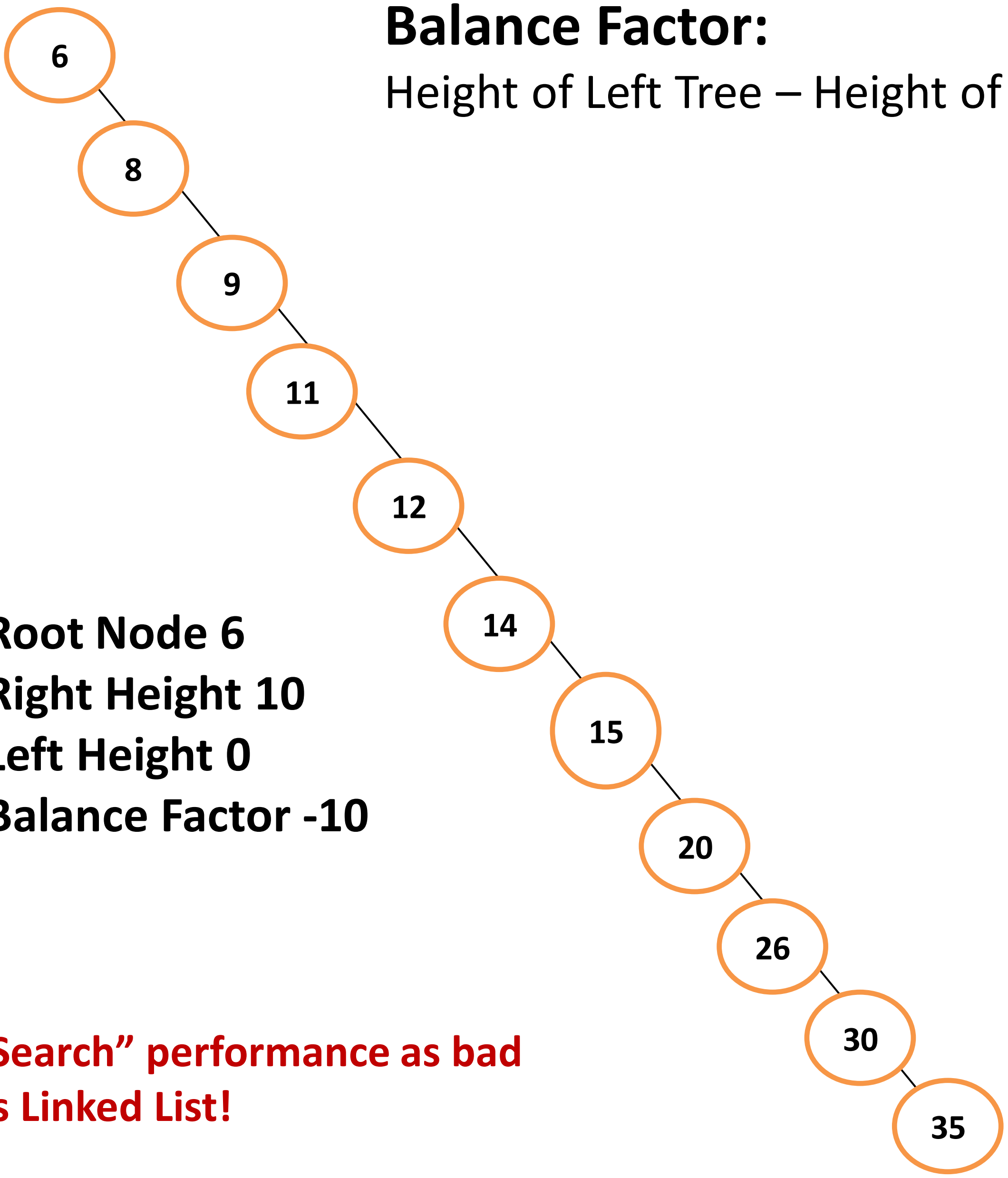
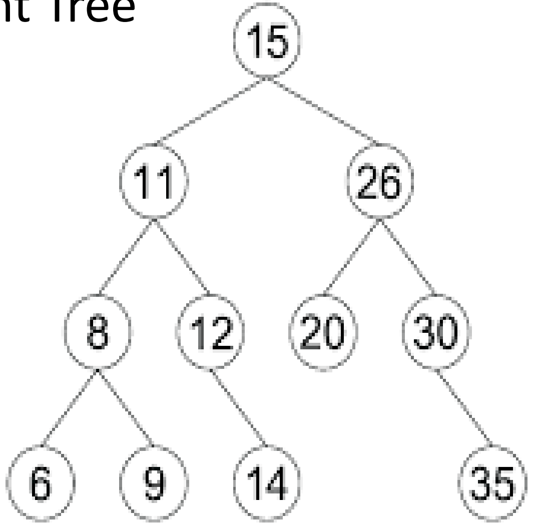
Tree Balancing:

Which BST Operates more effectively?



Balance Factor:

Height of Left Tree – Height of Right Tree



Root Node 6
Right Height 10
Left Height 0
Balance Factor -10

Root Node 15
Right Height 3
Left Height 3
Balance Factor 0

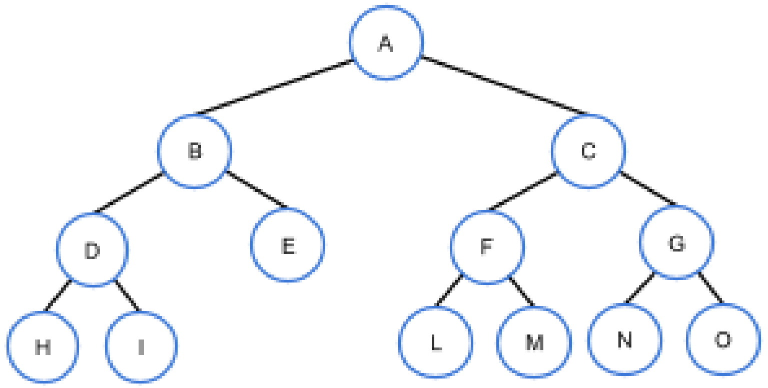
"Search" performance as bad as Linked List!

Binary Tree Terminology: Review

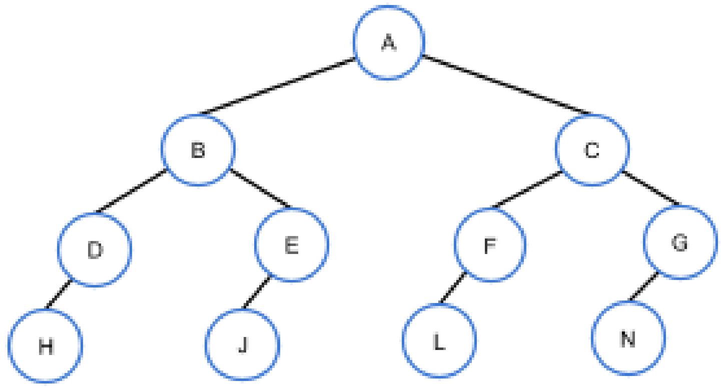
- **Node Depth:** # edges until Root Node
- **Tree Height:** Largest depth of any node
- **Root Node**: Topmost Node, if none, Tree is empty
- **Leaf Node**: Possessing neither an LHC or RHC

- **Complete Tree:** All levels except last contain all possible children (all Leaf nodes within 1 depth)
- **AVL Tree:** For each node, height of left/right subtrees differ by 0 or 1 (Balance).

