# AVL Balance Condition!



Left and right subtrees **recursively** have heights differing by at most one.

**Definition (`balance`)**

$$\text{balance}(\texttt{n}) = \text{abs}(\text{height}(\texttt{n.left}) - \text{height}(\texttt{n.right}))$$

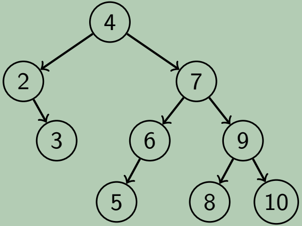
**Definition (AVL Balance Property)**

An AVL tree is balanced when:

For every node $n$, $\text{balance}(n) \leq 1$


- This ensures a small depth
- It's relatively easy to maintain

## AVL Tree




**Structure Property**:
0, 1, or 2 children

**BST Property**:
Keys in Left Subtree are smaller
Keys in Right Subtree are larger

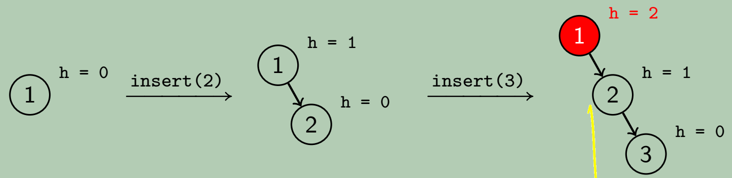
**AVL Balance Property**:
Left and Right subtrees have heights that differ by at most one.

That is, **all AVL Trees are BSTs**, but the reverse is not true.

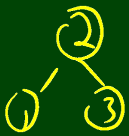

AVL Trees rule out **unbalanced BSTs**.

# The BST Worst Case

## Worst Case



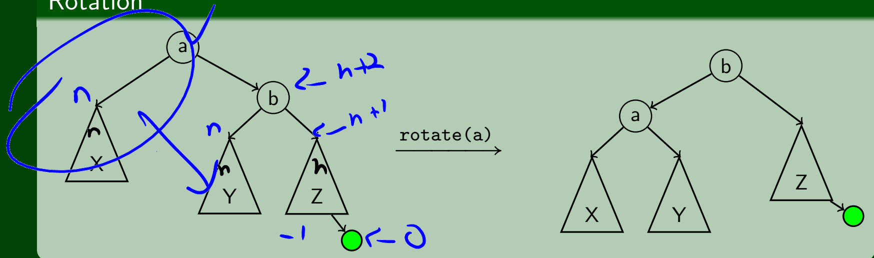

When we insert 3, we violate the AVL Balance condition. What to do?



# AVL Rotation 

This “fix” is called a rotation. We’re “rotating” the child node “up”:

Rotation




**This is the only fundamental of AVL Trees!**

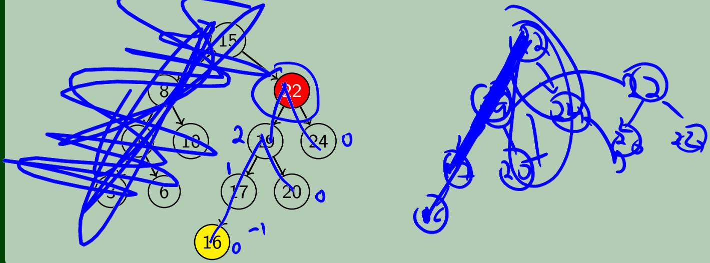
You can either look at this as “the only way to correctly rearrange the subtrees” or it’s helpful to think of it as gravity.

# More Complicated Now. . .



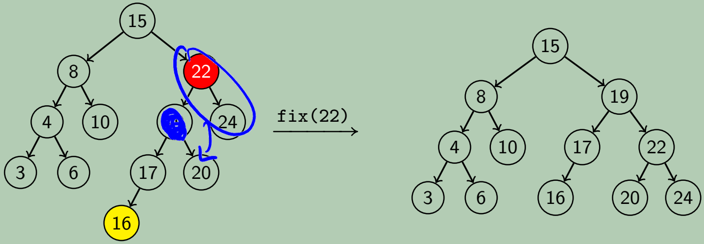
## Inserting 16

Is the result an AVL tree? If not, how do we fix it?



## Inserting 16

Is the result an AVL tree? If not, how do we fix it?

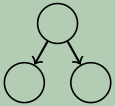



This is just the same rotation in the other direction!

# AVL Rotations. . . Are We Done?



## We Want. . .

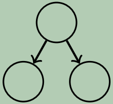


## Cases We've Handled

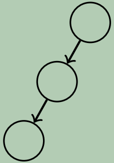
## Cases To Handle

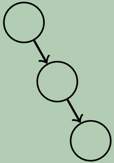
# AVL Rotations. . . Are We Done?

