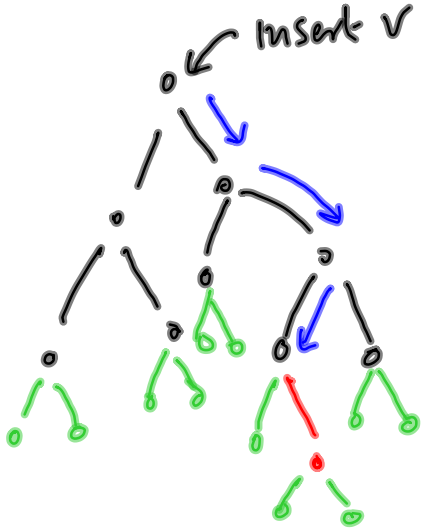
Height balanced trees

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

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

Assume tree is height balanced

insert / delete → restore balance



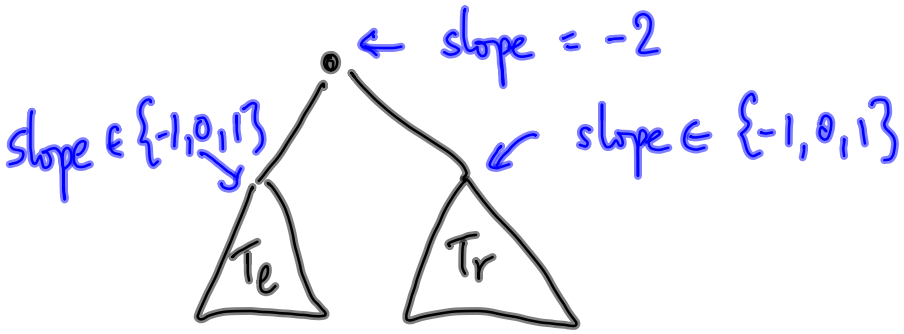
Fix balance at current node, assuming subtrees are balanced