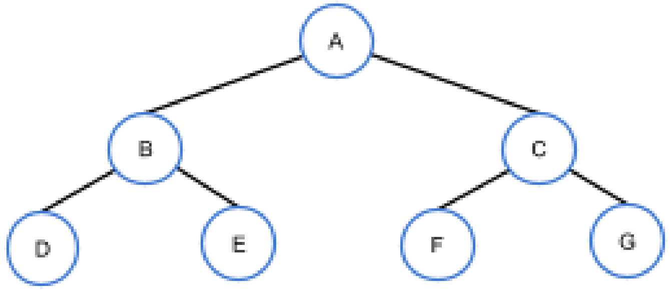
Complete Trees vs. AVL Trees



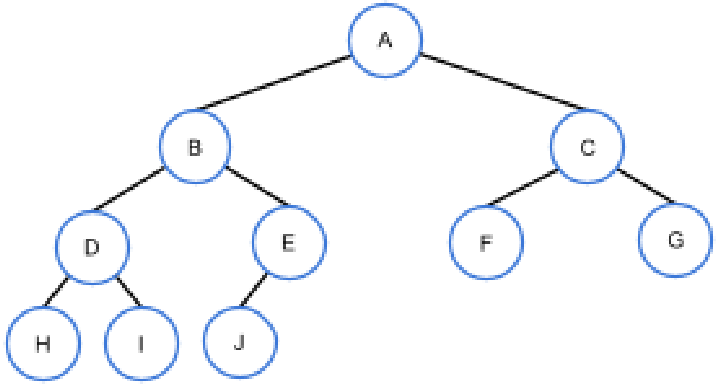
Tree 1



Tree 2



Tree 3

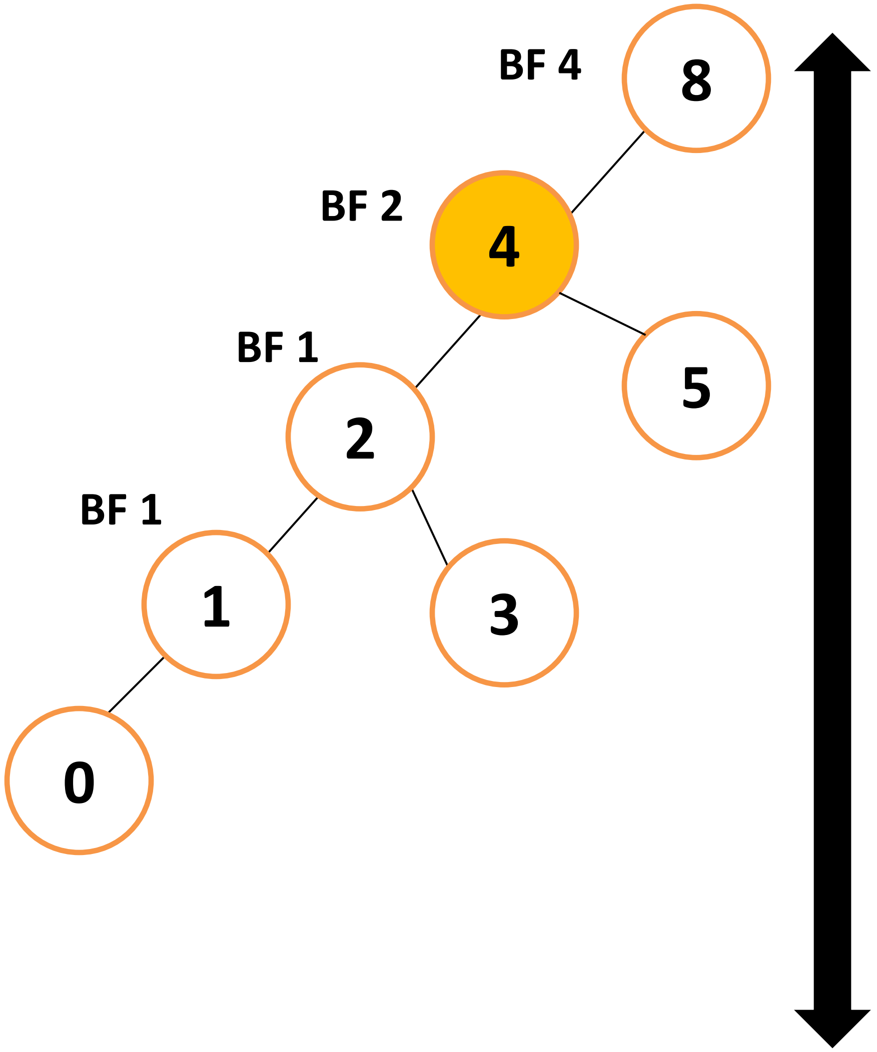


Tree 4

A. Complete, AVL / B. Not Complete, AVL

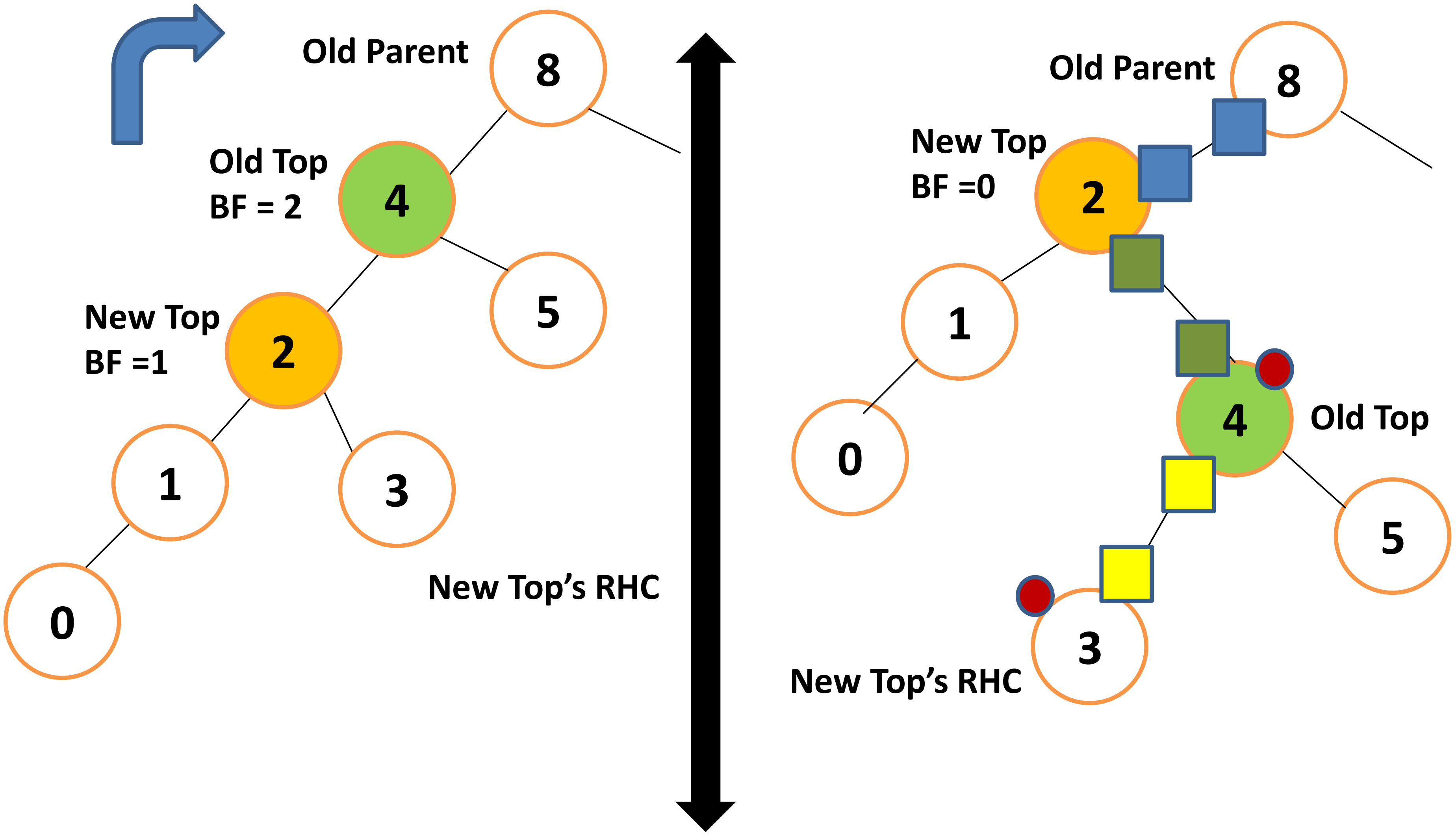
C. Complete, Not AVL / D. Not Complete, not AVL

