Recitation 5/26

Heap

→ def : Binary Tree like structure which is as complete as possible

→ data structure

heap ← some other low level DS $\boxed{\text{(array)}}$

(conceptual) (implementation)

→ **property** / invariant : → Max heap

→ the root is max (recursively true)

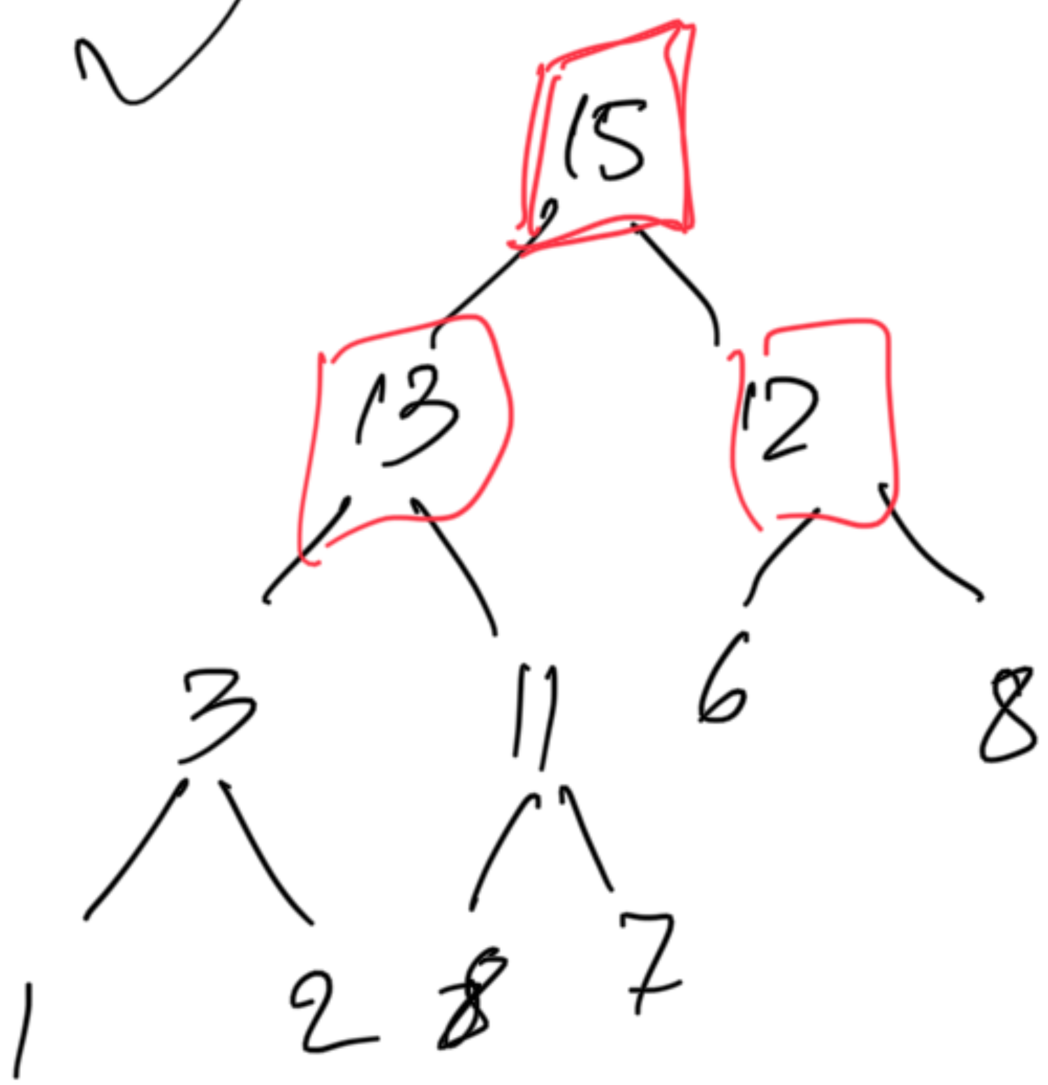
→ operations

	complexity
→ find_max	$O(1)$
→ insert	$O(\log n)$
→ delete	$O(\log n)$

→ build $O(n)$

→ sort $O(n \log n)$

→ code implementation



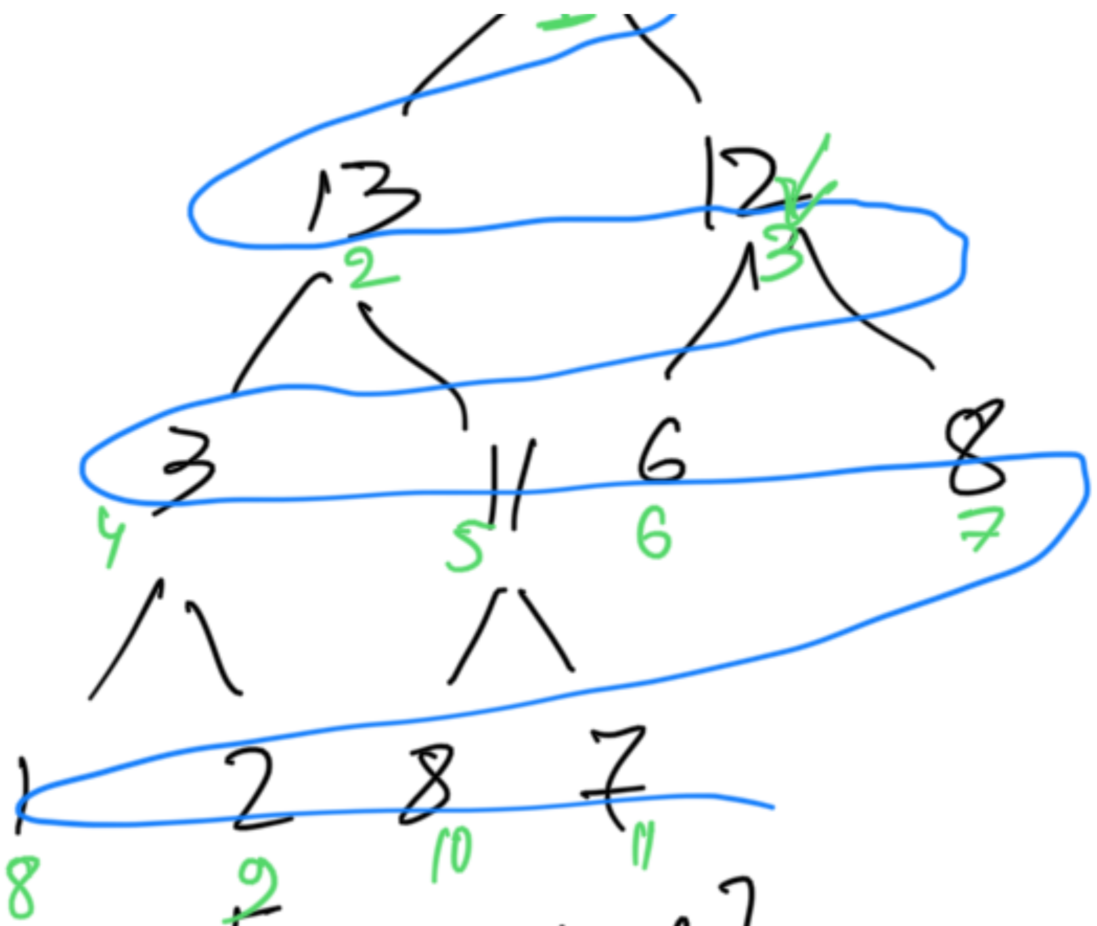
datastructure

heap ← Array (low level ds)

