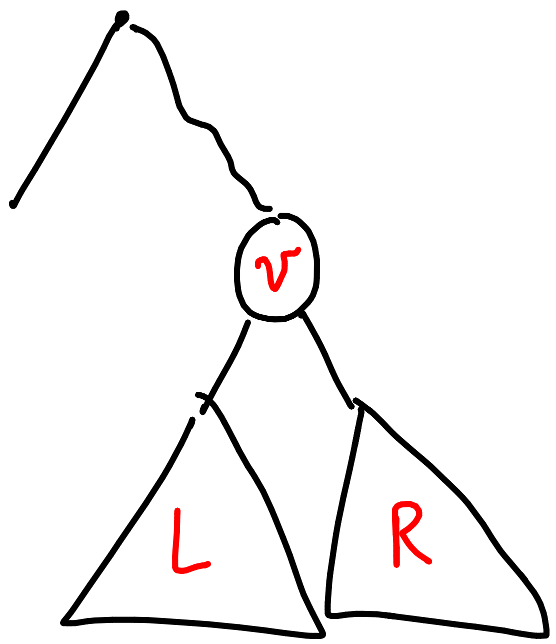
Search Trees : Delete



General case

Move max from L to v

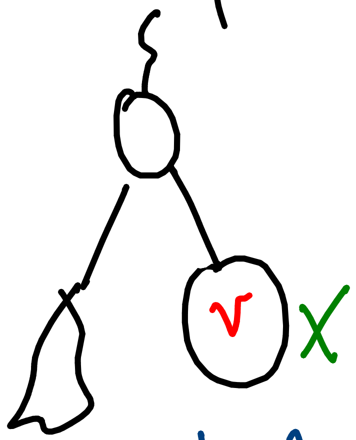
Delete max from L

Symmetrically,

Move min from R

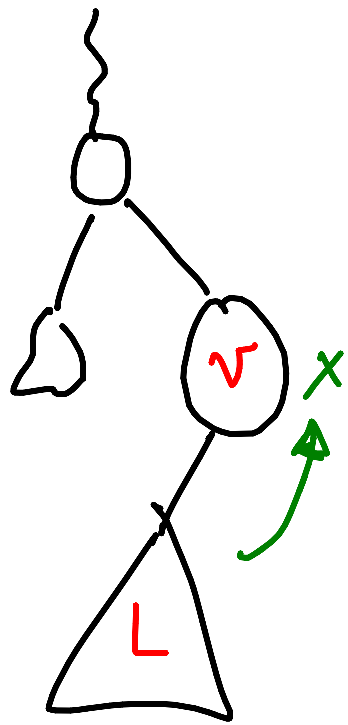
Delete min in R

3 cases for delete:



Leaf

Case1



Only one subtree

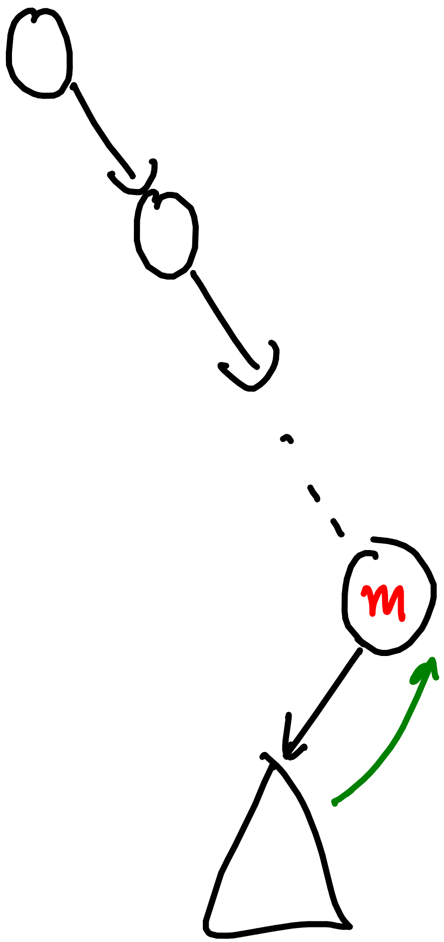
Case 2

Case 3

Both subtrees exist

Delete max etc

Delete max is Case 2 (or 1)



