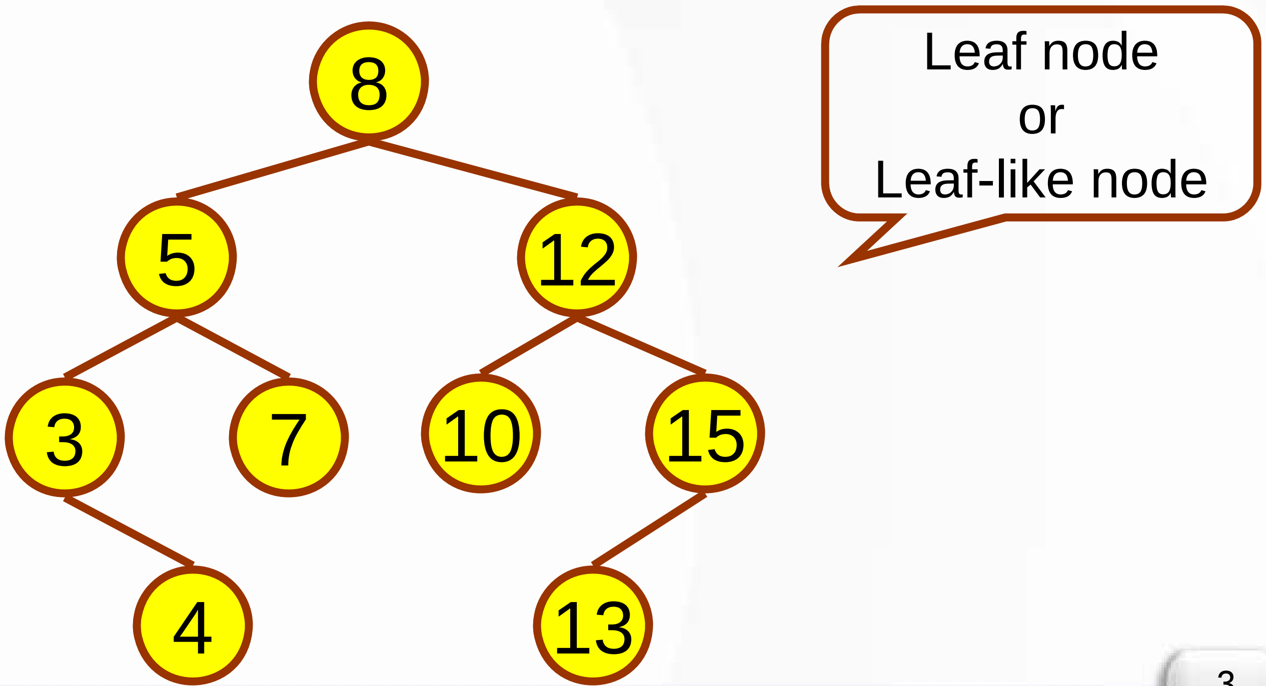
Binary Search Tree and AVL Tree

Definition of BST

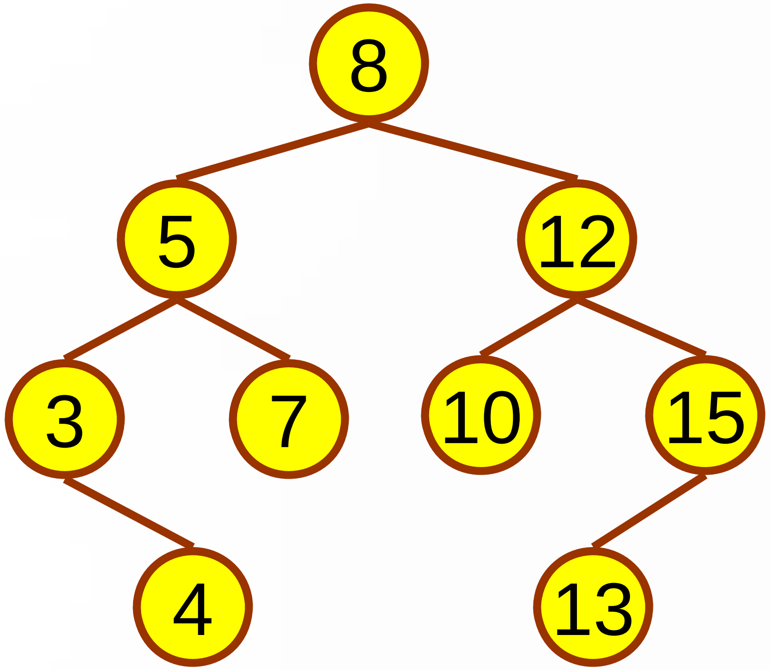
- Binary tree
- Relationship among values in nodes
  - All values in the left subtree < value in the root
  - All values in the right subtree ≧ value in the root
- For all subtrees

Nice Property

- Where is the smallest node?
- Where is the largest node?



Traversals on BST



- Preorder
  - 8, 5, 3, 4, 7, 12, 10, 15, 13
- Postorder
  - 4, 3 7, 5, 10, 13, 15, 12, 8
- Inorder
  - 3, 4, 5, 7, 8, 10, 12, 13, 15

Search

- Start from the root
- Check the value of the root and the key
- If key < root
  - Go to the left subtree
- Else
  - Go to the right subtree
- Repeat until the key is found in a node
  - or no node is found

How to Construct a BST?

- A set of data → a set of nodes
- Insert each node to a BST

- *Inserted node is a leaf node.*

- Data structure, if using a linked list
  - Value
  - Left subtree
  - Right subtree

Insertion

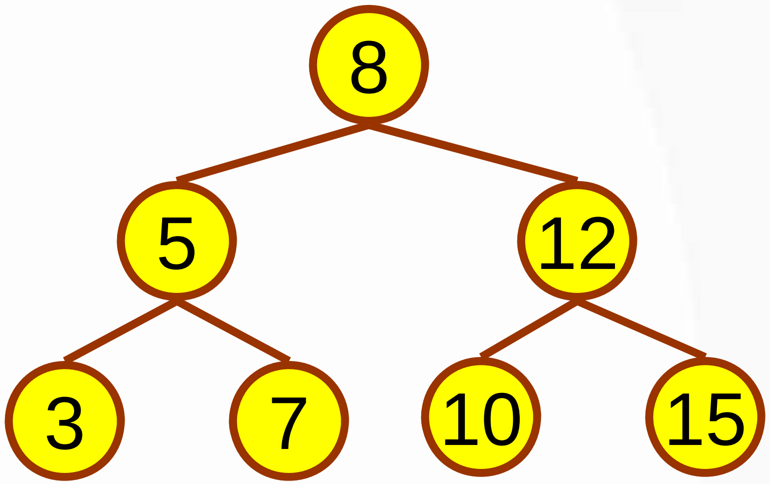
- Case 1
  - Insert to an empty BST
- Case 2
  - Insert to the root's left or right
- Case 3
  - If no space available under root
    - Perform "Insert" to the subtree

Recursive Algorithm

```
Insert (BST, Node)
{
   if BST is empty
         BST is Node
   else if (Node's value < root's value)
         Insert (BST's left, Node)
   else
         Insert (BST's right, Node)
}
```

