AVL Trees

Advanced Data Structre

1

Binary Search Tree - Best Time

- All BST operations are O(h), where d is tree height
- minimum h is $h = \lfloor \log_2 N \rfloor$ for a binary tree with N nodes
  - What is the best case tree?
  - What is the worst case tree?
- So, best case running time of BST operations is O(log N)

2

Binary Search Tree - Worst Time

Worst case running time is O(N)

What happens when you Insert elements in ascending order?

- Insert: 2, 4, 6, 8, 10, 12 into an empty BST

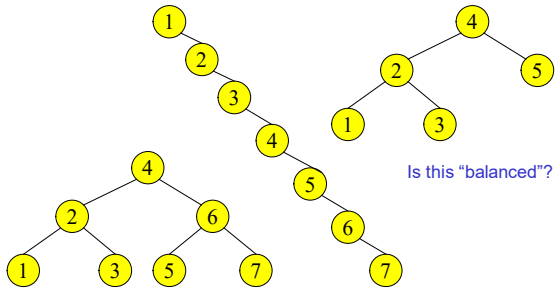
Problem: Lack of "balance":

- compare depths of left and right subtree

Unbalanced degenerate tree

3

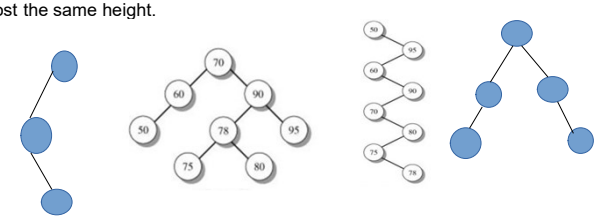
Balanced and unbalanced BST



4

Definition :: Balanced Tree

- Trees whose height in the worst case turns out to be O (log N) are known as Balanced trees or height balanced trees Or
- A balanced search tree is one where all the branches from the root have almost the same height.
-



Balance Factor is a property of node which is used to decide whether the tree is balnced or not

5

Approaches to balancing trees

- Don't balance
  - › May end up with some nodes very deep
- Strict balance
  - › The tree must always be balanced perfectly
- Pretty good balance
  - › Only allow a little out of balance
- Adjust on access
  - › Self-adjusting

6

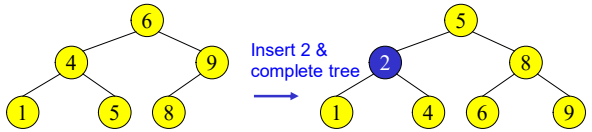
Balancing Binary Search Trees

- Many algorithms exist for keeping binary search trees balanced
  - › Adelson-Velskii and Landis (AVL) trees (height-balanced trees)
  - ›
  - › Splay trees and other self-adjusting trees
  - › B-trees and other multiway search trees

7

Perfect Balance

- Want a complete tree after every operation
  - › tree is full except possibly in the lower right
- This is expensive
  - › For example, insert 2 in the tree on the left and then rebuild as a complete tree



8

AVL - Good but not Perfect Balance

- AVL trees are height-balanced binary search trees
- Balance factor of a node **bf**
  - › height(left subtree) - height(right subtree)
- An AVL tree has balance factor calculated at every node
  - › For every node, heights of left and right subtree can differ by no more than 1,0,-1
  - › Store current heights in each node

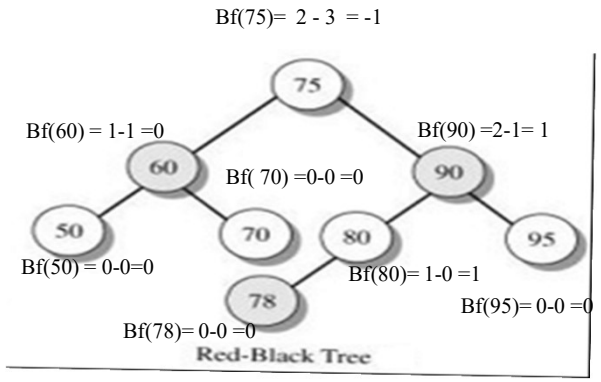
AVL Tee

If balanceFactor > 1 or < -1 then the tree is unbalanced, and needs 'rearranging' to make it more balanced