Exercise

Write the code

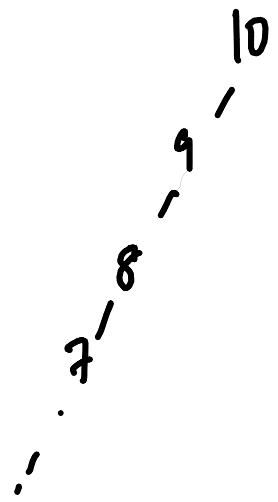
Problem is to look ahead one level to correctly remove a leaf node

How efficient are find/insert/delete?

Each operation explores a single path from root to leaf

Height: No nodes on longest path from root to leaf

Degenerate cases:



Want a guarantee on height vs size

Balanced tree – height is log(size)

What does balanced mean?

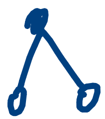
If we insist size(left) = size(right)

– perfectly balanced

– must have $2^h - 1$ nodes for height h

$h = 0$ empty size = $2^0 - 1 = 0$

$h = 1$ root size = $2^1 - 1 = 1$

$h = 2$  $2^2 - 1 = 3$

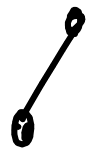
Relax balance $\quad |\text{size}(\text{left}) - \text{size}(\text{right})| \leq 1$

Want to maintain balance while manipulating tree : insert / delete

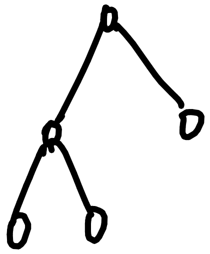
Difficult to do this for size balance

Still weaker balance

Height balance $\quad |\text{height}(\text{left}) - \text{height}(\text{right})| \leq 1$



size balanced
height balanced

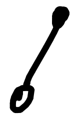


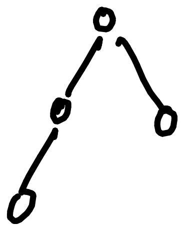
height balanced
not size balanced

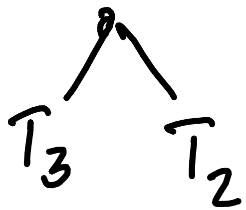
Why are height balanced trees enough to guarantee height = log (size)

Construct smallest poss tree of given height

$h = 1$  T_1

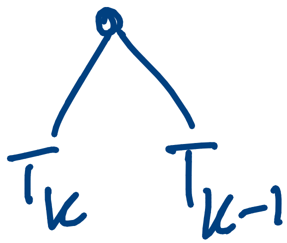
$h = 2$  T_2

$h = 3$?  T_3

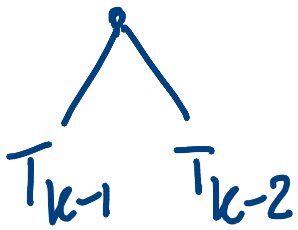
$h = 4$? 

T_k :

smallest height balanced tree of height k

$T_{k+1} =$ 

$$\text{size}(T_k) \stackrel{?}{=} 1 + \text{size}(T_{k-1}) + \text{size}(T_{k-2})$$



