CS151 Intro to Data Structures

Balanced Search Trees, AVL Trees

Announcements

LAST HOMEWORK - HW8 (AVL Trees) due Thursday 12/11

Warmup / Review

1. Is this tree balanced?
   a. If not, perform a rotation to balance it

2. Insert 5

3. Is this tree balanced?
   a. If not, perform a rotation to balance it

4. insert 13, 21, and 12 (in that order)

5. Is this tree balanced?
   a. If not, perform a rotation to balance it

```
   20
  /
10
```

AVL Trees

AVL Trees

- "*self balancing* binary search tree"
- For every internal node, **the heights of the two children differ by at most 1**
- does rotations upon insert/removal if necessary

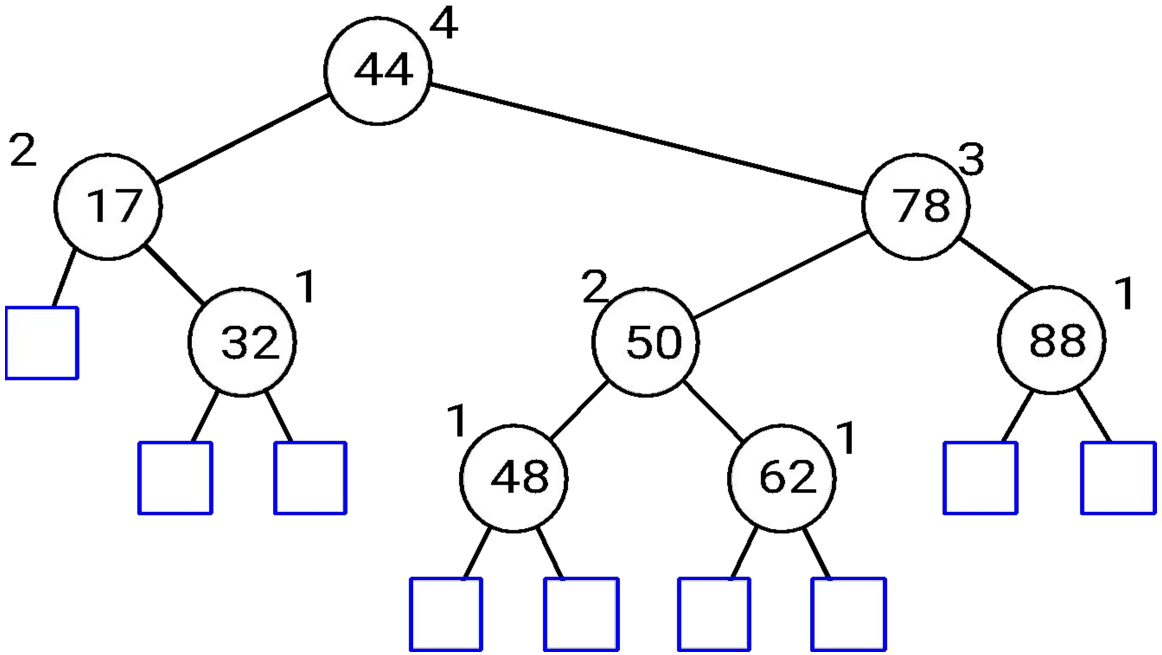
AVL Height

- We keep track of the height of each node as a field for quick access
  - height of a leaf is 1

