

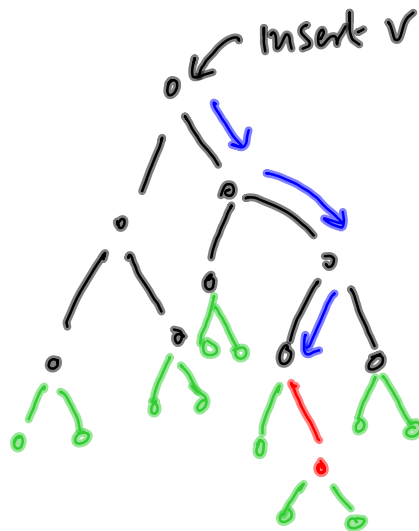
Height balanced trees

$$|h(\text{left}) - h(\text{right})| \leq 1$$

$h(t)$ = # of nodes on longest path from root to leaf

Assume tree is height balanced

insert/delete \rightarrow restore balance



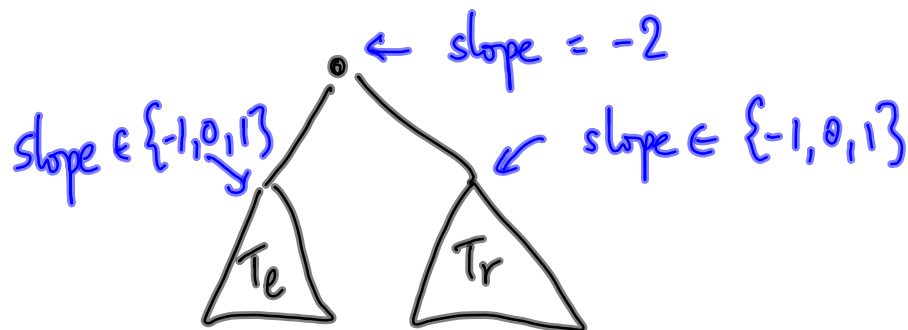
Fix balance at current node, assuming subtrees are balanced

Amount of imbalance is limited

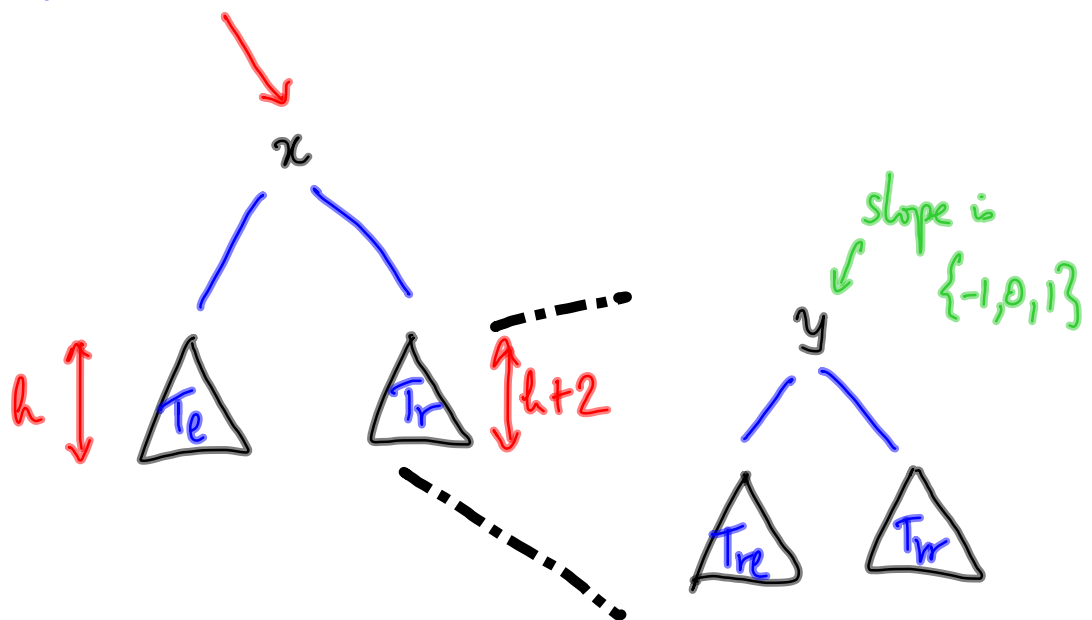
$$\text{slope} = h(\text{left}) - h(\text{right})$$

