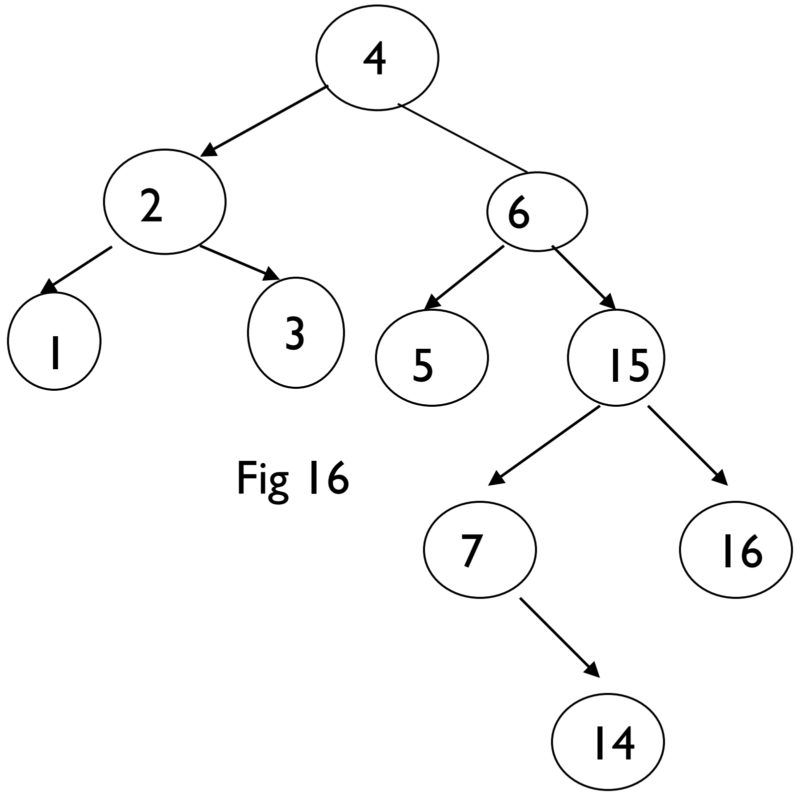


Fig 16

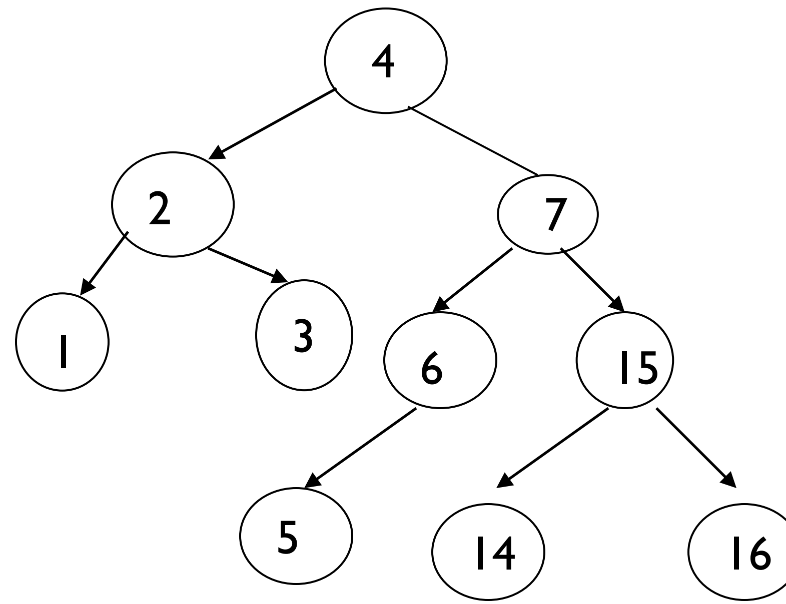


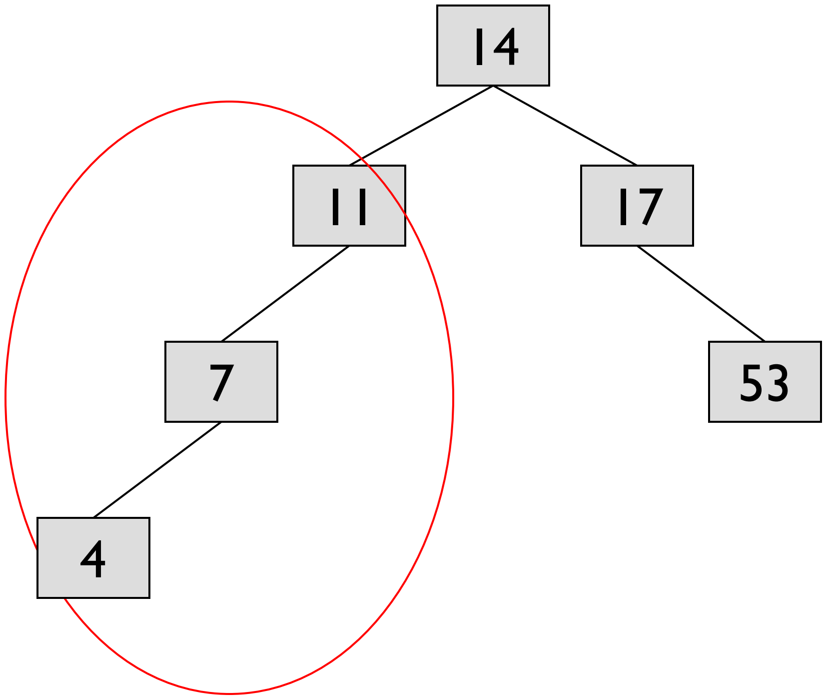
Fig 17

Deletion

- Delete by a BST deletion by copying algorithm.
- Rebalance the tree if an imbalance occurs.
- There are three deletion cases:
  1. Deletion that does not cause an imbalance.
  2. Deletion that requires a single rotation to rebalance.
  3. Deletion that requires two or more rotations to rebalance.

