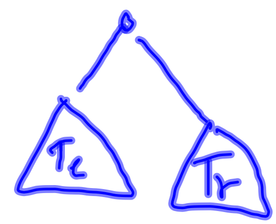
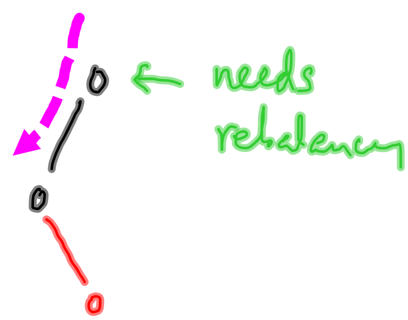
Height balanced search trees

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



After each insert/delete, we rebalance the tree bottom-up



When rebalancing a node, both subtrees are already balanced

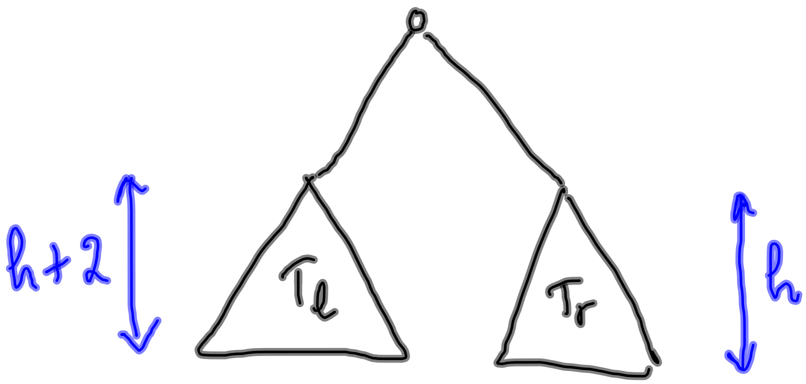
Start with slope = height(T_l) − height(T_r)