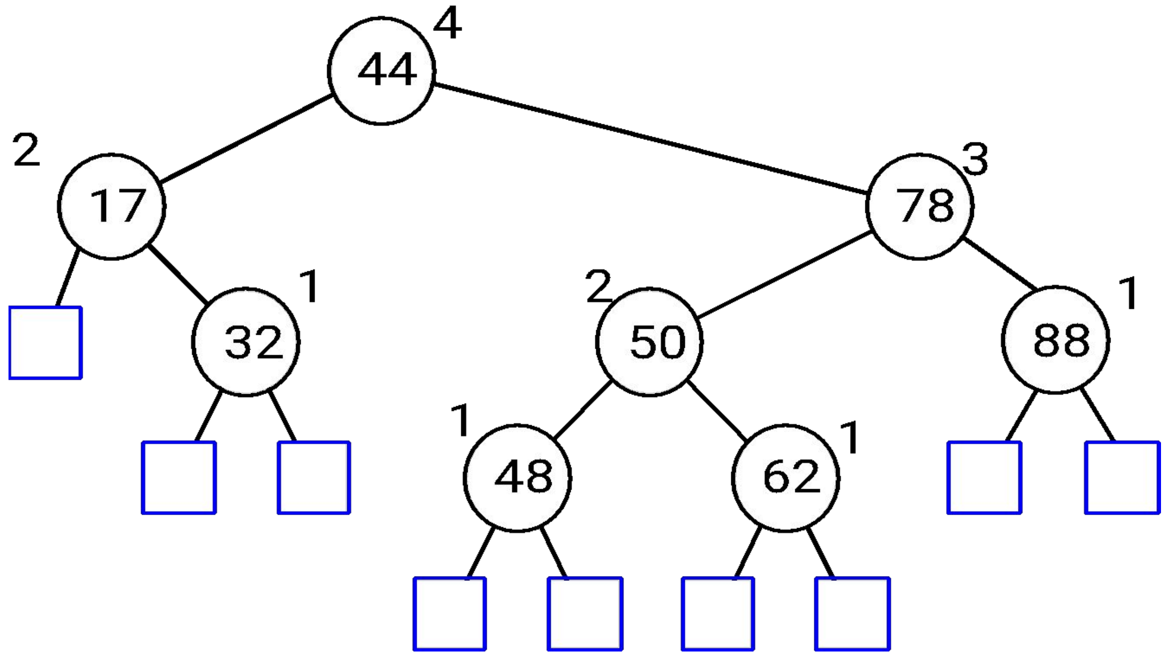
- The height of an AVL tree is logn
  - Always balanced