$$\text{fib}(k) = \text{fib}(k-1) + \text{fib}(k-2)$$

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

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

$$s(T_0) = 0$$

$$s(T_1) = 1$$

$$\text{size}(T_k) \geq \text{fib}(k)$$

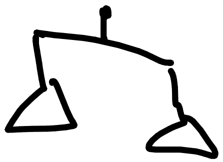
But $\text{fib}(k)$ is exponential in k

Goal: Maintain height balance

Incrementally

Balanced tree ⟶ Insert/Delete ⟶ Rebalance

Imbalance is restricted to one node added/removed



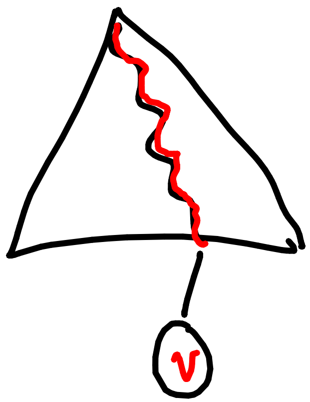
slope(node) = height(left) – height(right)

Height balanced ⇒ slope is $\{-1, 0, +1\}$

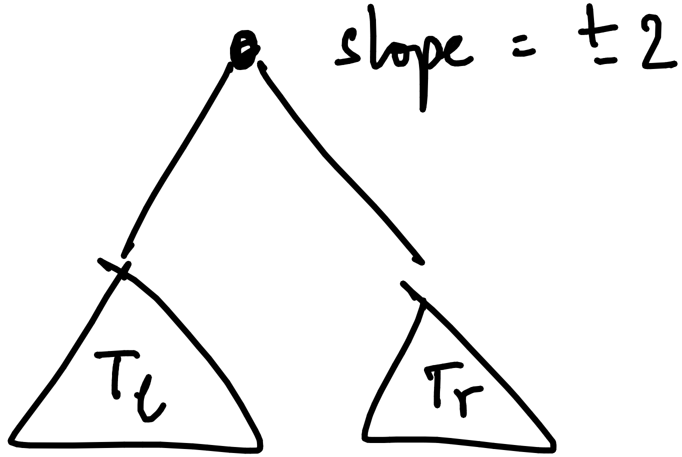
After one insert delete ⇒ slope $\{-2, .., +2\}$