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

After a single insert/delete

$\in \{\underset{=}{-2}, -1, 0, 1, \underset{=}{2}\}$

Slope = 2



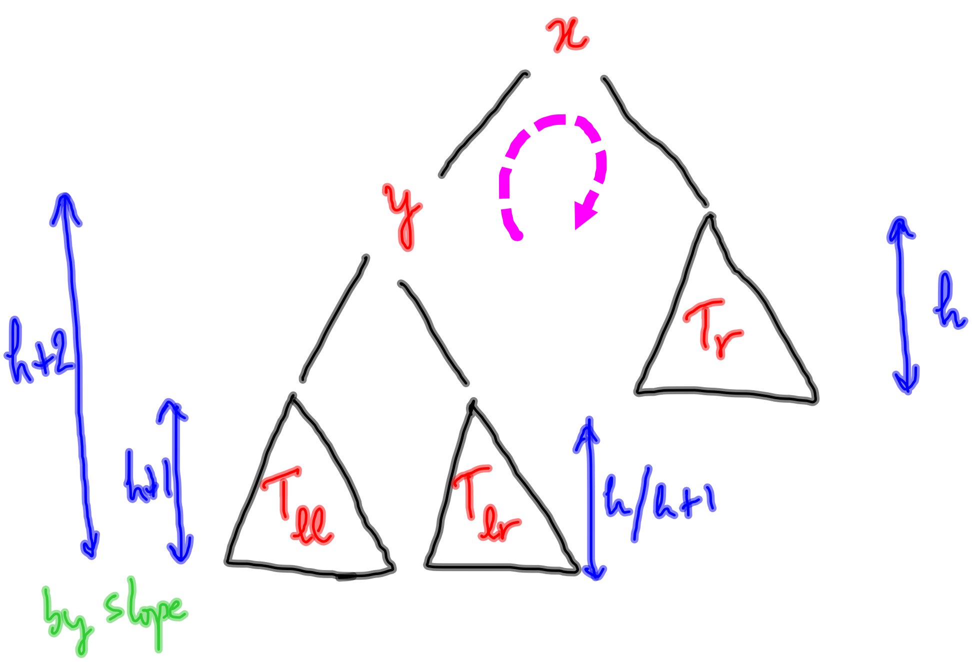
Balanced Balanced

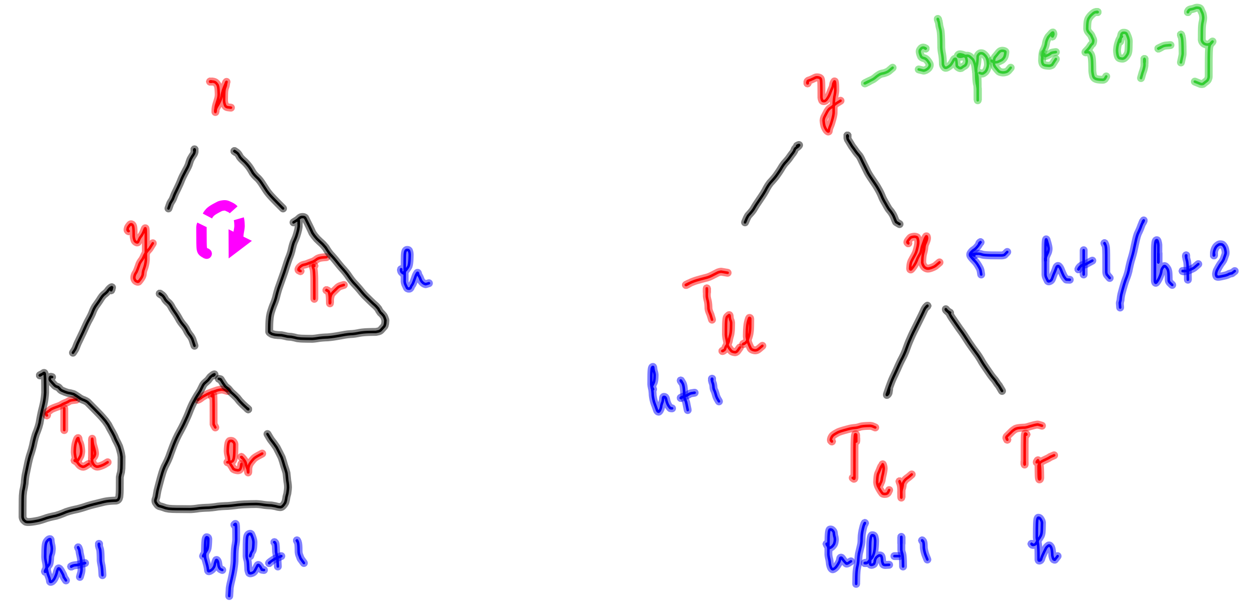
Slope $\in \{-1,0,1\}$ Slope $\in \{-1,0,1\}$