AVL tree

Height of a node

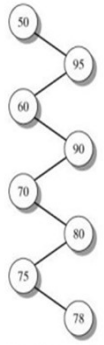
- The height of a leaf is 1. The height of a null pointer is zero.
- The height of an internal node is the maximum height of its children plus 1



:: Definiton

- An empty binary tree is an AVL tree.
- If non empty tree the binary tree T is an AVL tree if
  - T_L and T_R the left and right subtrees of T are also AVL trees.
  - $|h(T_L) - h(T_R)| <= 1$ where $h(T_L)$ and $h(T_R)$ are the heights of the left and right subtrees.

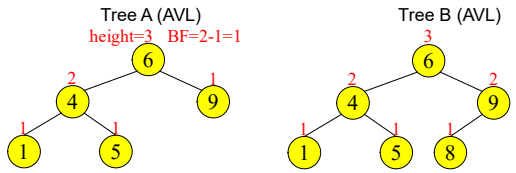
Calculate the balance factor for given tree



Height of an

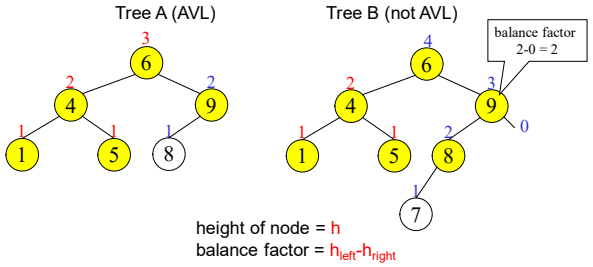
- Suppose we have n nodes in an AVL tree of height h.
  - $n \geq N(h)$ (because N(h) was the minimum)
  - $n \geq \phi^h$ hence $\log_\phi n \geq h$ (relatively well balanced tree!!)
  - $h \leq 1.44 \log_2 n$ (i.e., Find takes O(logn))

Node Heights



height of node = h
balance factor = h_{left}-h_{right}
empty height = 0

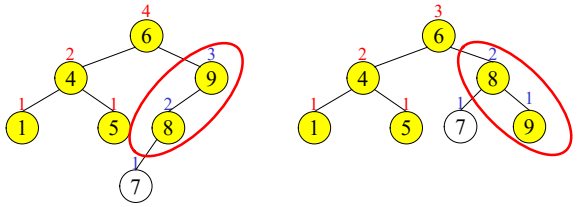
Node Heights after Insert 7



Insert and Rotation in s

- Insert /delete operation may cause balance factor to become 2 or –2 for some node
  - › only nodes on the path from insertion point to root node have possibly changed in height
  - › So after the Insert, go back up to the root node by node, updating heights
  - › If a new balance factor (the difference h_{left}-h_{right}) is 2 or –2, adjust tree by *rotation* around the node

Single Rotation in an



Types of rotation

Insertions in s

Let the node that needs rebalancing be α.

There are 4 cases:

Outside Cases (require single rotation) :

1. Insertion into left subtree of left child of α.
2. Insertion into right subtree of right child of α.

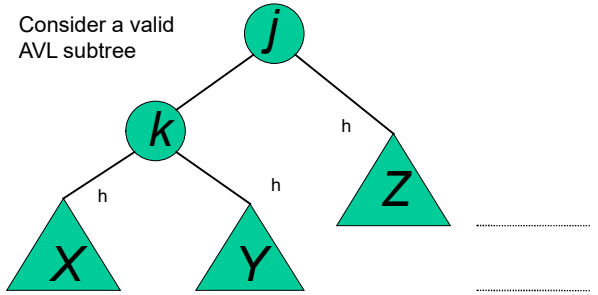
Inside Cases (require double rotation) :

3. Insertion into right subtree of left child of α.
4. Insertion into left subtree of right child of α.

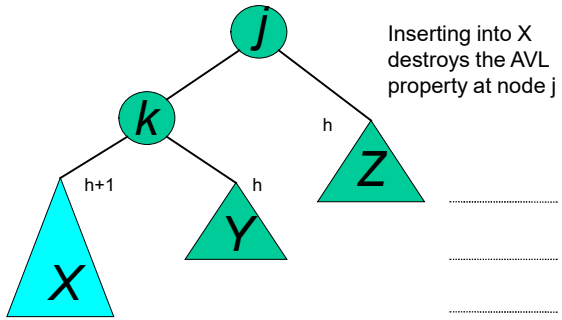
The rebalancing is performed through four separate rotation algorithms.

