

## CS 331 — AVL Trees

Dr. Mattox Beckman

Illinois Institute of Technology  
Department of Computer Science

Spring, 2012

- 1 Introduction
  - Objectives

## 2 Balancing Trees

### 3 Rotations

- Explain why height-balanced trees are necessary.
- Explain how to perform two of the four kinds of rotations:
  - left, right
- Identify the proper kind of rotation for a particular situation.

## Motivation

Do you remember how long it takes...

- To insert an element into a BST on average?
- To insert an element into a BST worst case?
- To delete an element from a BST on average?
- To delete an element from a BST worst case?

## Motivation

Do you remember how long it takes...

- To insert an element into a BST on average? //  $\mathcal{O}(\lg n)$
- To insert an element into a BST worst case? //  $\mathcal{O}(n)$
- To delete an element from a BST on average? //  $\mathcal{O}(\lg n)$
- To delete an element from a BST worst case? //  $\mathcal{O}(n)$

## Some Good Insertions

## Insert These Nodes

8 6 16 30 7 2 12

## Some Good Insertions

## Insert These Nodes

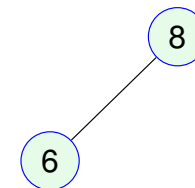
8 6 16 30 7 2 12



## Some Good Insertions

## Insert These Nodes

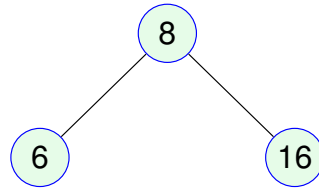
8 6 16 30 7 2 12



## Some Good Insertions

Insert These Nodes

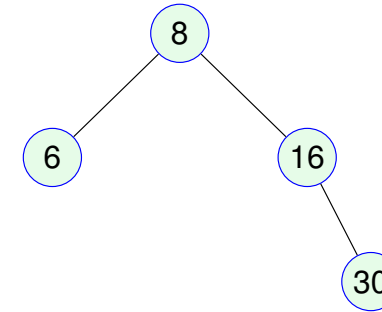
8 6 16 30 7 2 12



## Some Good Insertions

Insert These Nodes

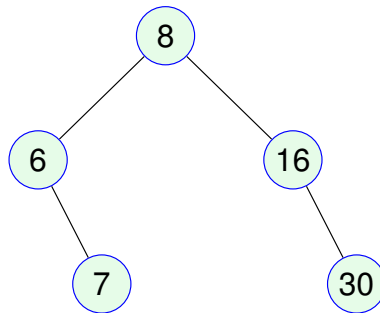
8 6 16 30 7 2 12



## Some Good Insertions

Insert These Nodes

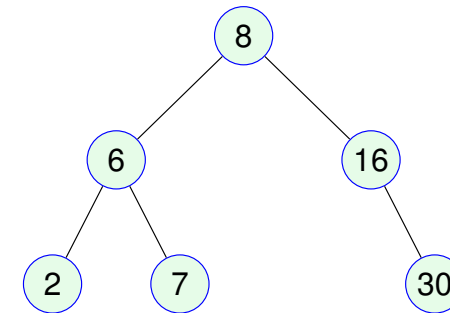
8 6 16 30 7 2 12



## Some Good Insertions

Insert These Nodes

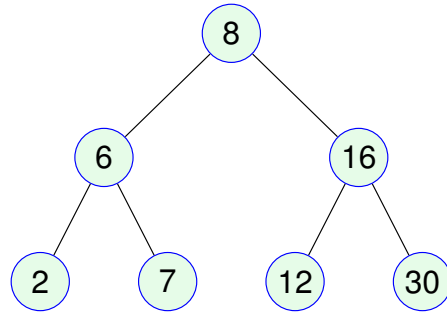
8 6 16 30 7 2 12



## Some Good Insertions

Insert These Nodes

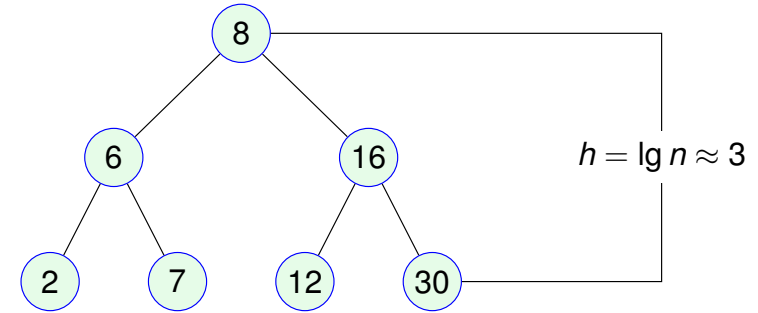