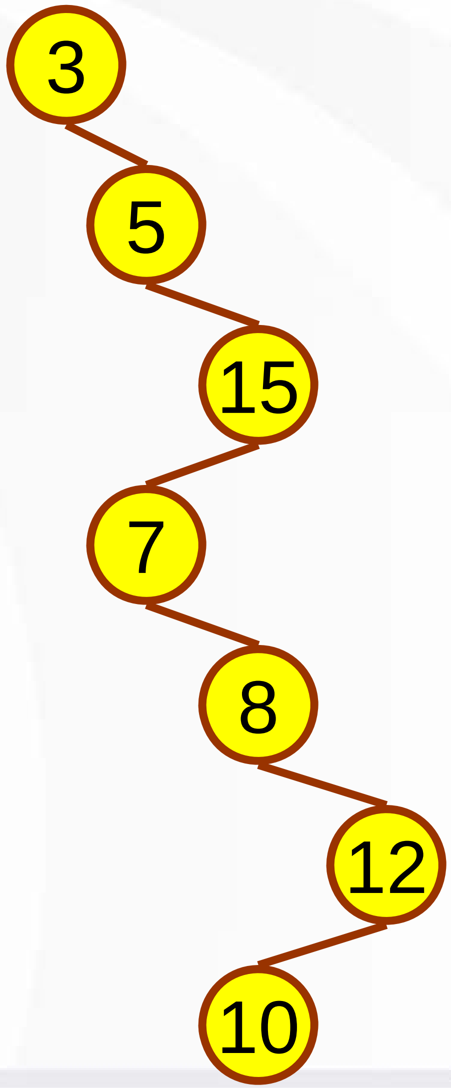
Illustration

- Keys: 8, 12, 10, 5, 15, 3, 7



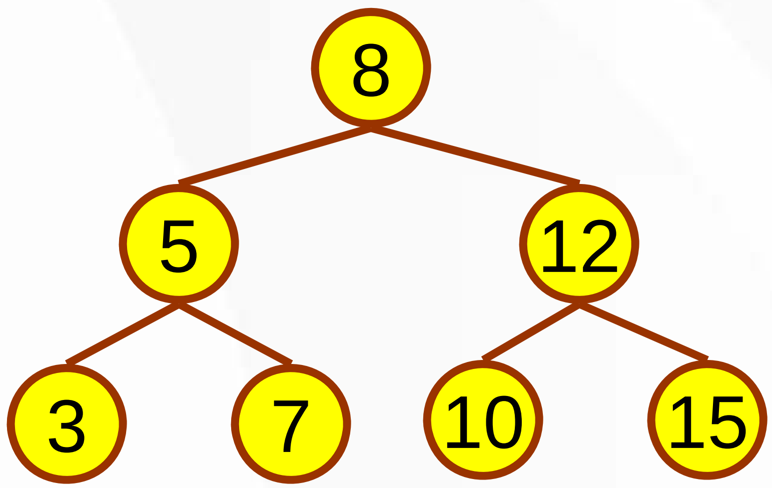
Search

- Search for 8
  - 5 steps
- Search for 5
  - 2 steps
- Search for 3
  - 1 step
- Search for 10
  - 7 steps



Search in a Complete BST

- Search for 8
  - 1 step
- Search for 5
  - 2 steps
- Search for 3
  - 3 steps



Steps Needed

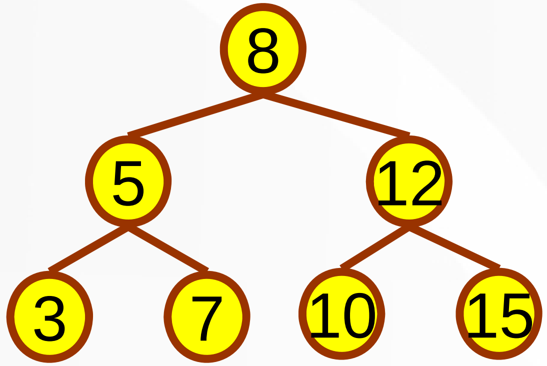
- Chain-like tree
  - Best case
    - 1 step
  - Worst case
    - N steps
- Complete tree
  - Best case
    - 1 step
  - Worst case
    - log N + 1 steps

Just like a linked list.

Height of a BST

Search Algorithm

```
BST_search (tree, key)
{
   if (tree is empty)
         return FALSE
   else if (tree's root is equal to key)
         return TRUE
   else if (tree's root > key)
         BST_search (tree's left, key)
   else BST_search (tree's right, key)
}
```



Algorithm Analysis

- Step 1
  - N nodes
- Step 2
  - N/2 nodes
- Step 3
  - N/4 nodes
- Step 4
  - N/8 nodes
- ...
- Step k
  - $N/2^{k-1}$ nodes

$$N/2^k = 1$$
$$2^k = N$$
$$k = \log N$$

What happens when there is no more nodes?

AVL Trees

Adelson-Velskii and Landis Trees

Problem with Insertion

- Shape of the tree:
  - Affected by the node sequence
    - Well balanced
    - Tilted
- Efficiency vs. Shape
  - Balanced BST
- Important issue
  - How to keep the tree balanced?

Possible Approach

- Approach 1
  - Rearrange the node sequence
- Approach 2
  - Reorganize the tree
- Approach 3
  - Reorganize the tree while it is constructed

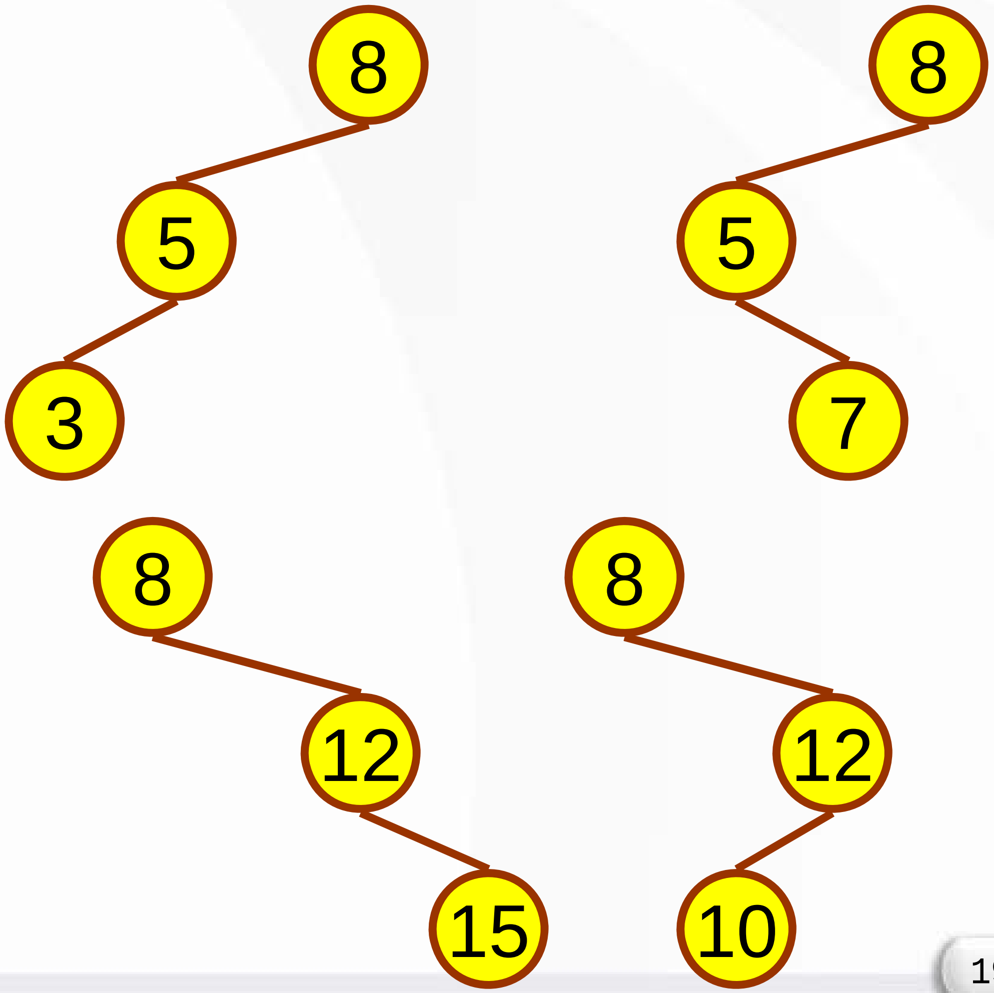
Possible Violation of "Balanced-ness"

- Goal
  - |Height of LST – height of RST| $\leq$ 1
- Violation caused by inserting a node
  - $|H_L - H_R| = 1$
  - One more node to the "tilted" side
    - Case 1: $H_L - H_R = 1$
      - One more node to the LST
    - Case 2: $H_R - H_L = 1$
      - One more node to the RST

Violation!!