AVL Insertion: Outside Case

Consider a valid AVL subtree

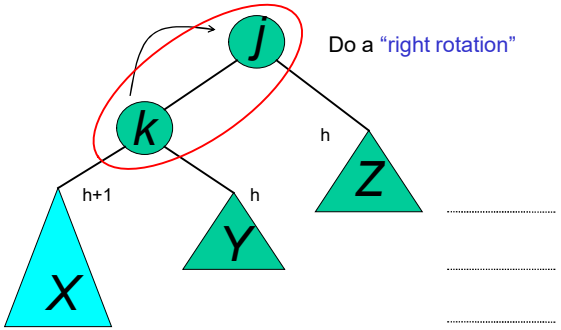


AVL Insertion: Outside Case



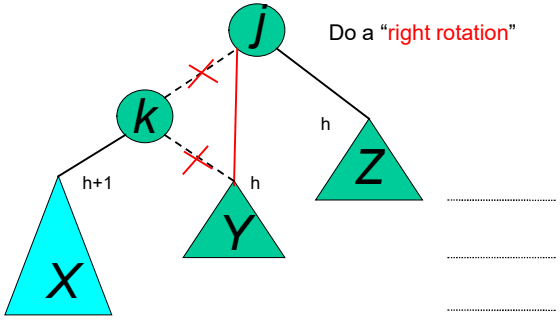
Inserting into X destroys the AVL property at node j

AVL Insertion: Outside Case



Do a "right rotation"

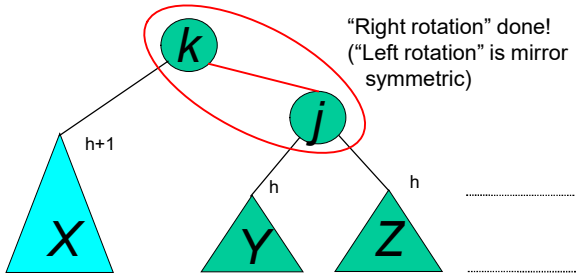
Single right rotation



Do a "right rotation"

25

Outside Case Completed



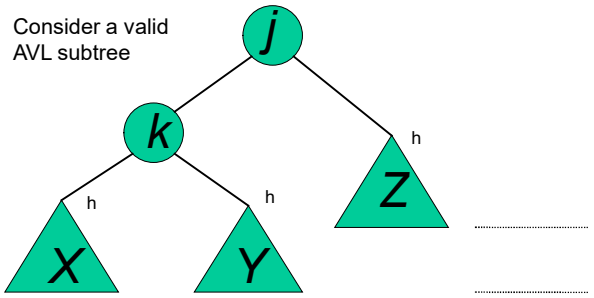
"Right rotation" done!
("Left rotation" is mirror symmetric)

AVL property has been restored!

26

AVL Insertion: Inside Case

Consider a valid AVL subtree

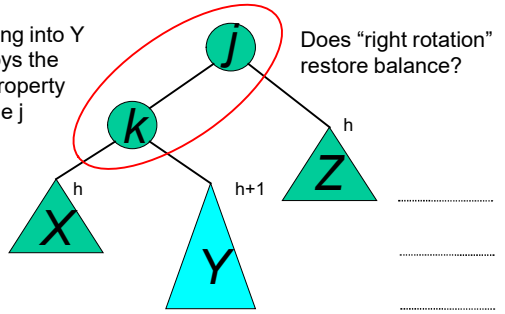


27

AVL Insertion: Inside Case

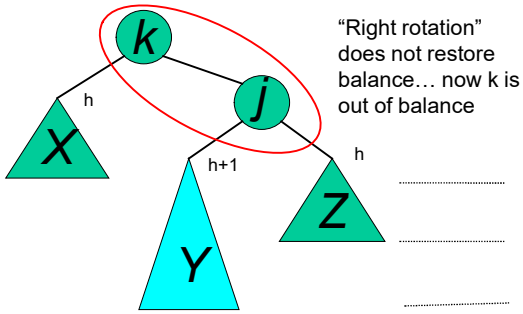
Inserting into Y destroys the AVL property at node j