Balanced: $\text{slope} \in \{-1, 0, 1\}$

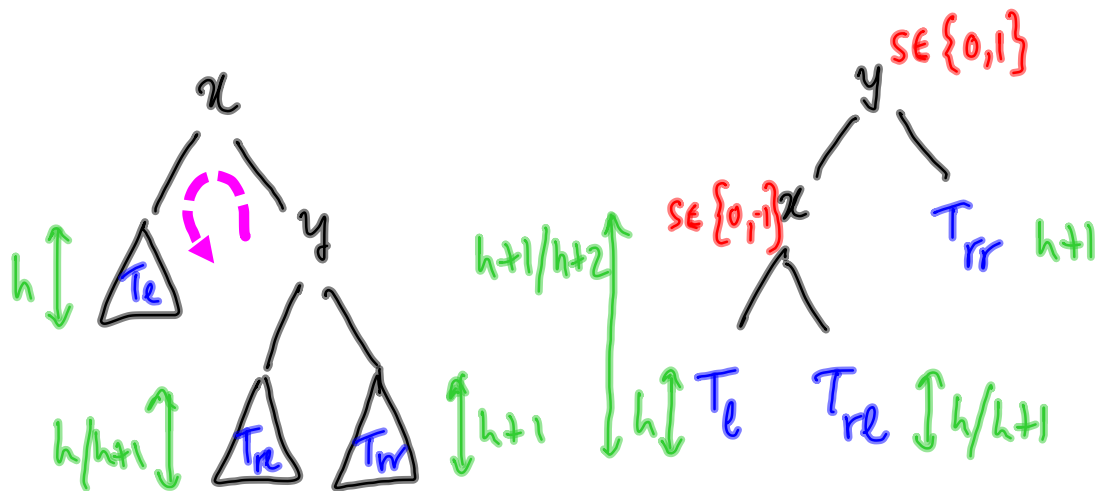
Unbalanced: $\text{slope} \in \{-2, -1, 0, 1, 2\}$



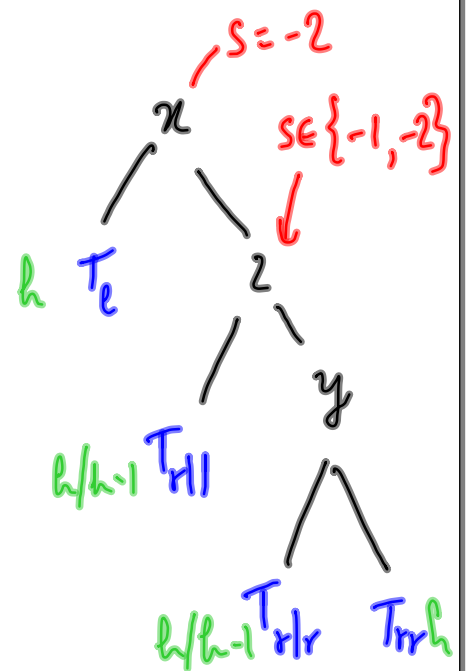
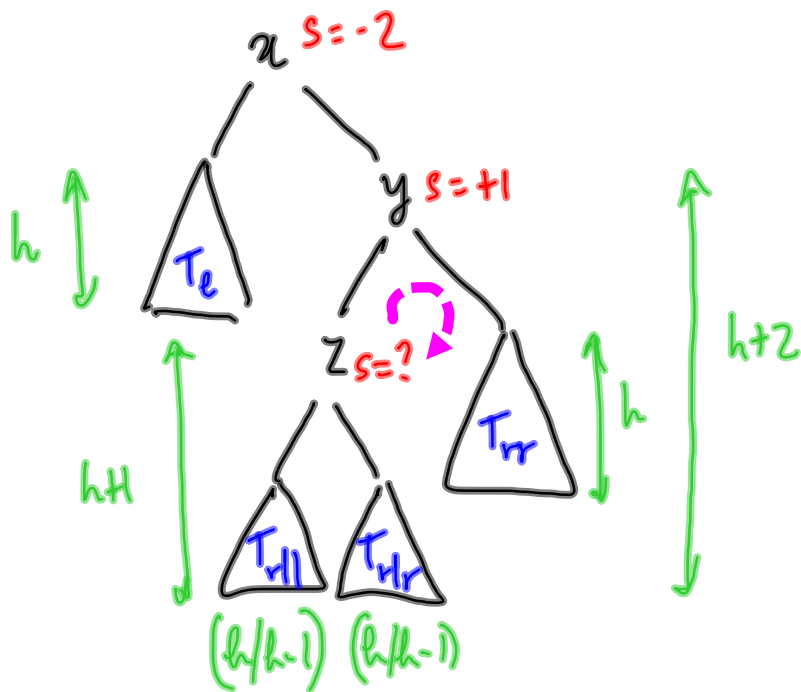
Case: Slope = -2