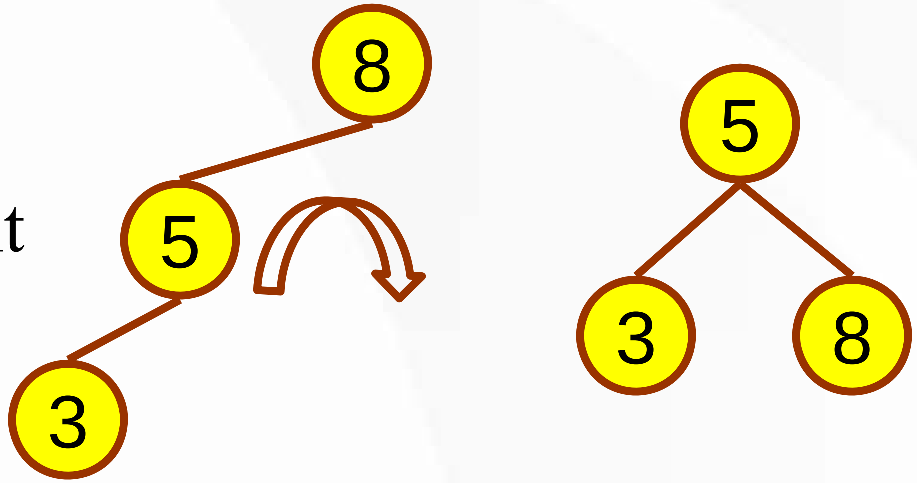
Possible Violation Cases

- Case 1
  - Left of left
- Case 2
  - Right of left
- Case 3
  - Right of right
- Case 4
  - Left of right

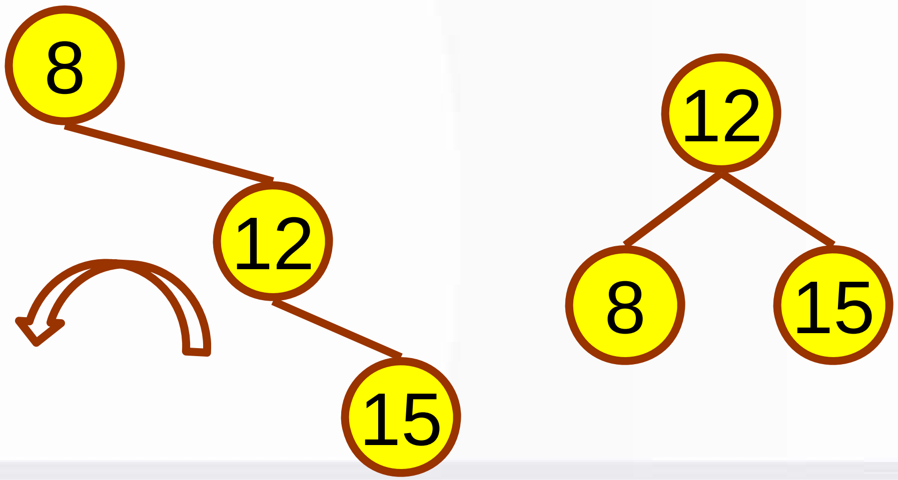


Solution

- Left of left
  - Rotation to right

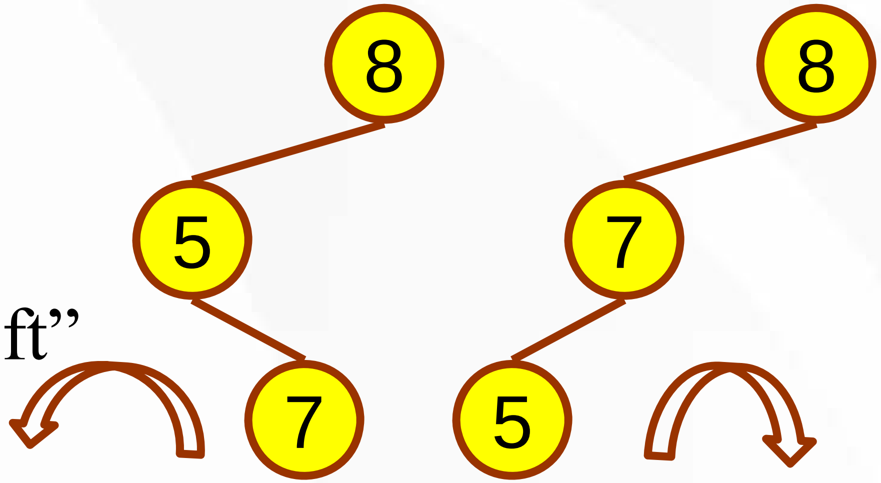


- Right of right
  - Rotation to left

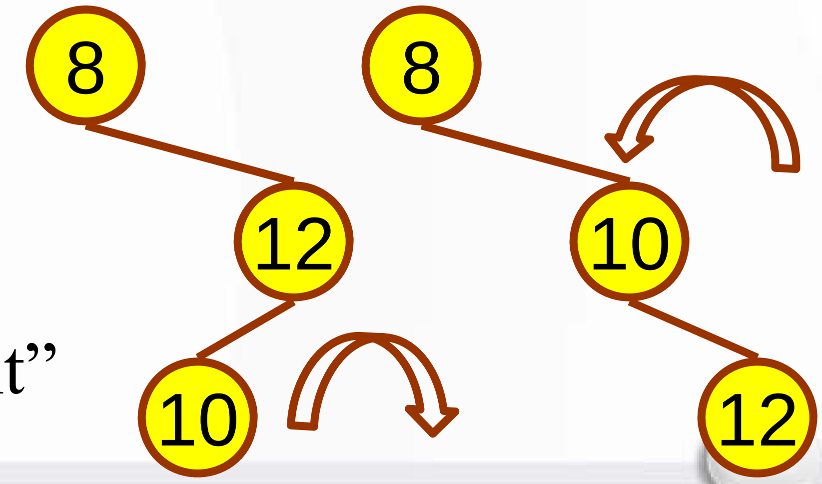


Solution

- Right of left
  - Rotation to left
  - Treat it as “left of left”