Does "right rotation" restore balance?



28

AVL Insertion: Inside Case

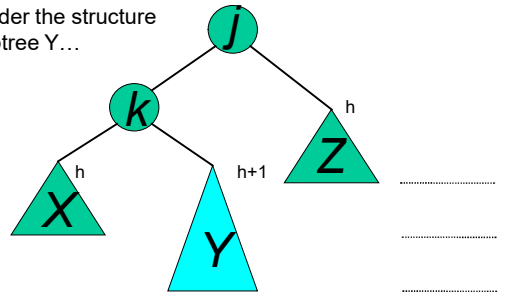


"Right rotation" does not restore balance… now k is out of balance

29

AVL Insertion: Inside Case

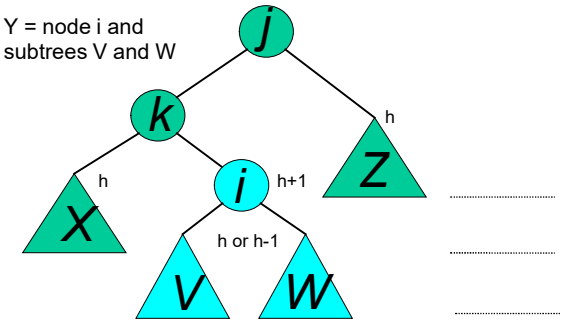
Consider the structure of subtree Y…



30

AVL Insertion: Inside Case

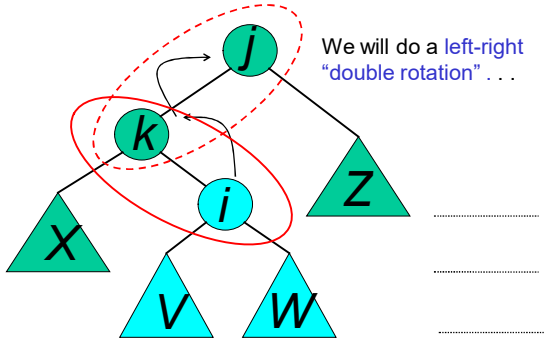
Y = node i and subtrees V and W



31

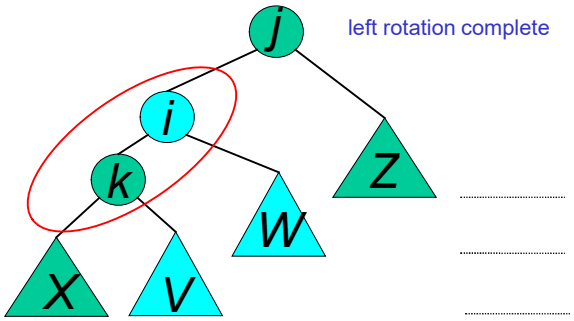
AVL Insertion: Inside Case

We will do a left-right "double rotation" . . .