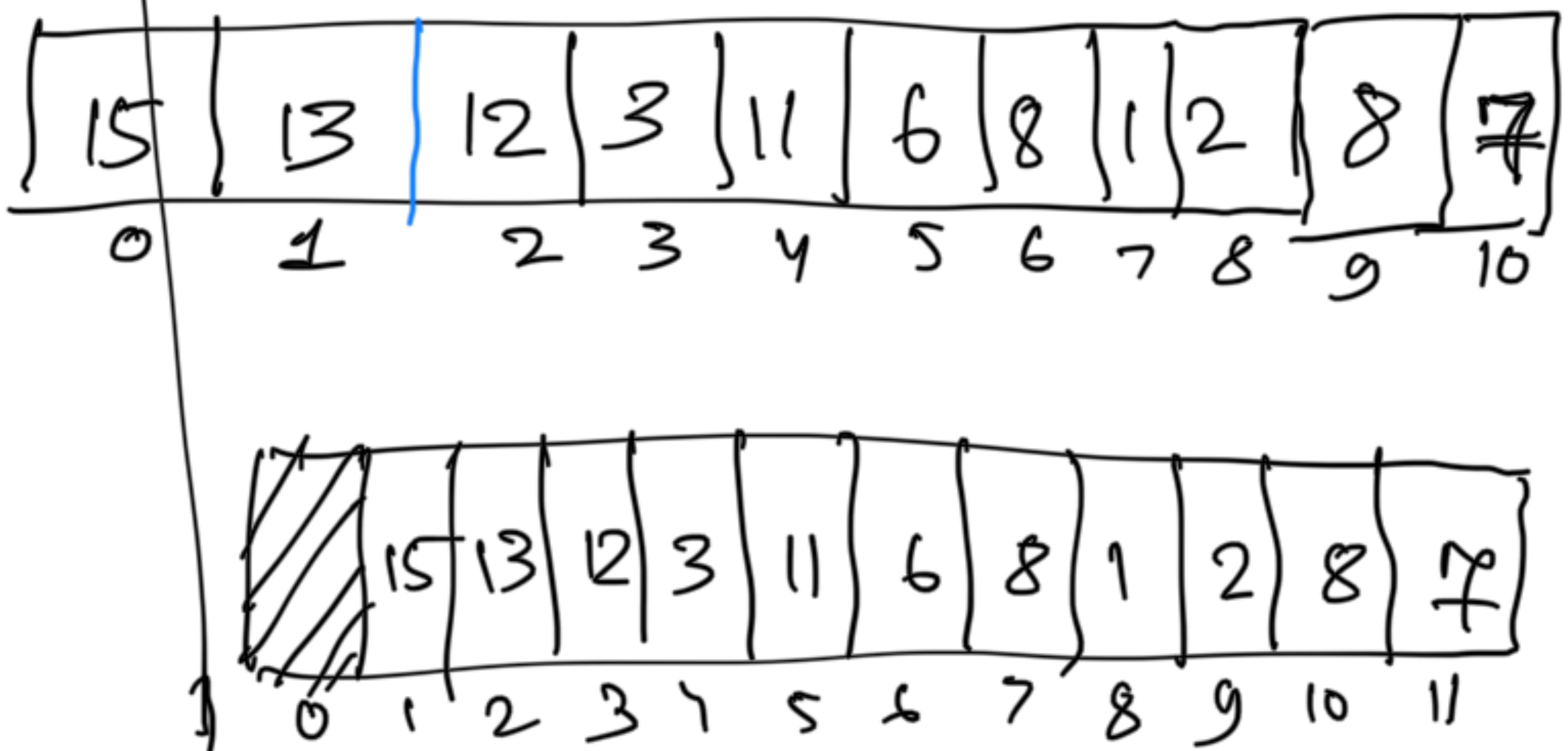
Heap

15

Array



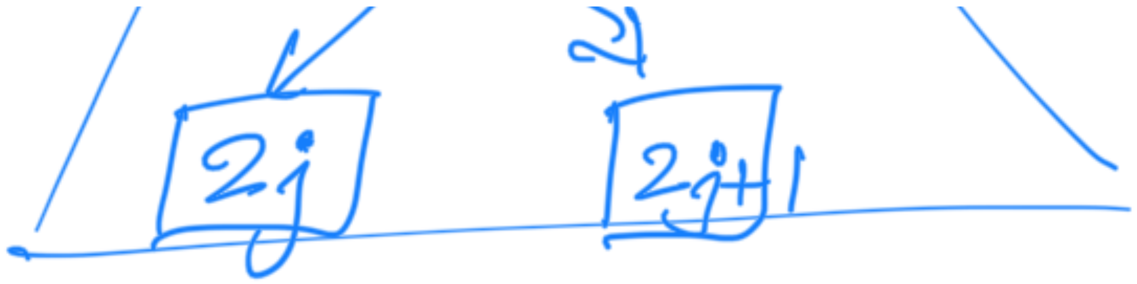
[conceptual]



[Reality]