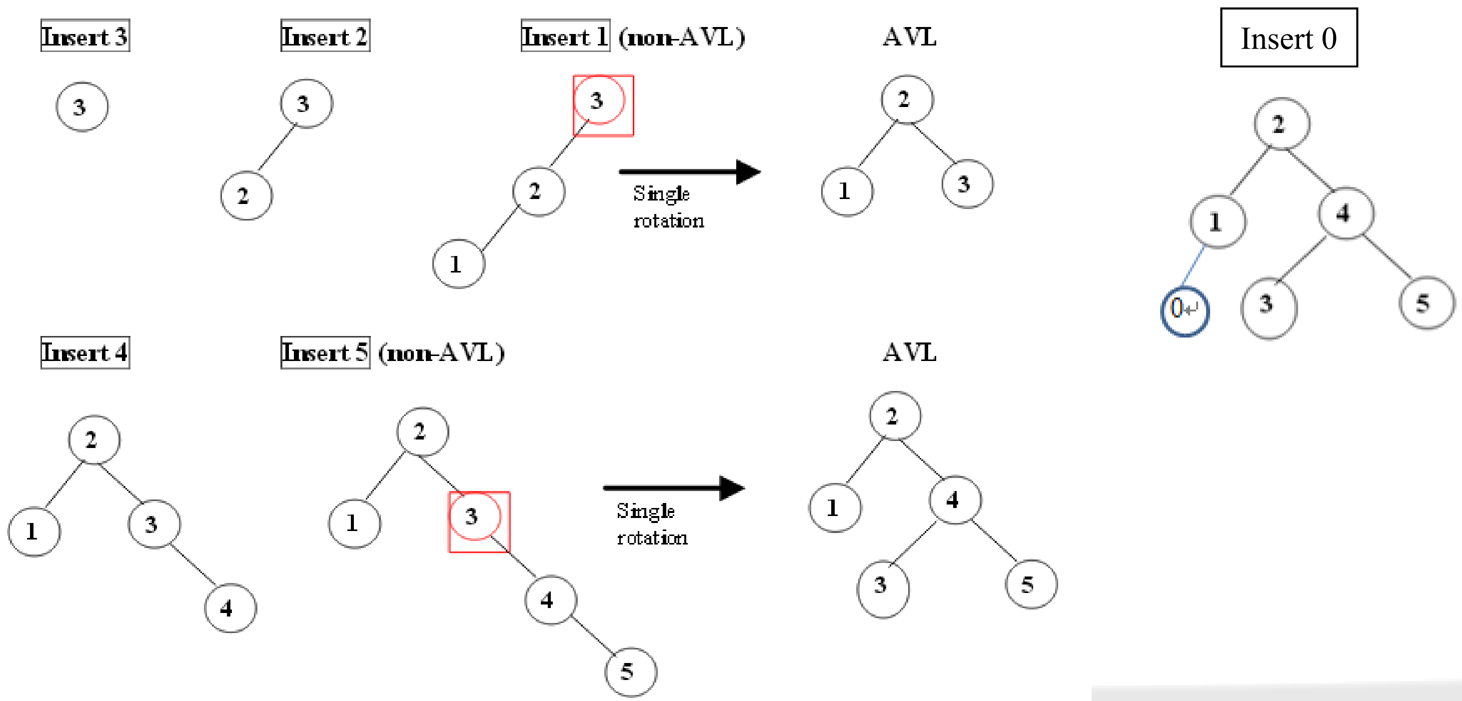


- Left of right
  - Rotation to right
  - Treat it as “right of right”



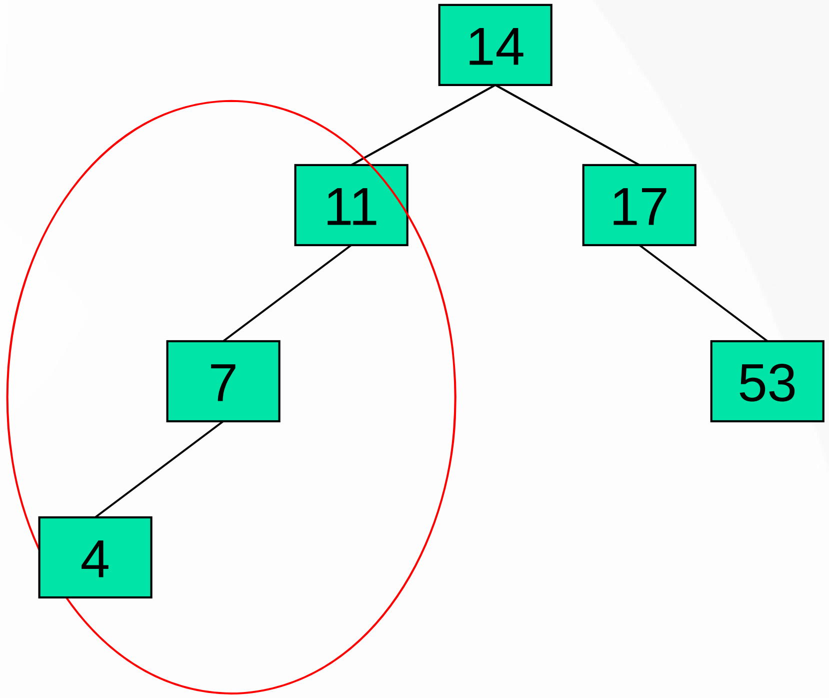
Suppose we insert the following data keys in sequence: 3, 2, 1, 4, 5, 0

Please show the AVL tree for every step after you insert one key. Write down all the necessary operations (e.g., left of left rotation)



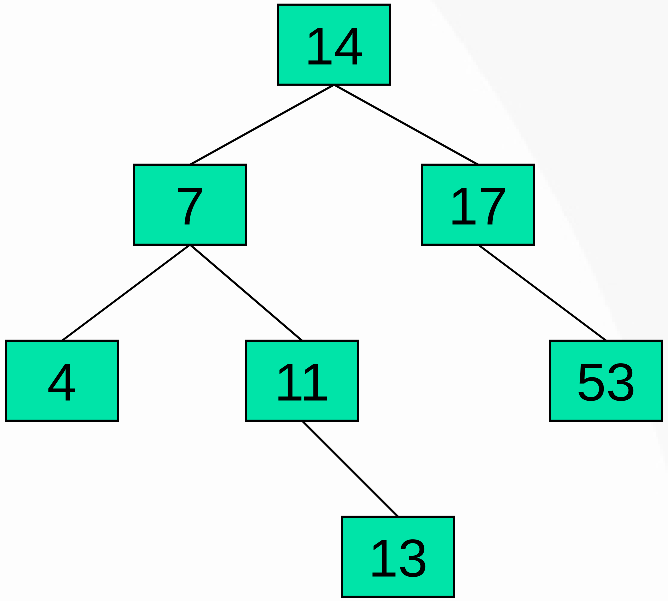
AVL Tree Example:

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



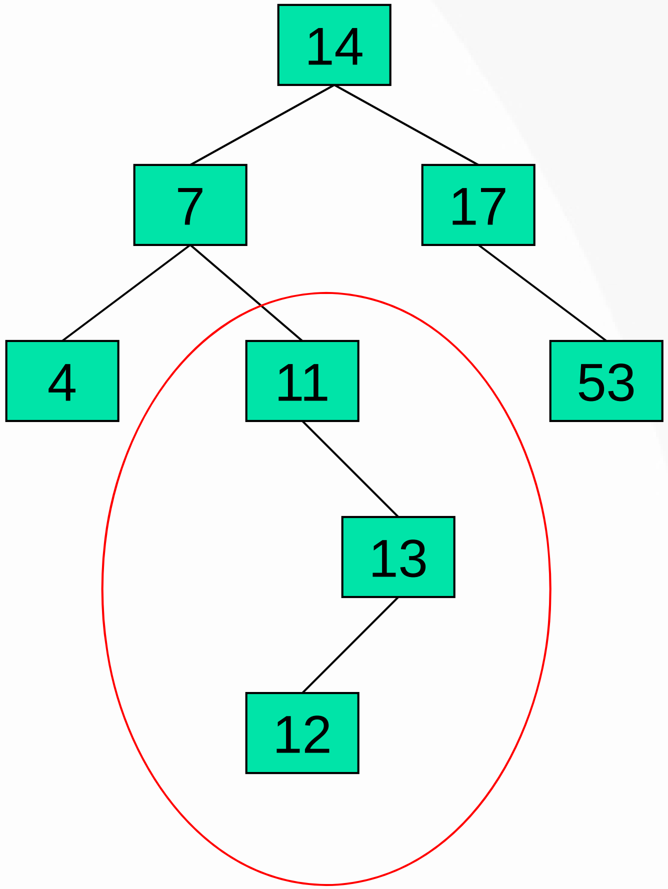
AVL Tree Example:

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



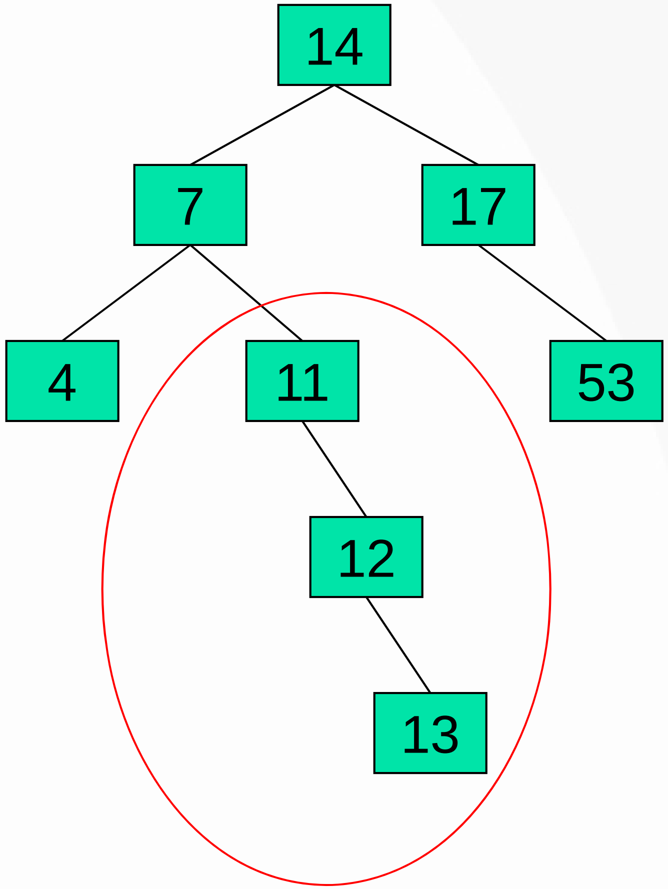
AVL Tree Example:

• Now insert 12



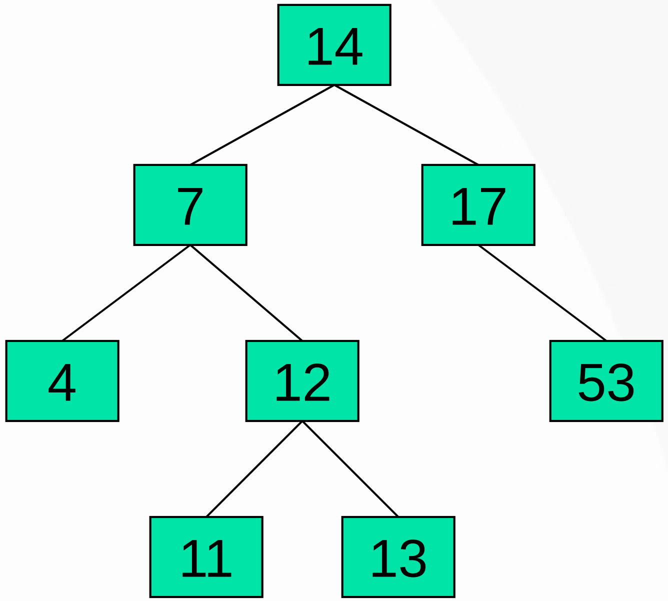
AVL Tree Example:

• **Now insert 12**



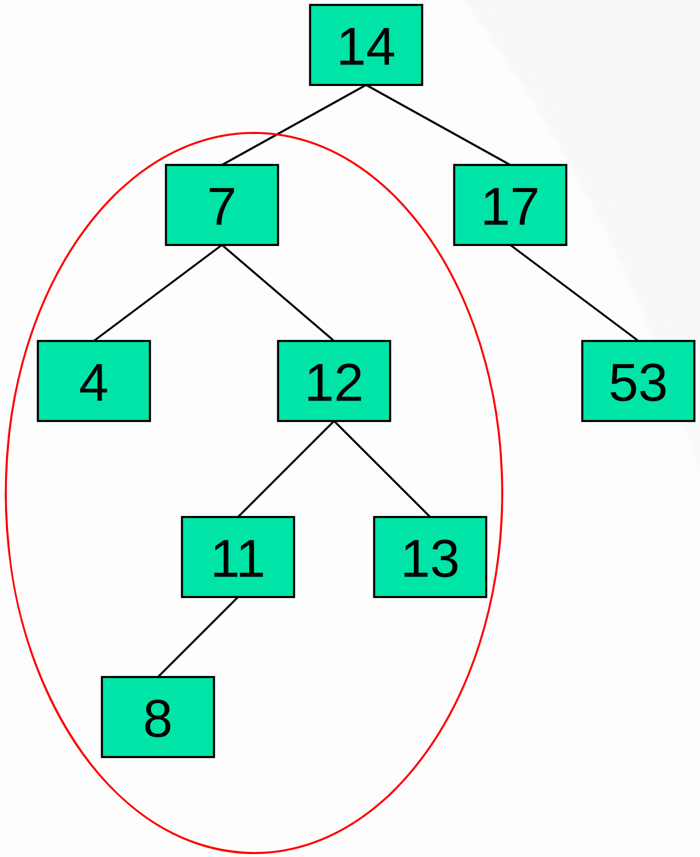
AVL Tree Example:

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



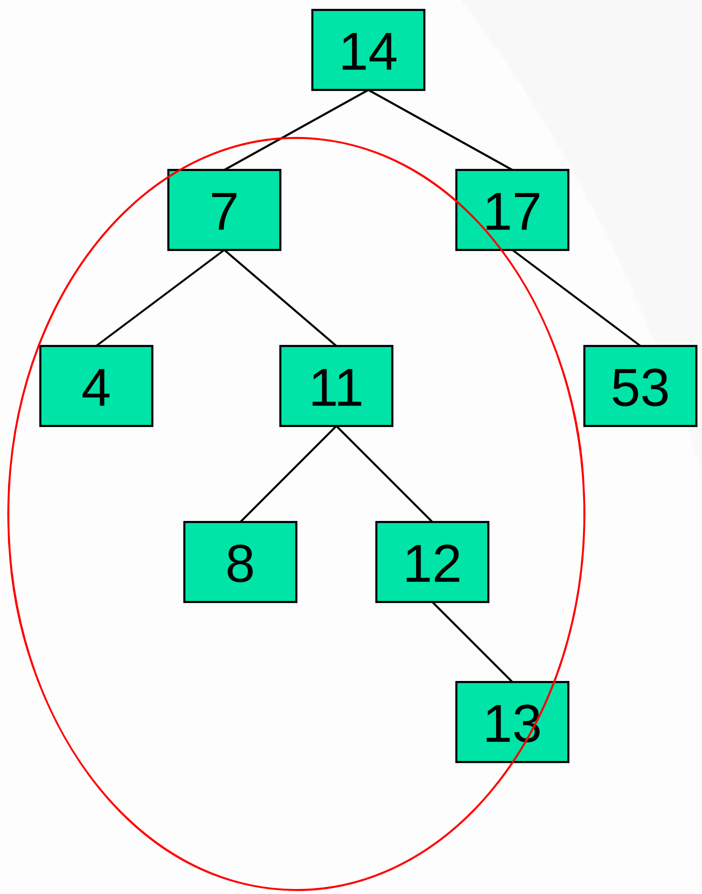
AVL Tree Example:

- **Now insert 8**



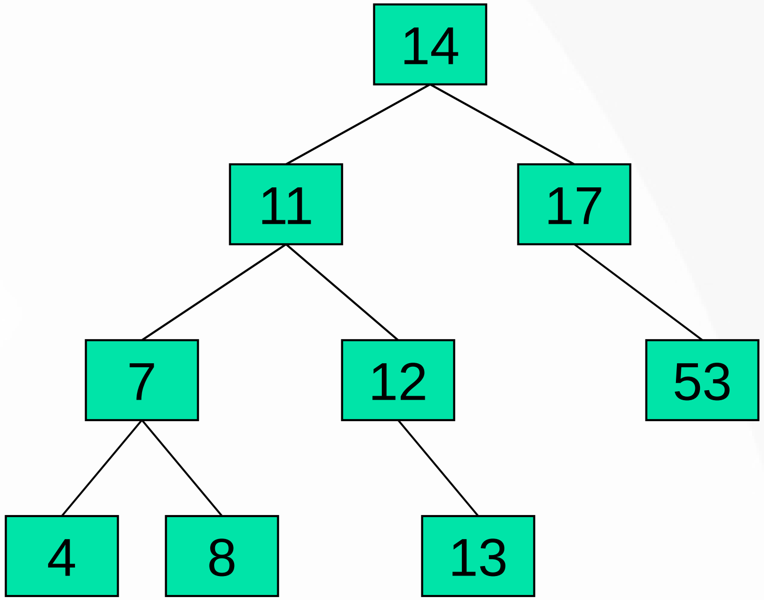
AVL Tree Example:

• **Now insert 8**



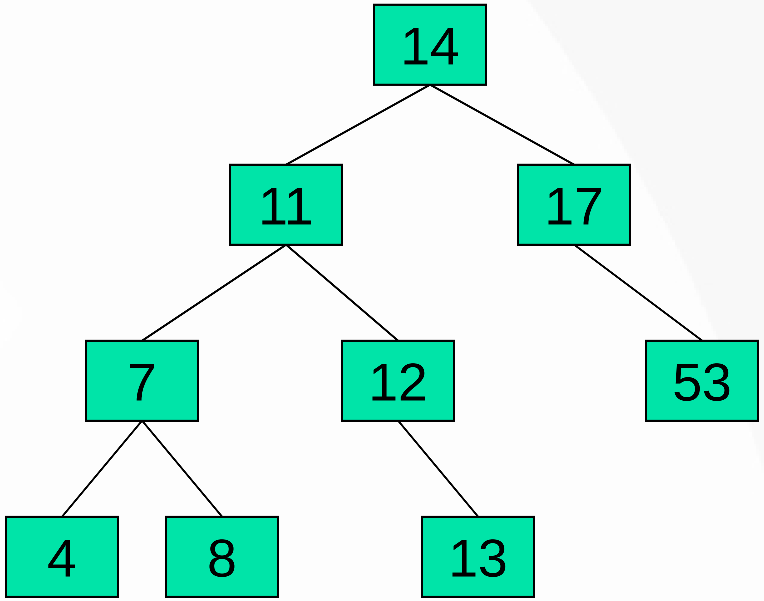
AVL Tree Example:

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



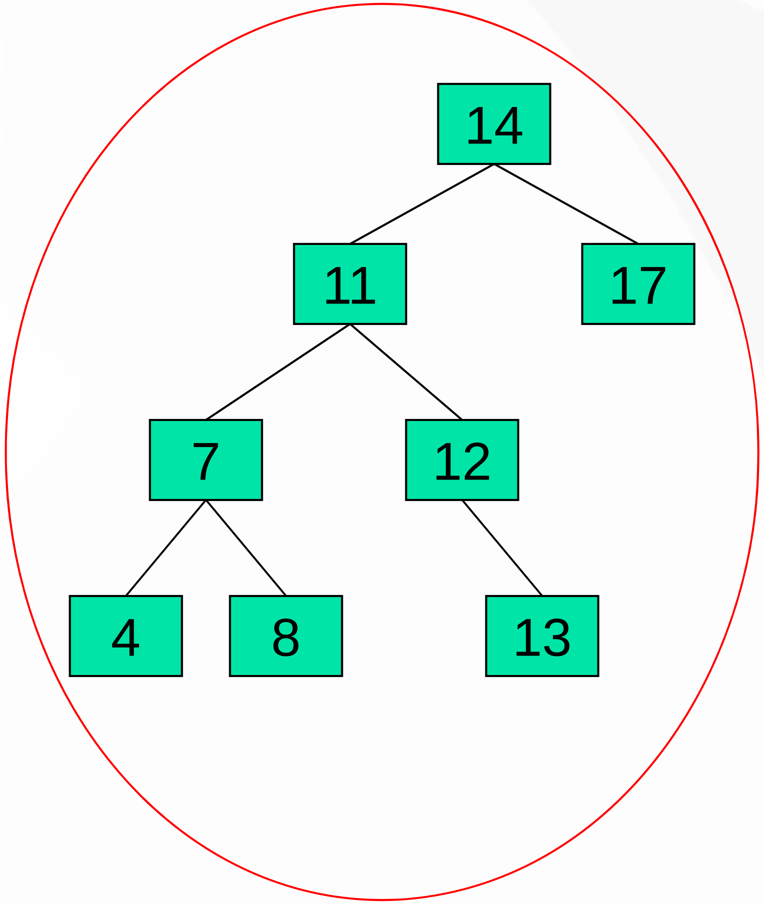
AVL Tree Example:

• **Now remove 53**



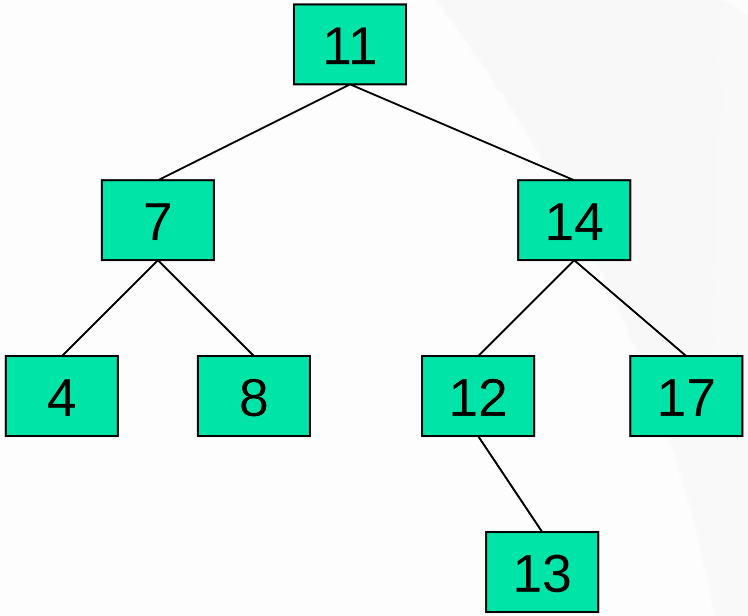
AVL Tree Example:

• **Now remove 53, unbalanced**



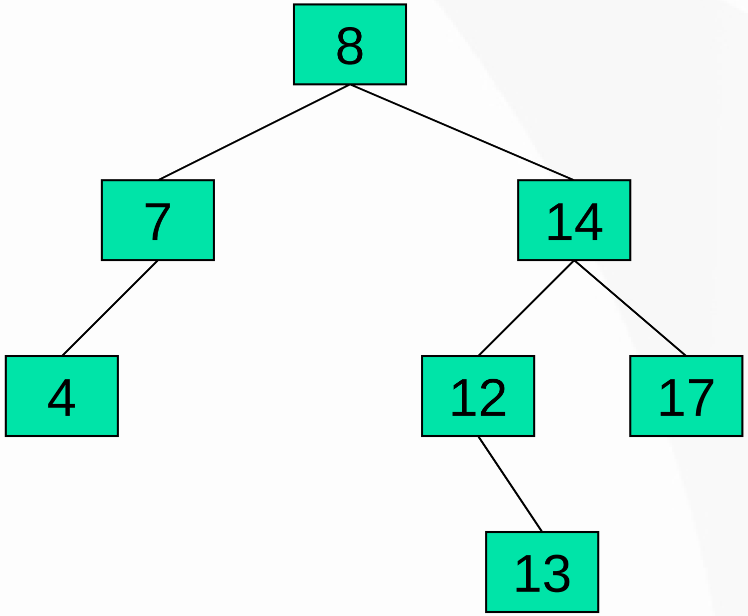
AVL Tree Example:

- **Balanced!**   **Remove 11**



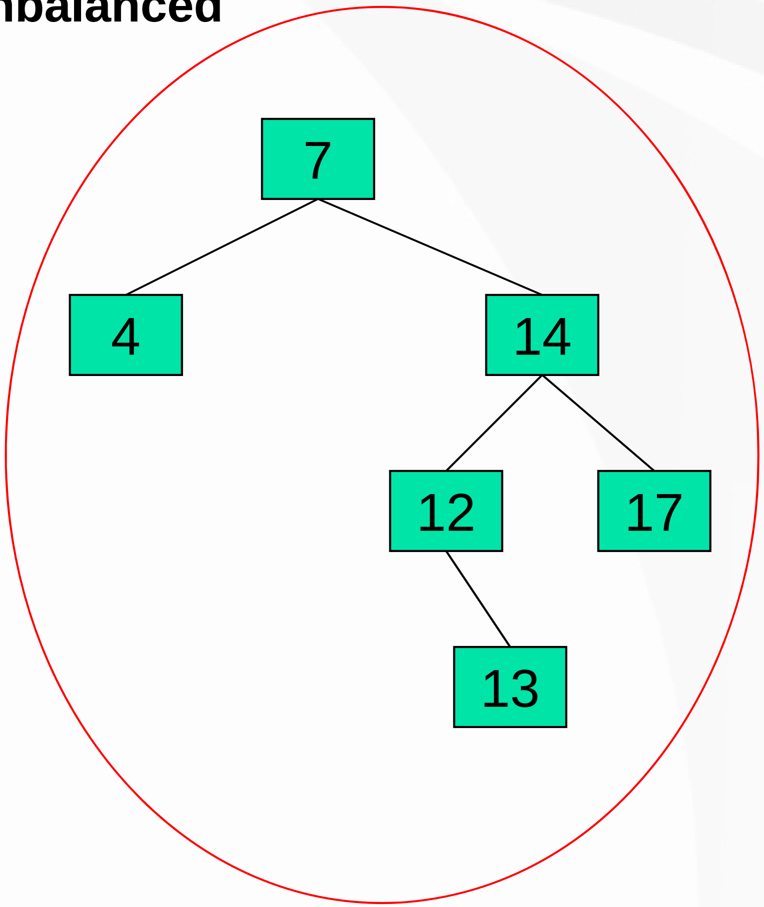
AVL Tree Example:

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



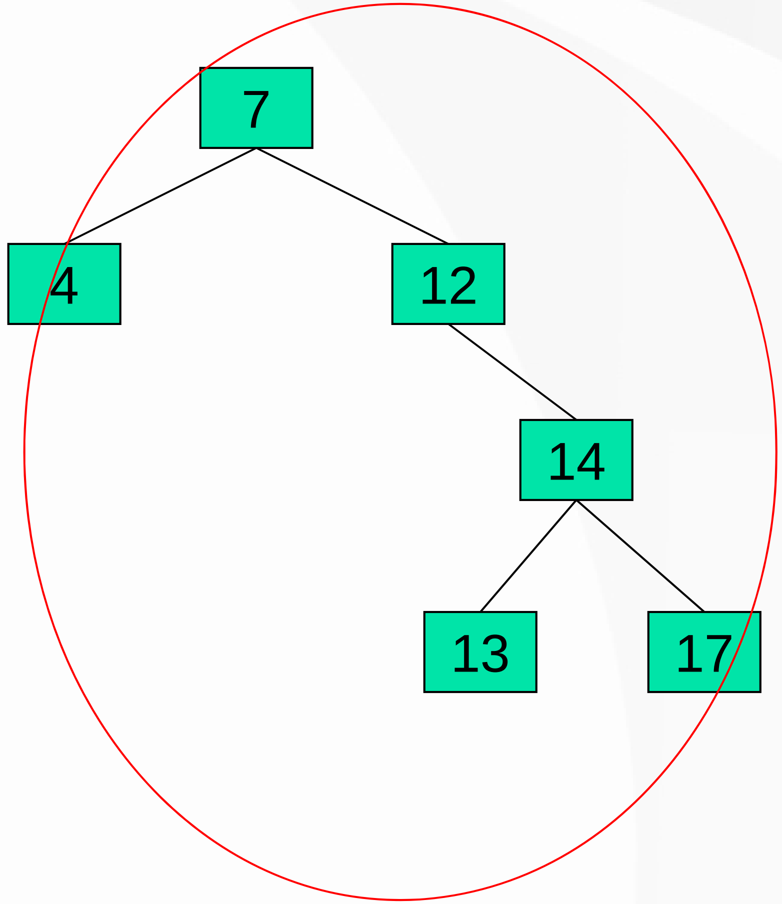
AVL Tree Example:

- Remove 8, unbalanced



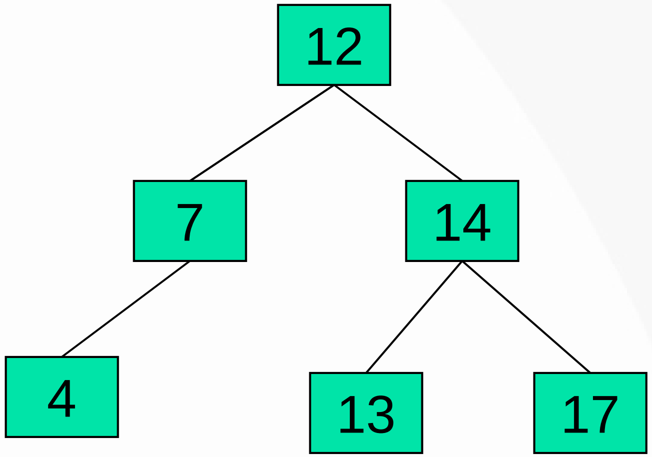
AVL Tree Example:

- **Remove 8, unbalanced**



AVL Tree Example:

- **Balanced!!**

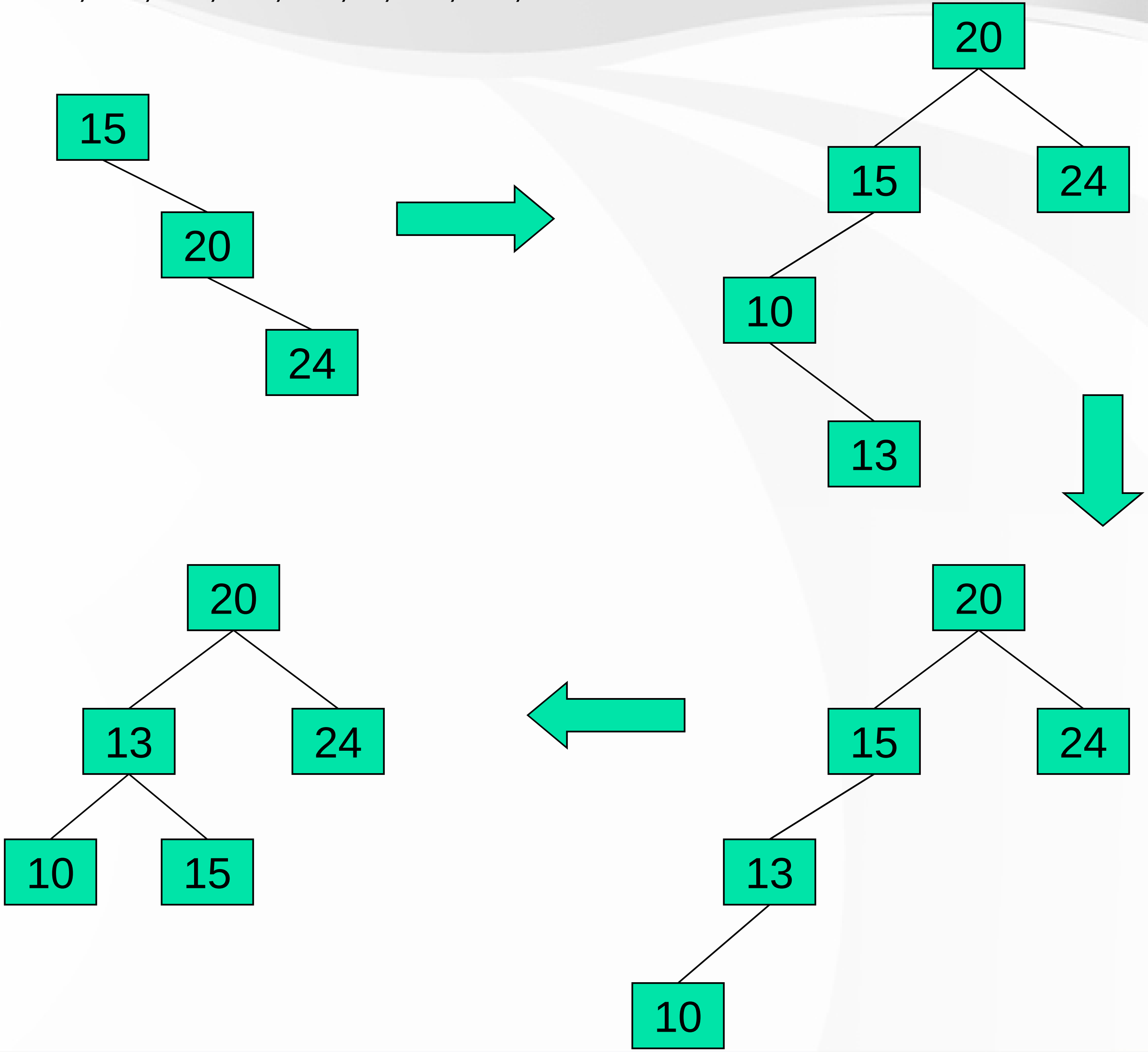


In Class Exercises

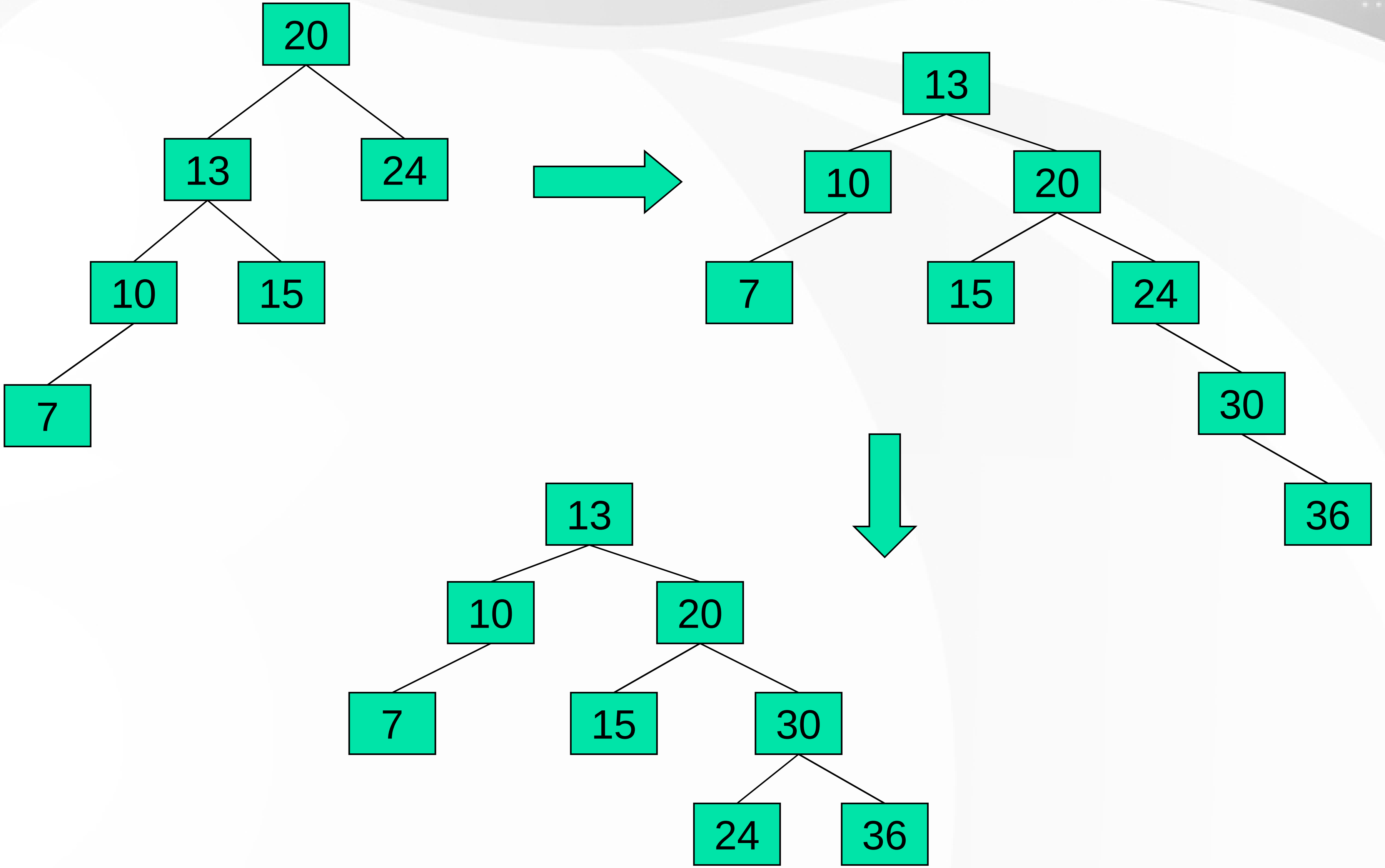
- Build an AVL tree with the following values:

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

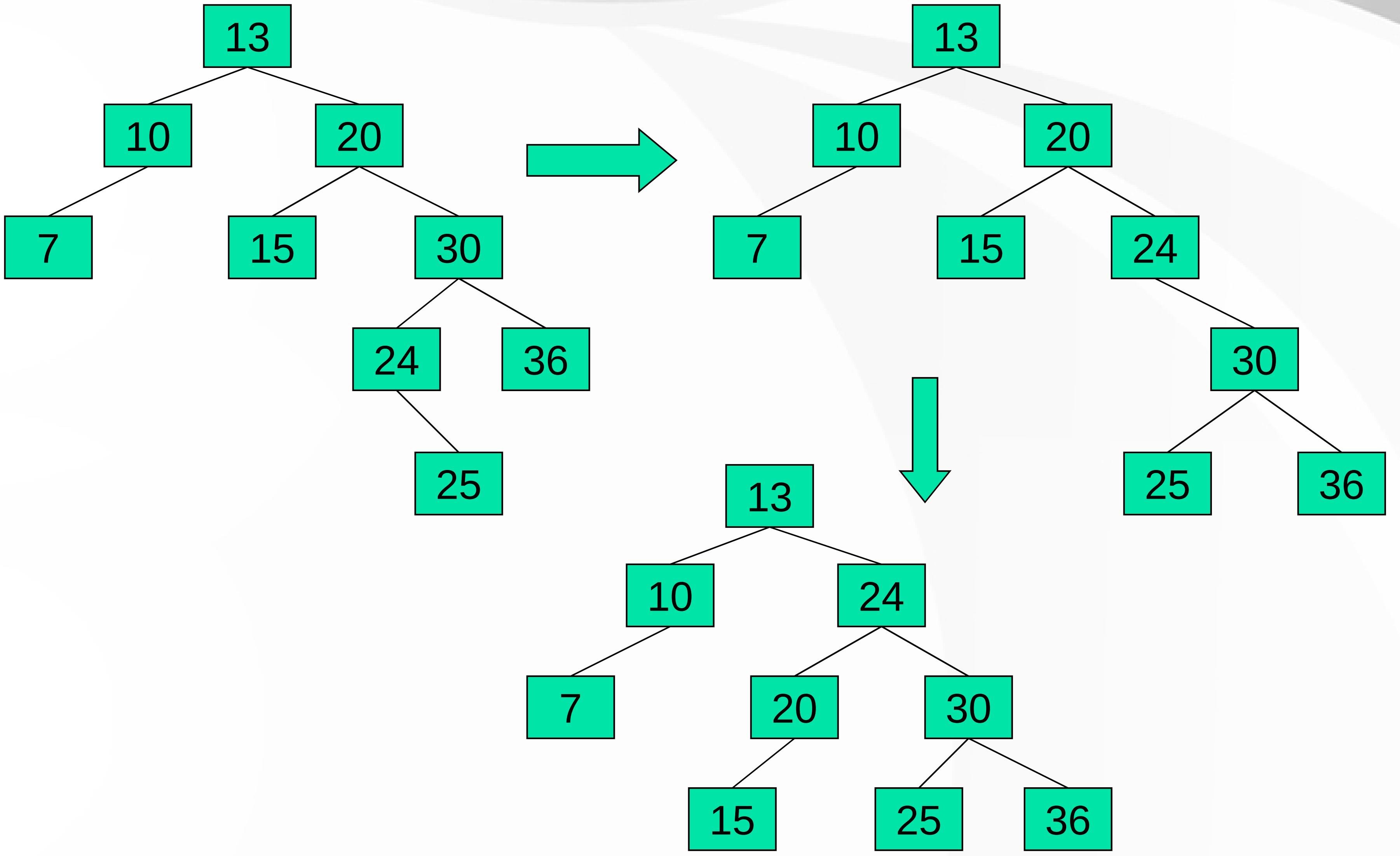
15, 20, 24, 10, 13, 7, 30, 36, 25



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.