Inductively assume rebalancing happens bottom up

Insert



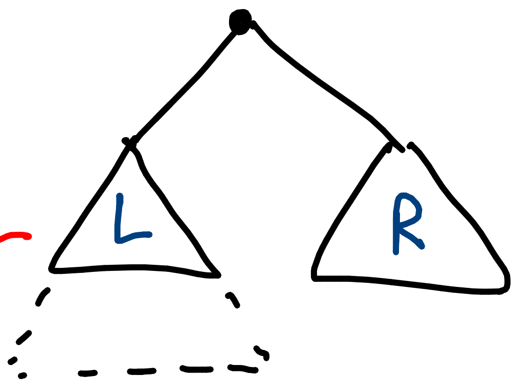
At rebalancing time



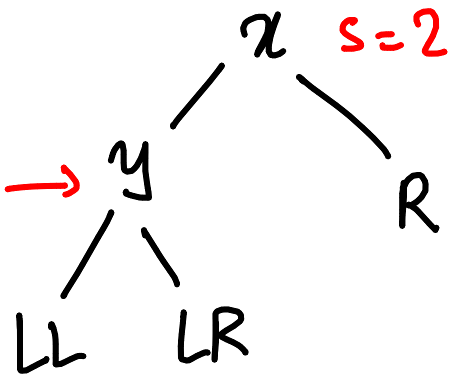
Both height balanced

Case 1   Slope +2

$h(L) - h(R) = 2$



expand L



balanced →

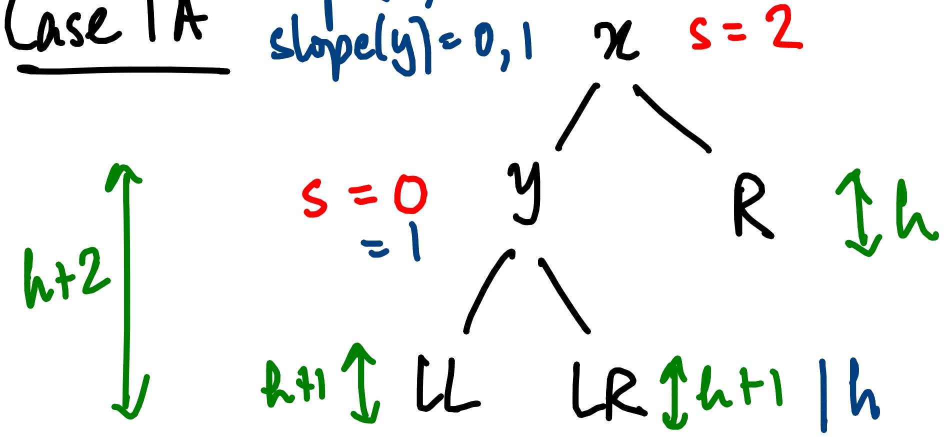
$s \in \{-1, 0, 1\}$

↓

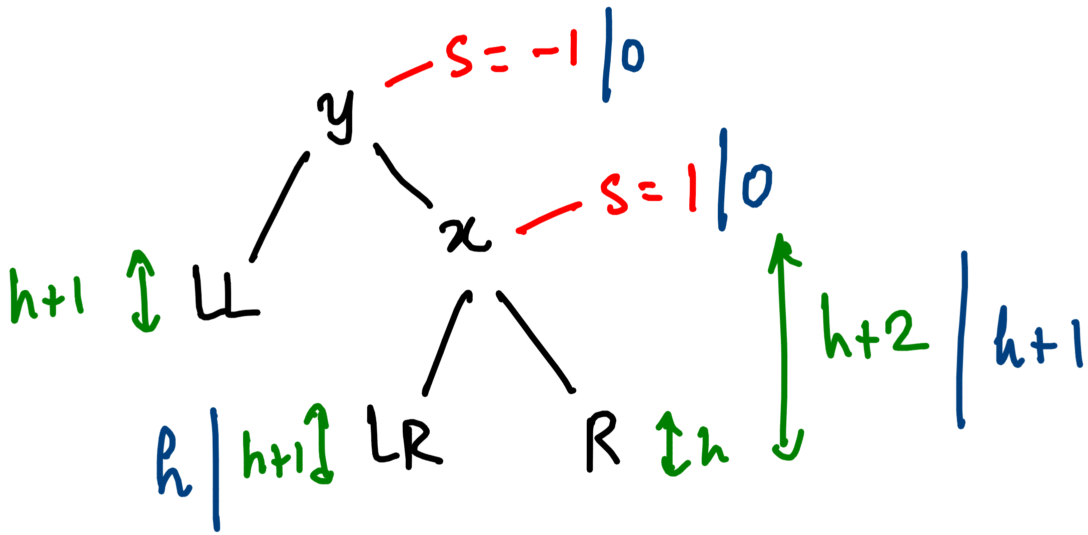
Case analysis

Case 1A

