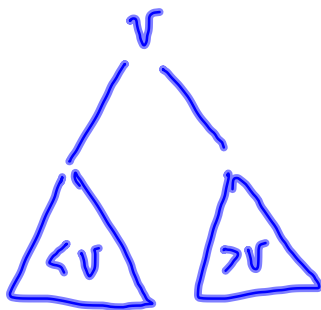


Search Trees

No duplicate values



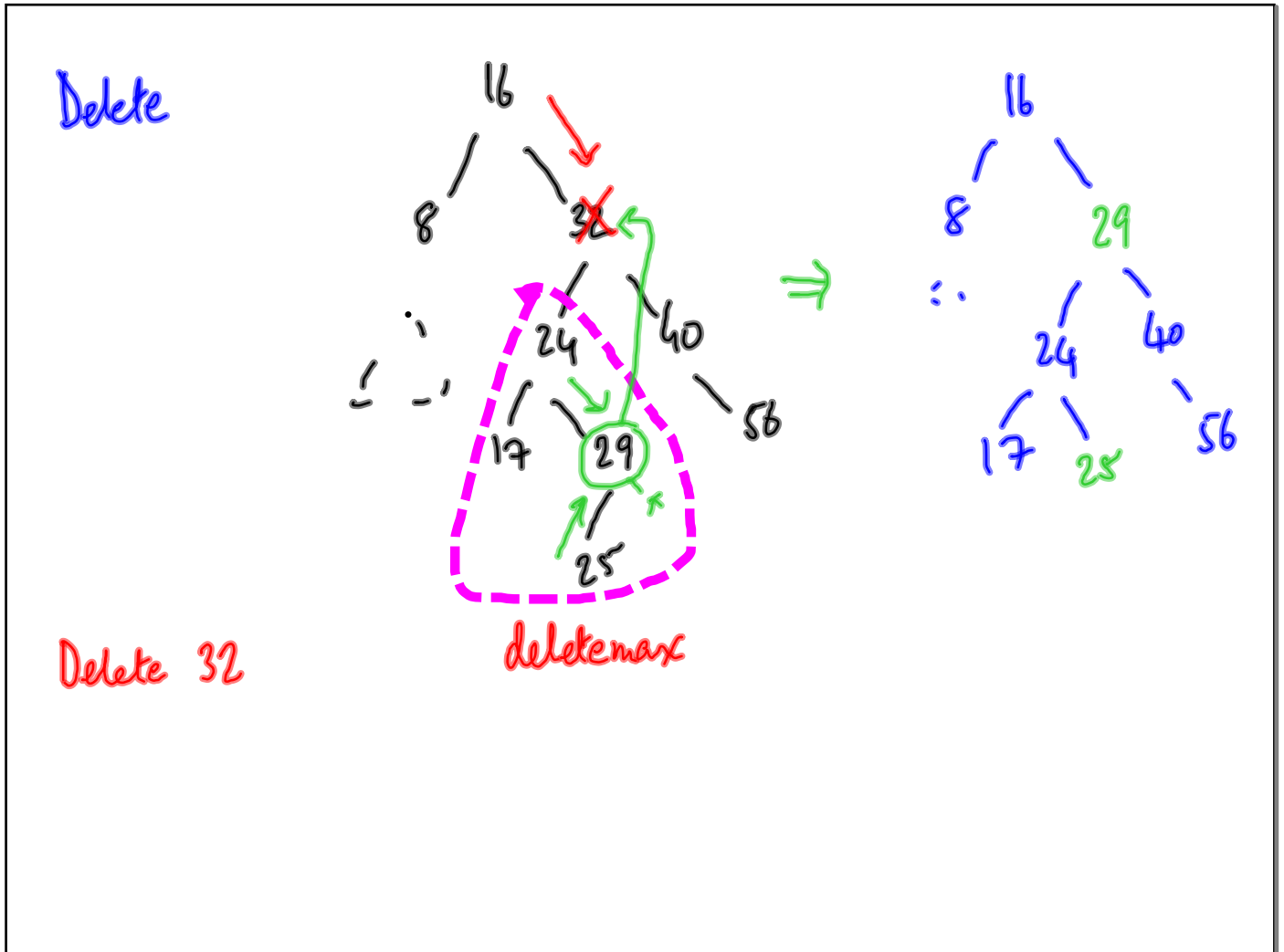
Insert & find
are similar



Find 10

Find 19 x

Insert 19



In Python:

```
class Tree:
```

```
    def __init__(self, x=None):
```

```
        self.value = x
```