Balance Check



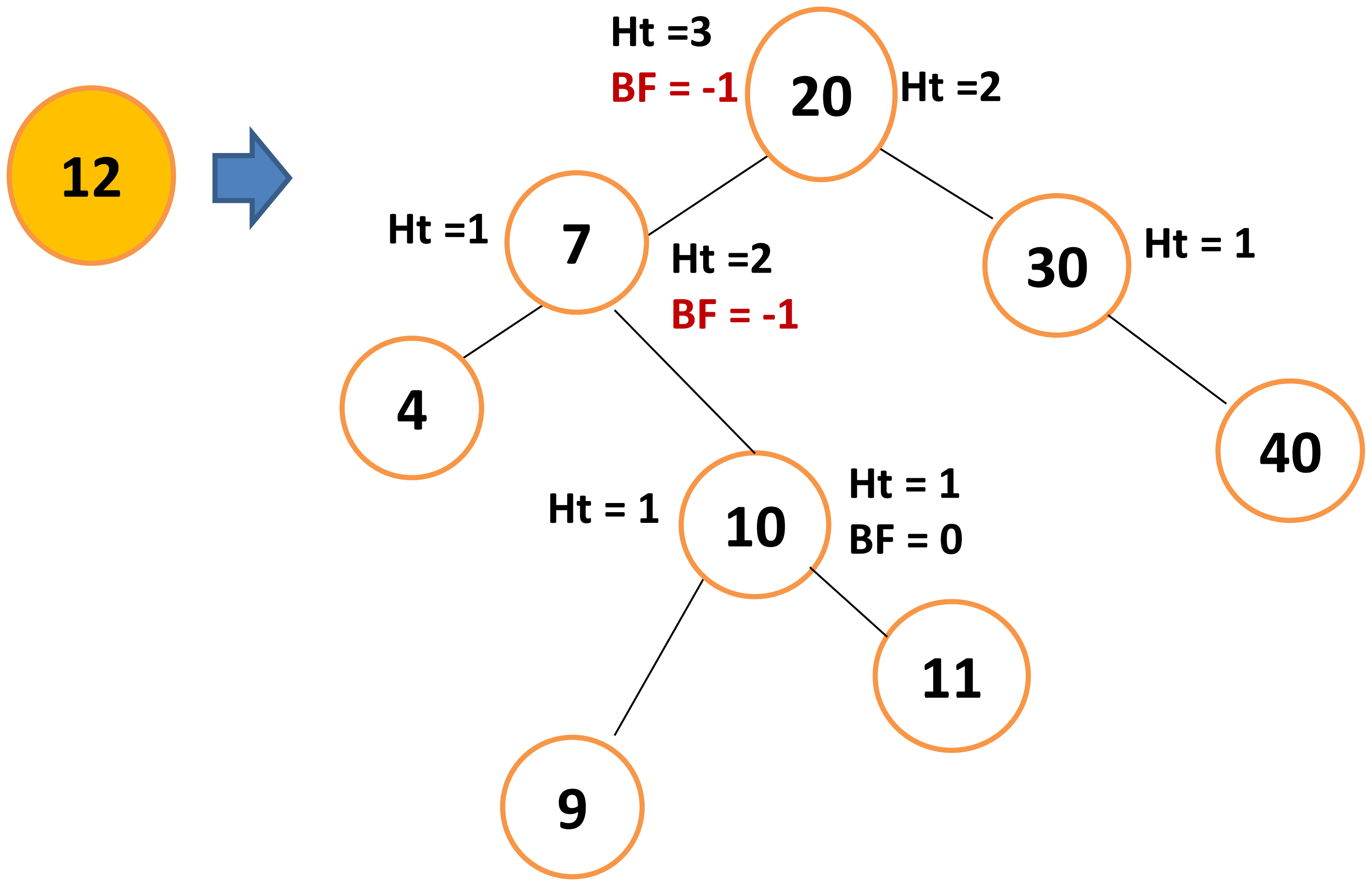
```
int getBalance (node *tnode) {
   int lht = 0;  rht = 0;
   if (tnode->lhc)
        lht =  getHt(tnode->lhc)+1;
   if (tnode->rhc)
        rht =  getHt(tnode->rhc)+1;
   return (lht – rht);
};
// As we know ...

int getHt  (node *tnode) {
    int lht = 0; int rht = 0;
    if ((tnode->lhc)
        lht =  getHt(tnode->lhc)+1;
    if ((tnode->rhc)
        rht =  getHt(tnode->rhc)+1;
    return ((lht > rht)? lht: rht);
};
```

