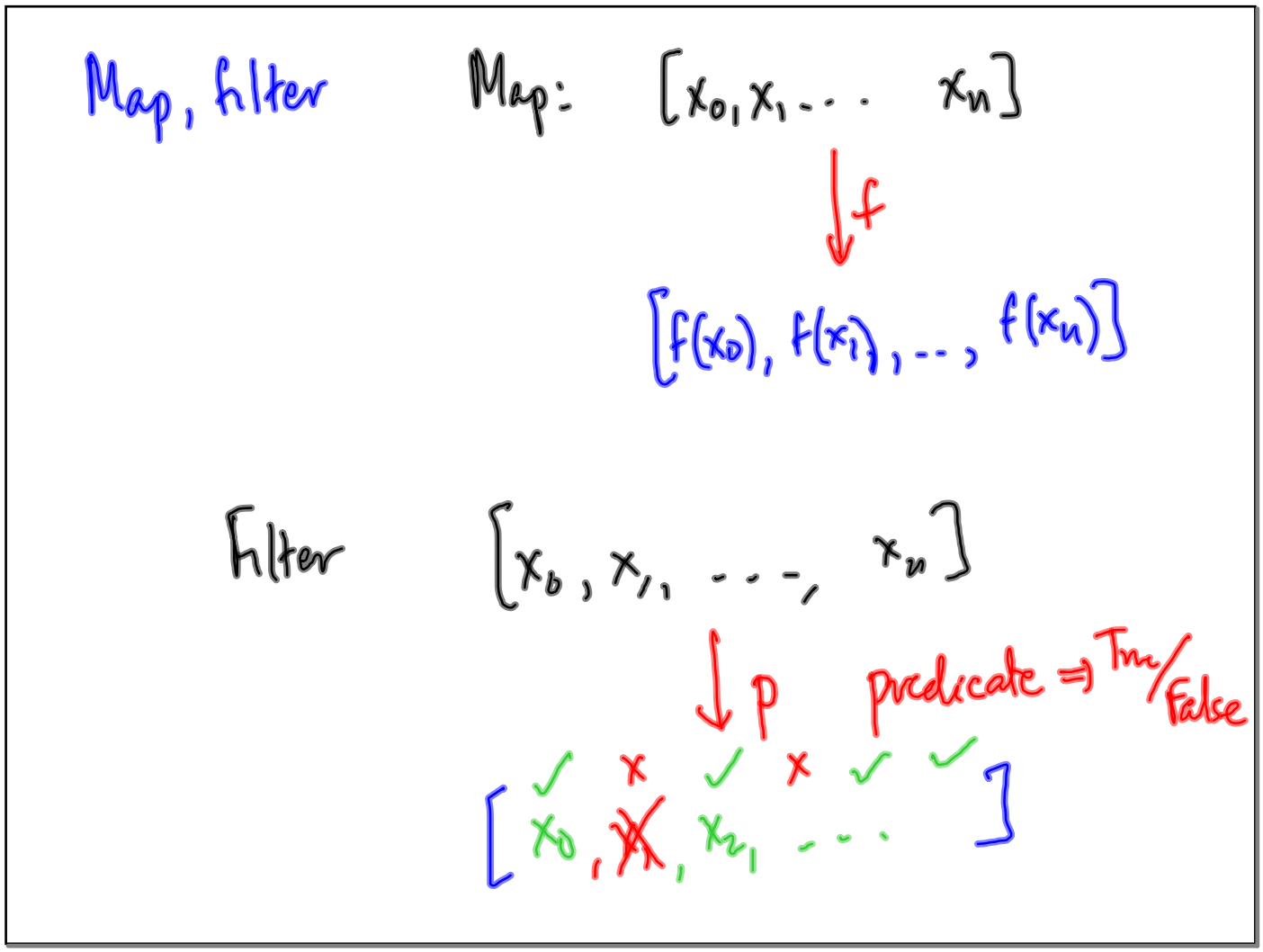

Map, filter
Map: [x0, x1 . . . xn]
f
[f(x0), f(x1), . . , f(xn)]
Filter [x0, x1, . . . , xn]
p predicate ⇒ True/False
[x0, x2, . . .]

List comprehension

map generator filter

```
[ x**2 for x in range(10) if iseven(x) ]
```

$$\{x^2 \mid x \in \{0,1,\ldots,9\}, x \text{ is even}\}$$

Multiple generators

map creates pairs

no filter

```
[(x,y) for x in range(10) for y in range(10)]
```

2 generators

⇓

like a nested for

```
for x in range(10):
    for y in range(10):
```

```
[(x,y,z) for x in range(1,100)
         for y in range(x,100)
         for z in range(y,100)
             if x**2 + y**2 == z**2]
```