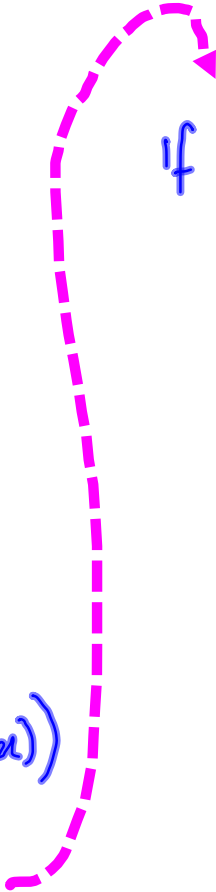
```
        self.left = None
```

```
        self.right = None
```

```
    def isempty(self):
```

```
        return (self.value == None)
```

```
def find(self, x):  
    if self.isempty():  
        return(False)  
    if self.value == x:  
        return(True)  
    if x < self.value:  
        if self.left:  
            return(self.left.find(x))  
        else:  
            return(False)  
    if x > self.value:  
        if self.right:  
            return(self.right.  
                find(x))  
        else:  
            return(False)
```



insert

```

def find(self, x):
    if self.isempty():
        return(False) self.value = x
    if self.value == x:
        return True
    if x < self.value:
        if self.left:
            return(self.left.find(x))
        else:
            return(False) self.left = Tree(x)
    if x > self.value:
        if self.right:
            return(self.right
                insert) find(x)
        else:
            return(False)
            self.right = Tree(x)

```


```


def delete(self, x):
    if self.isempty():
        return

    if x < self.value:
        if self.left:
            self.left.delete(x)
            if self.left.value == None:
                self.left = None
        return
    Symmetric for x > self.value

# self.value == x
self.value = None
if self.left == None and self.right == None:
    return
if self.right == None: # right empty # left is not
    self.value = self.left.value
    self.right = self.left.right
    self.left = self.left.left
    return

```





```
if self.left == None:
```

```
# Symmetric
```

```
self.value = self.right.value
```

```
self.left = self.right.left
```

```
self.right = self.right.right
```

```
return
```

```
# both self.left & self.right exist
```

```
self.value = self.left.deletemax() ← to write!
```

```
if self.left.value == None:
```

```
self.left = None
```

```

def deletemax (self):
    if self.right:
        self.right.deletemax()
        if self.right.value == None
            self.right = None
    return

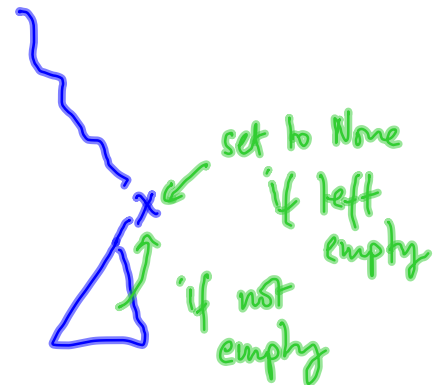
```

this is the max.

```

if self.left:
    self.value = self.left.value
    self.right = self.left.right
    self.left = self.left.left

```



else:

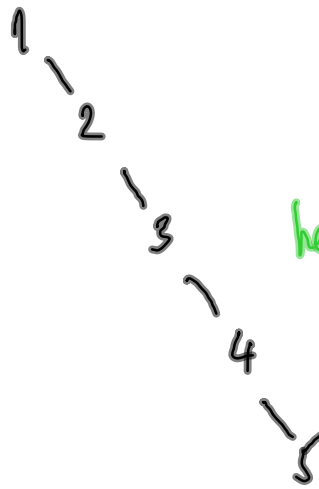
```

self.value = None

```


Balance?

insert 1,2,3,4,5 in that order

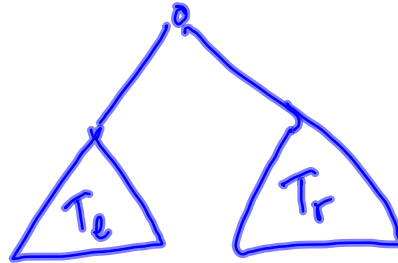


find etc takes
time proportional
height to length of longest
path = size of
tree!