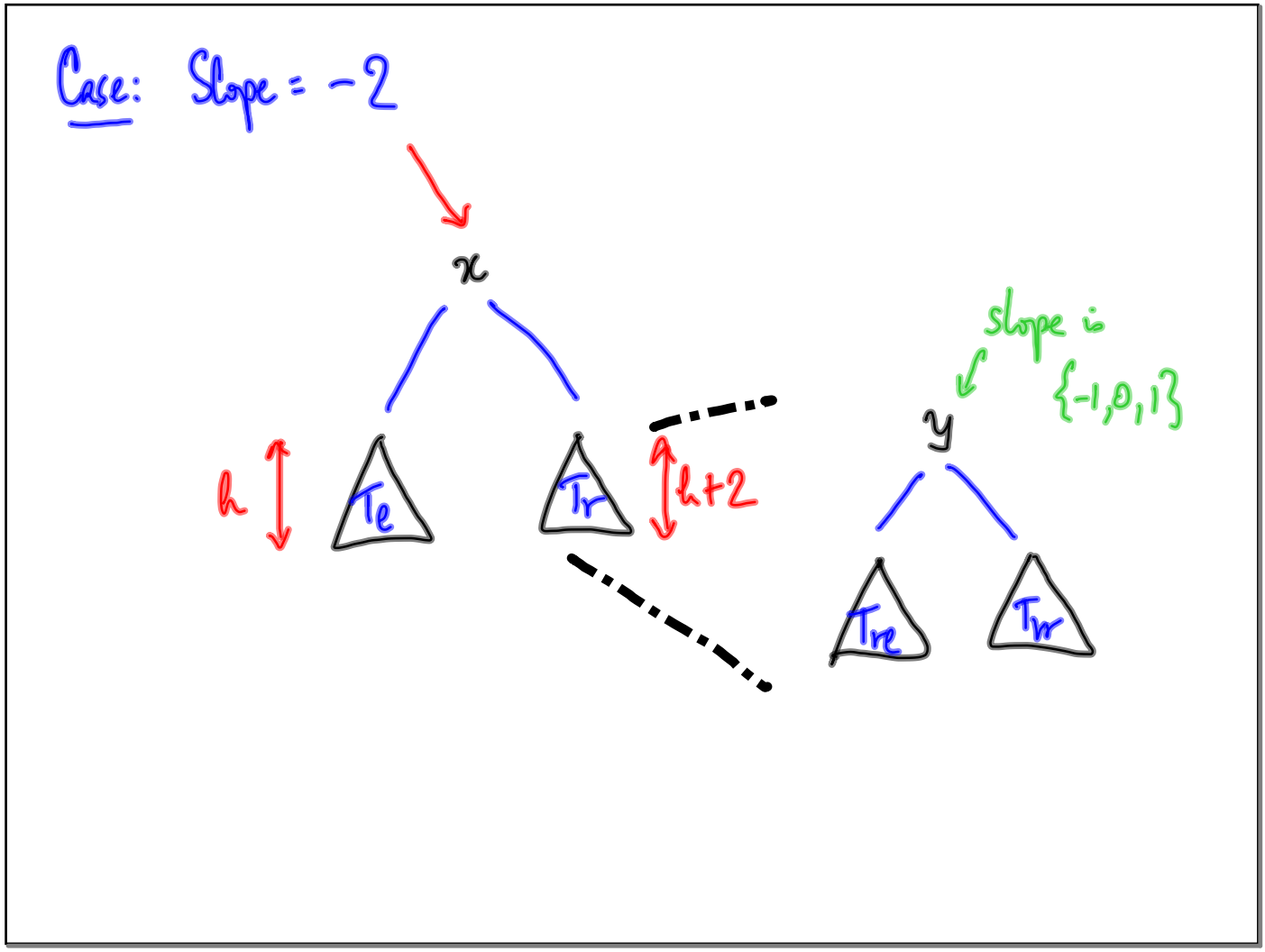
Amount of imbalance is limited

slope = h(left) - h(right)

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

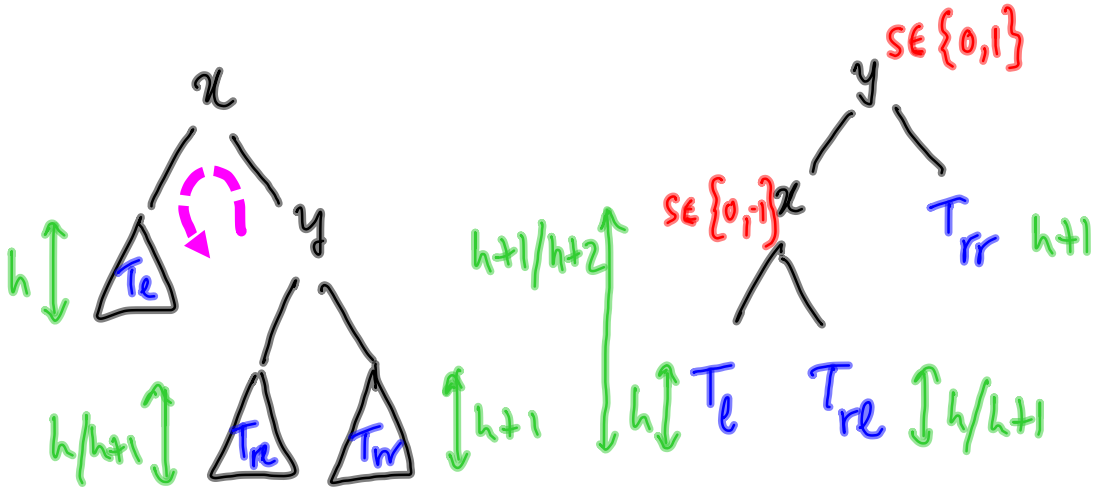
Unbalanced: Slope $\in \{-2, -1, 0, 1, 2\}$