8 6 16 30 7 2 12



## Some Good Insertions

Insert These Nodes

8 6 16 30 7 2 12



## Some Bad Insertions

Insert These Nodes

30 2 16 6 7 12 8



## Some Bad Insertions

Insert These Nodes

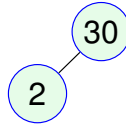
30 2 16 6 7 12 8

## Some Bad Insertions

## Some Bad Insertions

Insert These Nodes

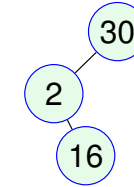
30 2 16 6 7 12 8



## Some Bad Insertions

Insert These Nodes

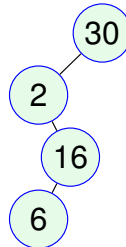
30 2 16 6 7 12 8



## Some Bad Insertions

Insert These Nodes

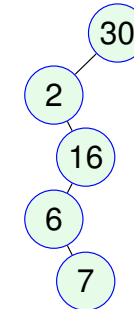
30 2 16 6 7 12 8



## Some Bad Insertions

Insert These Nodes

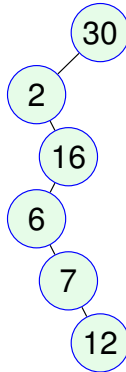
30 2 16 6 7 12 8



## Some Bad Insertions

Insert These Nodes

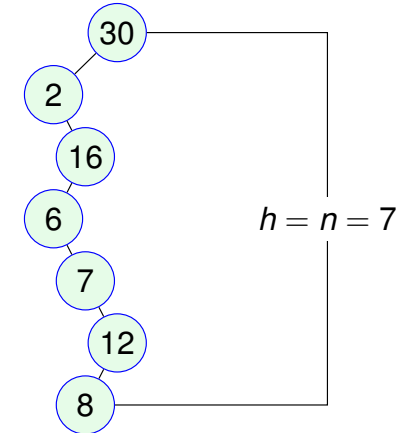
30 2 16 6 7 12 8



## Some Bad Insertions

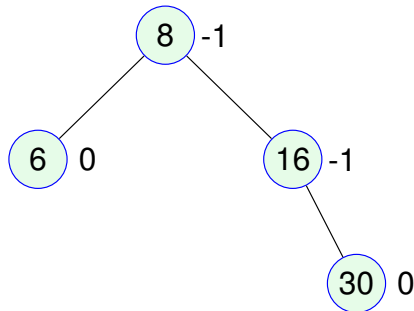
Insert These Nodes

30 2 16 6 7 12 8



## Balance

- The **balance** of a node is the depth of the left subtree minus the depth of the right subtree.



- Depth is the **longest path** from the node to a leaf.
- Leaves always have balance of zero.

## Some Insertions, with Balances

Insert These Nodes

30 2 16 32 37 12

## Some Insertions, with Balances

Insert These Nodes

30 2 16 32 37 12

## Some Insertions, with Balances

Insert These Nodes

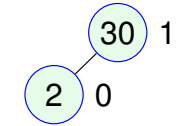
30 2 16 32 37 12



## Some Insertions, with Balances

Insert These Nodes

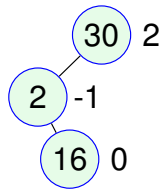
30 2 16 32 37 12



## Some Insertions, with Balances

Insert These Nodes

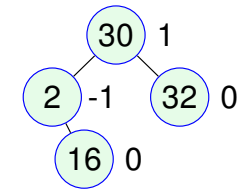
30 2 16 32 37 12



## Some Insertions, with Balances

Insert These Nodes

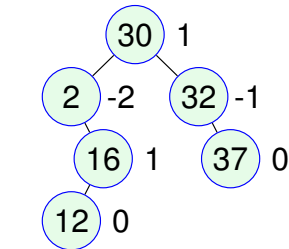
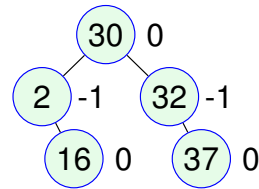
30 2 16 32 37 12