Crucial operation in Implementation:-

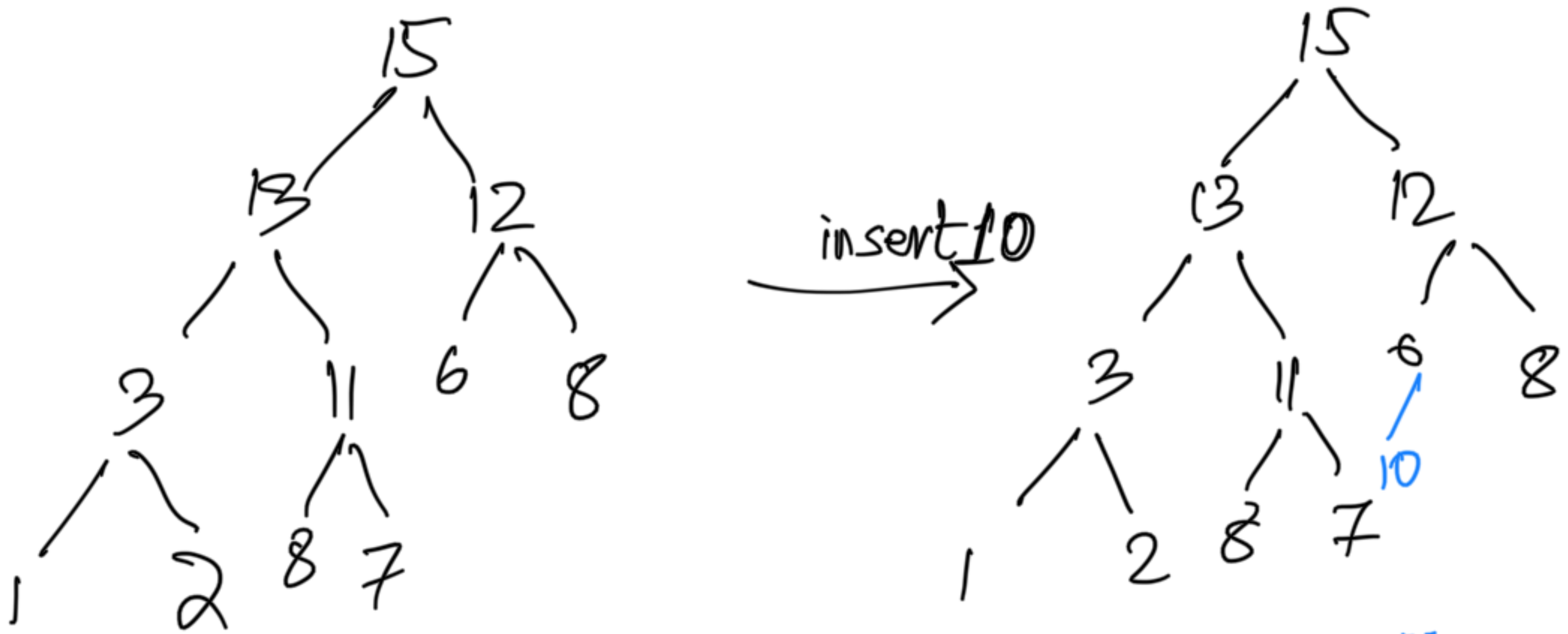
Comparison



Operations

→ find_max : O(1) lookup

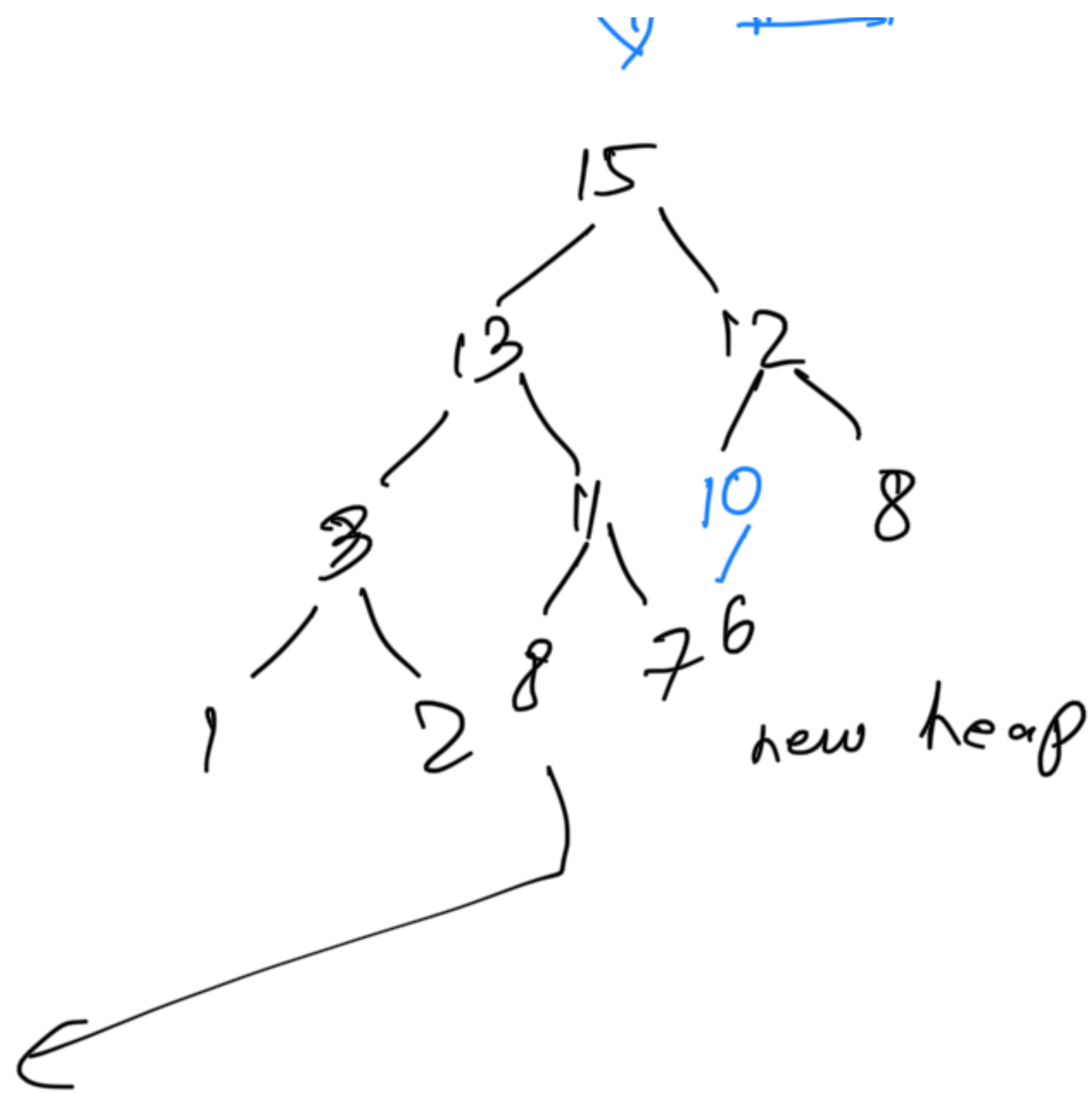
→ insert :