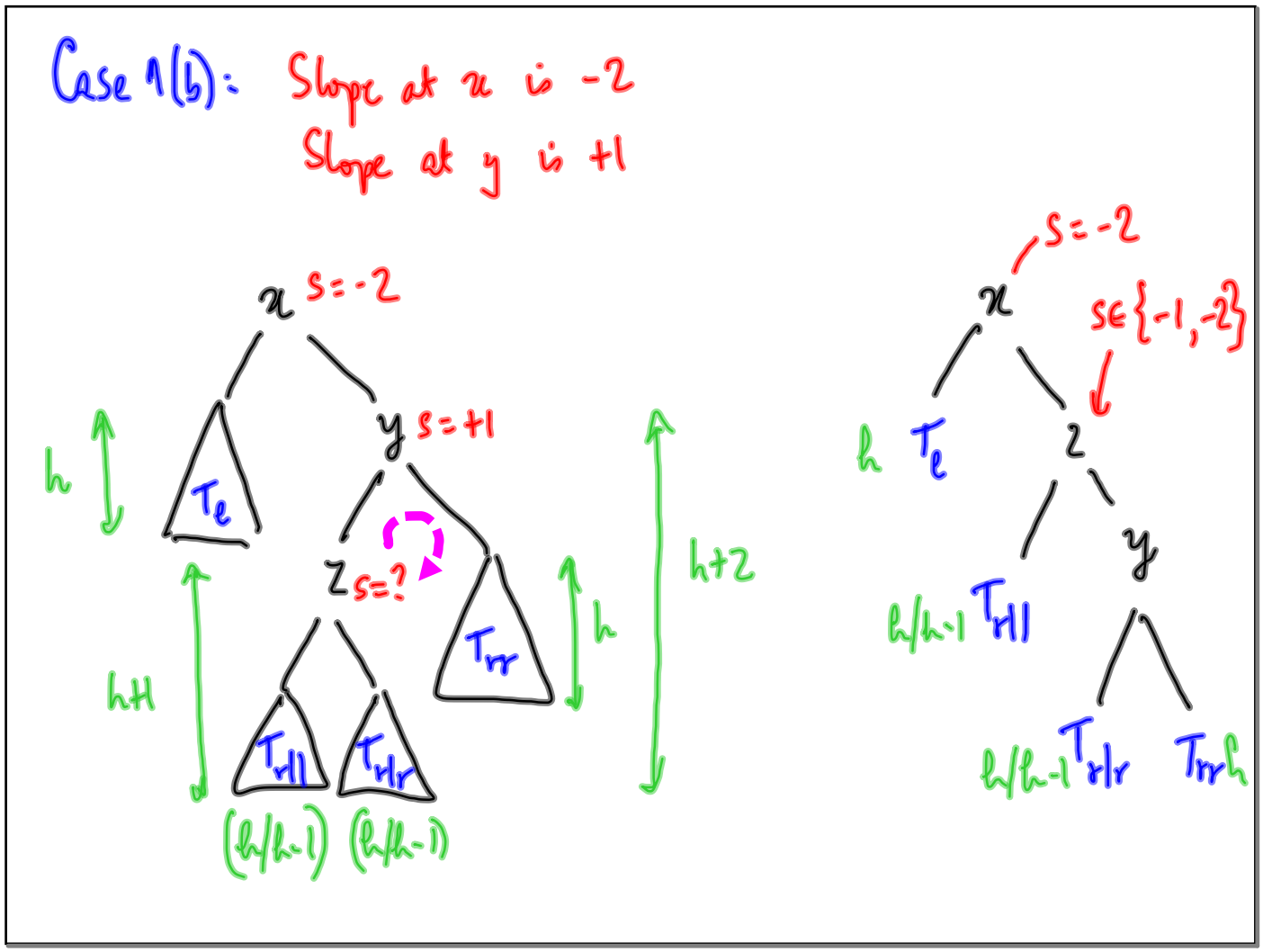



Case: Slope = -2
x
h
Tl
Tr
h+2
slope is
{-1,0,1}
y
Tre
Trr

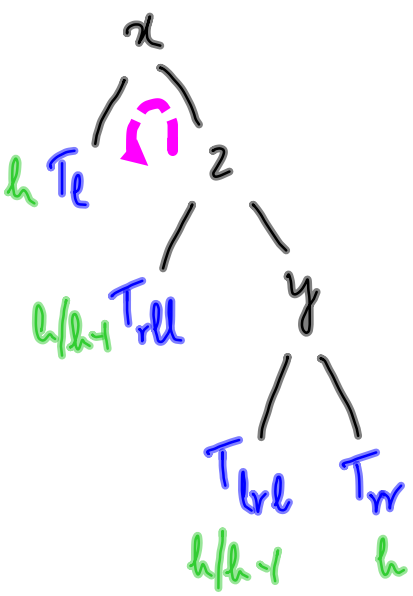
Case 1(a): Slope at x is -2

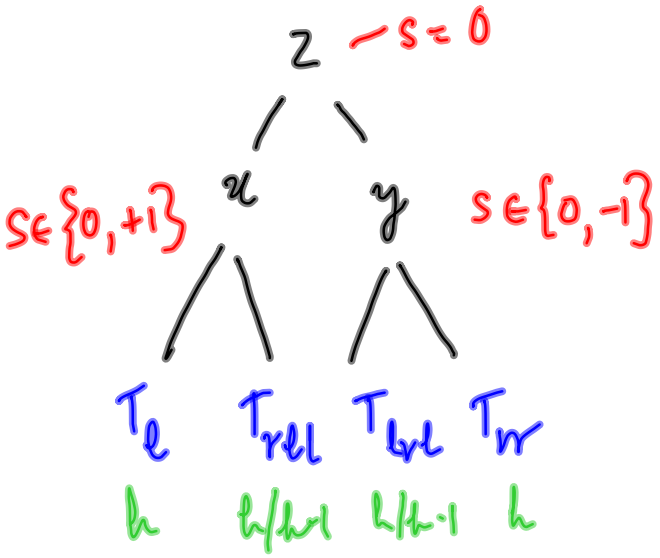
Slope at y is $\{0,-1\}$




Case 1(b): Slope at x is -2
Slope at y is +1
x s=-2
y s=+1
z s=?
h
h+1
h+2
h
(h/h-1)
(h/h-1)
s=-2
x
s∈{-1,-2}
z
y
h/h-1
h/h-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

```
insert x []     = [x]
insert x (y:ys)