Refine this

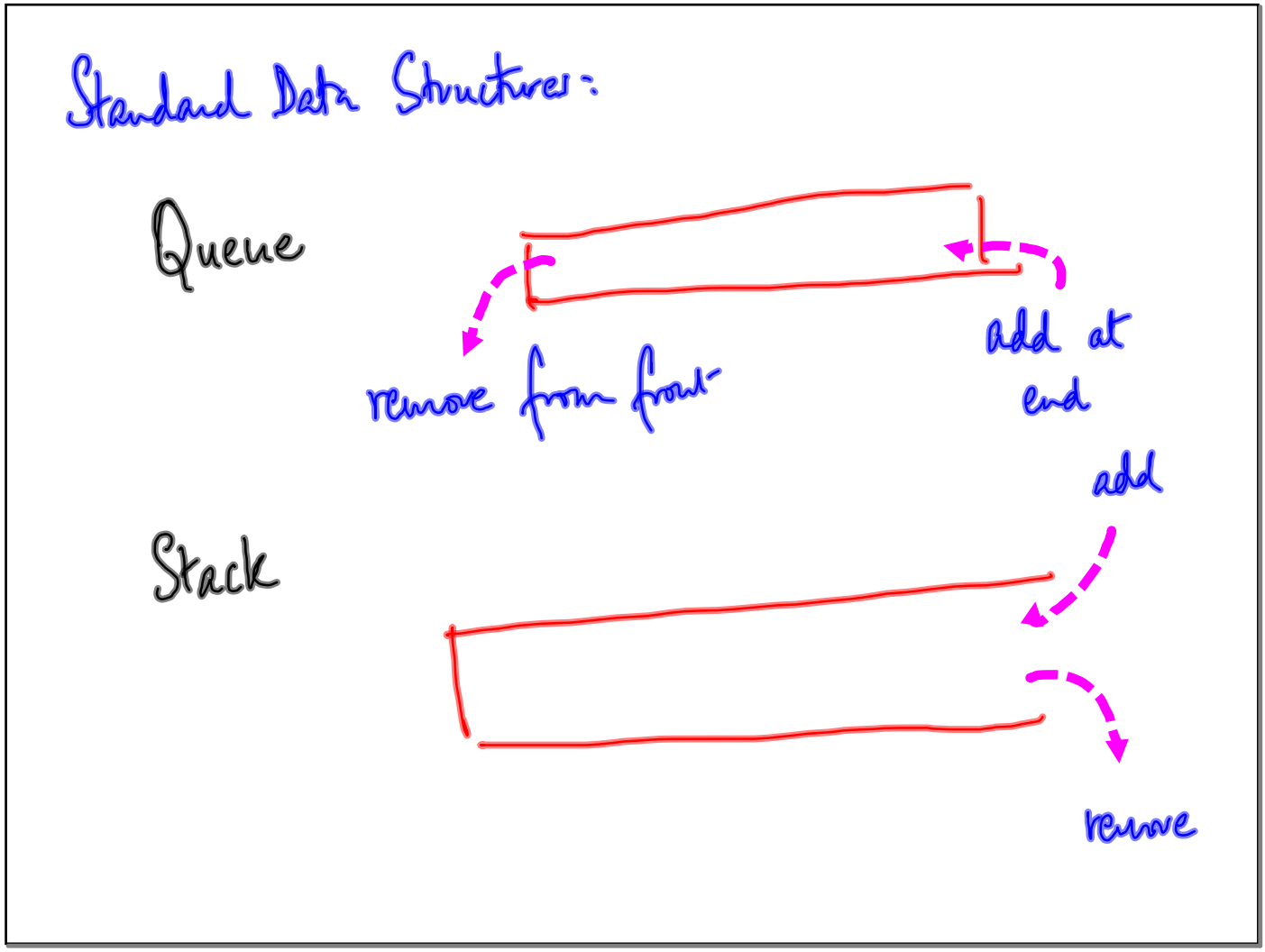
```
for x in range(1,100)
    for y in range(x,100)
        for z in range(y,100)
```

Niklaus Wirth

Algorithms + Data Structures = Programming

Sorting a list helps searching

Data structure: way of organizing data
to optimize specific operations


Standard Data Structures:
Queue
remove from front
add at end
Stack
add
remove

Priority Queue

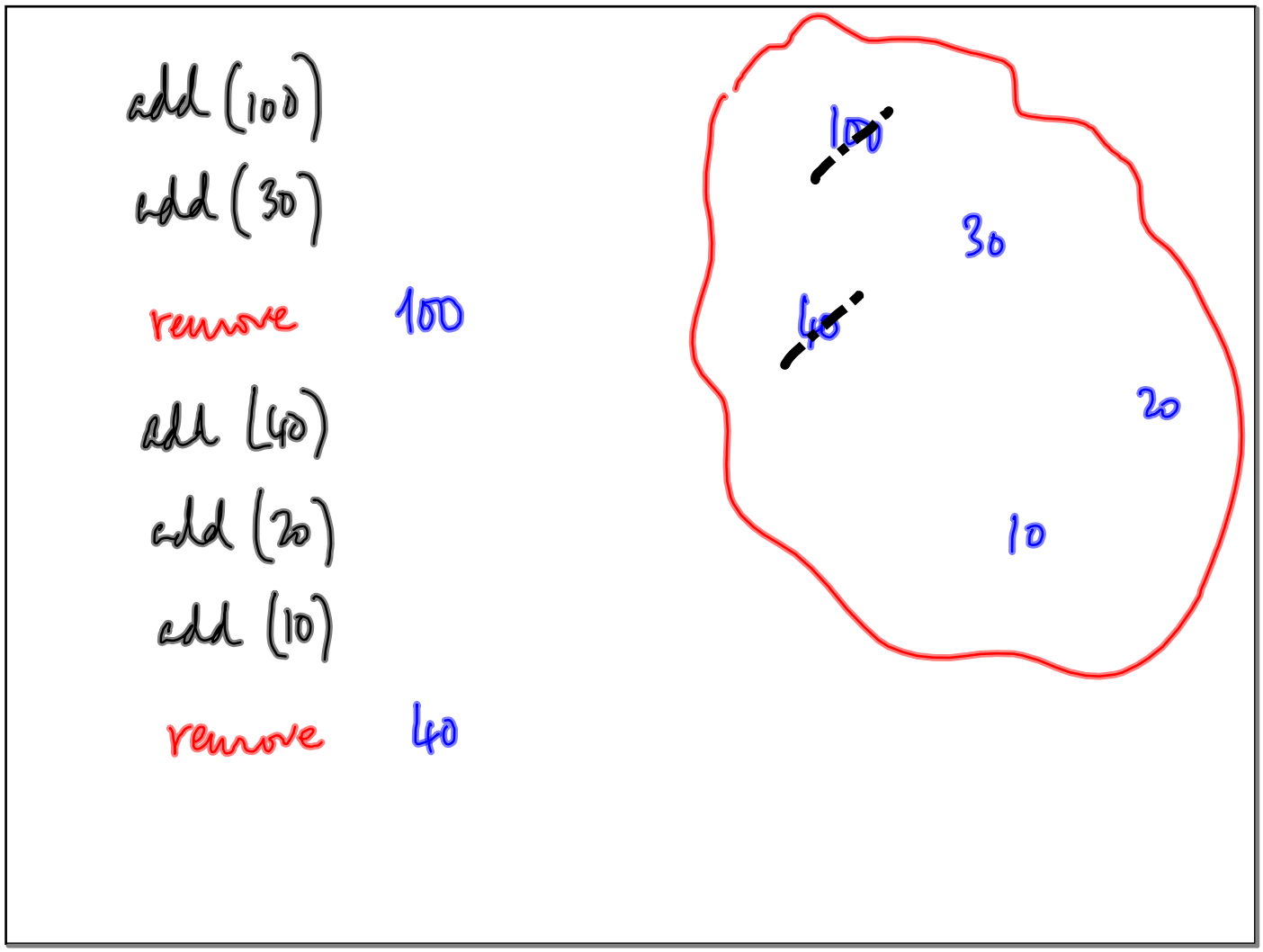
Like a queue, but pending requests/customers have priorities