(insert at the last spot)

→ invariant is false

→ $\boxed{\text{fix}}$ the heap



Algo :

→ insert at the last spot

→ bubble it up until you hit the right spot, swapping elements as you go.

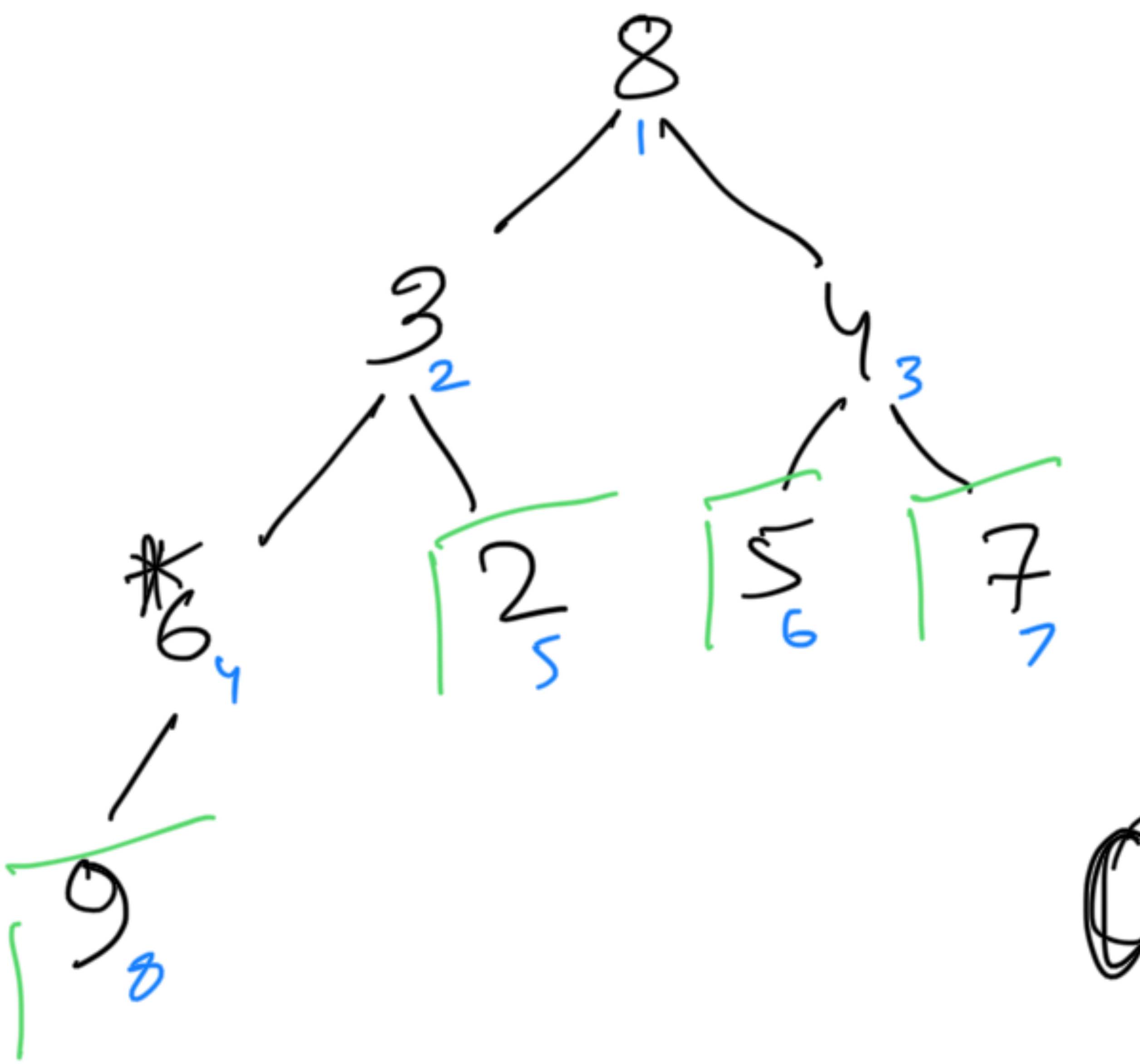
build

→ All leaves are already heaped

→ bubble down the non-heap elements

→ keep building the heap bottom up.