Tree Balancing: Ex: Rotate Right Lowers the Balance Factor

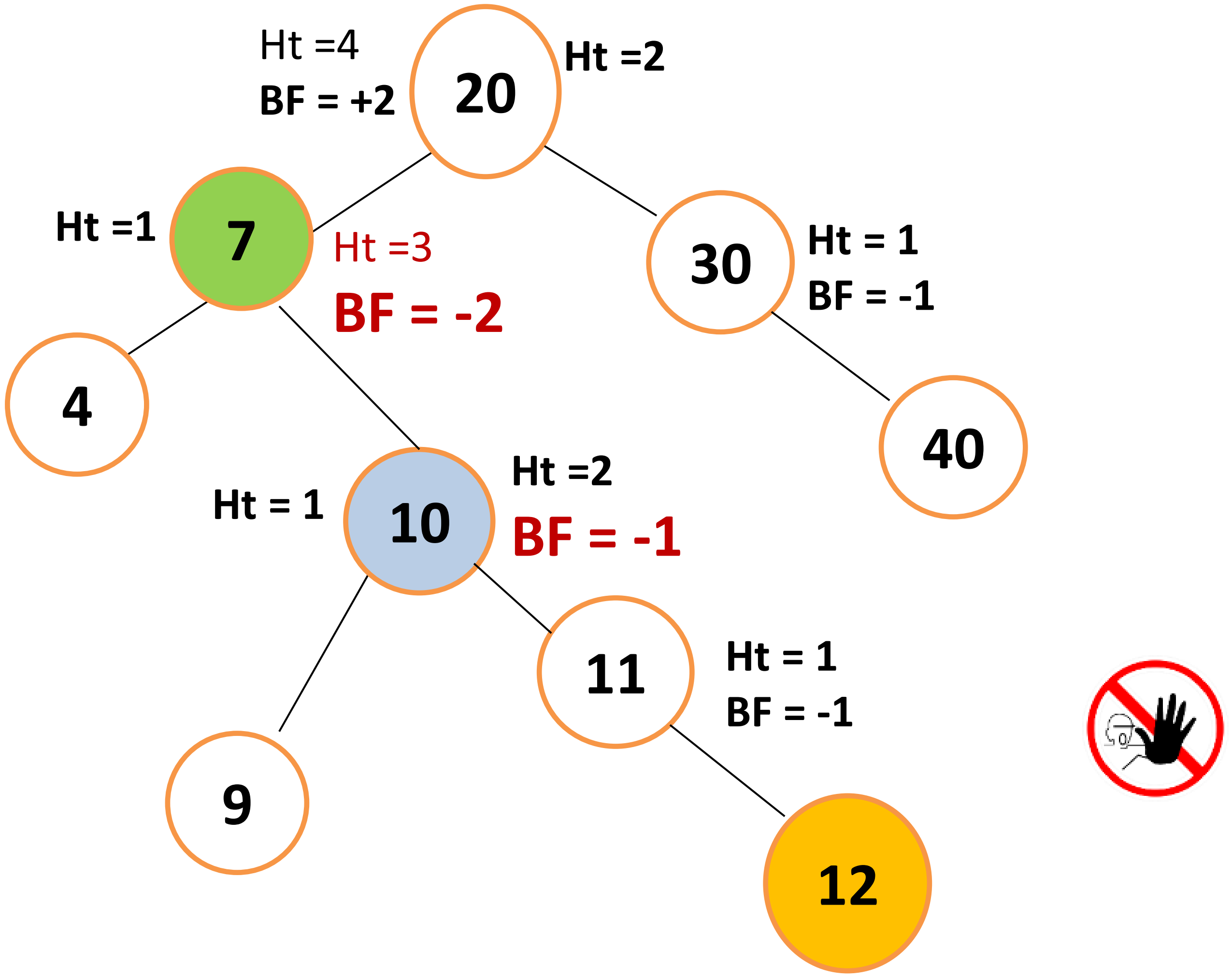


Note: If "Old Top" is Root, Old Parent is BST "toRoot"

Leaf Node Insertion often ➔ Ancestor Rebalancing

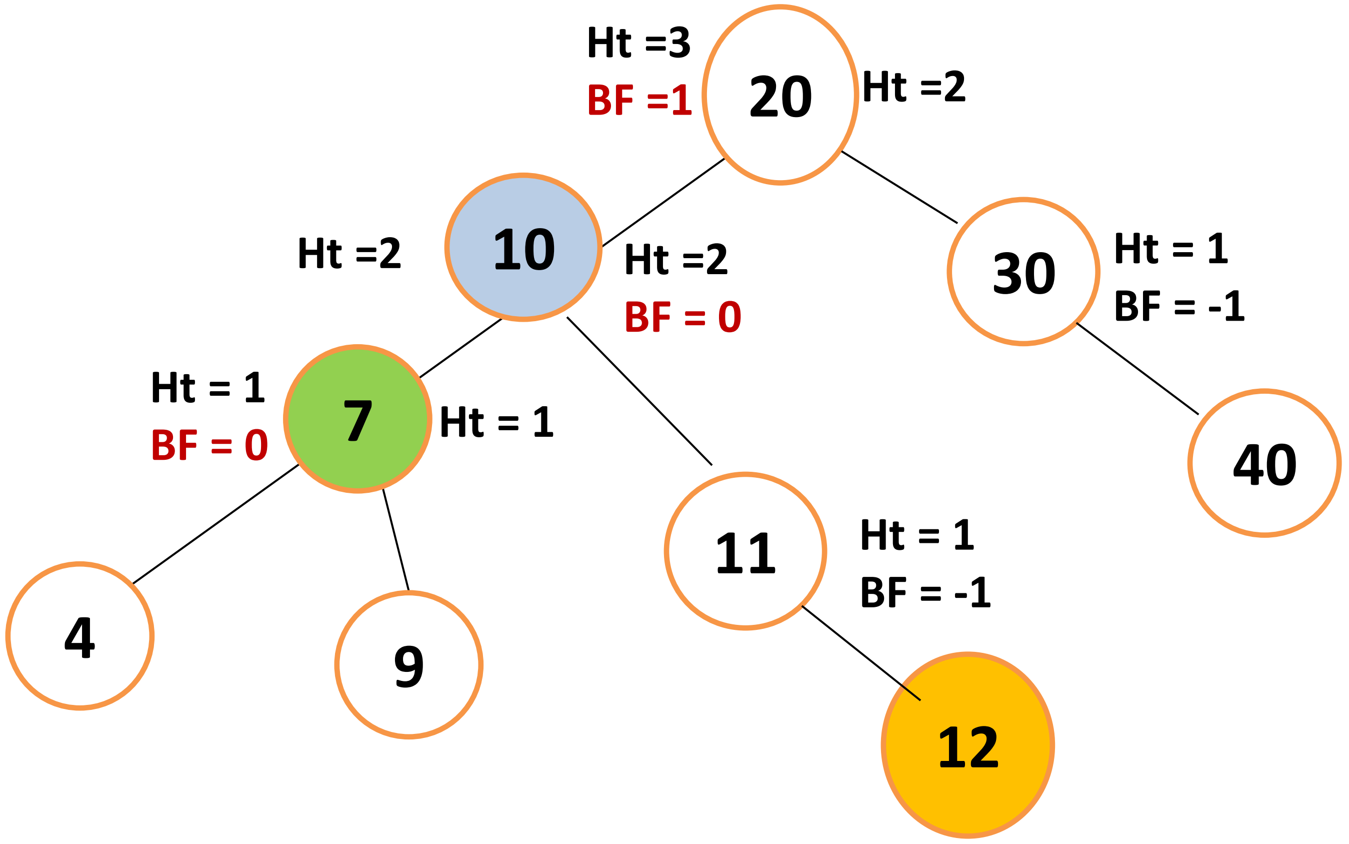


Insertion could ➔ rebalancing ancestors



Which node will need rebalancing after Leaf insertion?
Rotate left or rotate right?

Left Rotation on node 7



Rebalancing BST after Node Insertion

Work your way up from the leaf node just added, computing the BF of each Parent.

If you find a Parent with a BF >1 or < -1, stop. If the tree was previously balanced, there are 4 possibilities for the BF of the Parent and the Child.

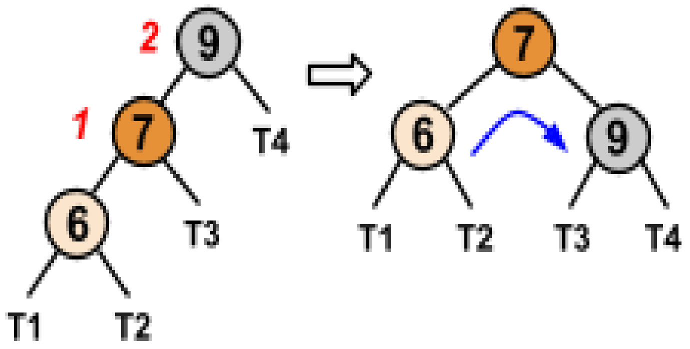
-2, -1 (the case here) // +2, +1 // -2, +1 // +2, -1.

Each requires different node rotations to rebalance the Tree on insertion.