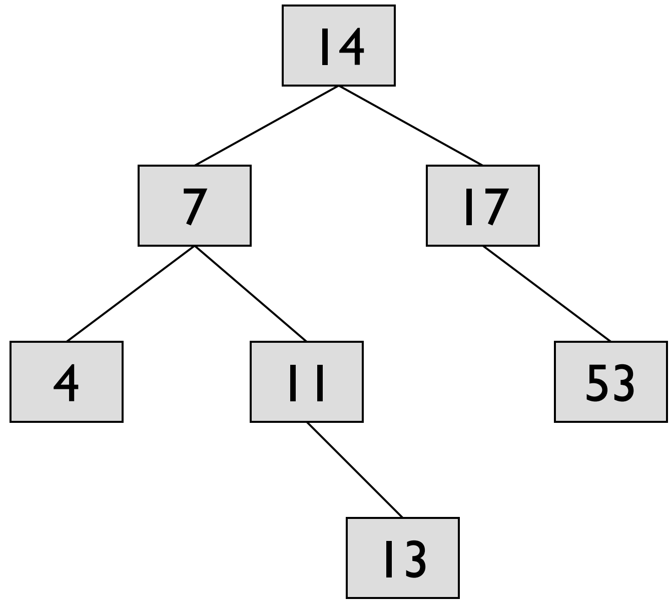
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



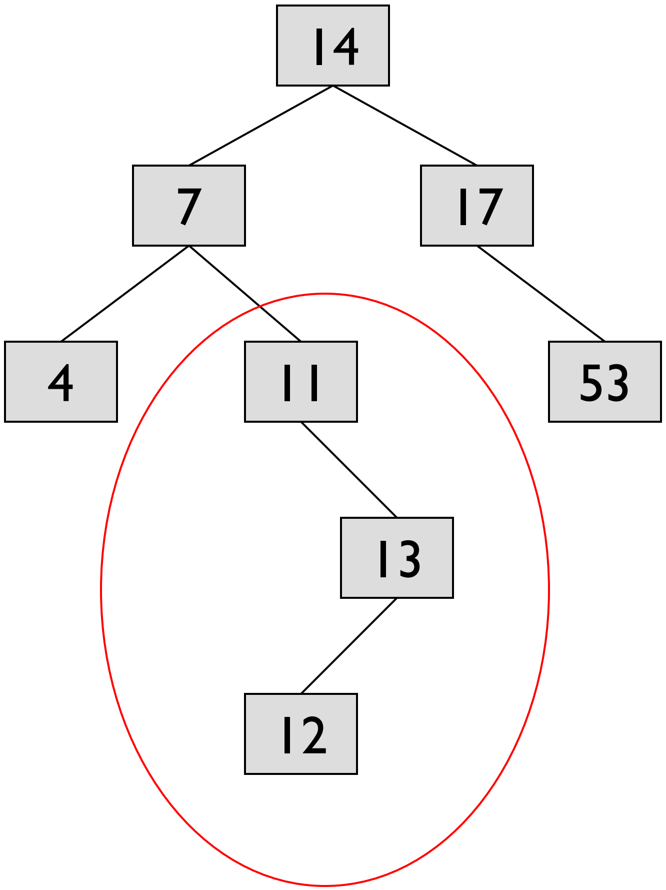
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



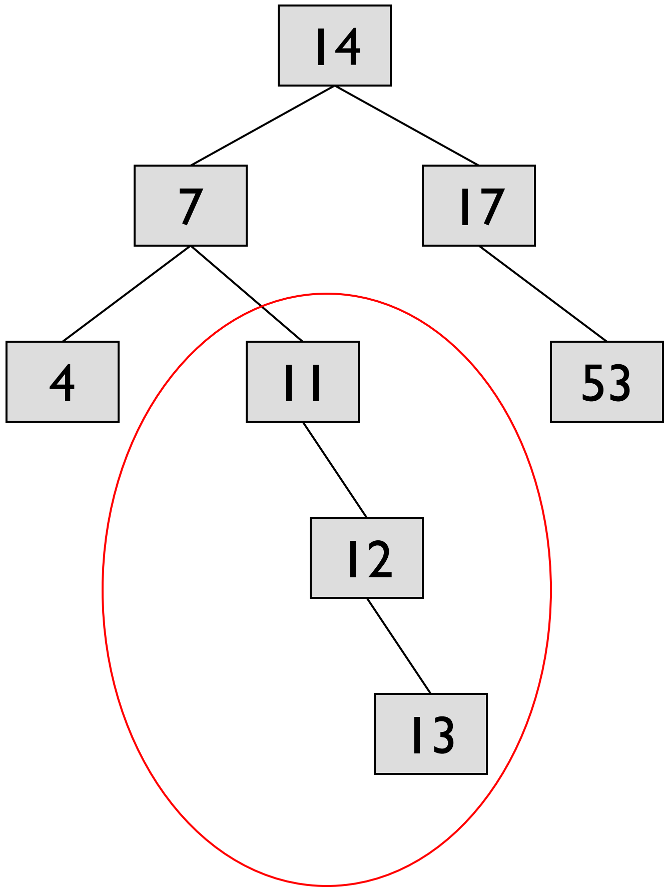
AVL Tree Example:

- **Now insert 12**



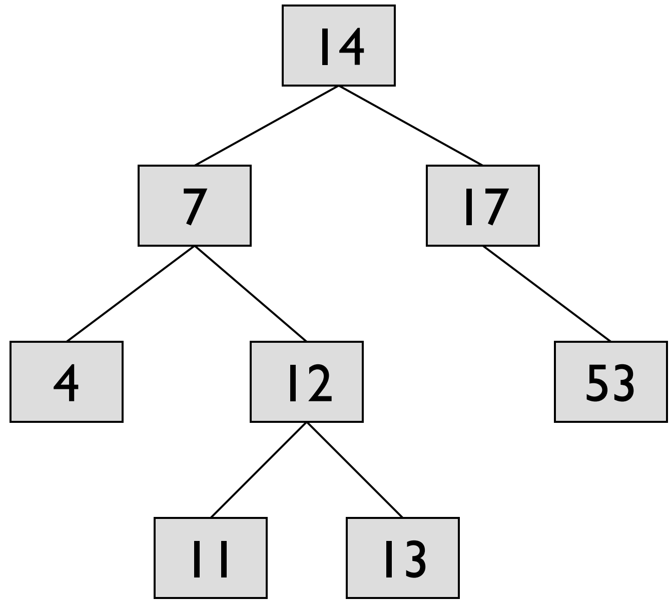
AVL Tree Example:

- **Now insert 12**



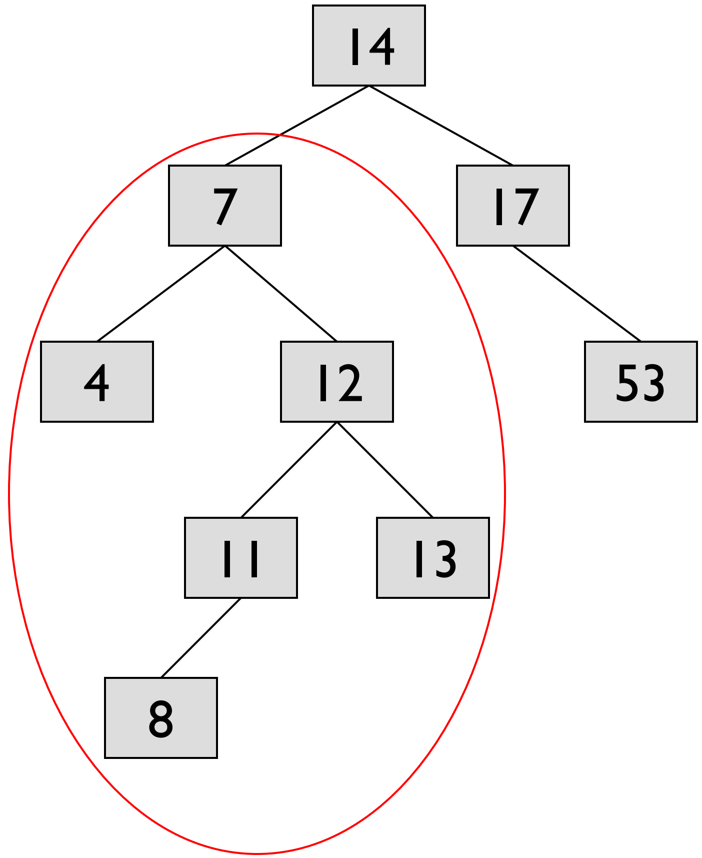
AVL Tree Example:

- **Now the AVL tree is balanced.**



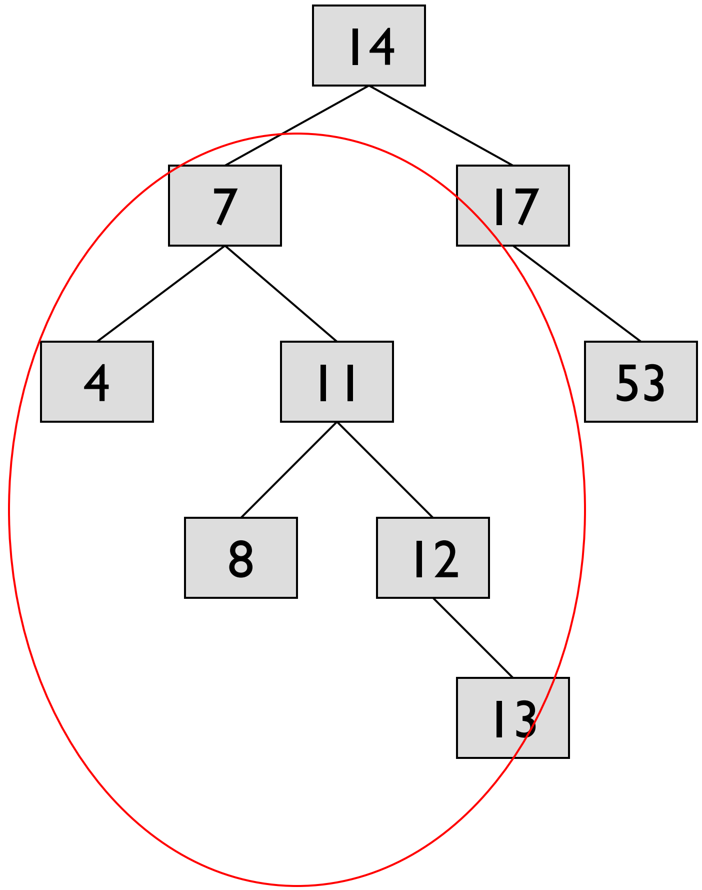
AVL Tree Example:

- **Now insert 8**



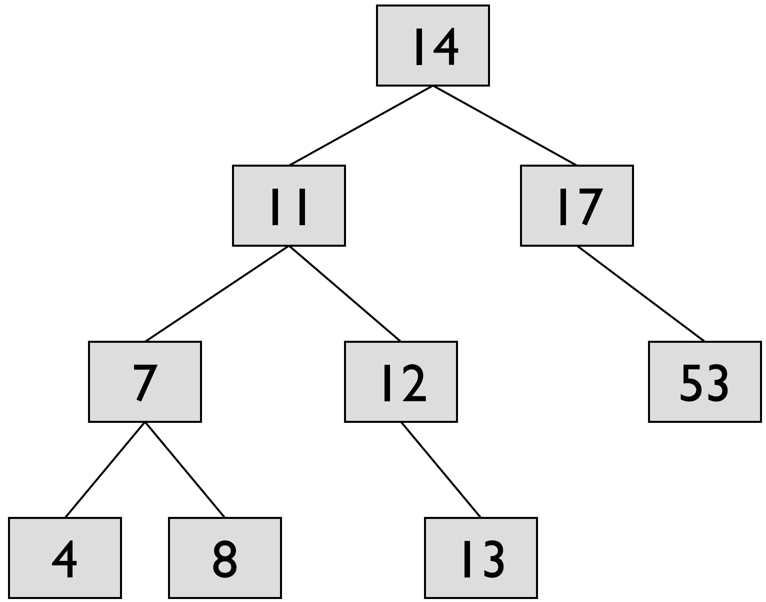
AVL Tree Example:

- **Now insert 8**



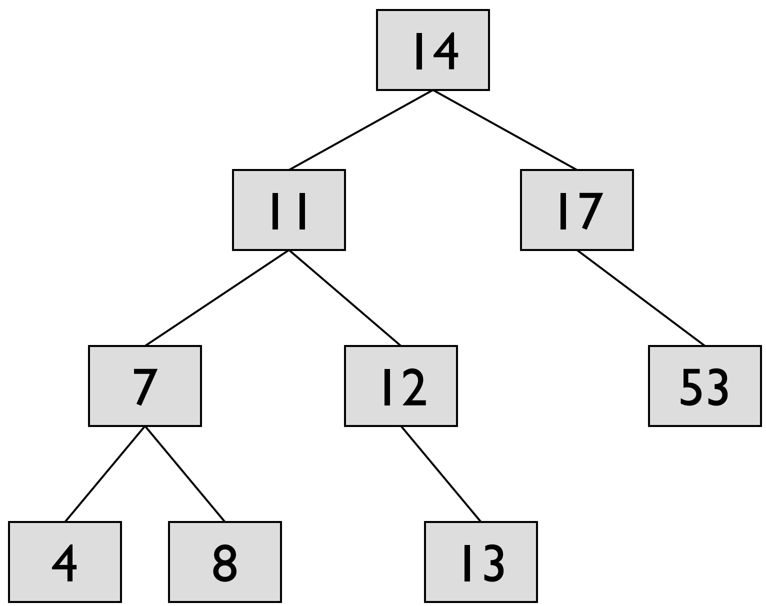
AVL Tree Example:

- **Now the AVL tree is balanced.**



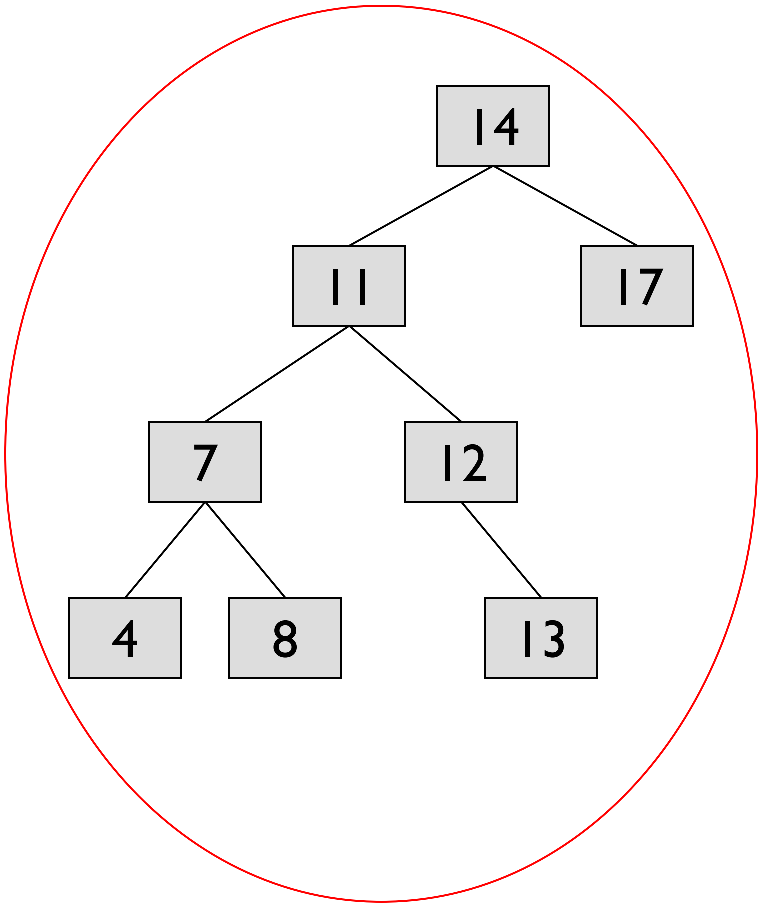
AVL Tree Example:

- **Now remove 53**



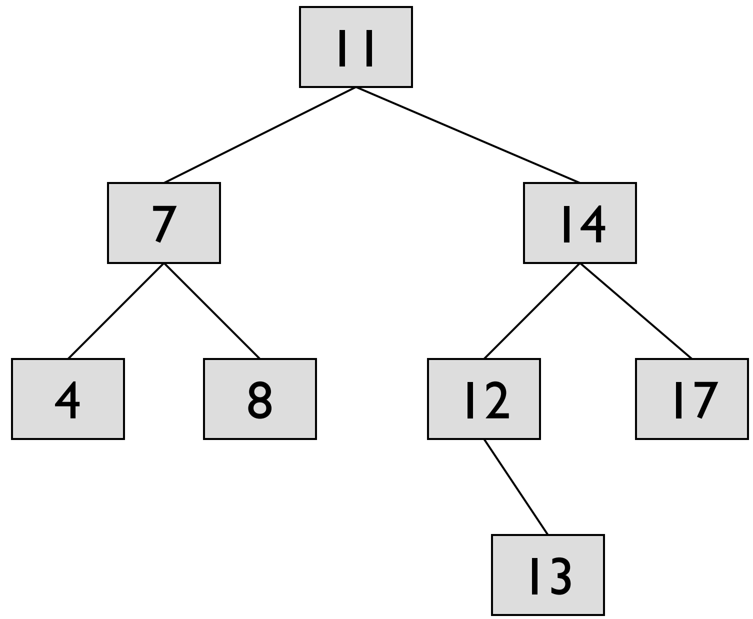
AVL Tree Example:

- **Now remove 53, unbalanced**



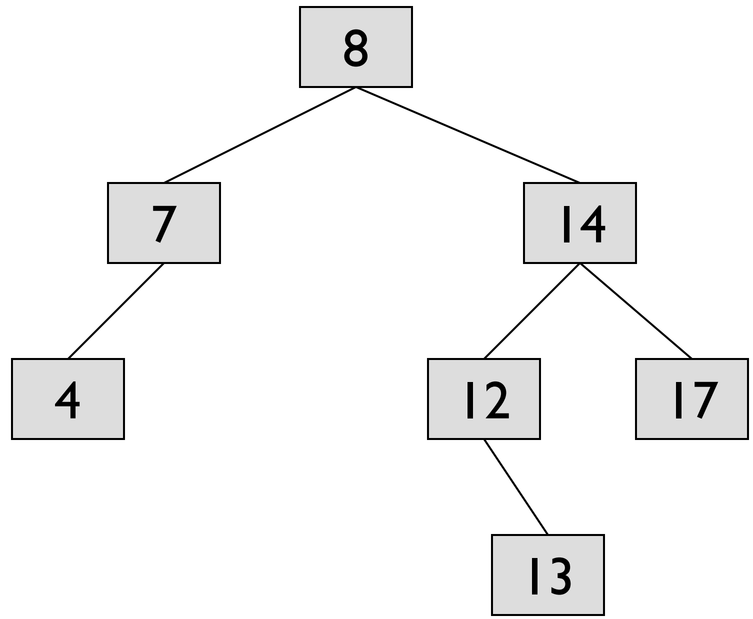
AVL Tree Example:

- **Balanced!   Remove 11**



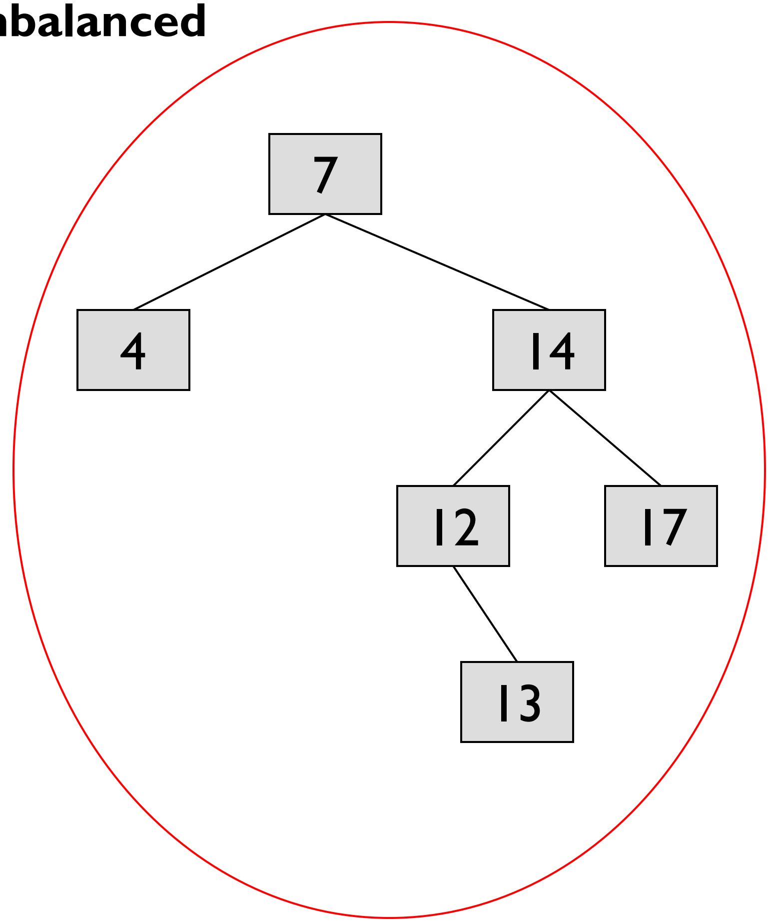
AVL Tree Example:

- **Remove 11, replace it with the largest in its left branch**



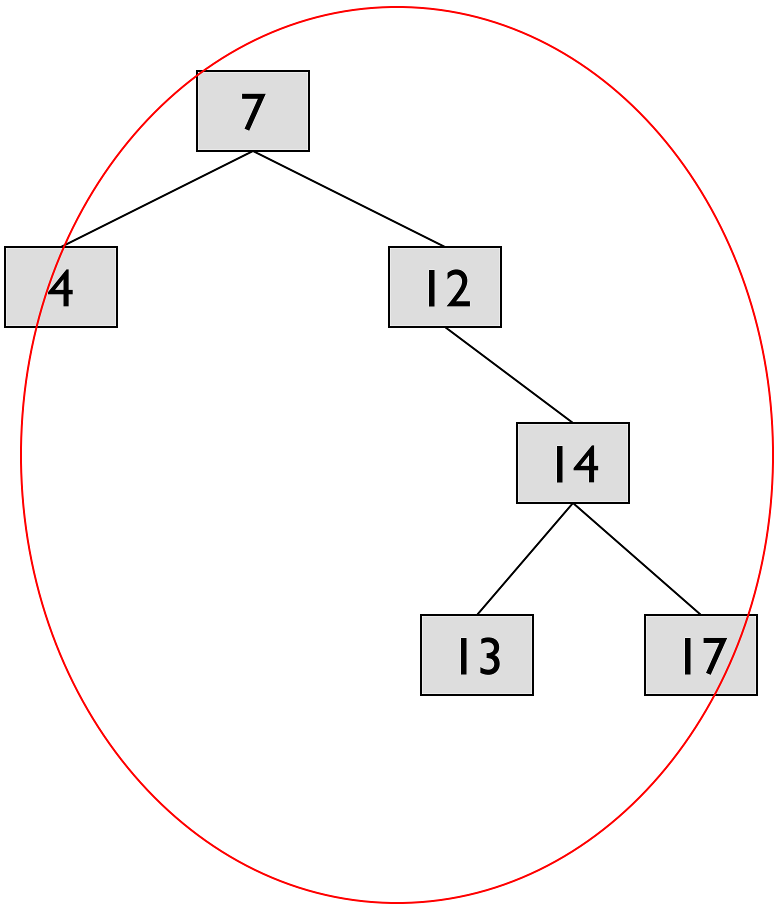
AVL Tree Example:

- **Remove 8, unbalanced**



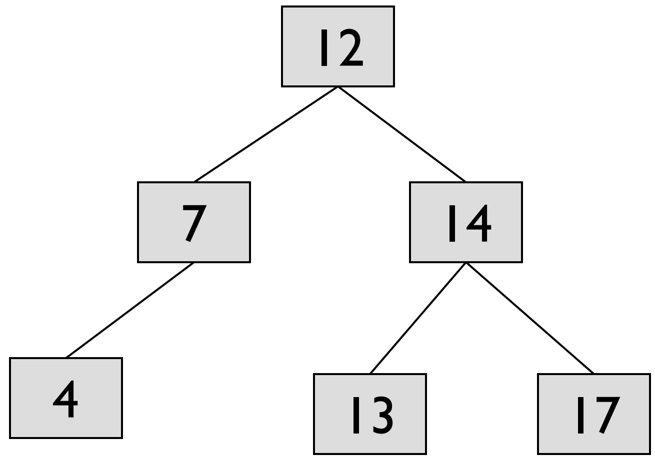
AVL Tree Example:

- **Remove 8, unbalanced**



AVL Tree Example:

- **Balanced!!**

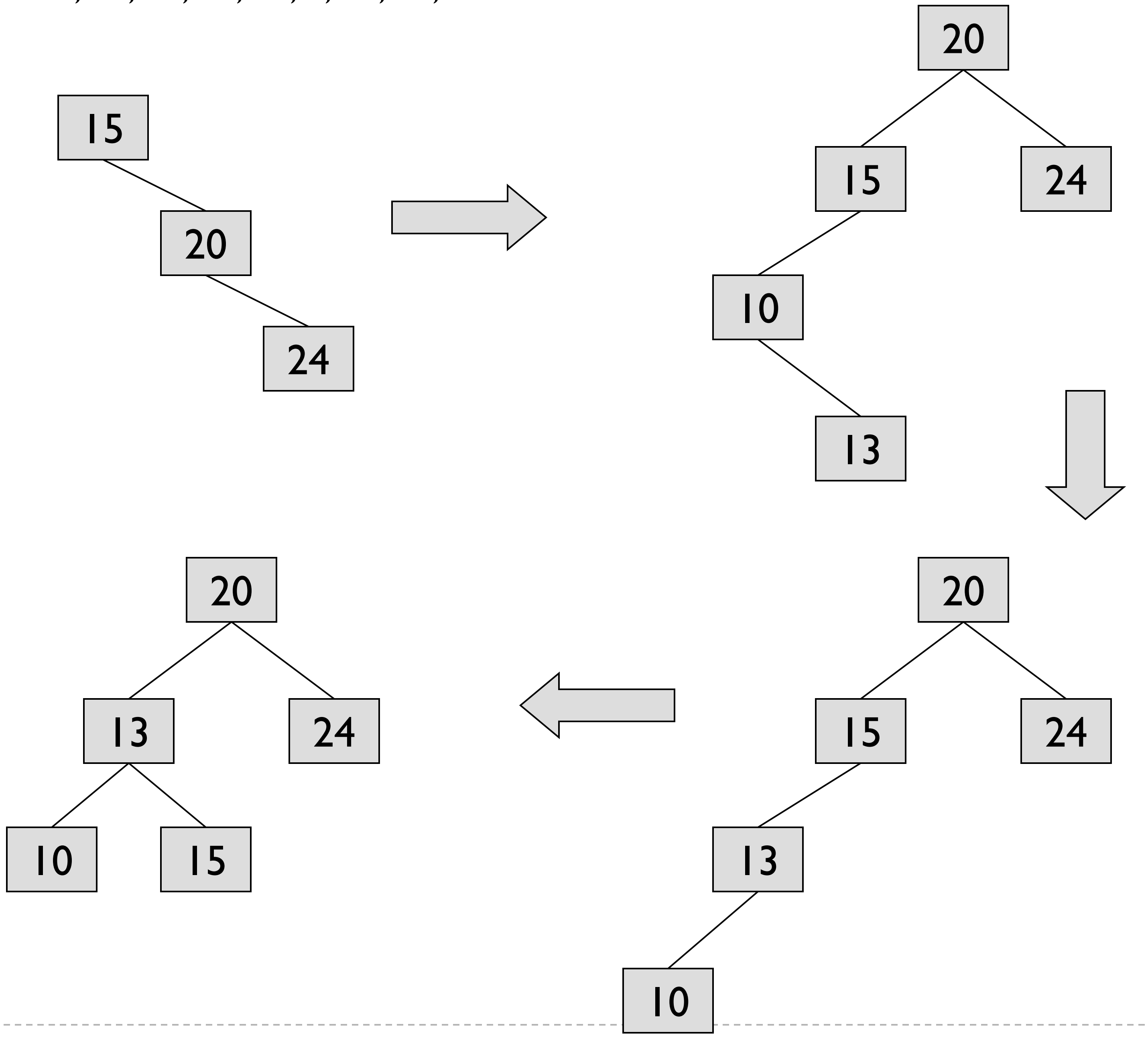


In Class Exercises

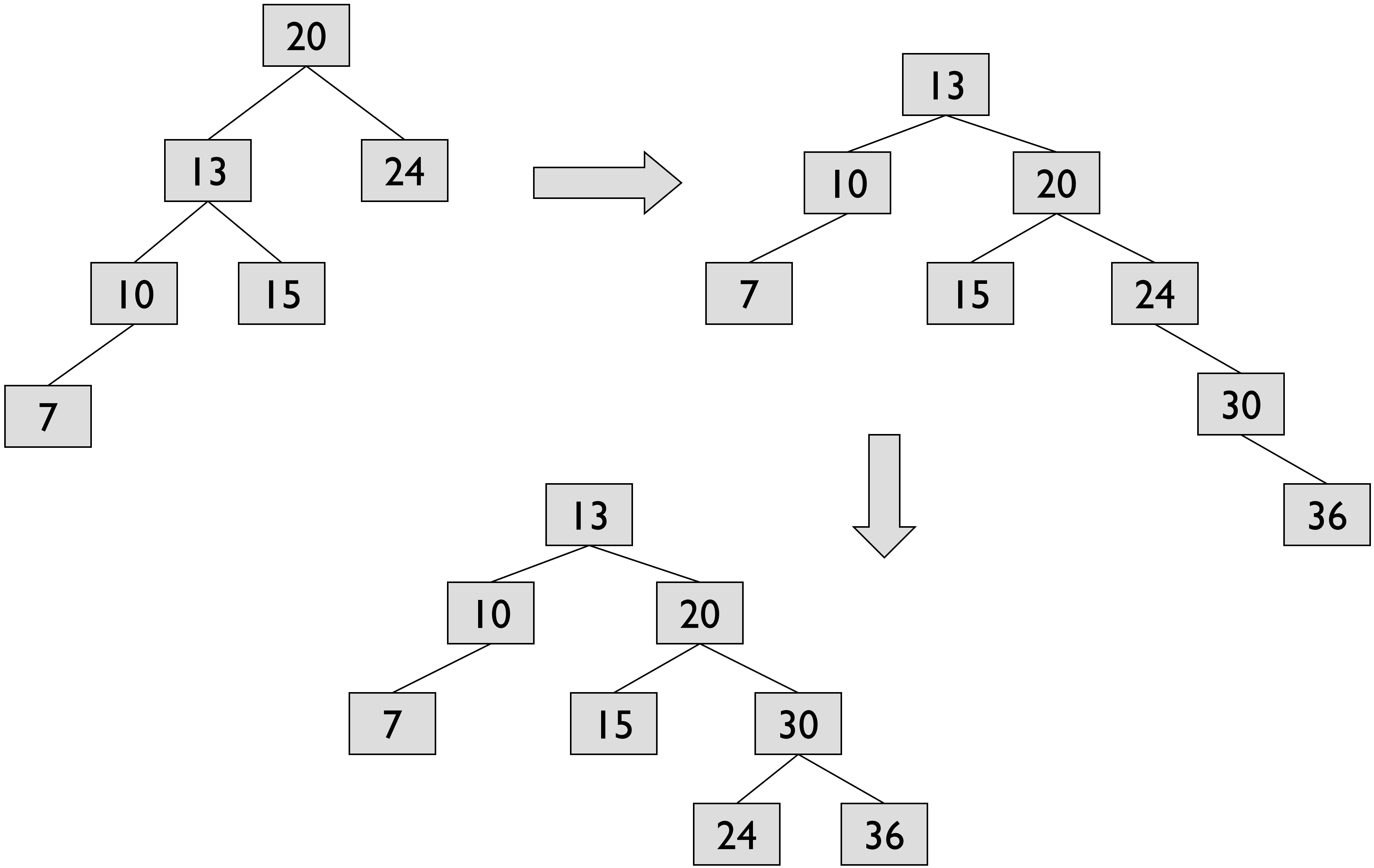
- Build an AVL tree with the following values:

15, 20, 24, 10, 13, 7, 30, 36, 25

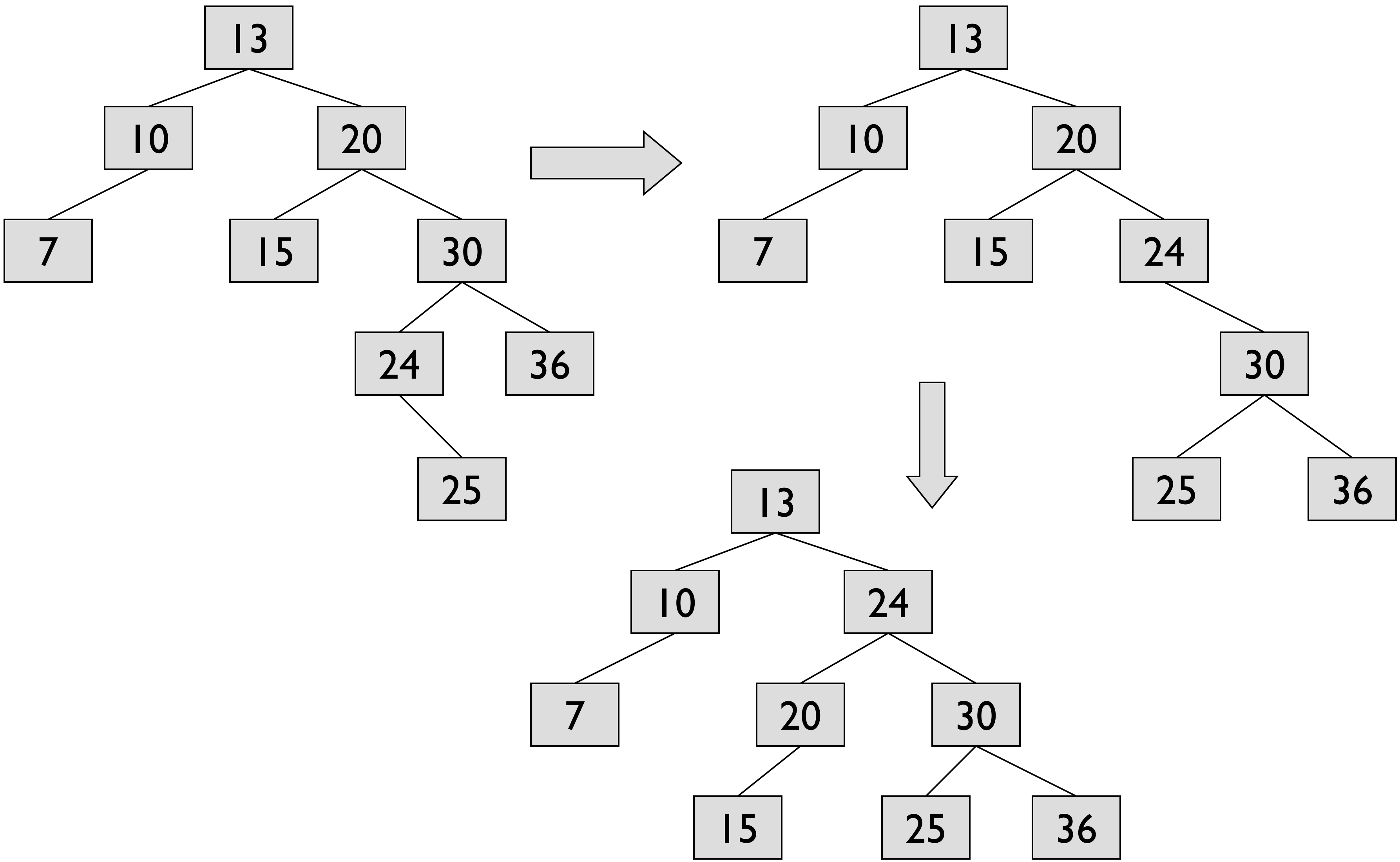
15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25



Remove 24 and 20 from the AVL tree.