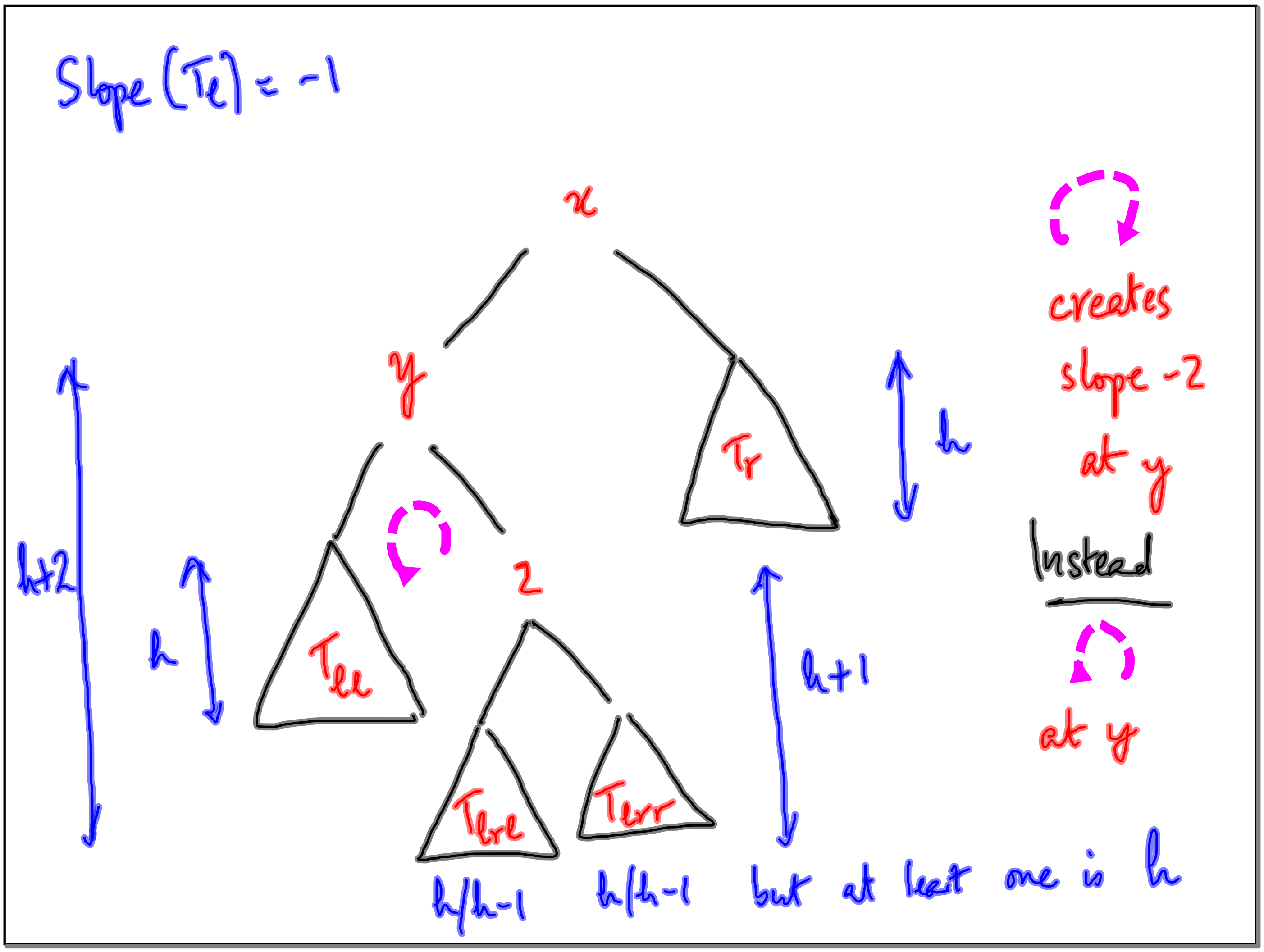
Focus on T_l

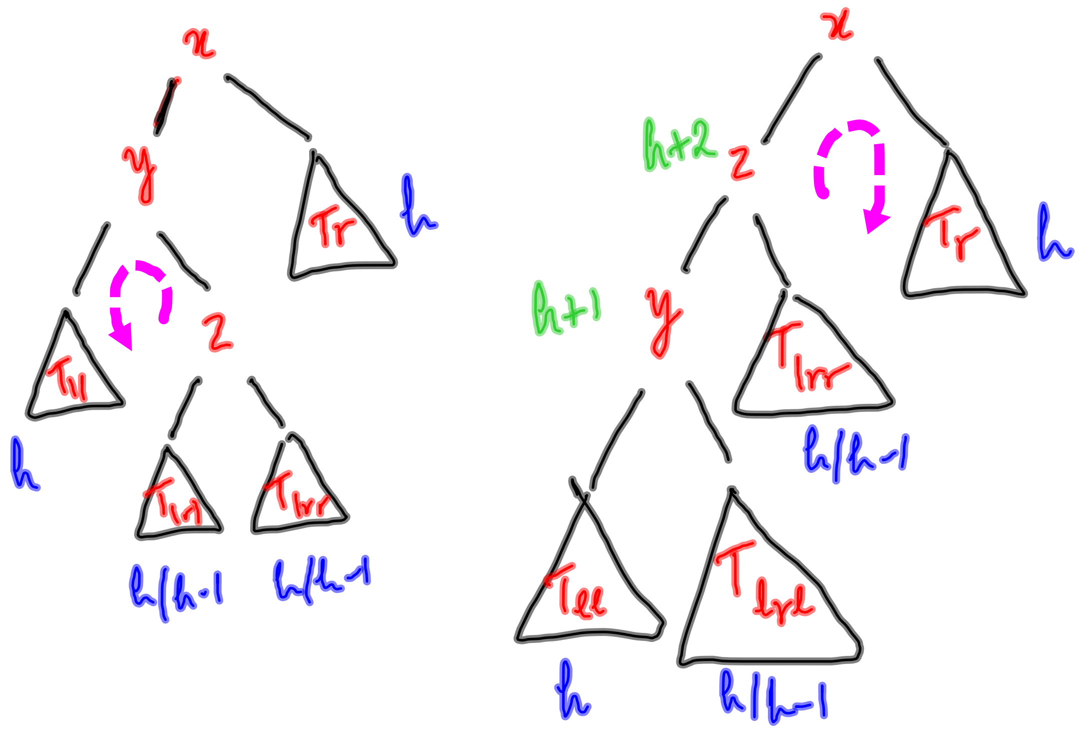
Expand the picture

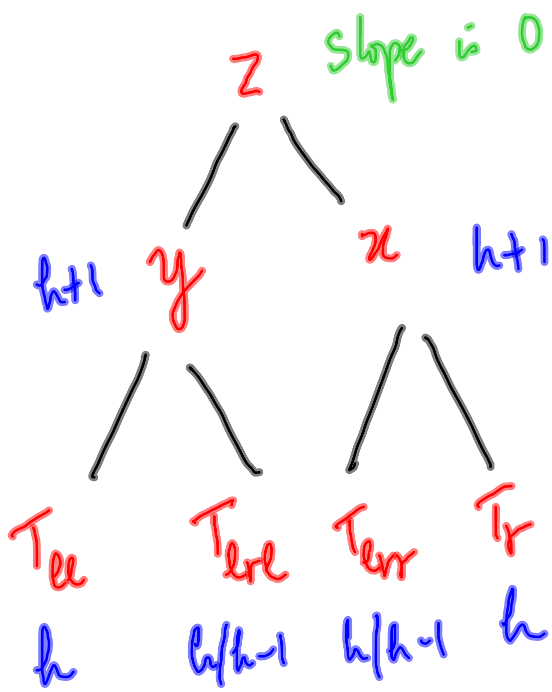
$Slope(T_\ell) \in \{0, 1\}$




x
y
T_r
h
T_{ll}
T_{lr}
h+1
h/h+1
y
slope ∈ {0,-1}
T_{ll}
h+1
x
h+1/h+2
T_{lr}
T_r
h/h+1
h


Slope (Tl) = -1
x
y
z
Tr
h
Tll
h
h+2
Tlrl
Tlrr
h+1
h/h-1
h/h-1
but at least one is h
creates
slope -2
at y
Instead
at y


x
y
Tr
h
Tll
z
h
Tlr
Trr
h/h-1
h/h-1
x
h+2
z
Tr
h
h+1
y
Tlrr
h/h-1
Tll
Tlrl
h
h/h-1


z
slope is 0
h+1
y
x
h+1
T_{ll}
T_{lrl}
T_{lrr}
T_r
h
h/h-1
h/h-1
h

if slope is 2

if left slope is -1

left rotate T_l

right rotate root

Each rotation takes a constant amount of time

Overall log n rebalances take $O(\log n)$ time if slope is -2 # symmetric

if slope of T_r is +1

right rotate T_r

left rotate at root

In search tree code, insert a rebalance after every recursive insert/delete

```
if x < self:
    if self.left:
        self.left.insert(x) / self.left.delete(x)
        self.rebalance()
```

How to compute slope?