## Some Insertions, with Balances

Insert These Nodes

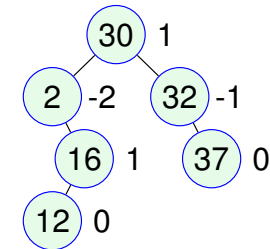
30 2 16 32 37 12



## Some Insertions, with Balances

Insert These Nodes

30 2 16 32 37 12



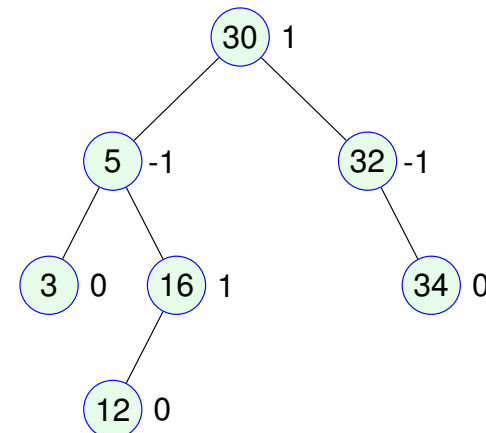
## Updating Balance During Add

- Perform an add as normal, using recursion.
- The leaf will have balance zero.
- Upon return:
  - If you went left, increment your balance.
  - If you went right, decrement your balance.
  - If the balance becomes zero, stop updating balances. (Why?)
  - If the balance is  $+/- 1$ , return to the parent.
  - If the balance is  $+/- 2$ , rebalance the node.

## Add Example 1

Example

Insert a 1.



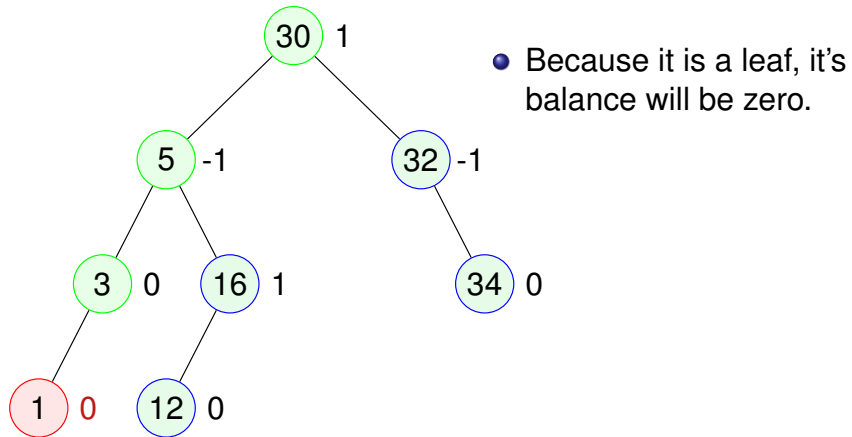
- The node 1 goes to the left of 3.



## Add Example 1

## Example

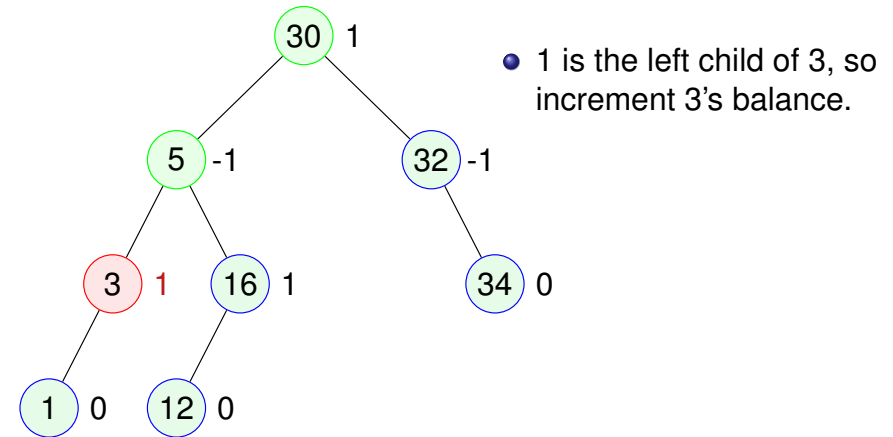
Insert a 1.