32

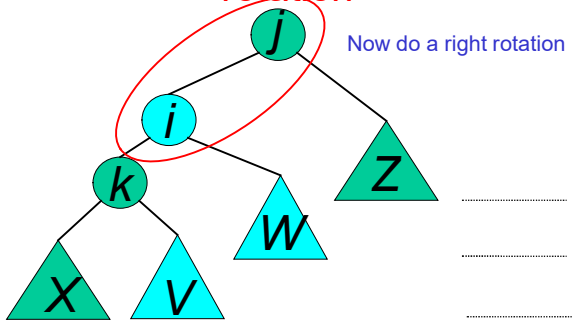
Double rotation : first rotation



left rotation complete

33

Double rotation : second rotation



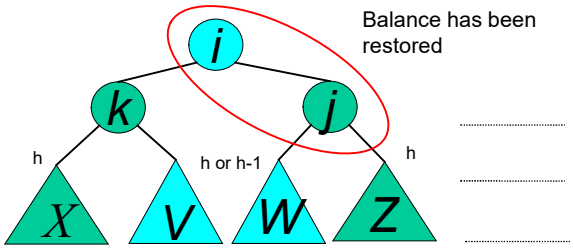
Now do a right rotation

34

Double rotation : second rotation

right rotation complete

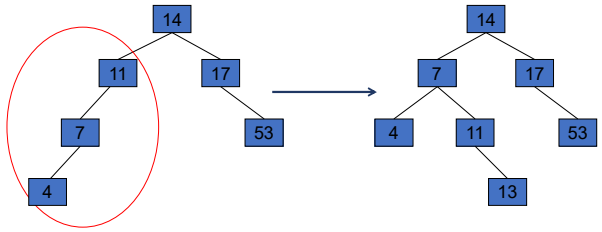
Balance has been restored



35

Example:

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

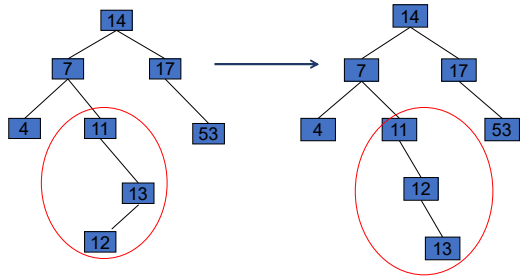


Balancing left subtree nodes - right rotation
Inserting 13

36

Example:

- **Now insert 12**



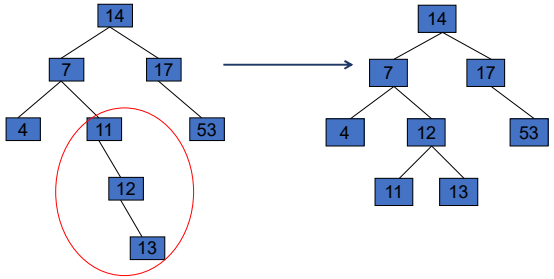
Inserting 12
Balancing subtree- a right rotation

37

37

Example:

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



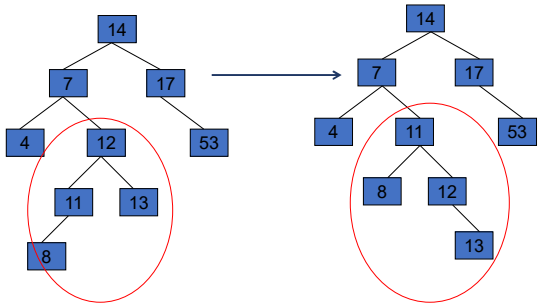
Balancing subtree- double left rotation

38

38

Example:

- **Now insert 8**



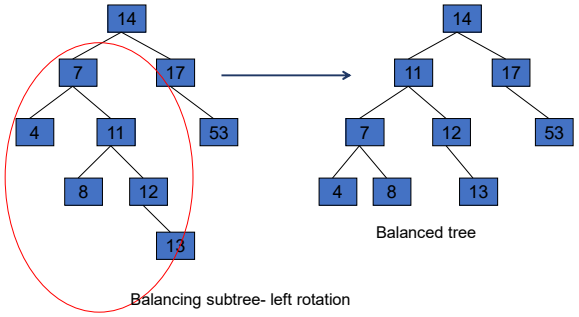
Inserting 8
Balancing subtree- right rotation

39

39

Example:

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



Balanced tree

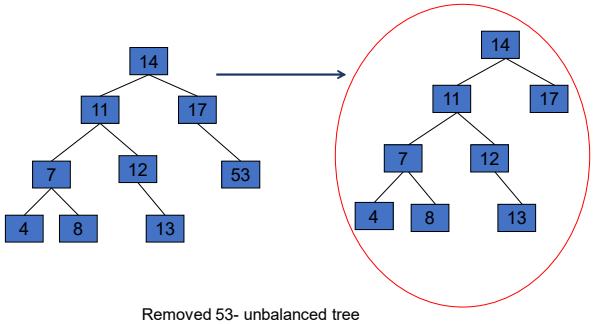
Balancing subtree- left rotation

40

40

Example:

- **Now remove 53**



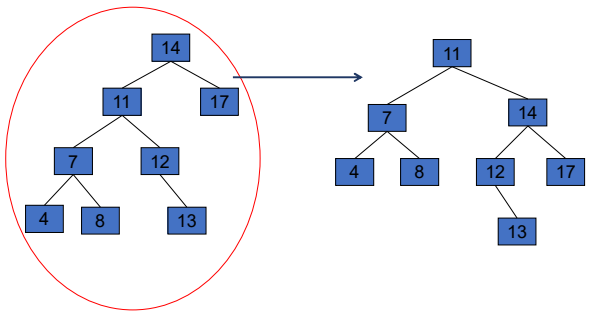
Removed 53- unbalanced tree

41

41

Example:

- **Balanced!**



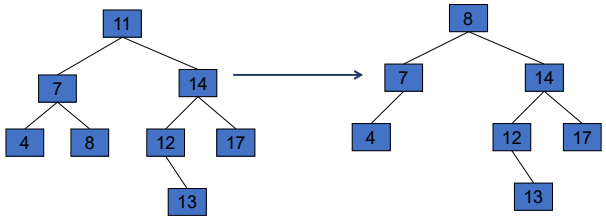
Balanced tree- with single right rotation

42

42

Example:

- **Remove 11**



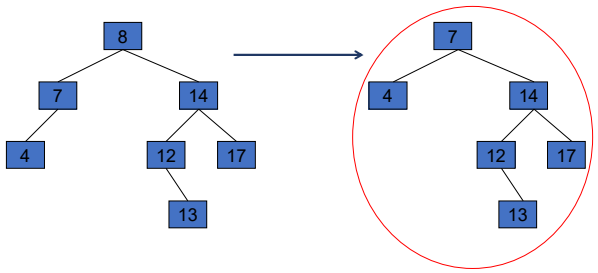
To remove 11- replacing 11 with the largest node in its left branch

43

43

Example:

- **Remove 8**



Remove 8

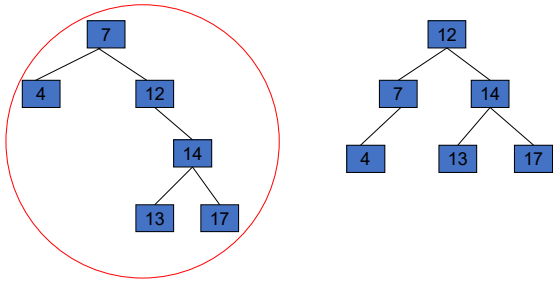
Replacing 8 with its largest node in left branch

44

44

Example:

Balanced!!



Balancing tree - one left rotation

45

45

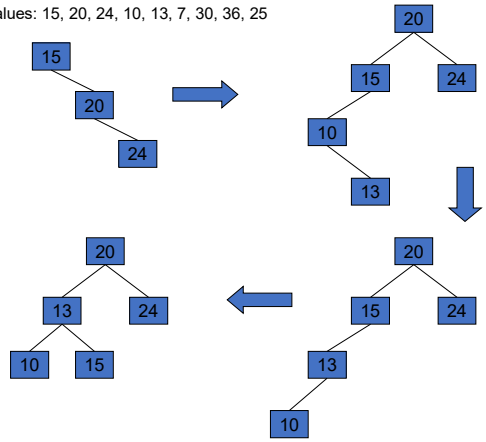
Exercises

- Build an AVL tree with the following values:
  15, 20, 24, 10, 13, 7, 30, 36, 25