## We Want. . .

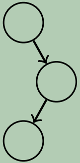


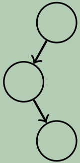
## Cases We've Handled





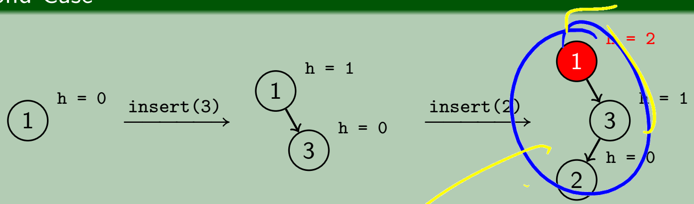
## Cases To Handle



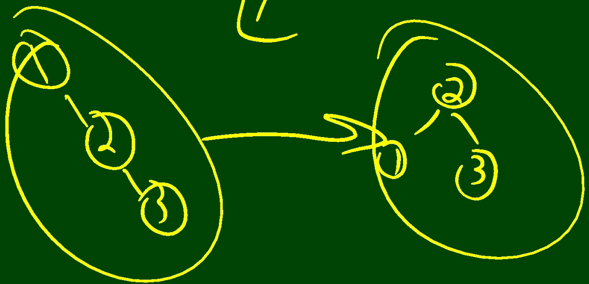


# Another Case

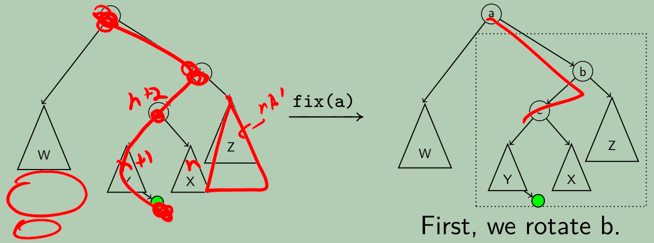
## Second Case




When we insert 2, we violate the AVL Balance condition. What to do?



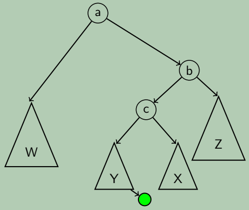
## **Double** Rotation



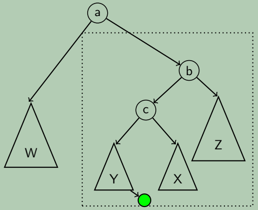

First, we rotate b.

# It Doesn't Look Like a Single Rotation Will Do...

## **Double** Rotation

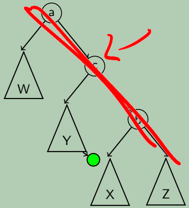



`fix(a)` →




First, we rotate b.

`rotate(b)` →




Now, we're back to the **line** case.

We must **guarantee** that the AVL property gives us a small enough tree.
Our approach: Find a big **lower bound** on the number of nodes necessary to make a tree with height $h$.

$h \approx \lg n$

I know AVL is height h

$n \longrightarrow h$

# Does an AVL Tree Work? 

We must **guarantee** that the AVL property gives us a small enough tree.
Our approach: Find a big **lower bound** on the number of nodes necessary to make a tree with height $h$.

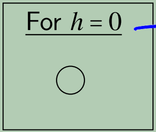
**What is the smallest number of nodes to get a height $h$ AVL Tree?**

For $h = 0$ $\rightarrow$ 1

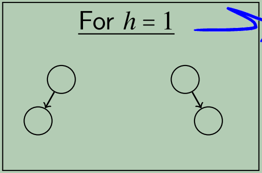
For $h = 1$

We must **guarantee** that the AVL property gives us a small enough tree. Our approach: Find a big **lower bound** on the number of nodes necessary to make a tree with height $h$.

What is the **smallest** number of nodes to get a height $h$ AVL Tree?



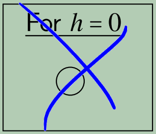

1

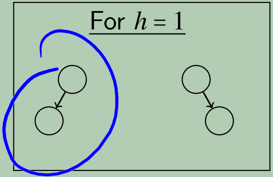



2

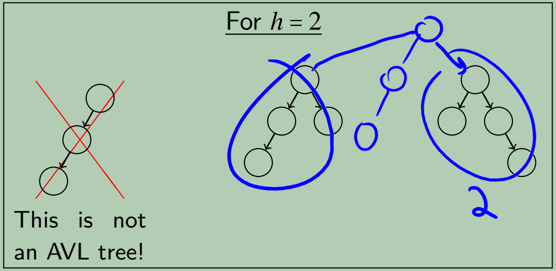
We must **guarantee** that the AVL property gives us a small enough tree. Our approach: Find a big **lower bound** on the number of nodes necessary to make a tree with height $h$.

**What is the smallest number of nodes to get a height $h$ AVL Tree?**

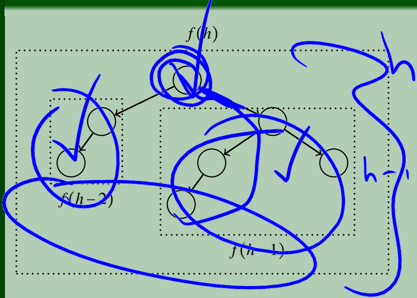



*(3)



What is the **smallest** number of nodes to get a height $h$ AVL Tree?




The general number of nodes to get a height of $h$ is:

$$f(h) = f(h-2) + f(h-1) + 1$$

We break down where each term comes from. We want a tree that has the **smallest** number of nodes where each branch has the AVL Balance condition.

- $f(h-1)$: To force the height to be $h$, we take the smallest tree of height $h-1$ as one of the children
- $f(h-2)$: We are allowed to have the branches differ by one; so, we can get a smaller number of nodes by using $f(h-2)$
- $+1$ comes from the root node to join together the two branches

So, now we solve our recurrence. How?

## Ratio Between Terms

A good way of solving a recurrence that we expect to be of the form $X^n$

$X^h$

is to look at the ratio between terms. If $\frac{f(h+1)}{f(h)} > X$, then

$$f(h+1) > Xf(h) > X(X(f(h-1)) > \cdots > X^h$$

So, we evaluate these ratios and see the following:

OUTPUT

```
>> 2.0
>> 2.0
>> 1.75
>> 1.7142857142857142
>> 1.6666666666666667
>> 1.65
>> 1.6363636363636365
>> 1.6296296296296295
>> 1.625
>> 1.6223776223776223
>> 1.6206896551724137
>> 1.6196808510638299
>> 1.619047619047619
>> 1.618661257606491
>> 1.618421052631579
>> ...
```

$n = f(h+1) > X^h$

$f(h+1) > \phi^h$

$\phi^2 = \phi + 1$

$\lg n > h$