Insertion

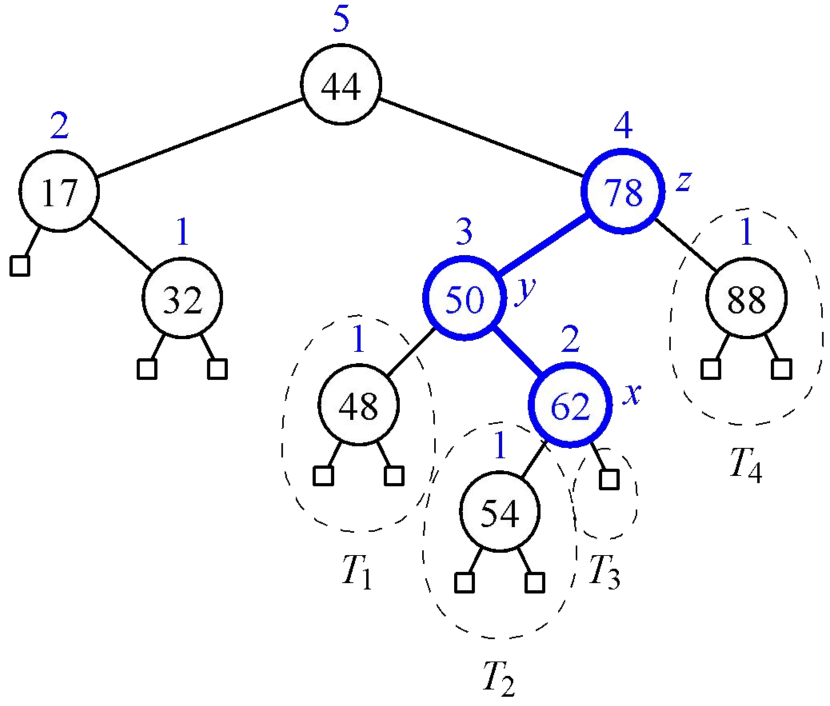
AVL Tree Example

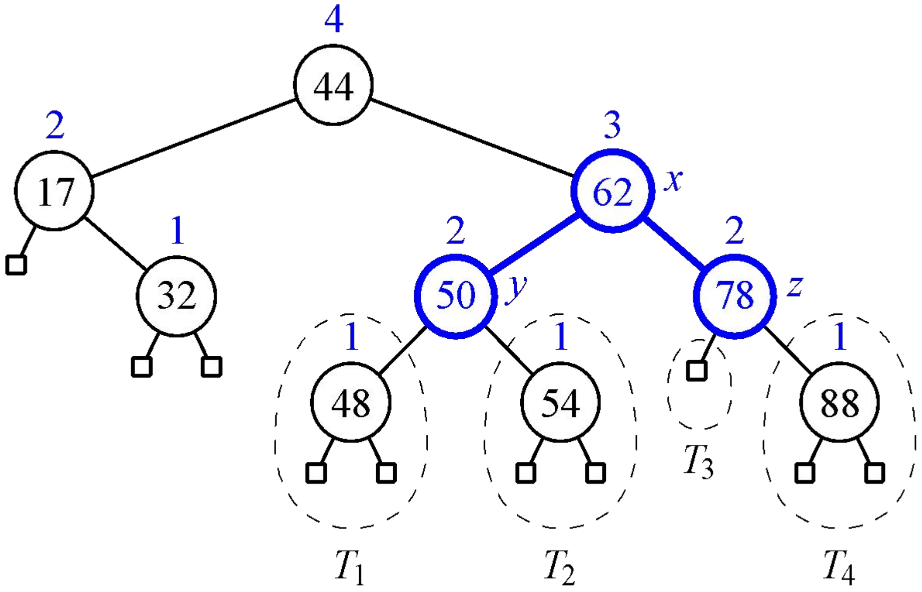


Insert 54



Insertion (54)