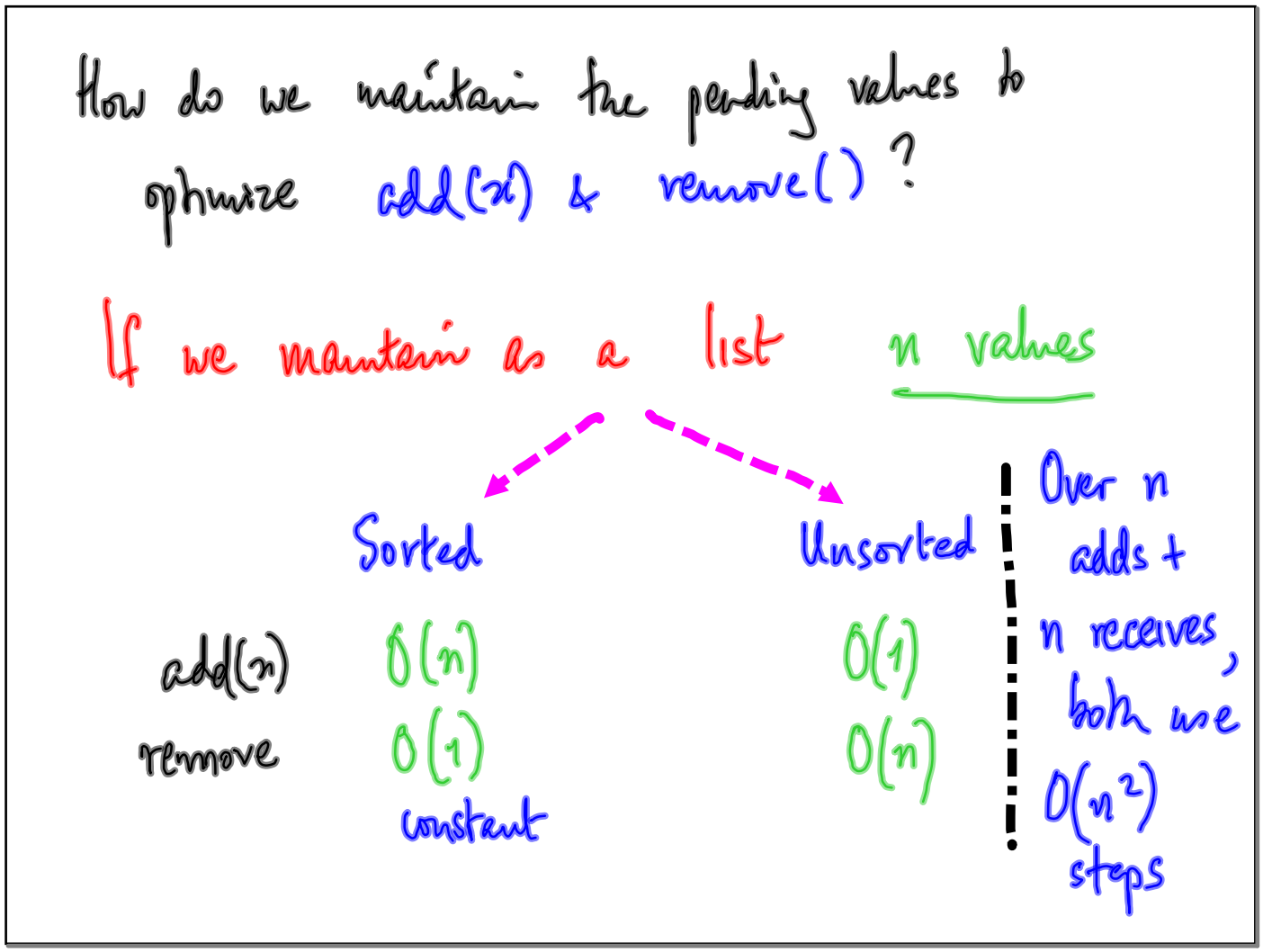
Next process the highest priority pending customer

Two operations

Add to queue

Remove from queue ← extract current value of highest priority

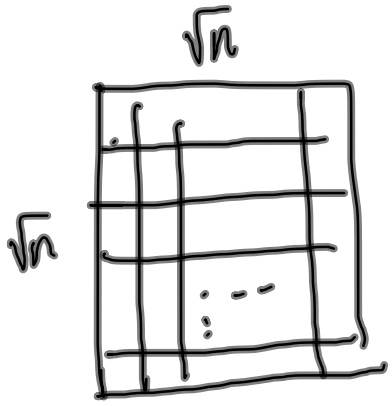

add (100)
add (30)
remove 100
add (40)
add (20)
add (10)
remove 40
100
30
40
20
10