## Add Example 1

## Example

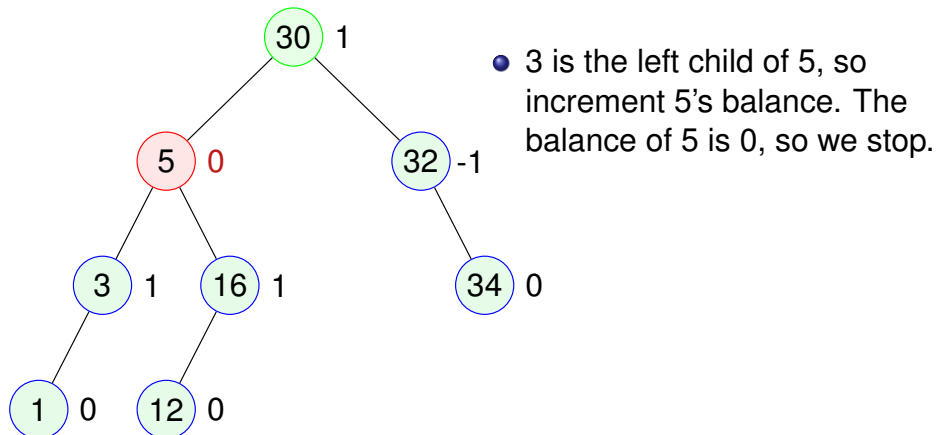
Insert a 1.



## Add Example 1

## Example

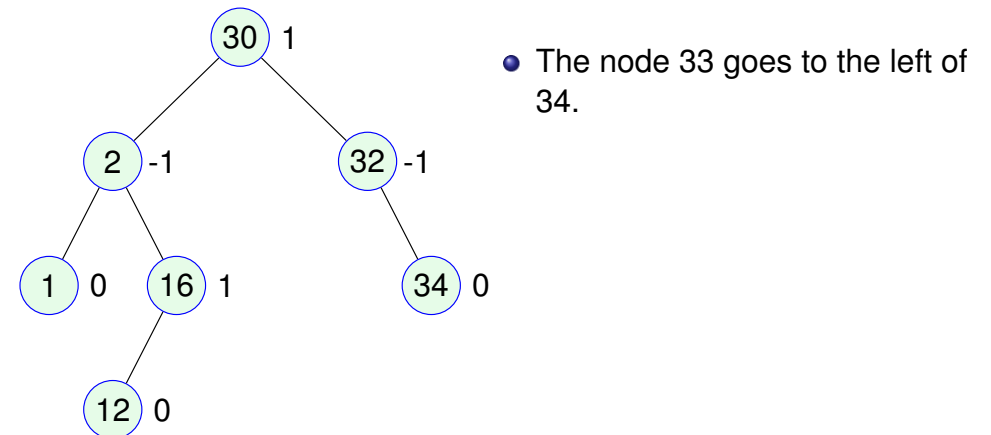
Insert a 1.



## Examples

## Example

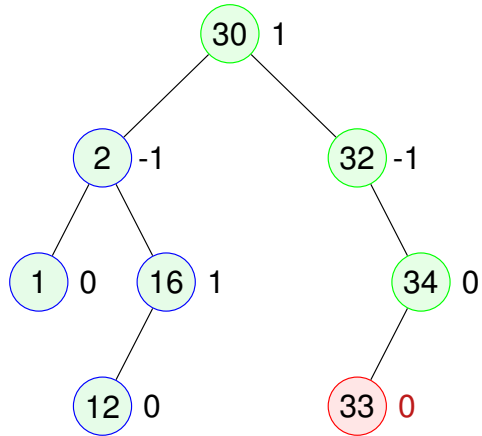
Insert a 33.



## Examples

### Example

Insert a 33.

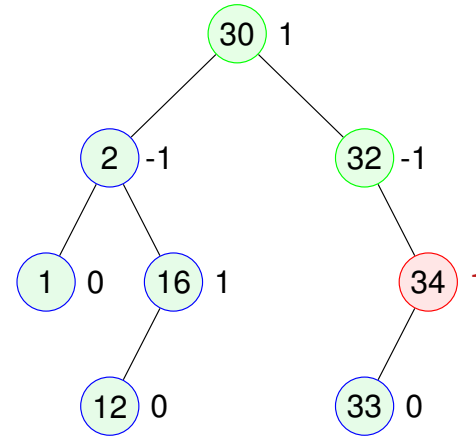


- Because it is a leaf, its balance will be zero.

## Examples

### Example

Insert a 33.