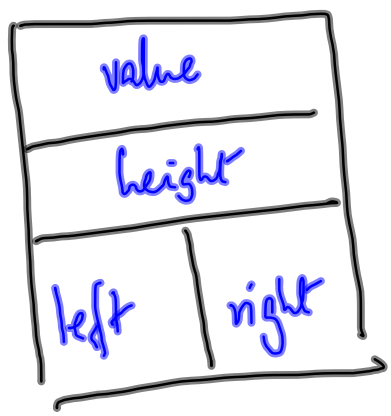
Compute height recursively each time

Requires time proportional to size(n) !!!!

At each node, maintain it's current height

Update with each change

Node looks like



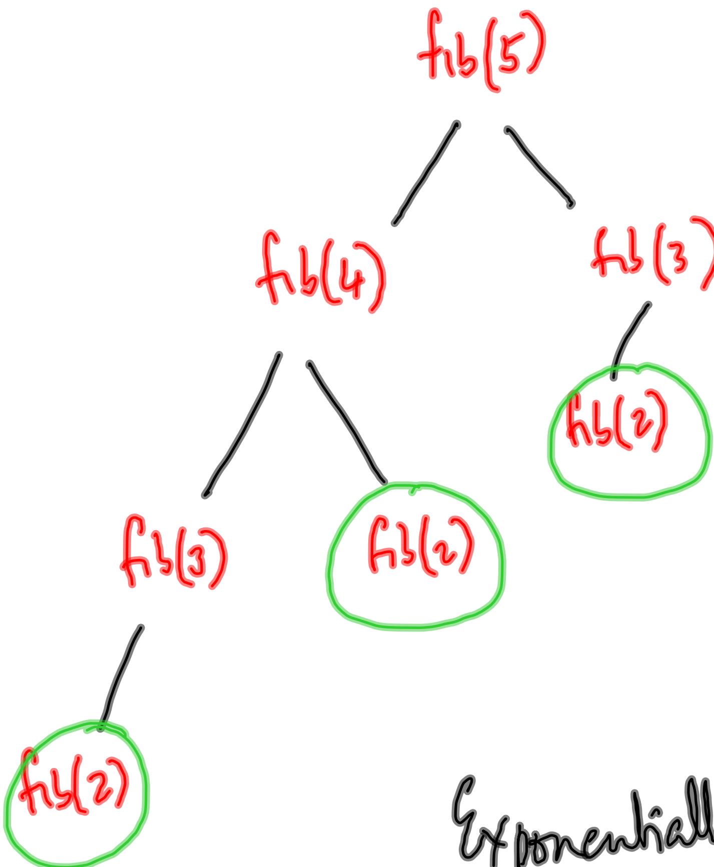
Evaluating recursive functions efficiently

$$\text{factorial}(n) = n \cdot \text{factorial}(n-1)$$

$$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$$

factorial(5) factorial(4) 3! 2! 1! 0!

120 24 6 2 1 1



Naively evaluate fib(n) from scratch each time I need it, repeated evaluation of same value