8, 3, 4, 6, 2, 5, 7, 9



	8	3	4	6	2	5	7	9
0	1	2	3	4	5	6	7	8

$O(n)$

```
def build( . )
    heaped = size
    while heaped > 0
        bubble_down(heaped)
        heaped --
```

$\boxed{\text{elements[heaped+1 :] have been heaped}}$

0th heaped = 8

[ ] ← Initially

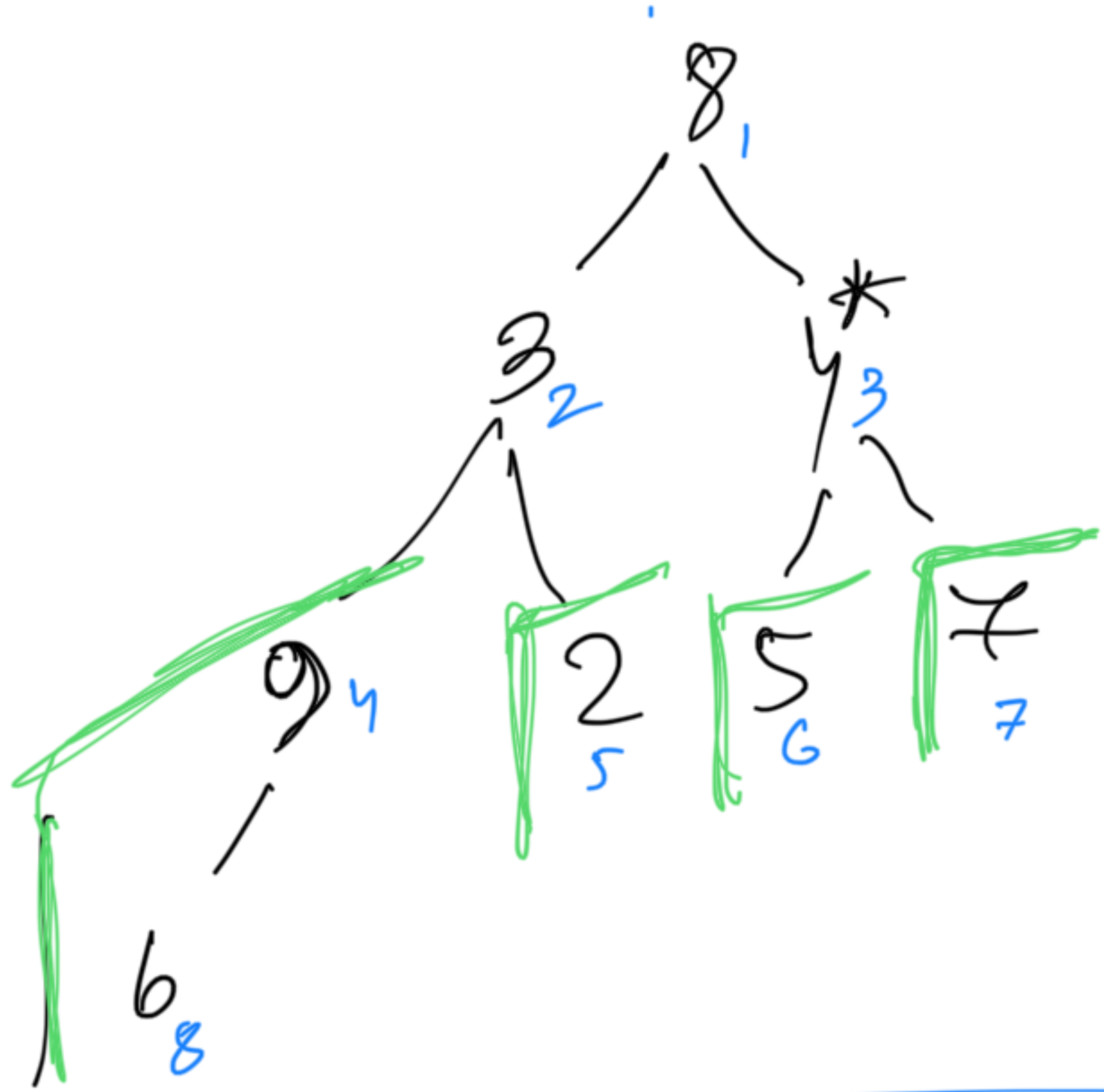
1st bubbledown(8)