  | x < y       = x : y : ys
  | otherwise   = y : (insert x ys)

isort []     = []
isort (x:xs) = insert x (isort xs)
```

O(n) where n = length of list

O(n²) : 1+2+...+n-1

Insertion sort produces a new list

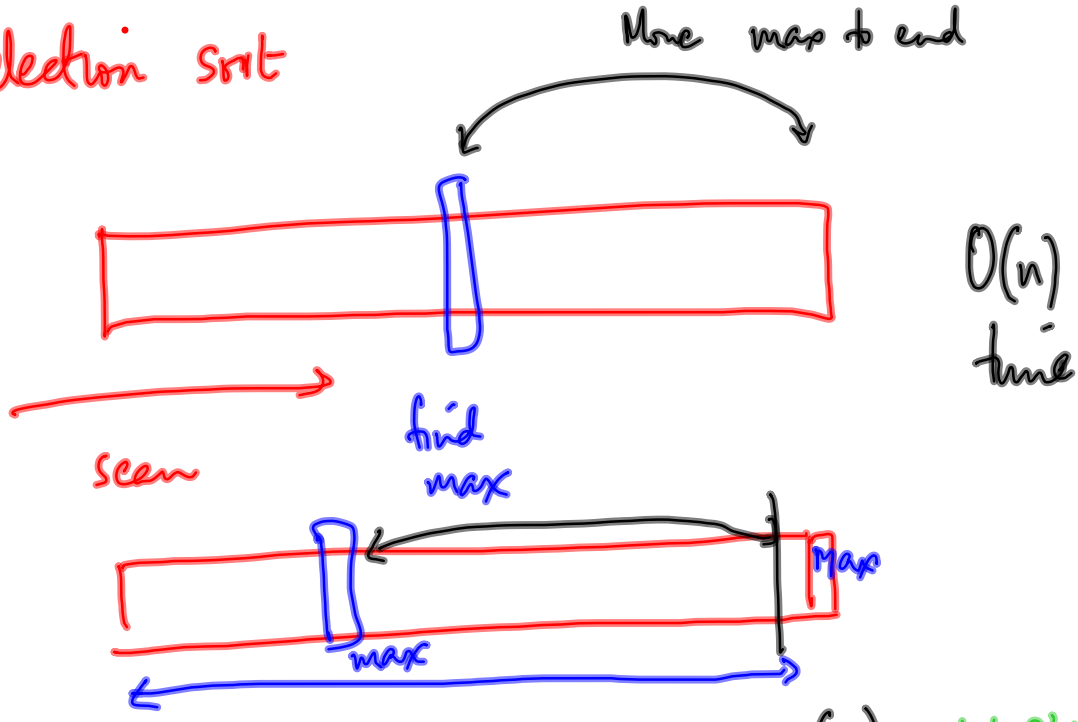
Traditionally – measure time taken by an algorithm

- What about space – In place sort
- Data movement

In place sort

Naive in place sort

Selection sort



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