How do we maintain the pending values to optimize add(x) & remove() ?
If we maintain as a list n values
Sorted
Unsorted
Over n adds + n receives, both use O(n²) steps
add(x)
remove
O(n)
O(1)
O(1)
O(n)
constant

Key observation

One dimension is inadequate

Store upto n values in 2D

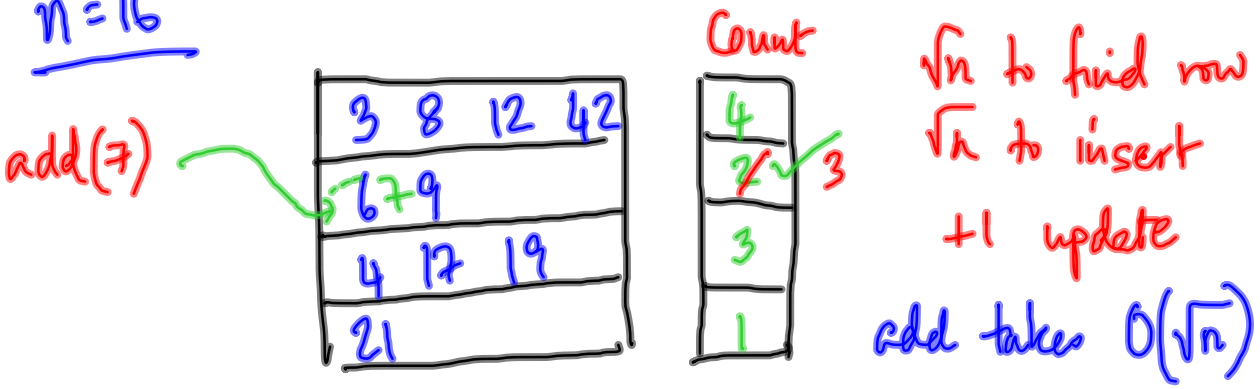


Suggestion:

Keep each row sorted

add: Insert into first row that has an empty square

$n = 16$



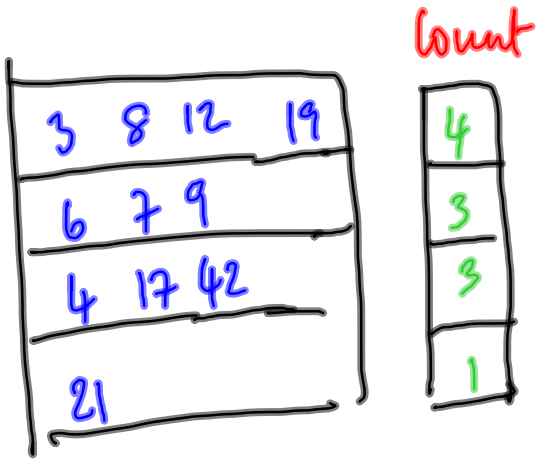
$\sqrt{n}$ to find row

$\sqrt{n}$ to insert

+1 update

add takes $O(\sqrt{n})$

remove



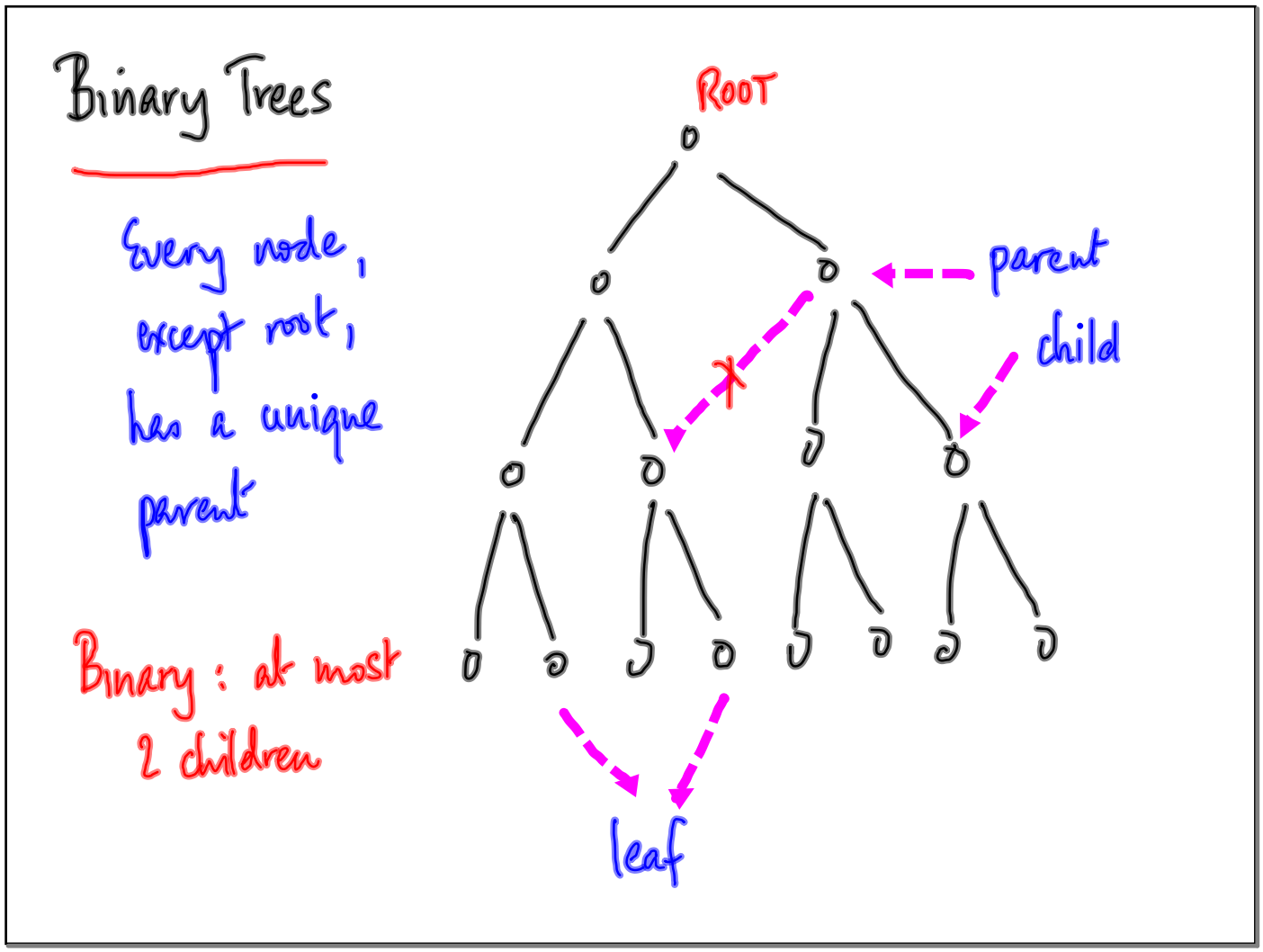
In each row, max value is at position "count"