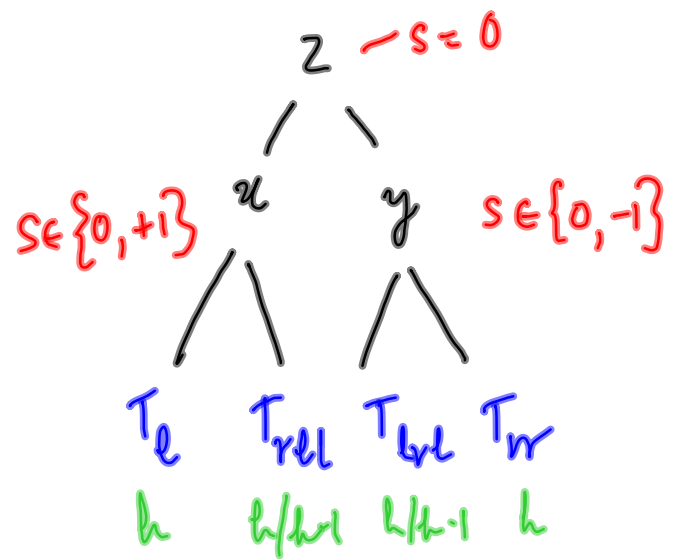
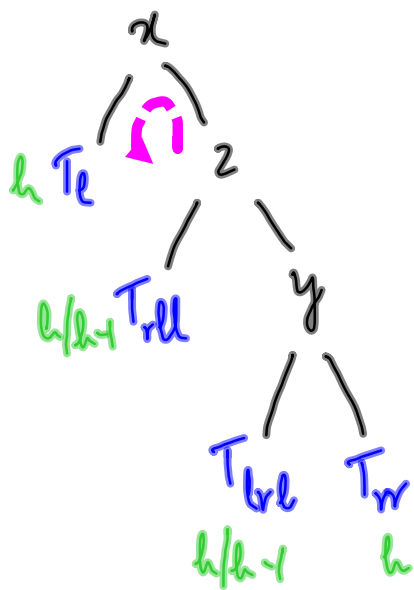
Case 1(a): Slope at x is -2
 Slope at y is $\{0, -1\}$



Case 1(b): Slope at x is -2
Slope at y is $+1$



Apply a second (left) rotation at x



```
def insert(self, x):  
    :  
    if x < self.value:  
        self.left.insert(x)  
        self.rebalance()
```

Same thing in all cases for
insert / delete

Crucial caveat

Need to compute slope
Computing height
explicitly costs $O(n)$

Store the current
height at each
node

Update with each
insert / delete

Sorting

Insertion sort

$O(n)$

where

$n = \text{length of list}$

$\text{insert } x [] = [x]$

$\text{insert } x (y:ys)$

$\mid x < y = x:y:ys$

$\mid \text{otherwise} = y:(\text{insert } x\ ys)$

$O(n^2)$

$= 1+2+\dots+n-1$

$\text{isort } [] = []$

$\text{isort } (x:xs) = \text{insert } x (\text{isort } xs)$

Insertion sort produces a new list

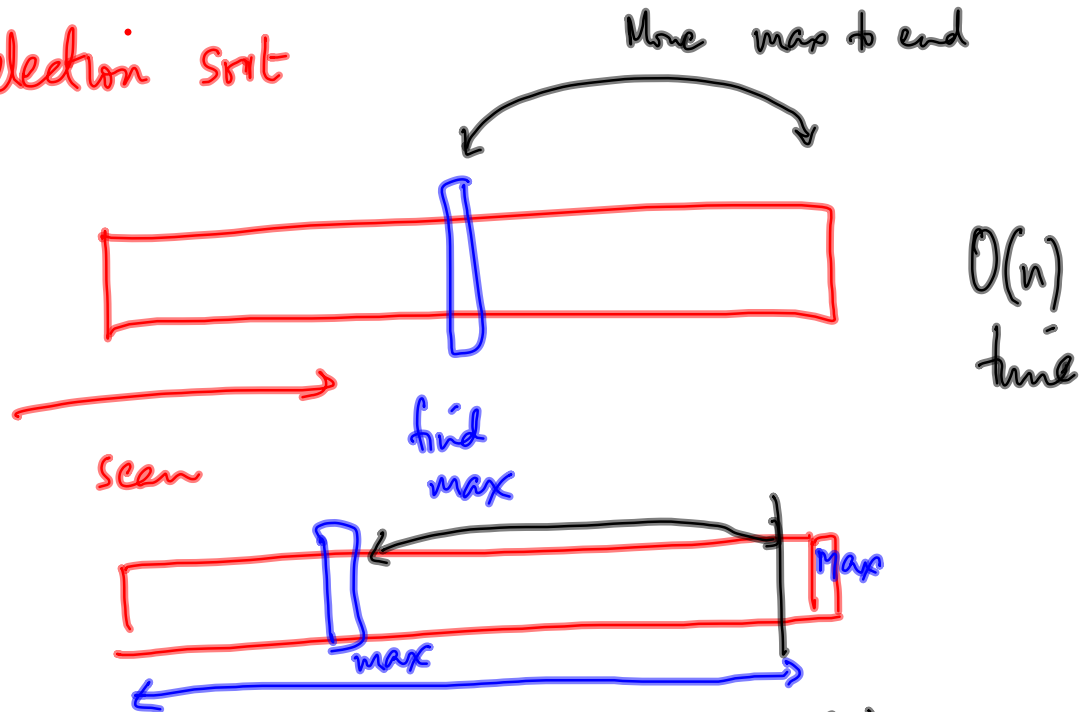
Traditionally – measure time taken by an algorithm

{ What about space – In place sort
└ Data movement

In place sort

Naive in place sort

Selection sort



Overall $n + n + 1 + \dots + 1 = O(n^2)$ IN PLACE!