heaped = 7

heaped = 4

bubble_down(4)

elements[4+1 :] have been heaped

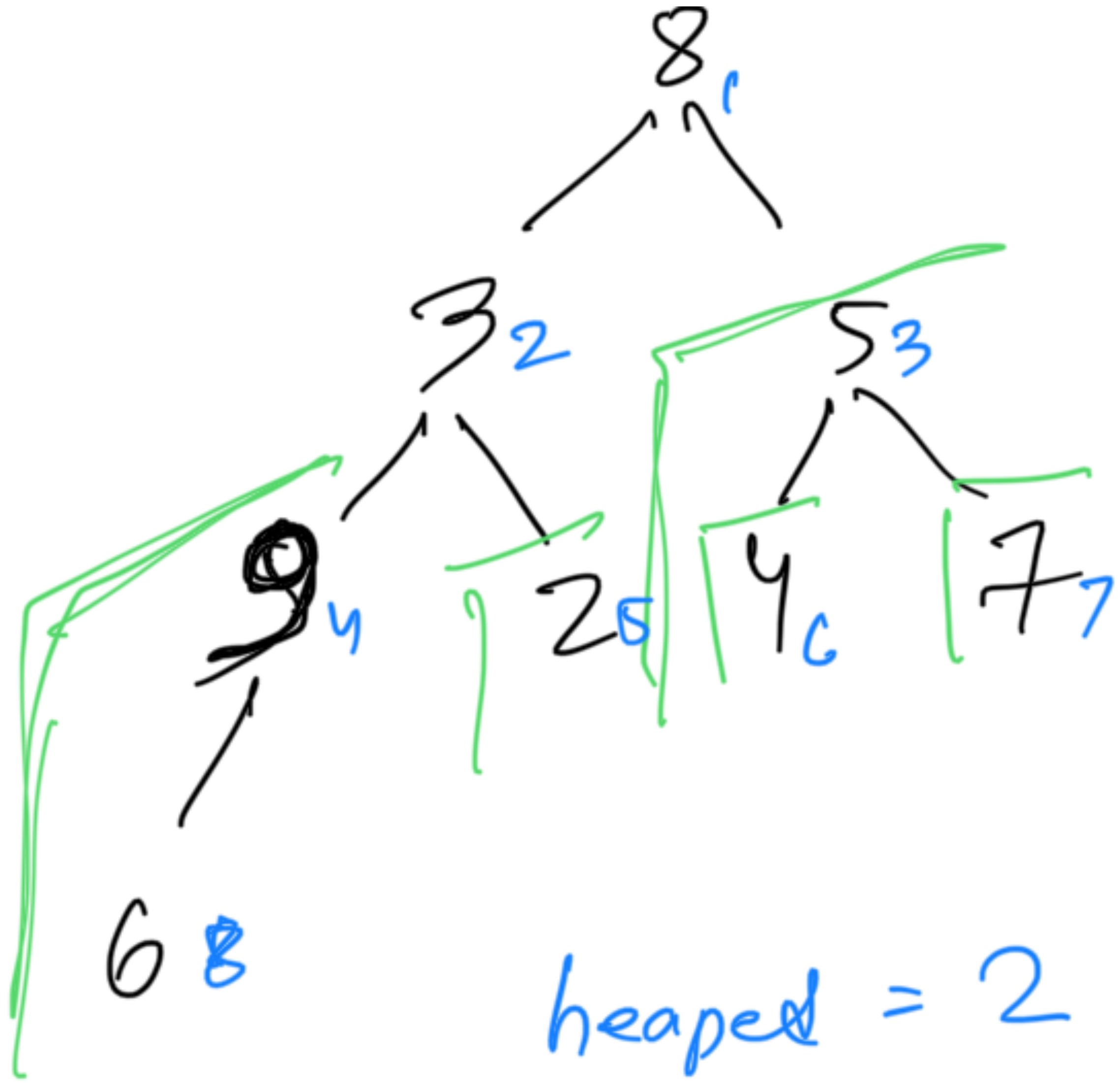


heaped = 3