46

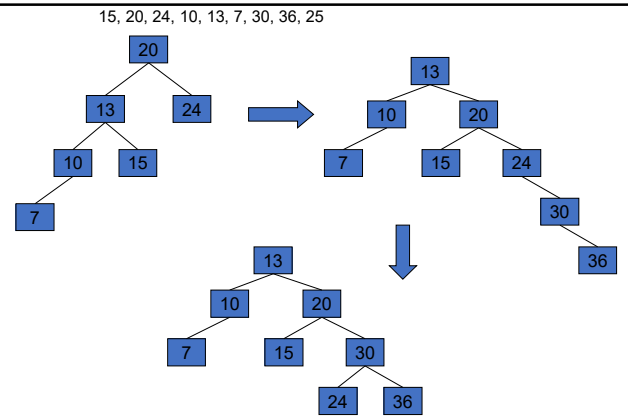
46

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



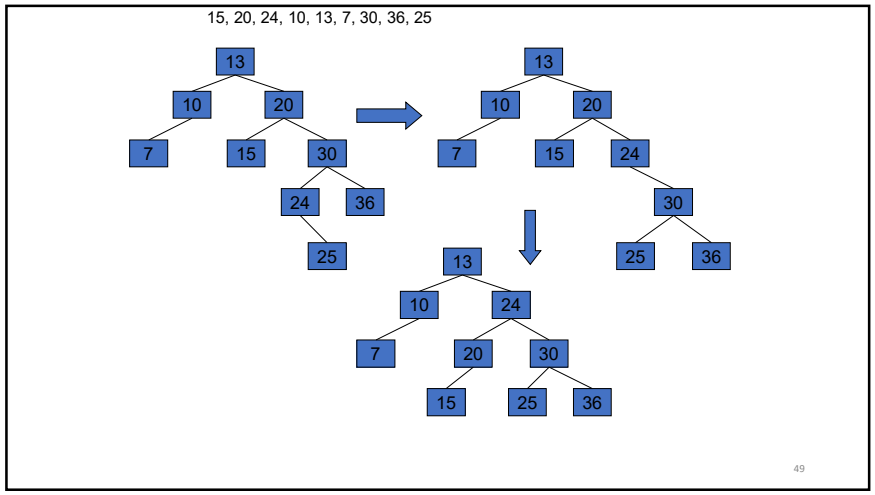
47

47

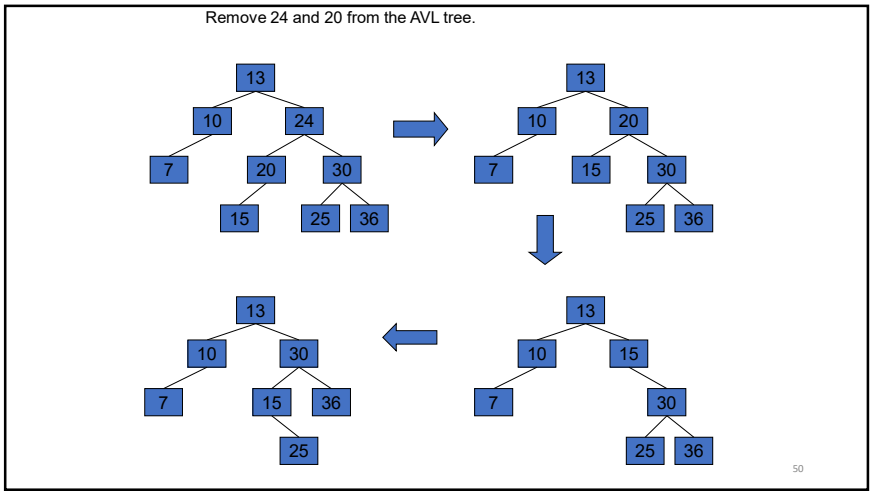


48

48


15, 20, 24, 10, 13, 7, 30, 36, 25
13
10
20
7
15
30
24
36
25
13
10
20
7
15
24
30
25
36
13
10
24
7
20
30
15
25
36
49

49


Remove 24 and 20 from the AVL tree.
13
10
24
7
20
30
15
25
36
13
10
20
7
15
30
25
36
13
10
15
7
30
25
36
13
10
30
7
15
36
25
50

50