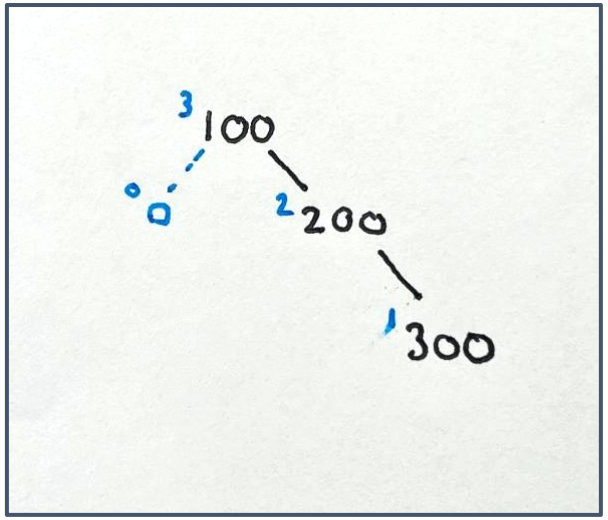


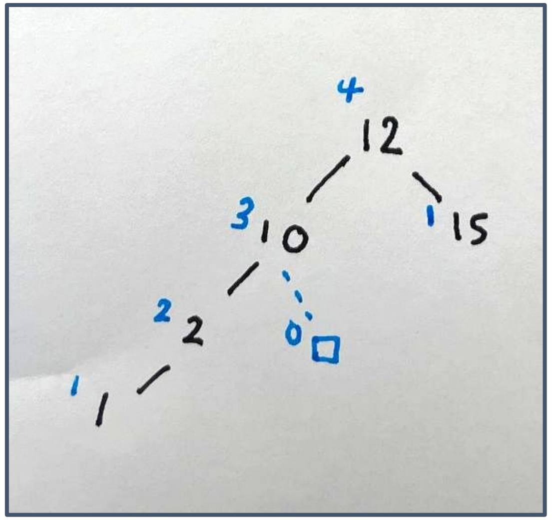


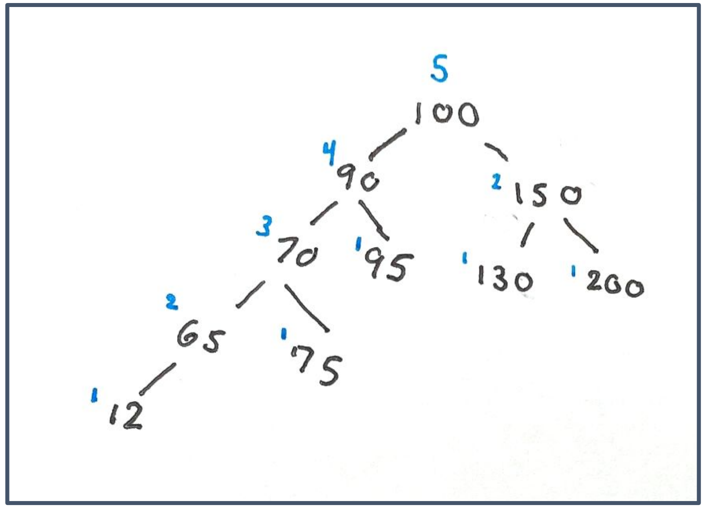
New node always has height 1

Parent may change height

Which node do we “rebalance over”?







lowest subtree with diff(heights) > 1

Exercise

- Create an AVL tree by inserting the nodes in this order:
    - M, N, O, L, K, Q, P, H, I, A

AVL Animation

Rebalance Algorithm