Examine $\sqrt{n}$ values & take the max

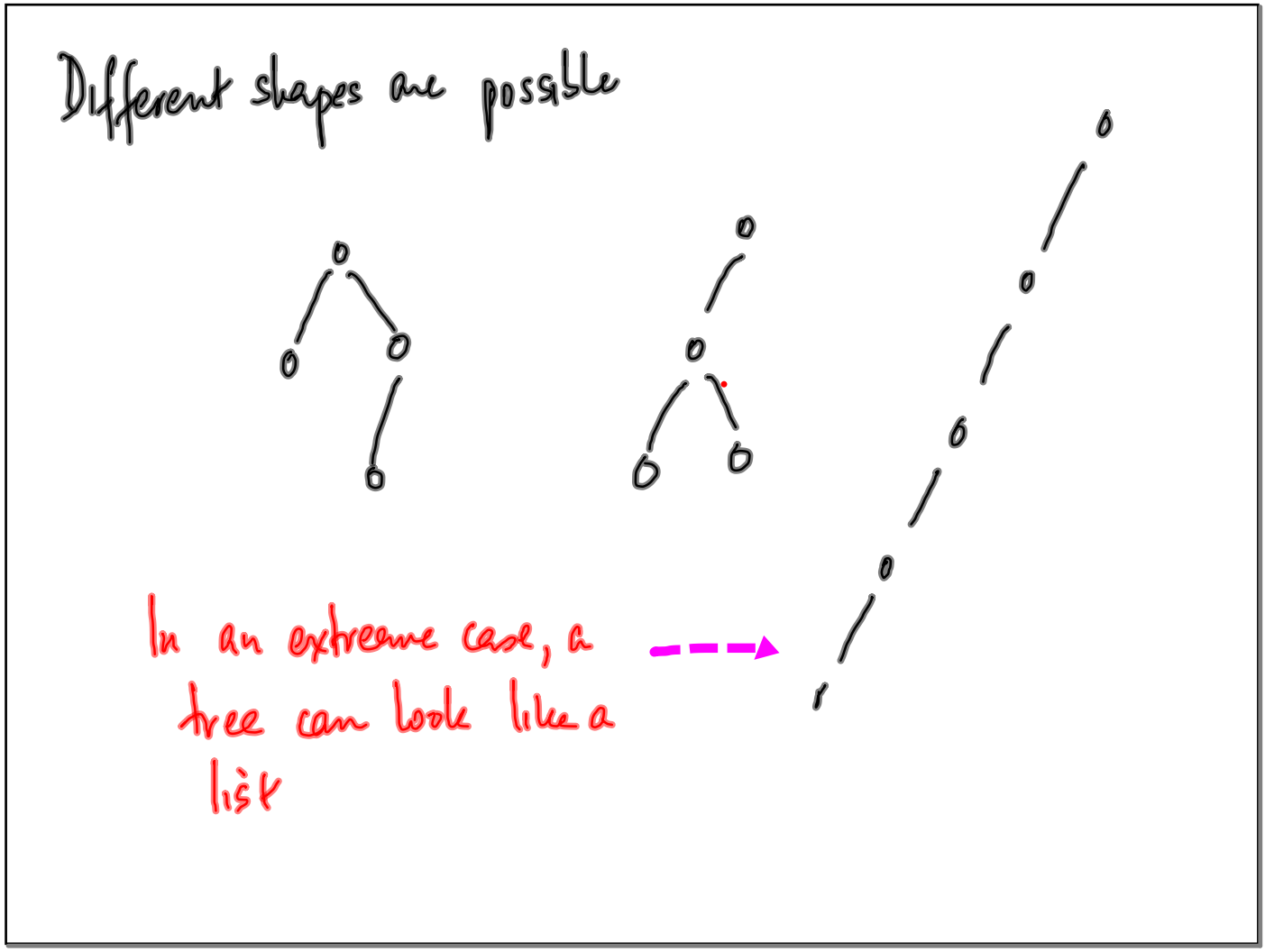
remove is also $\sqrt{n}$

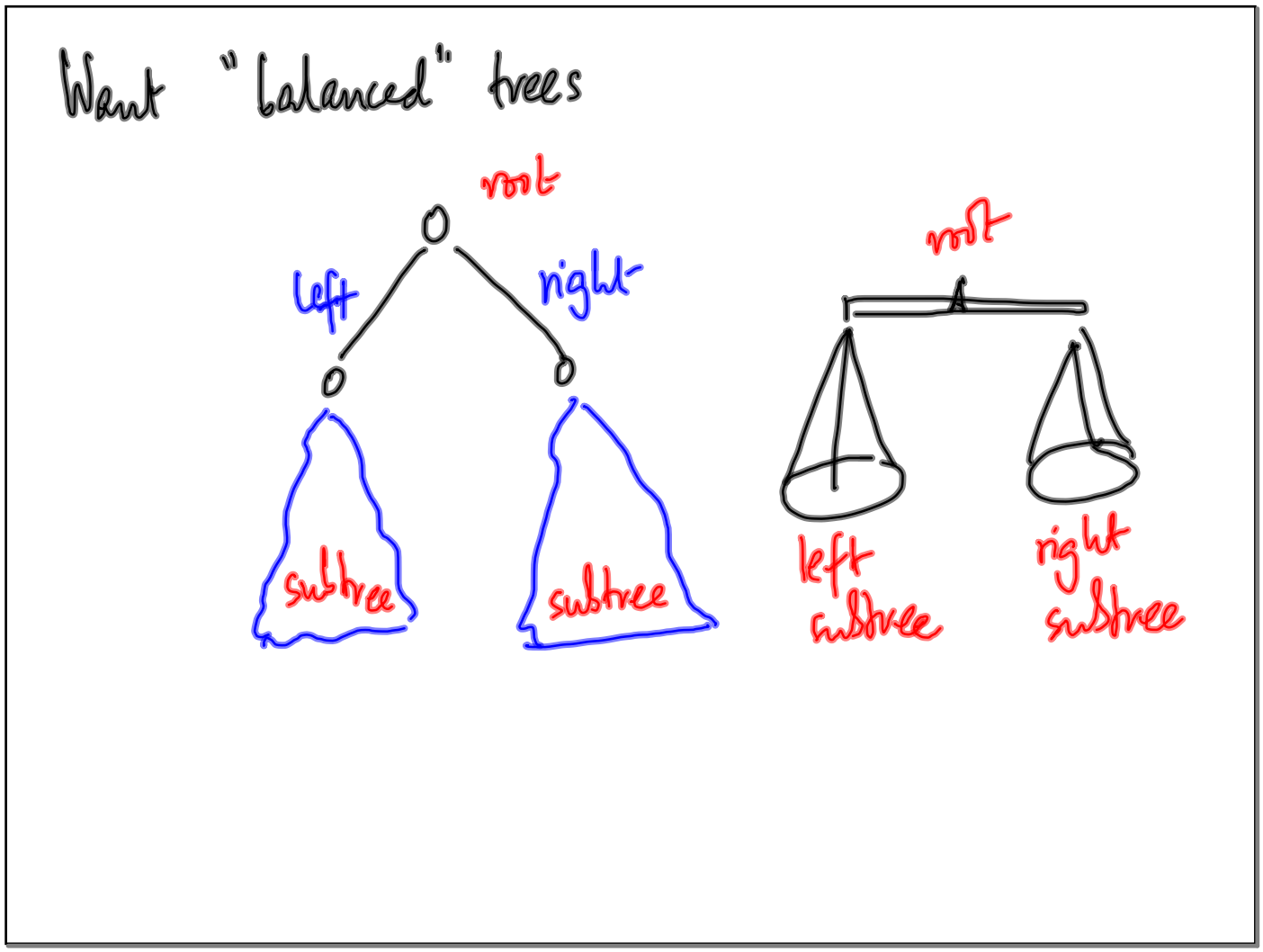
	add	remove	n ops
Sorted list	n	1	n^2
Unsorted list	1	n	n^2
$\sqrt{n} \times \sqrt{n}$ array	$\sqrt{n}$	$\sqrt{n}$	$n\sqrt{n}$

Can do much better.


Binary Trees
Every node, except root, has a unique parent
Binary : at most 2 children
ROOT
parent
child
leaf

Different shapes are possible




Want "balanced" trees
root
left
right
subtree
subtree
root
left subtree
right subtree

Simplest measure to balance a tree is

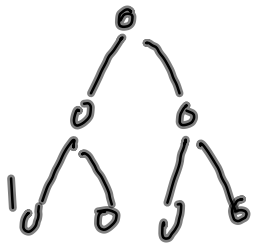
size = no. of nodes

Perfect balance:

At every node size(left) = size(right)

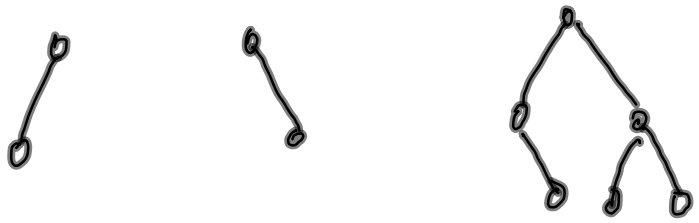
Restricts shapes:

"Complete" tree



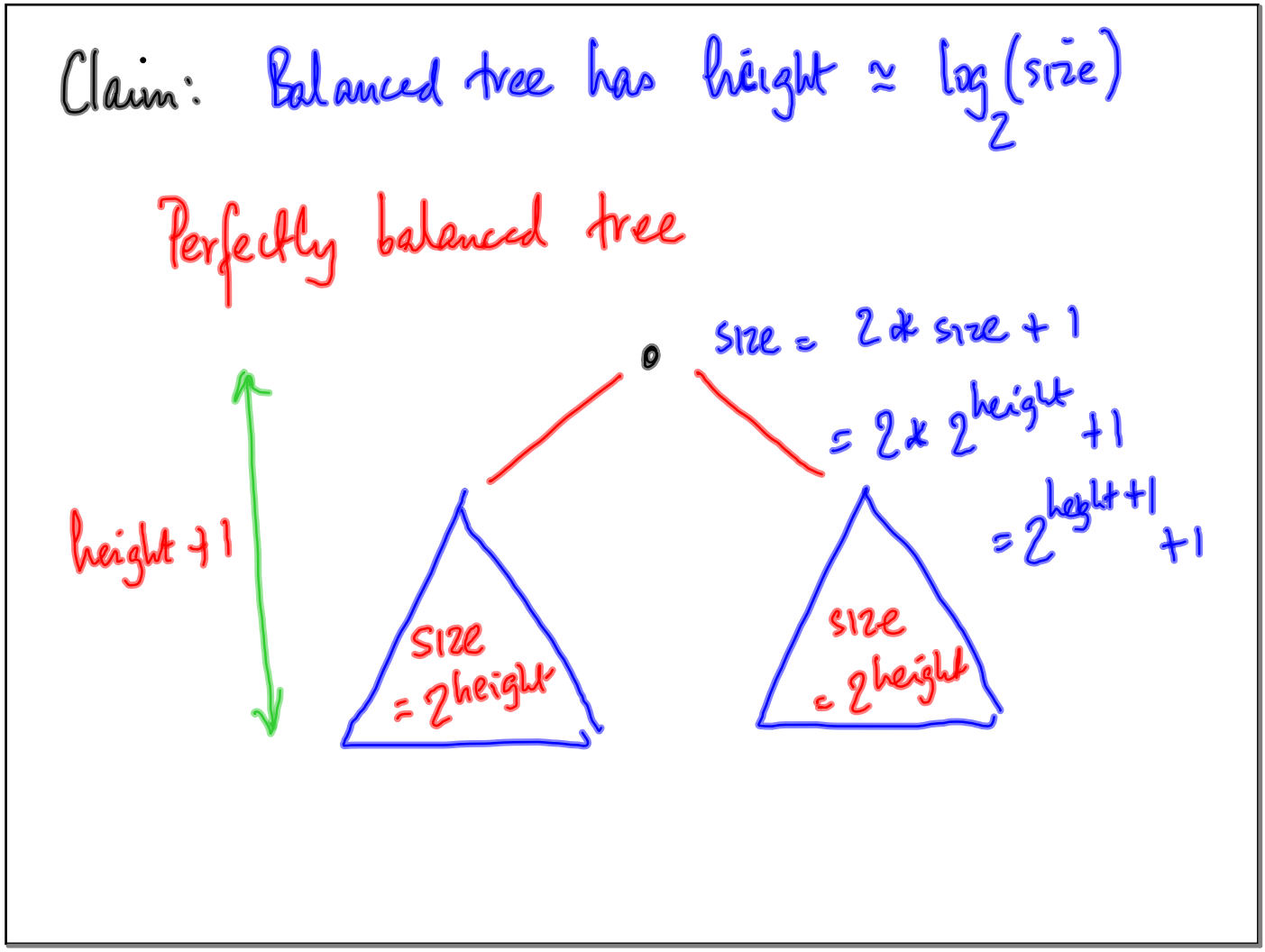
Slightly weaker:

$$|size(left) - size(right)| \leq 1$$



Balanced trees:

Relationship between size (no. of nodes) and height (longest path from root to leaf)


Claim: Balanced tree has height ≃ log2(size)
Perfectly balanced tree
size = 2 * size + 1
= 2 * 2^height + 1
= 2^(height+1) + 1
height + 1
size = 2^height
size = 2^height

Priority queue

Simultaneously optimize add(x), remove_max()

Maintain values in a special kind of tree called a heap

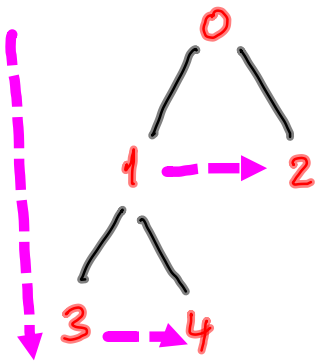
Heap property:
1. Nodes are added level by level, left to right
2. Every parent is ≥ both children

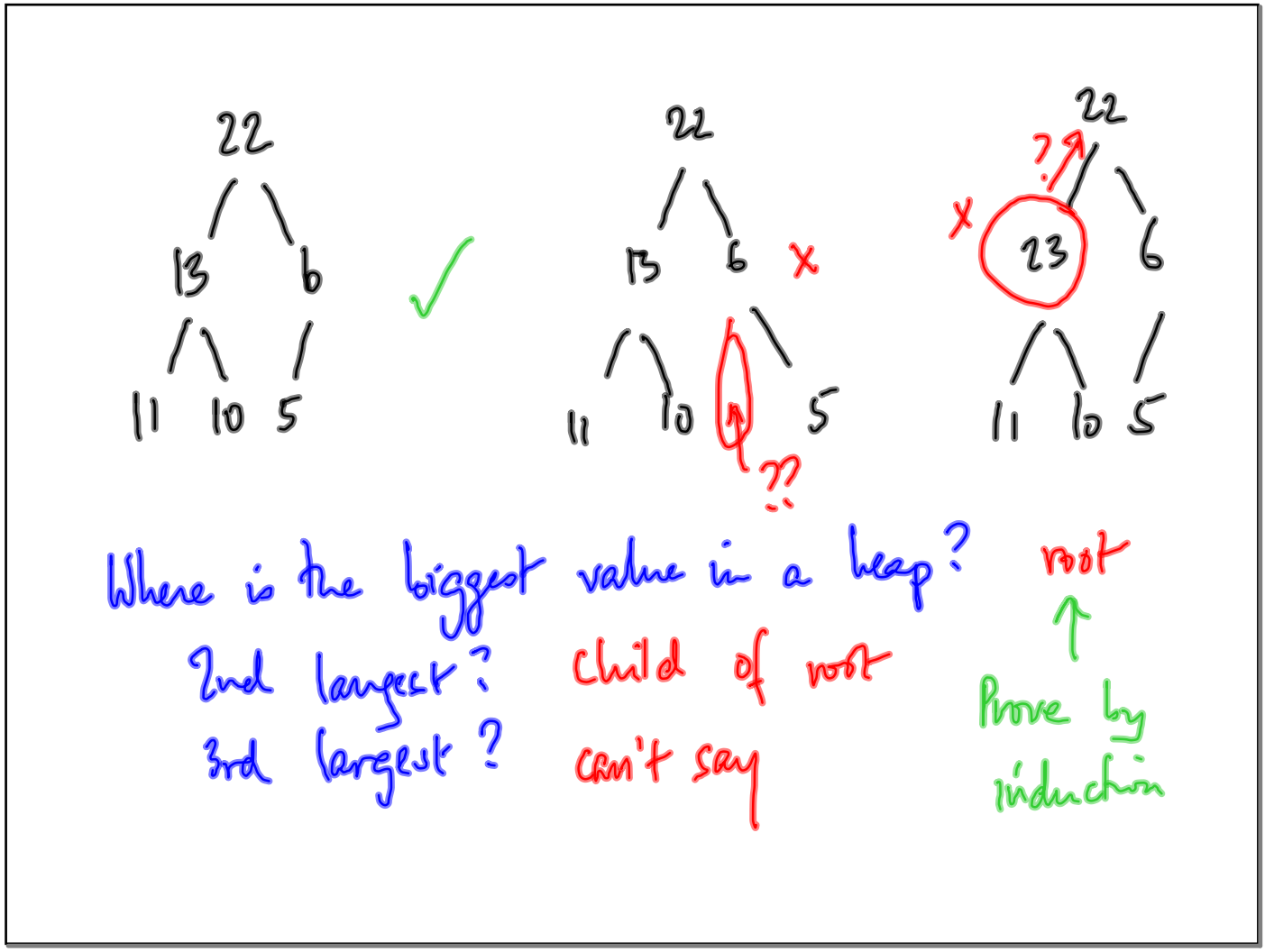
Heap property 1:

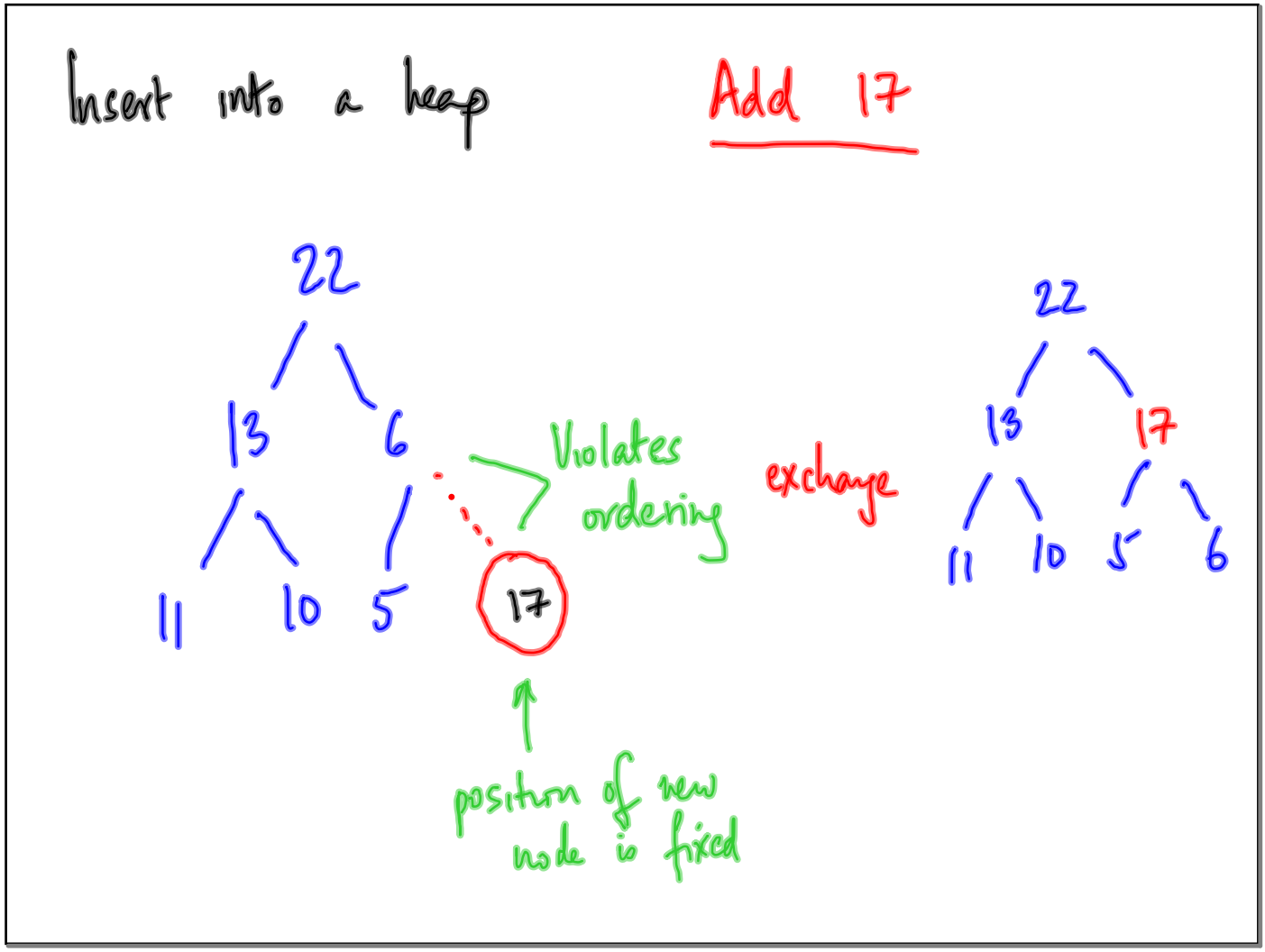
Tree grows as follows

Bottom row may be incomplete

No gaps above




22
13 6
11 10 5
22
13 6 x
11 10 5
??
22
?
x
23 6
11 10 5
Where is the biggest value in a heap? root
2nd largest? child of root
3rd largest? can't say
Prove by induction


Insert into a heap
Add 17
22
13
6
11
10
5
17
Violates ordering
position of new node is fixed
exchange
22
13
17
11
10
5
6

In general, may have to swap up till root