slope(x) = 2
slope(y) = 0, 1

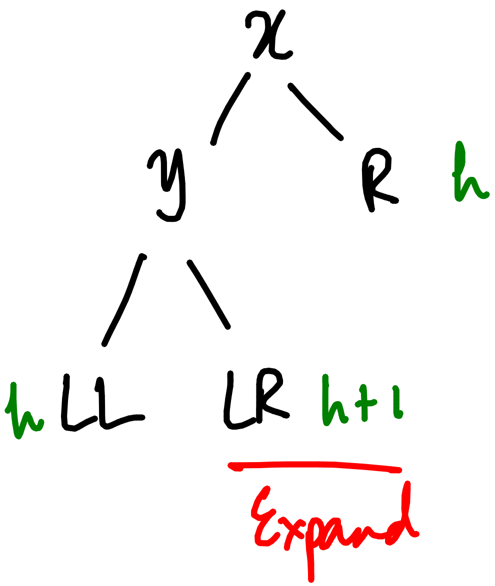


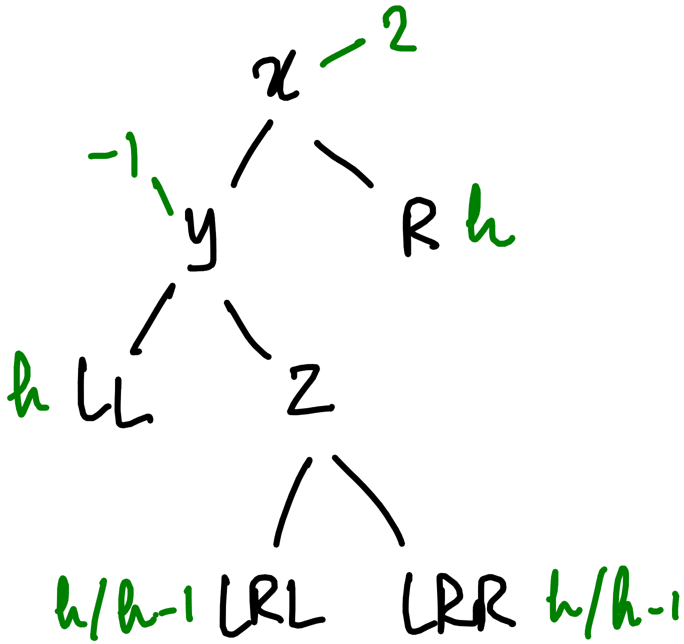
Rotate y, x

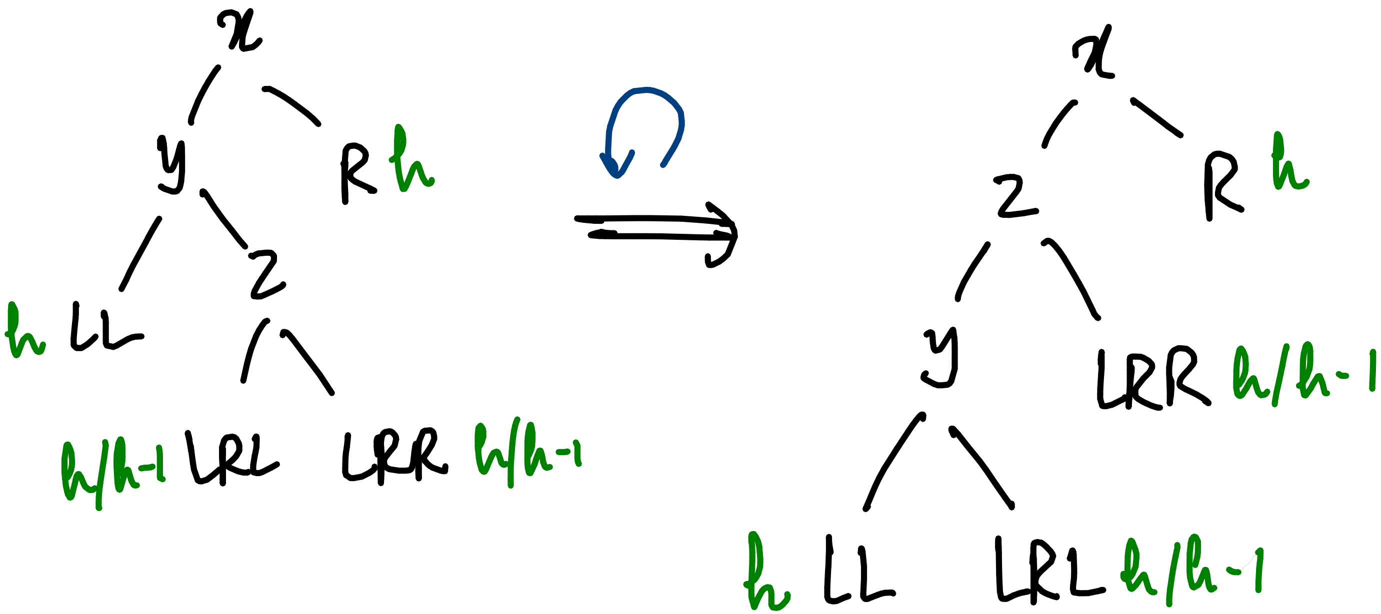


Case 1B

slope$(x) = 2$
slope$(y) = -1$



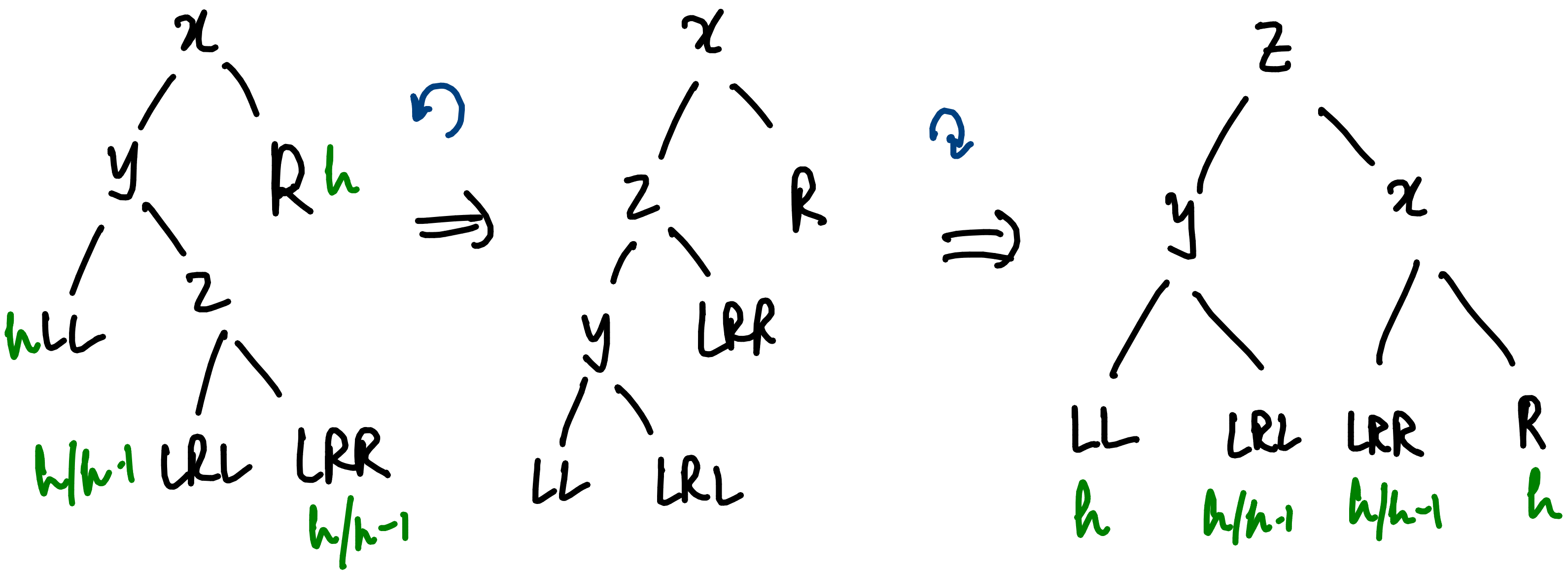




Slope(z) may be +2

But this is an intermediate step!

Now ↻ at z – x



if slope(x) == 2
    if slope(y) == -1
        left rotate at y
    right rotate at x

if slope(x) == -2
    if slope(y) == +1
        right rotate at y
    left rotate at x

AVL trees Adelson-Velskii Landis

Maintain a dynamically changing set of values s.t. find, insert, delete are all $O(\log n)$

Sorted list – insert/delete $O(n)$

Think about

Maintain Sets as Search Trees (balanced)

Union, Intersection, Membership