```
If left.height > right.height + 1:
    if (left.right.height > left.left.height)  //double rotate
        rotateLeftRight(n)
    else:
        rotateRight(n)

else if right.height > left.height + 1:
    if (right.left.height > right.right.height) //double rotate
        rotateRightLeft(n)
    else:
        rotateLeft(n)
```

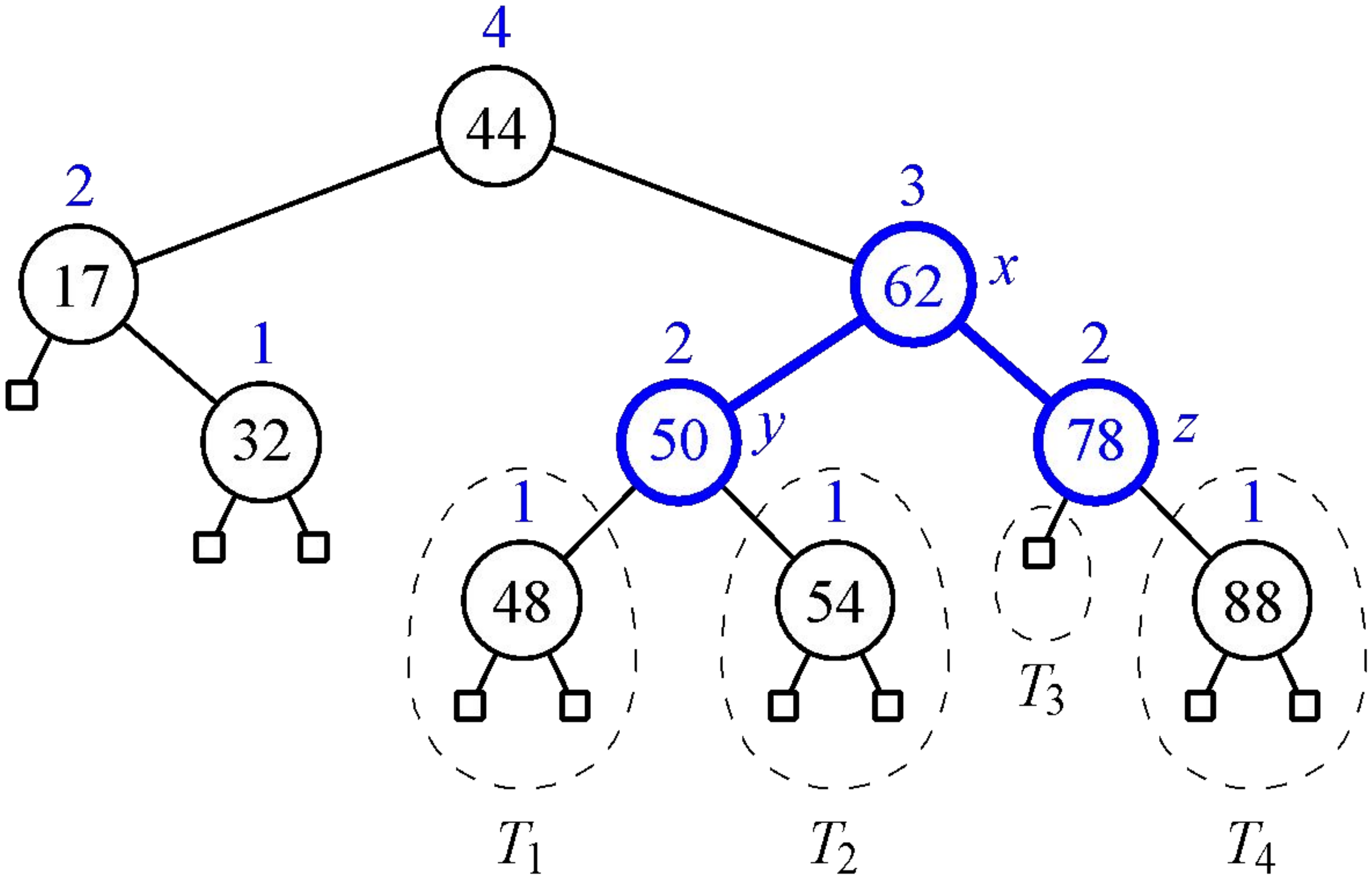
Runtime Complexity:

Insertion (plus rotation)

a. search + find node to rebalance + rotate
b. O(logn) + O(logn) + O(1) = **O(logn)**

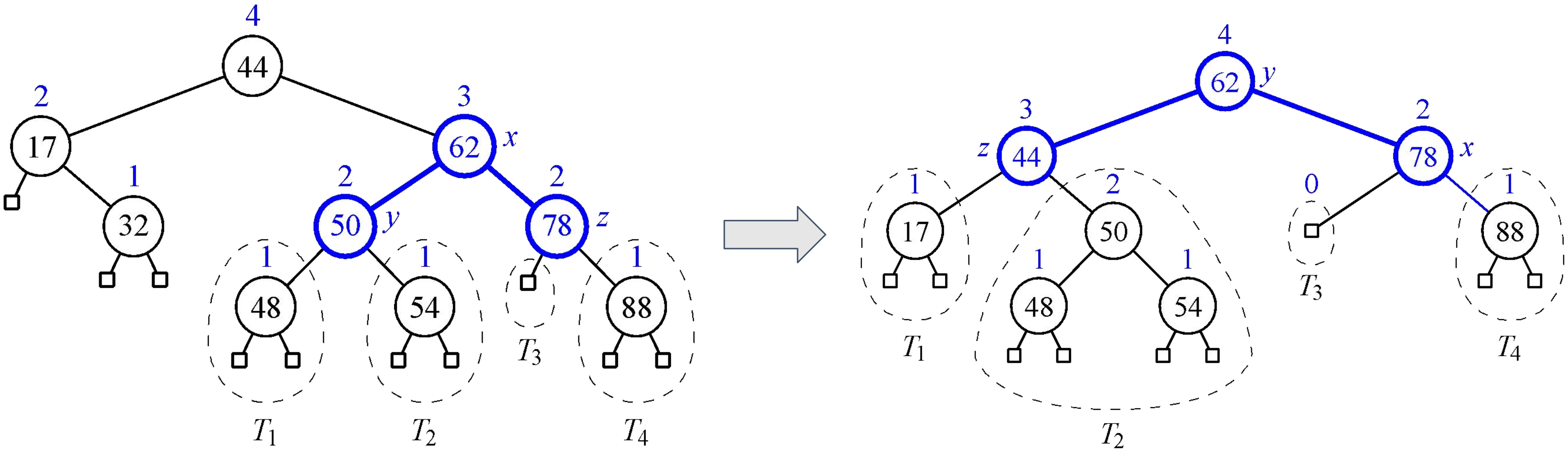
Deletion

Delete Example 1: 32

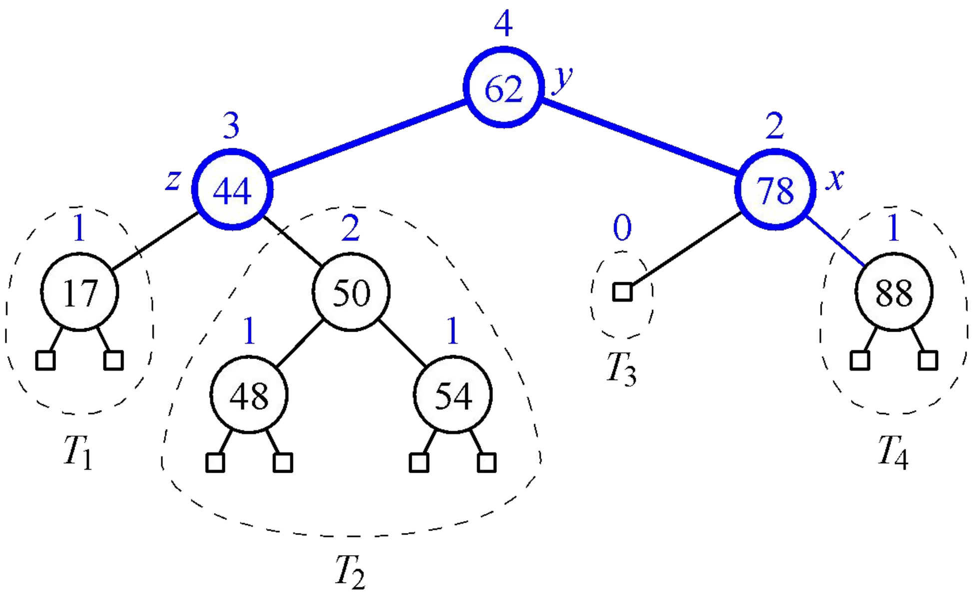


Delete Example 1: 32

rotateLeft

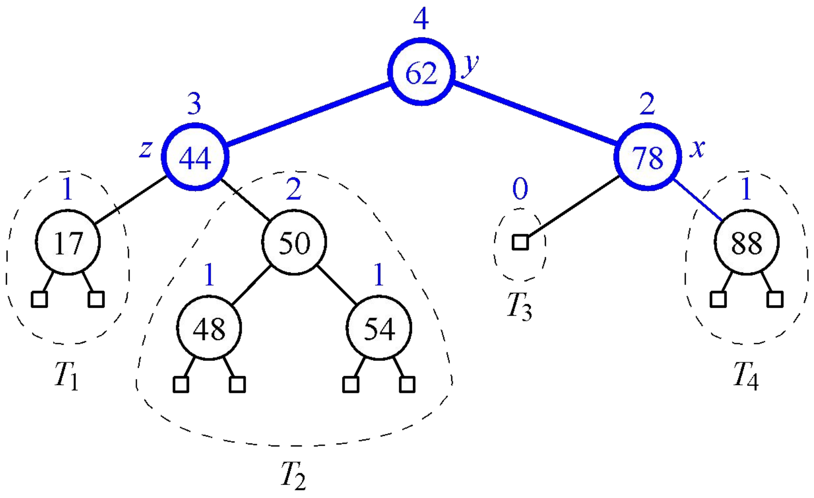


Delete Example 2: 78

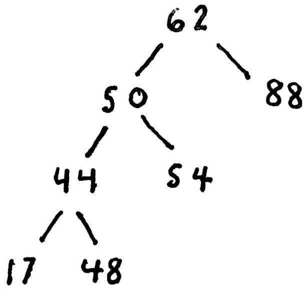


Delete Example 2: 78

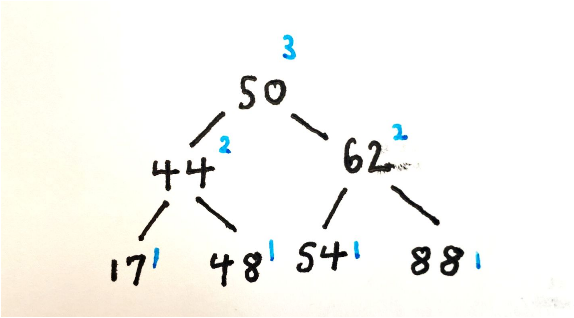
rotateLeftRight



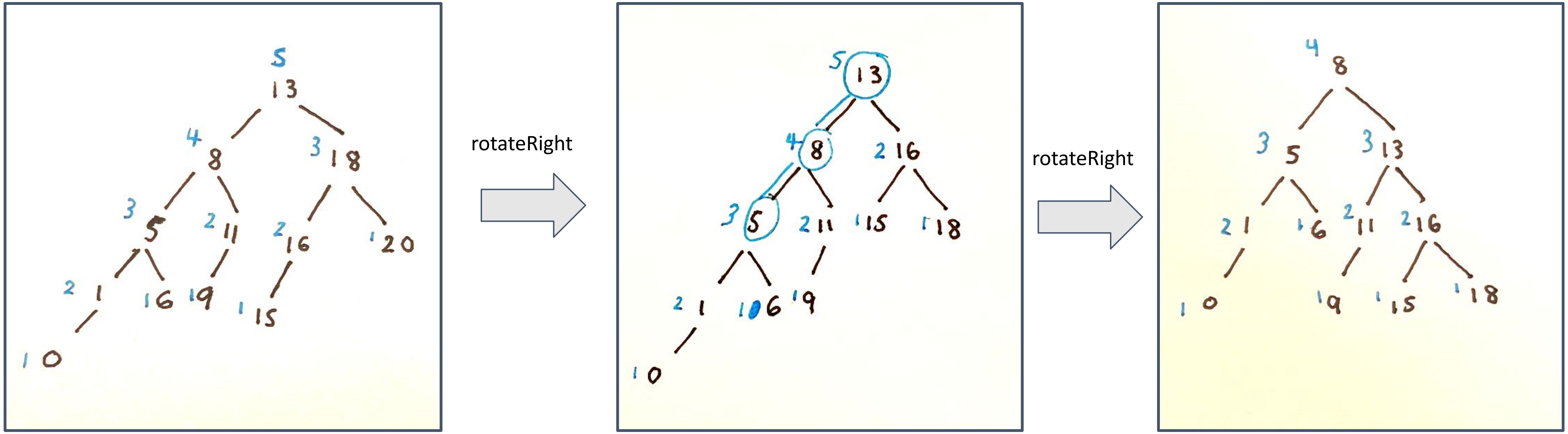
rotateLeft



rotateRight



Delete Example 3: 20



Delete Example 3: 20

- Deletion can cause more than one rotation

- Worst case requires O(logn) rotations
  - deleting from a deepest leaf node and rotating each subtree up to the root

Removal

Runtime Complexity?

a. search + find node to rebalance + rotate
b. O(logn) + O(logn) + O(1) = **O(logn)**

Still O(logn) even though we may need multiple rotations?

Why?

-> Even though we may need to find multiple nodes to rebalance we only traverse the height of the tree once

Performance of BSTs

Runtime complexity:

search?

BST:

O(n)

AVL:

O(logn)

Performance of BSTs

Runtime complexity:

insert?

- BST:
  - O(n)
- AVL:
  - O(logn)

Performance of BSTs

Runtime complexity:

remove?

- BST:
  - O(n)
- AVL:
  - O(logn)

Summary

AVL Trees:

- BST with a rotate operation which maintains tree balance
- O(logn) operations

Rotations:

- double rotation needed when
  - Left subtree is too heavy on the right side OR
  - Right subtree is too heavy on the left side (zig-zag pattern)

Rotations are constant time