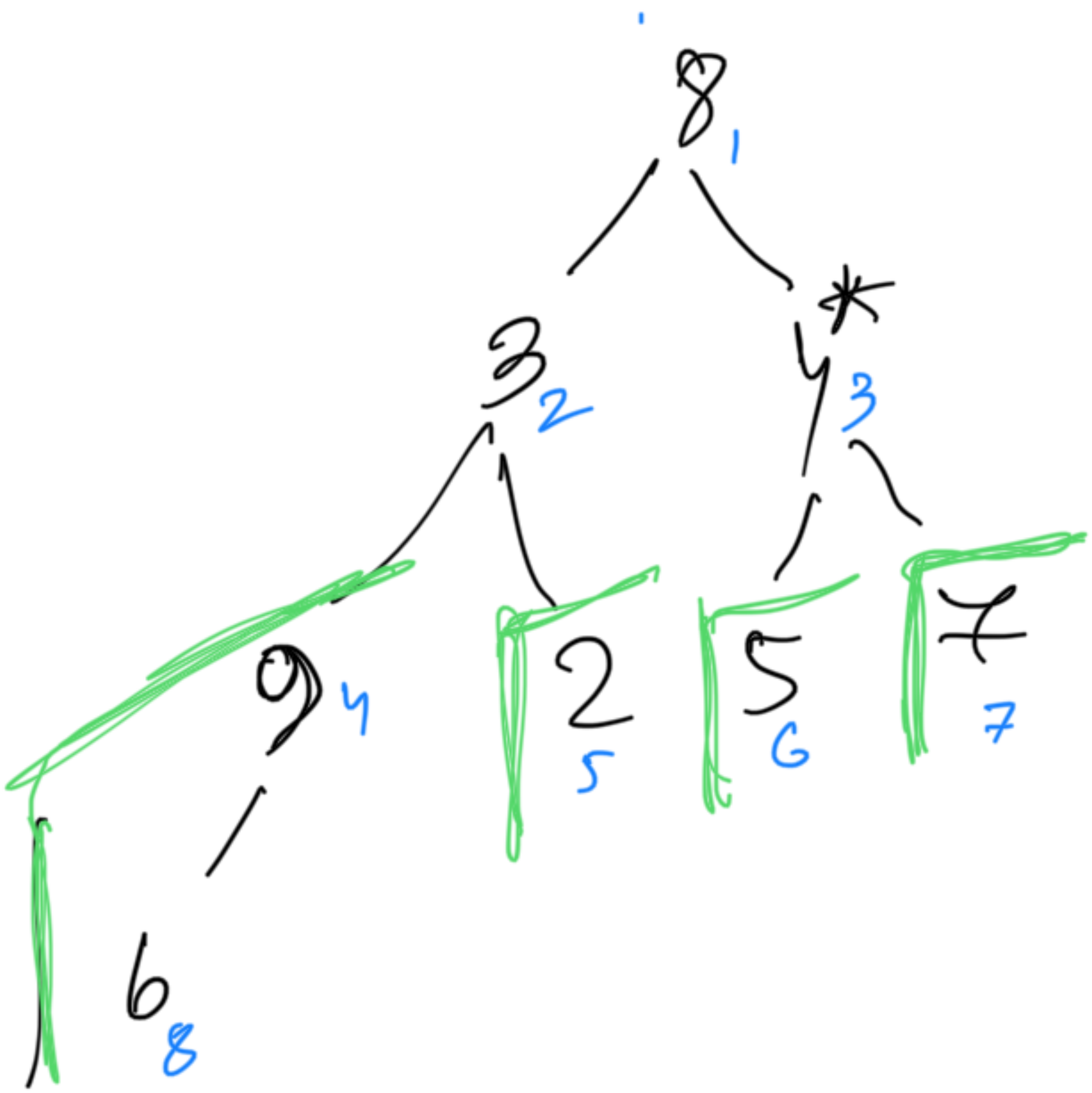
elements[3H :]

heaped

bubble_down(3)



heaped = 2



8 1
3 2
4 3
9 4
2 5
5 6
7 7
6 8