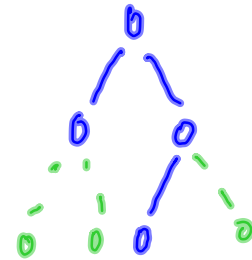
What is balance ?

Size balance

"
of nodes



$$| \text{size}(T_e) - \text{size}(T_r) | \leq 1$$



Size balanced tree:

Size is exponential wrt height

$\text{smallest}(h)$: size of smallest tree of height h

by defn: $h=0$ is empty tree

$h=1$ one root, no children

$$\text{smallest}(0) = 0$$

$$\text{smallest}(1) = 1$$

$$\begin{aligned} \text{smallest}(h+1) &= \text{smallest}(h) + (\text{smallest}(h)-1) + 1 \\ &= 2 \text{smallest}(h) \end{aligned}$$

at least one child has height h

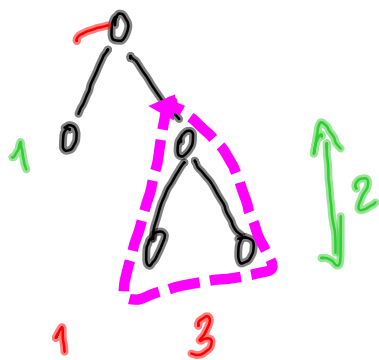
\therefore smallest $(h+1) = 2^h$

But, inductively maintaining size balance in
a search tree is expensive

Weaken the notion of balance

Height balance:

$$|\text{height}(T_e) - \text{height}(T_r)| \leq 1$$



Not size balanced

Height balanced ✓

Wrt
height
balance:

$$\text{smallest}(0) = 0$$

$$\text{smallest}(1) = 1$$

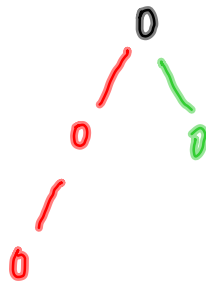
$$\text{smallest}(h+1) = \text{smallest}(L) + \text{smallest}(R) + 1$$

at least
as before, one
subtree has
height h

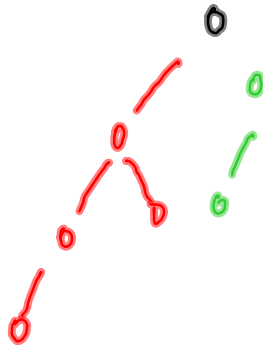
smallest (2)



smallest (3)



smallest (4)



$$\underline{f(n) = f(n-1) + f(n-2) + 1}$$

Fibonacci

Fibonacci grows exponentially w.r.t n

$$\text{Smallest}(n) \geq \text{fib}(n) \quad \forall n$$

\therefore smallest(n) grows exponentially

Maintain height balance as we go along

Before each insert/delete

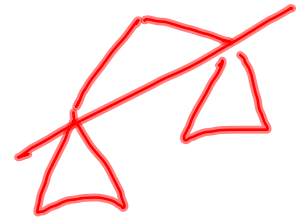
Assume tree is currently height balanced

After insert/delete

Fix balance if needed

Slope: $\text{height}(T_l) - \text{height}(T_r)$

height balance \rightarrow slope $\in \{-1, 0, +1\}$

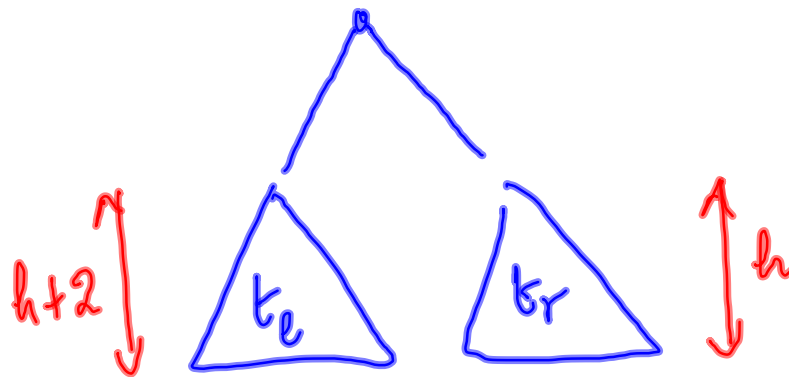


After one update

slope $\in \{-2, -1, 0, +1, +2\}$

Need to rebalance nodes with slope = ± 2

Slope = +2



Assumption: t_e & t_r are

themselves

height balance

"Rotating" at a
node

Rotation