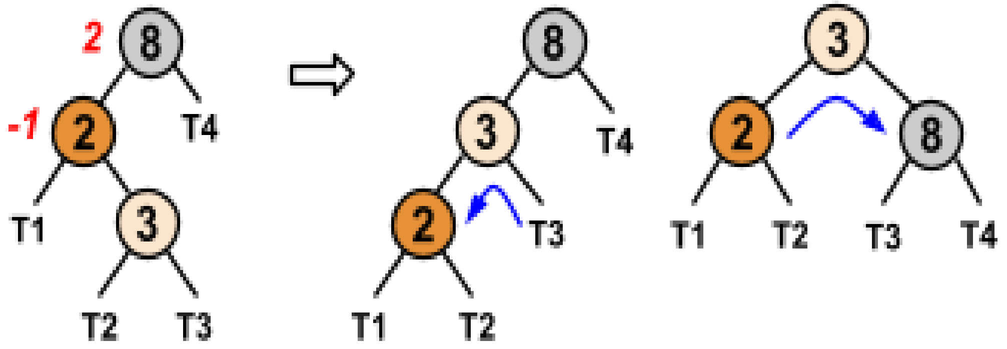
4 cases of Rebalancing

- **Different BF Signs (-2,1 or 2,-1)**
  - Two rotations
  - Child node is rotated down
  - (-2,1) Child weighted to left side. Rotate Right → (2,1)
  - (2,-1) Child weighted to right side. Rotate Left → (-2,-1)
- **Same BF Signs (2,1 or -2,-1)**
  - Only 1 rotation
  - Child node is rotated up
  - (2,1) Parent weighted to left side. Rotate Right
  - **(-2,-1) Parent weighted to Right side. Rotate Left**

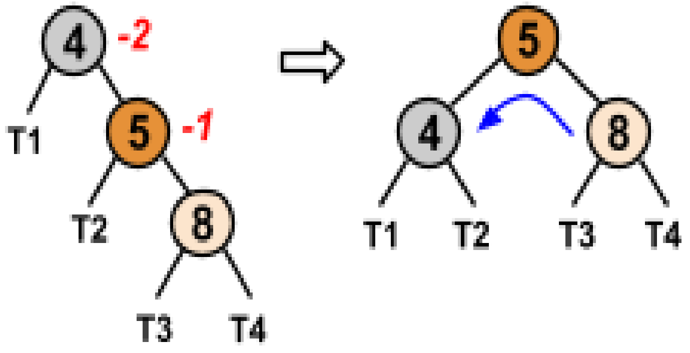
Four cases of Insert → Parent Imbalance (flatten BST)