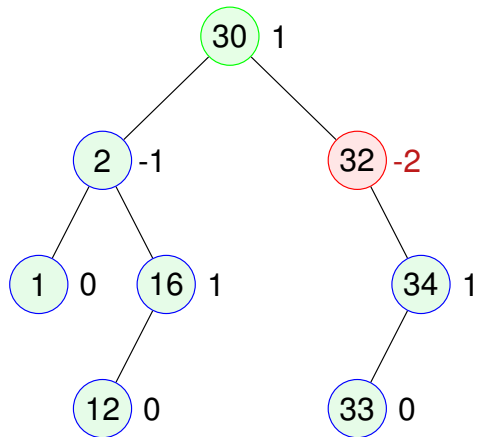


- 33 is the left child of 34, so increment 34's balance.

## Examples

### Example

Insert a 33.

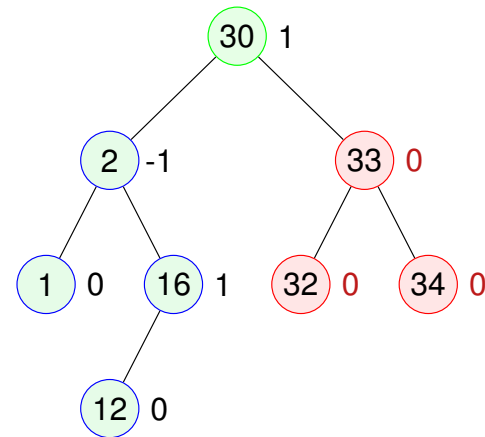


- 34 is the right child of 32, so decrement 32's balance. This node is out of balance, so we will rebalance here.

## Examples

### Example

Insert a 33.

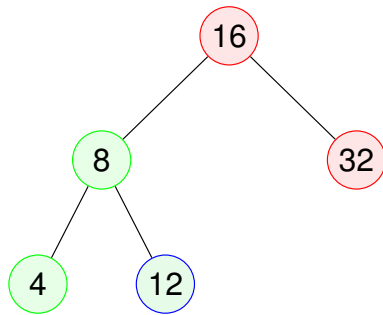


- Here is the result of rebalancing. Let's talk about that next....

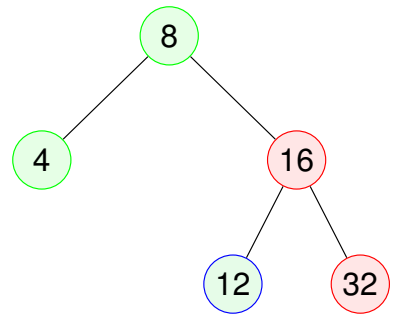
## Effect of a rotation

- This is a Right Rotation.

before



after

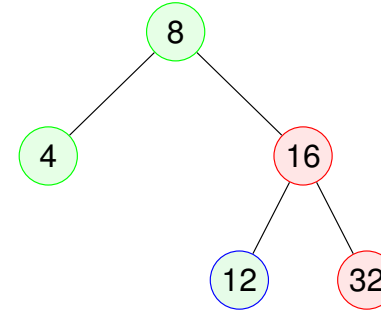


- What happens to the balance of the root?

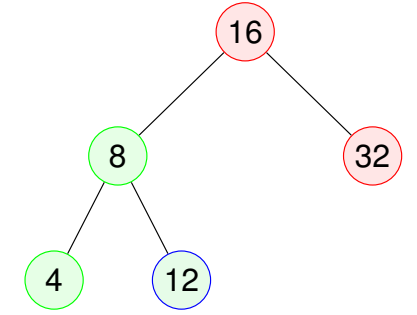
## Effect of a rotation

- This is a Left Rotation. It should look familiar.

before



after

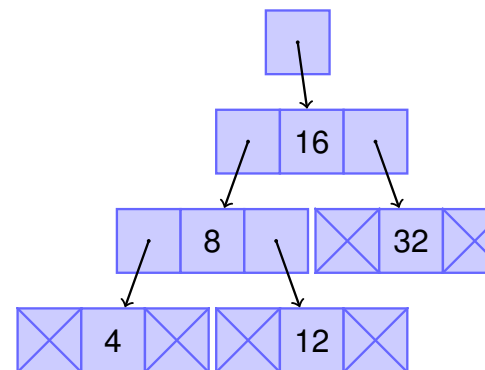


- What happens to the balance of the root?