Exponentially many repeated computations in general

Solution

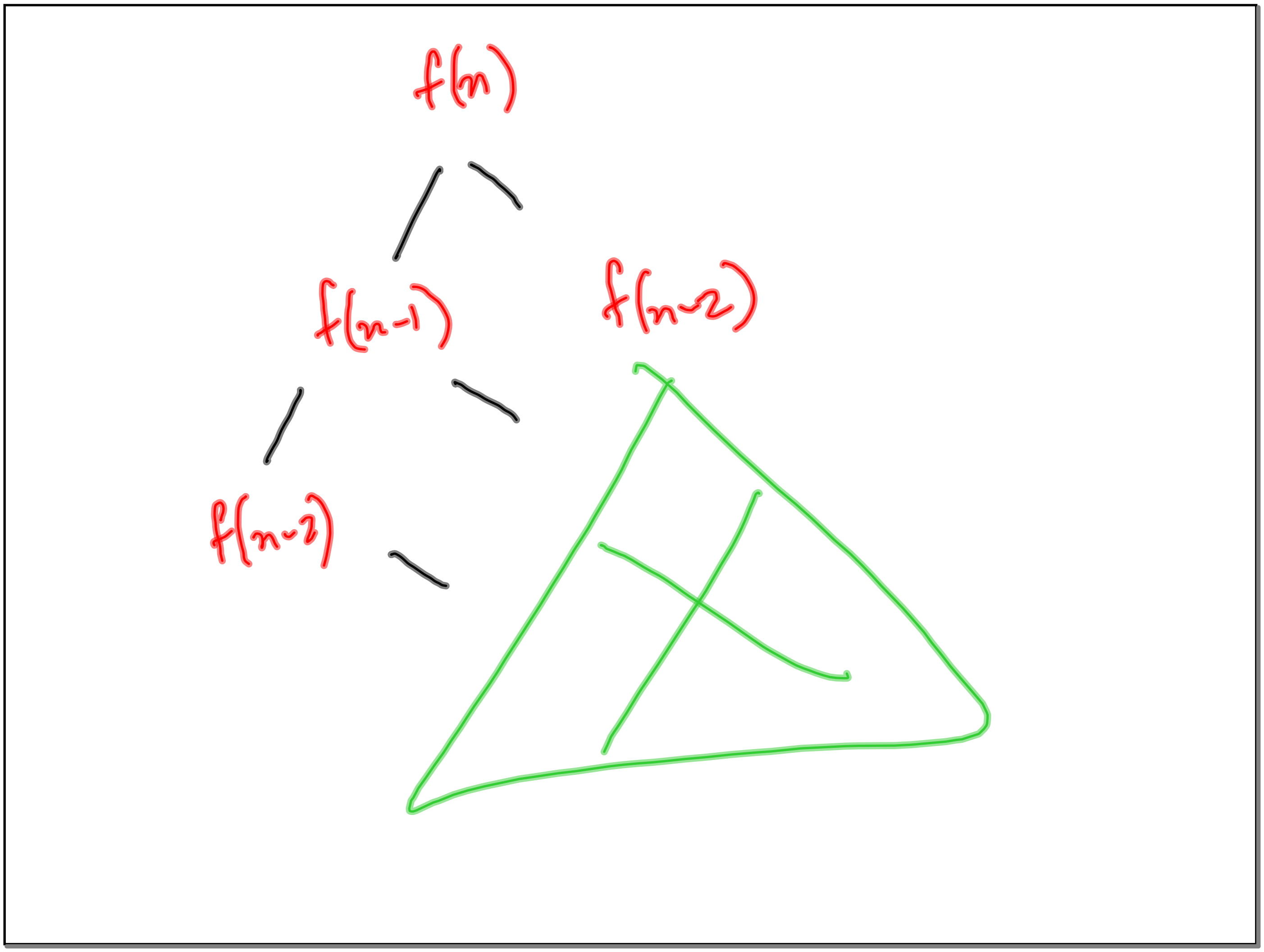
Remember previously evaluated values

fibonacci(n)

if fibonacci(j) exists in table
  look it up & do not compute
else
  do recursive computation
  store in table

Table

0	0
1	1
2	-1
3	-1
4	-1
5	-1
⋮	⋮
n-1	-1


f(n)
f(n-1)
f(n-2)
f(n-2)

Remembering in a table is called

Memoization

Write yourself a memo to remember

Dynamic Programming