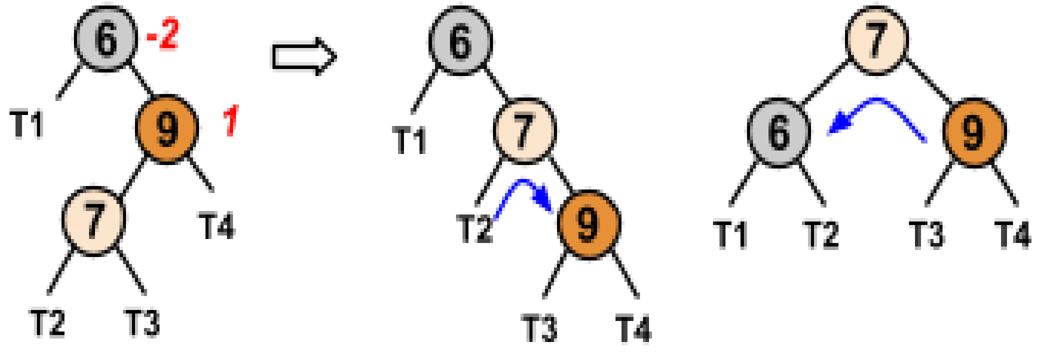
Left-left (2, 1) case



Left-right (2, -1) case



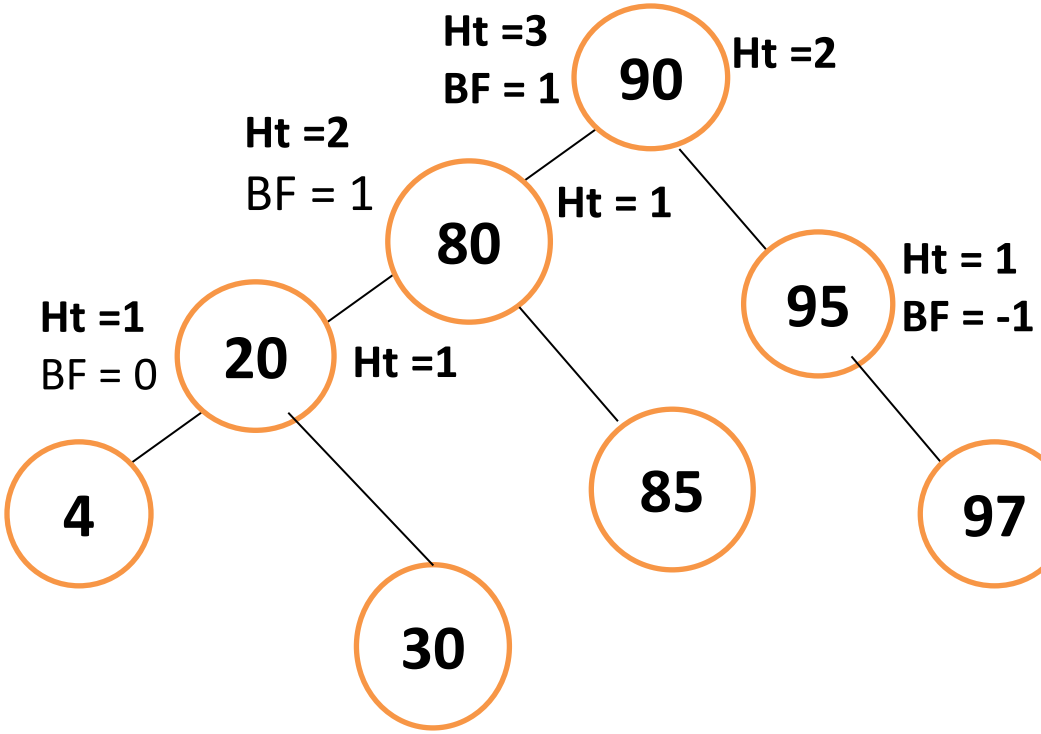
Right-right (-2, -1) case



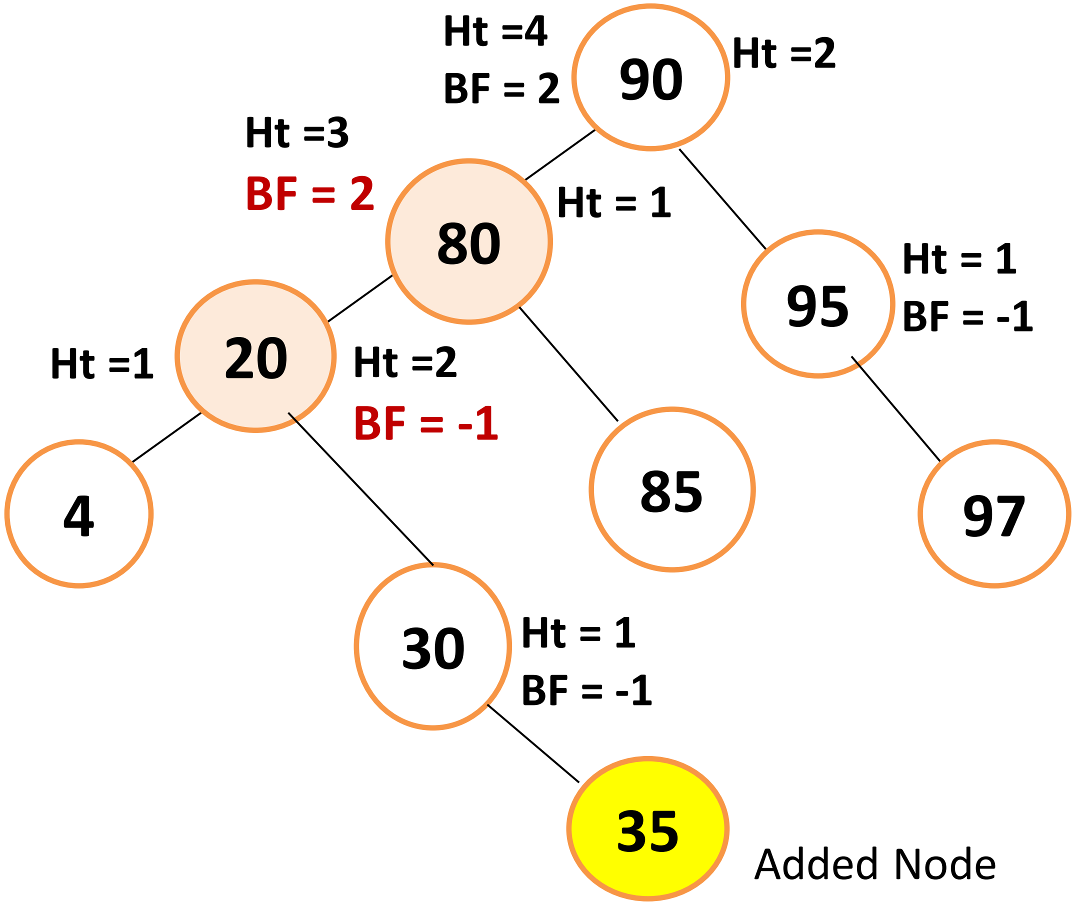
Right-left (-2, 1) case

Preexisting BST

Every Node BF = {-1. 0, +1}



Leaf Node insertion of (35) complete

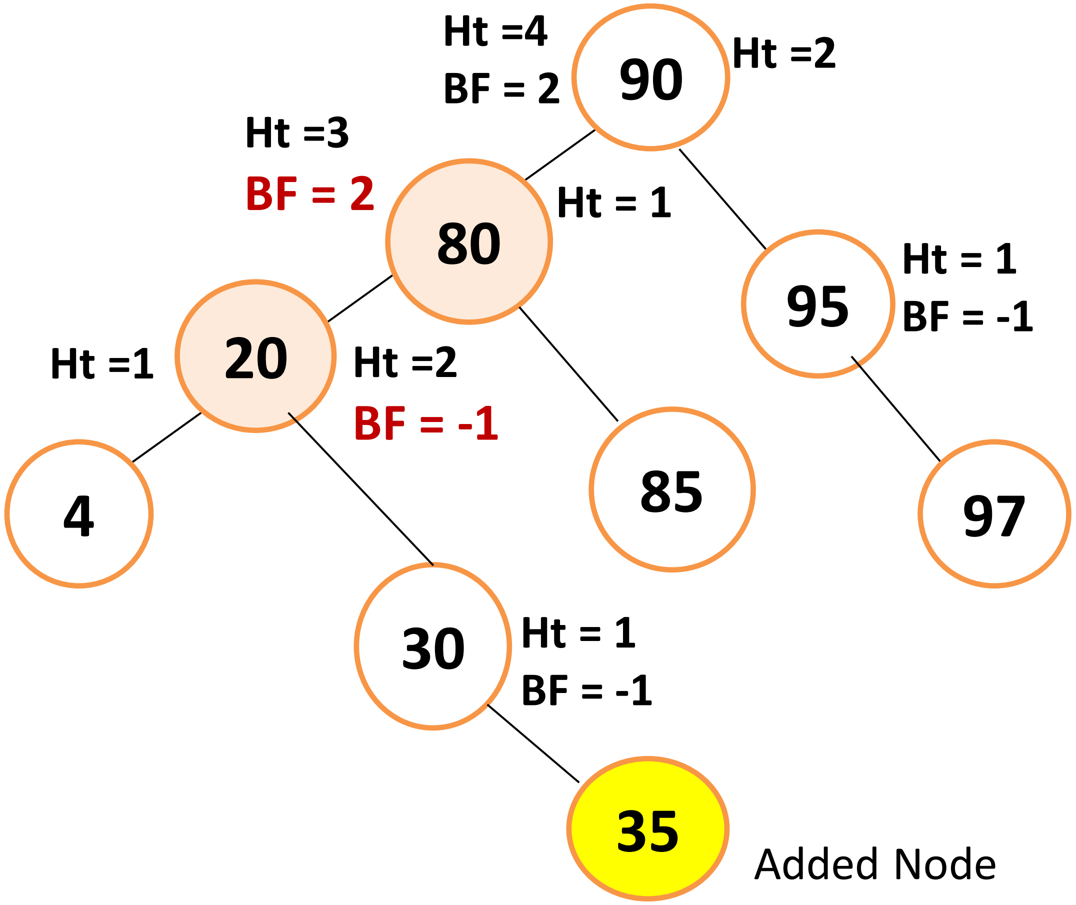


We have a BF >1. This means BST is overstocked to the left.

What is the Parent?
What is the Child?

Which turns needed for rebalancing (2,-1)?

Leaf Node insertion of (35) complete



BST is overstocked to the left.
What is the Parent?
What is the Child?

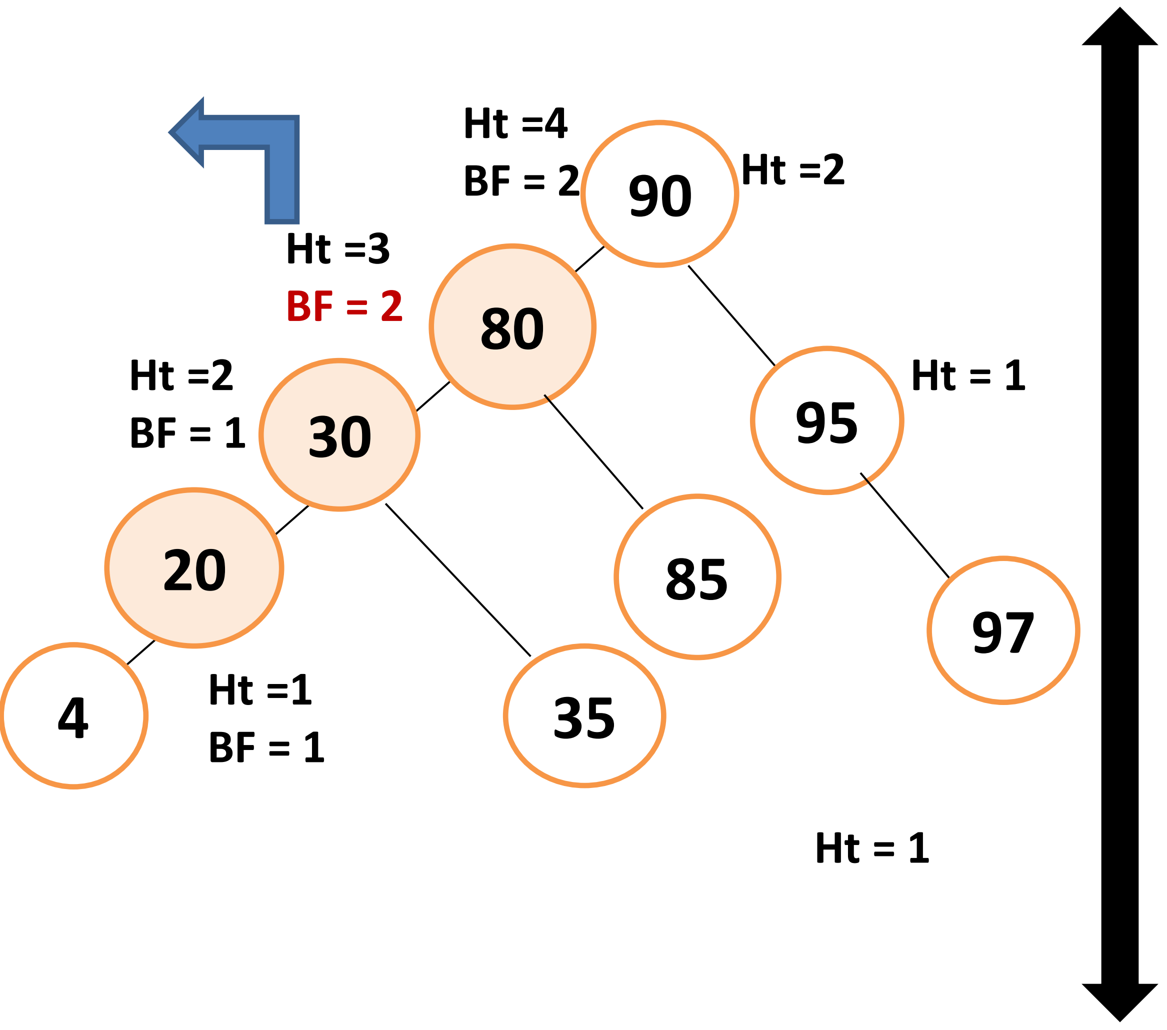
Parent 80 (BF = 2)
LHC 20 (BF = -1)

Which rotations needed for rebalancing (2,-1)?