## Effects of Rotations

- A Left Rotation adds 2 to the balance of the node. Use it when the balance is  $-2$ .
- A Right Rotation subtracts 2 from the balance of the node. Use it when the balance is 2.
- The “heavy” part of the tree needs to be on the *outer* side for this to work.

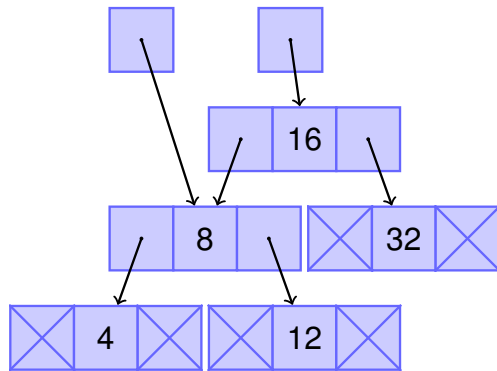
## Implementing a Right Rotation



```

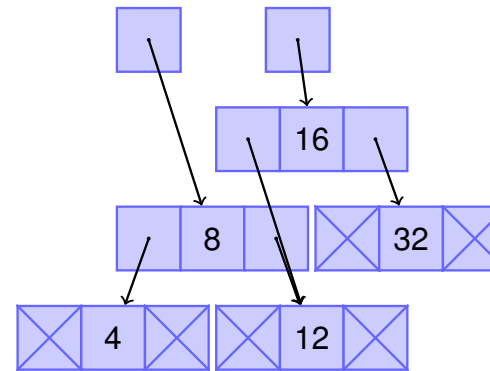
curr.balance += 2;
curr.left.balance++;
tmp = curr.left;
curr.left =
    curr.left.right;
tmp.right = curr;
curr.parent = tmp;
  
```

## Implementing a Right Rotation



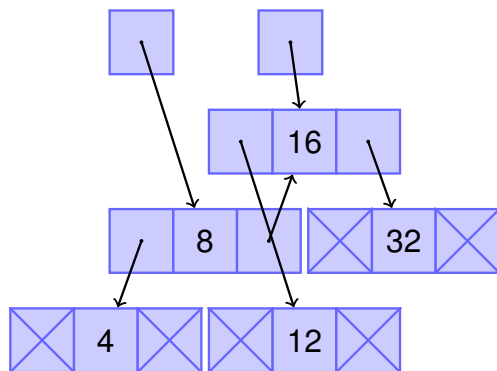
```
curr.balance += 2;
curr.left.balance++;
tmp = curr.left;
curr.left =
    curr.left.right;
tmp.right = curr;
curr.parent = tmp;
```

## Implementing a Right Rotation



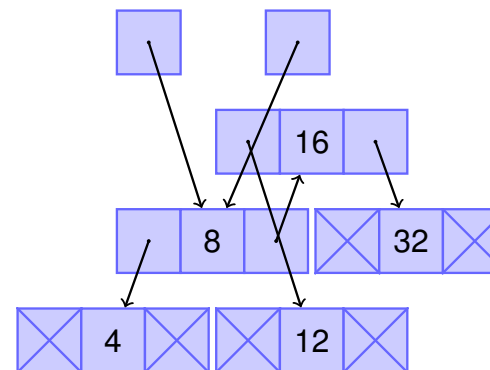
```
curr.balance += 2;
curr.left.balance++;
tmp = curr.left;
curr.left =
    curr.left.right;
tmp.right = curr;
curr.parent = tmp;
```

## Implementing a Right Rotation



```
curr.balance += 2;
curr.left.balance++;
tmp = curr.left;
curr.left =
    curr.left.right;
tmp.right = curr;
curr.parent = tmp;
```

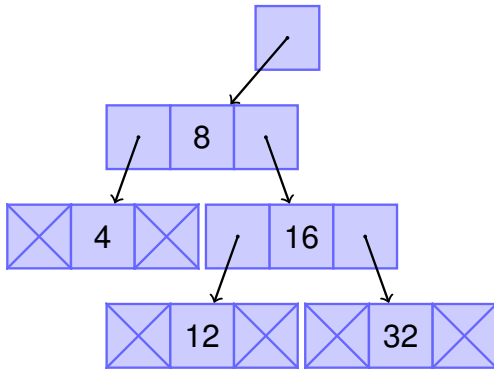
## Implementing a Right Rotation



```
curr.balance += 2;
curr.left.balance++;
tmp = curr.left;
curr.left =
    curr.left.right;
tmp.right = curr;
curr.parent = tmp;
```

- You have to update the parent's link also.