Left, Right:

Start by rotating LHC 20 down to the left

Double Rotation: Step 1 - Left



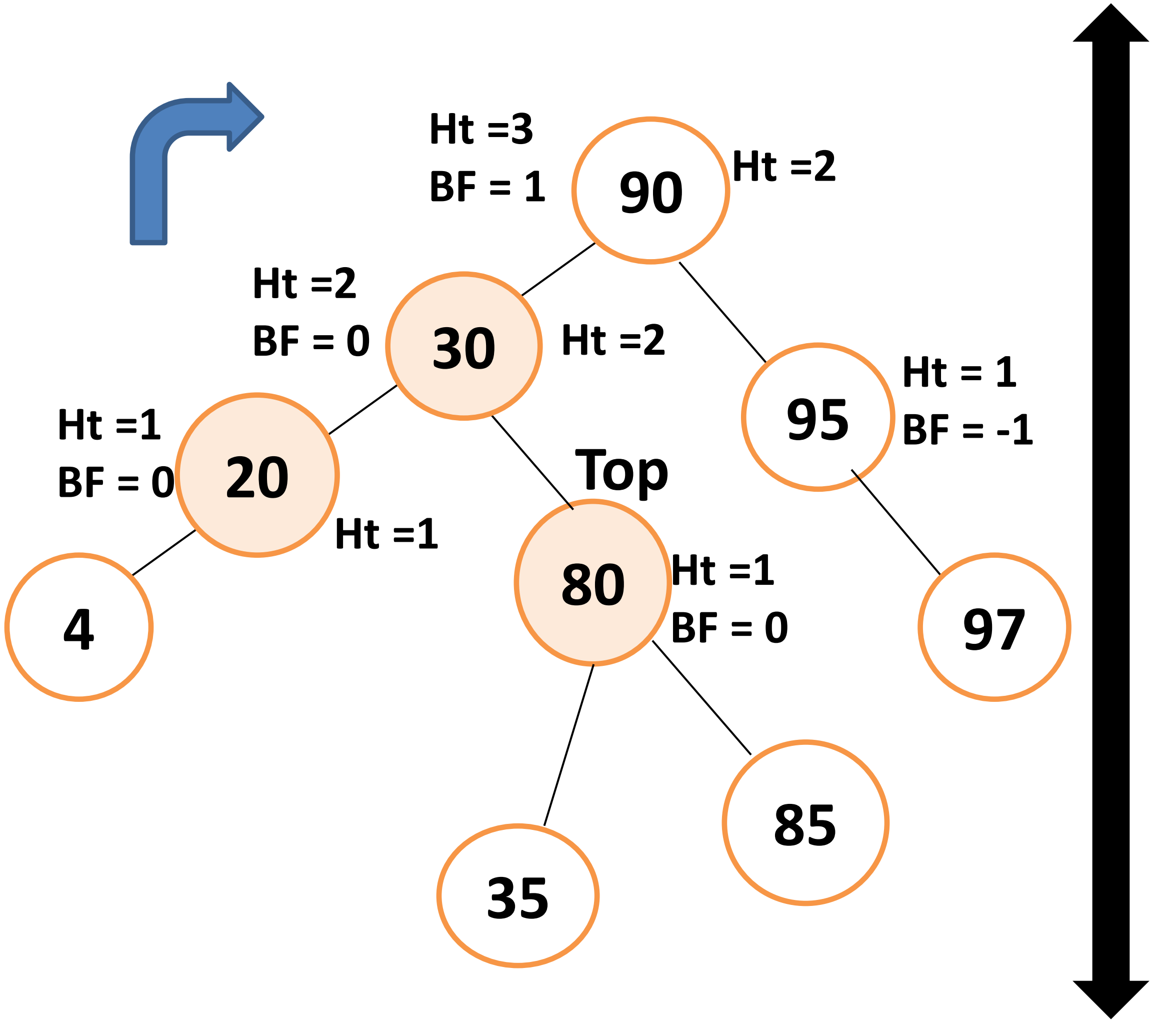
Parent 80’s BF still >1. This means BST is still overstocked to the left.

Parent 80 BF and new LHC 30’s BF’s are now the same sign!

What single rotation is needed to balance (2,1)?

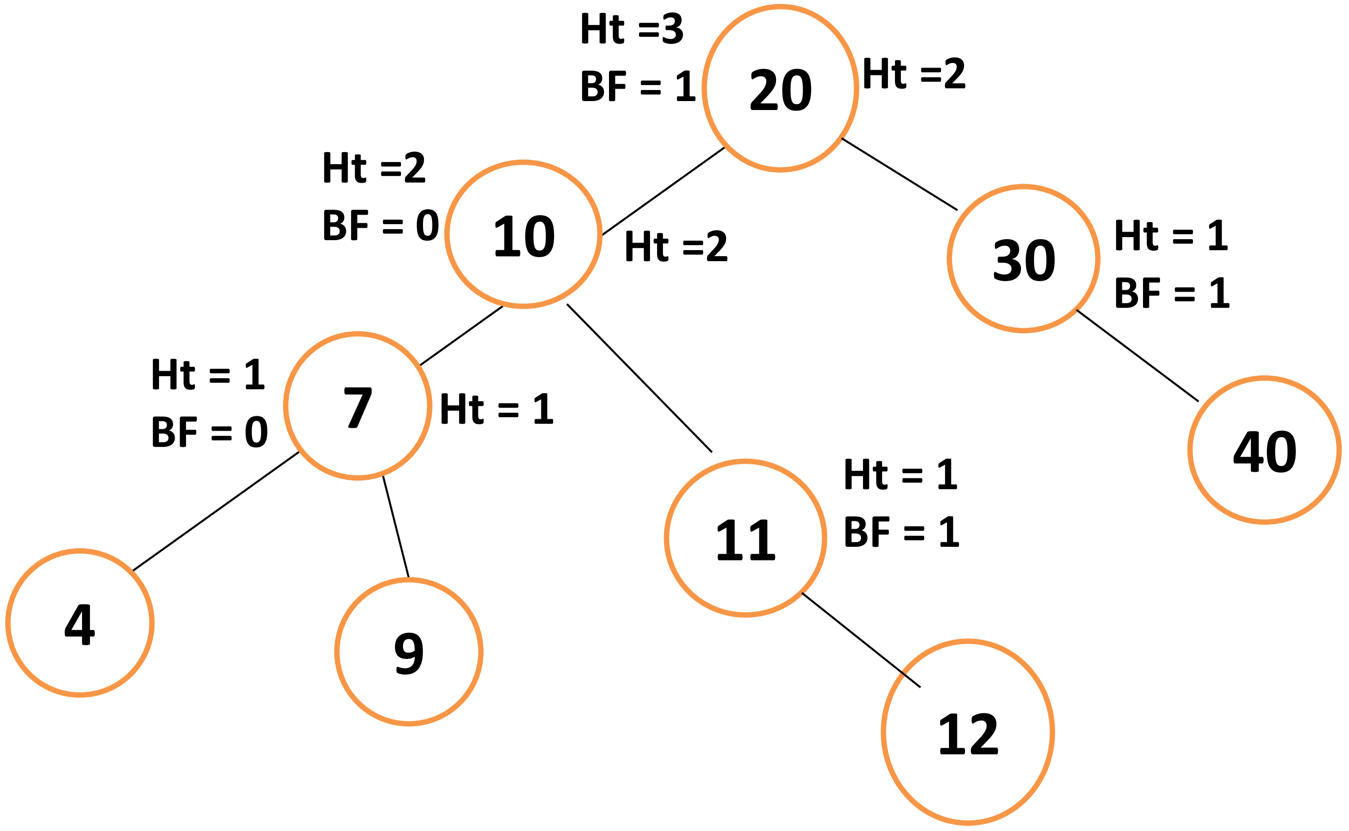
Right, with child node being rotated up

Double Rotation: Step 2 - Right



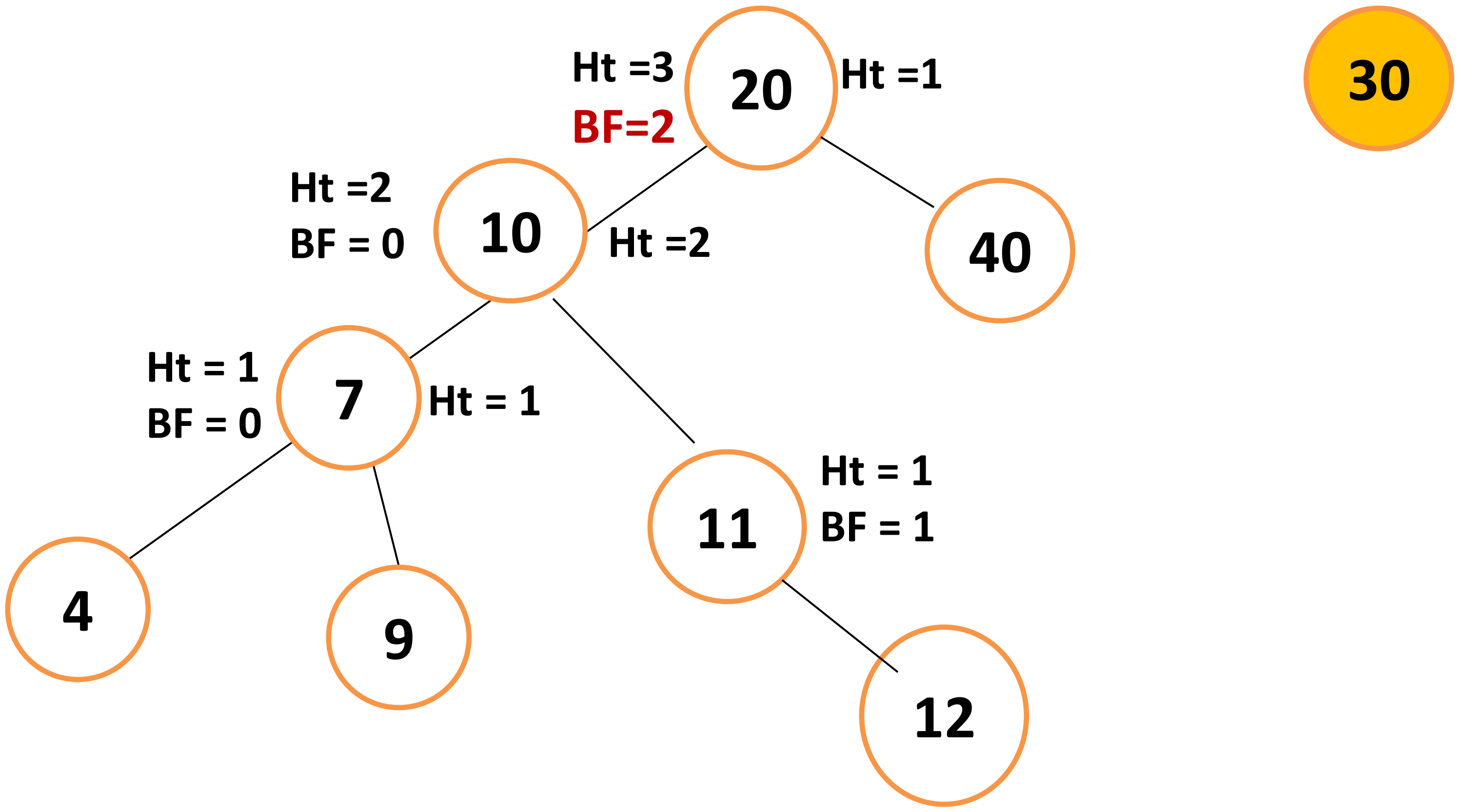
Rebalanced Tree with element 35 added.

Deletion could → ancestor rebalancing



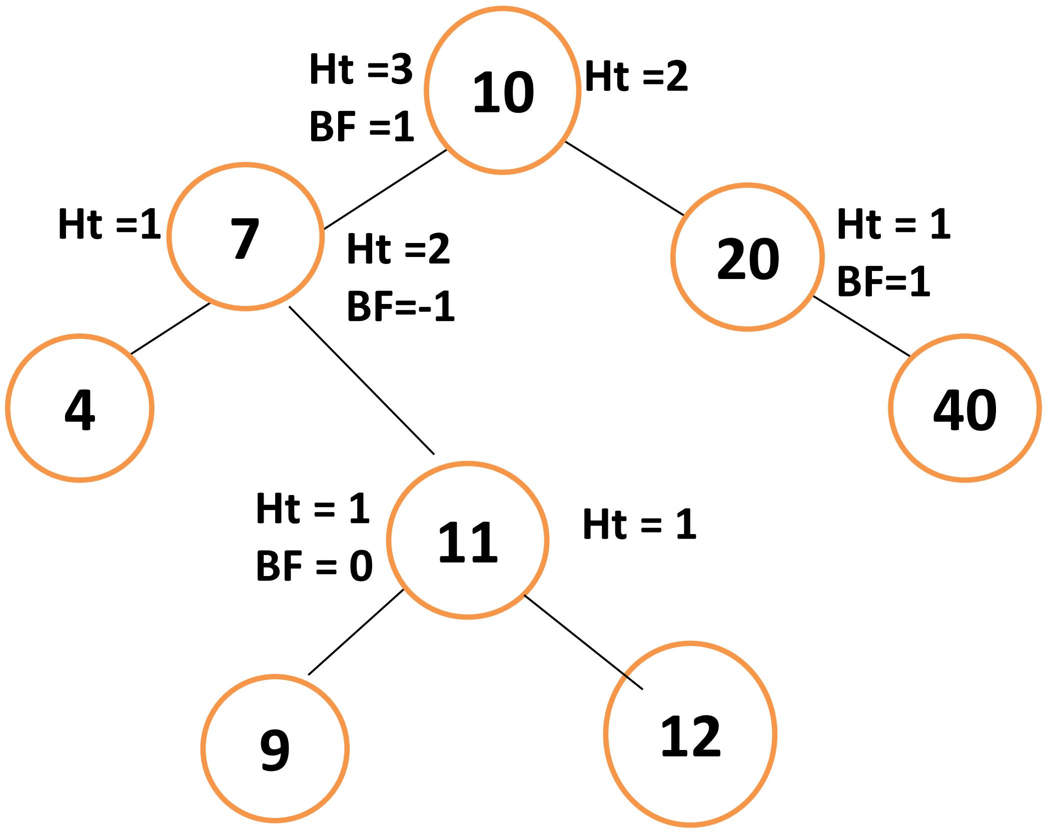
Now delete Node 30

Deletion could ➔ ancestor rebalancing



Deleting Node 30 or 40 causes leftward tilt at ancestor 20. Parent = 20, LHC = 10.
Right Rotate node 10 up

Balanced Tree



Questions?