## Implementing a Right Rotation



```

curr.balance += 2;
curr.left.balance++;
tmp = curr.left;
curr.left =
    curr.left.right;
tmp.right = curr;
curr.parent = tmp;
  
```

## Bad Insertions with Rotations

Insert These Nodes

1 2 3 4 5 6

## Bad Insertions with Rotations

Insert These Nodes

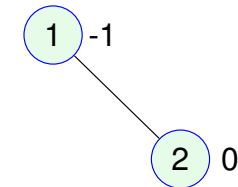
1 2 3 4 5 6



## Bad Insertions with Rotations

Insert These Nodes

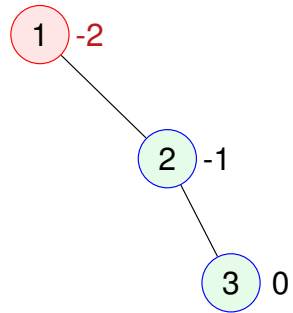
1 2 3 4 5 6



## Bad Insertions with Rotations

Insert These Nodes

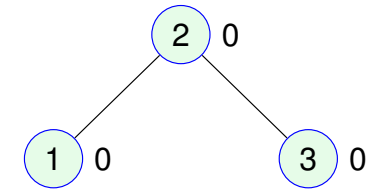
1 2 3 4 5 6



## Bad Insertions with Rotations

Insert These Nodes

1 2 3 4 5 6

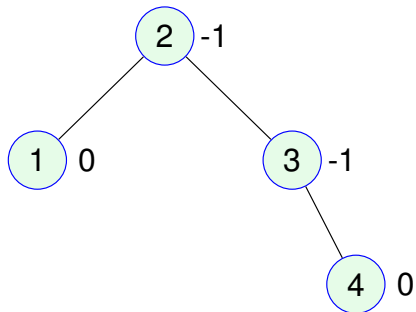


- Try inserting the next two yourself before looking ahead!

## Bad Insertions with Rotations

Insert These Nodes

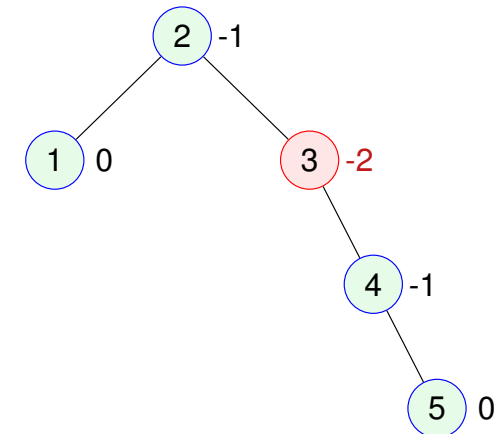
1 2 3 4 5 6



## Bad Insertions with Rotations

Insert These Nodes

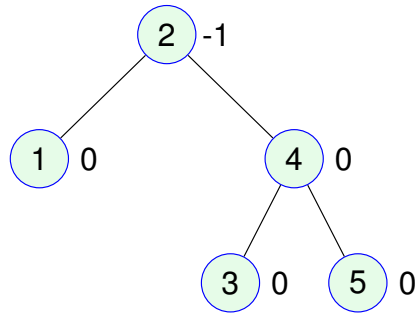
1 2 3 4 5 6



## Bad Insertions with Rotations

Insert These Nodes

1 2 3 4 5 6

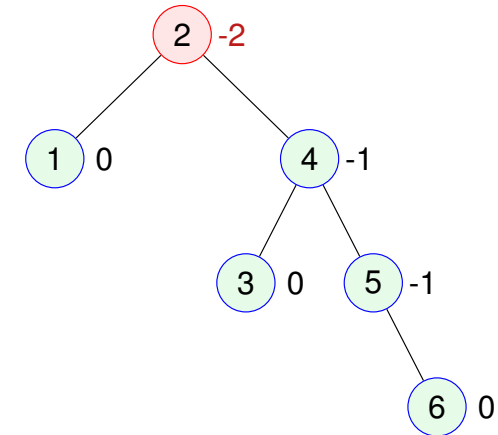


- Try inserting the next two yourself before looking ahead!

## Bad Insertions with Rotations

Insert These Nodes

1 2 3 4 5 6



## Bad Insertions with Rotations

Insert These Nodes

1 2 3 4 5 6

