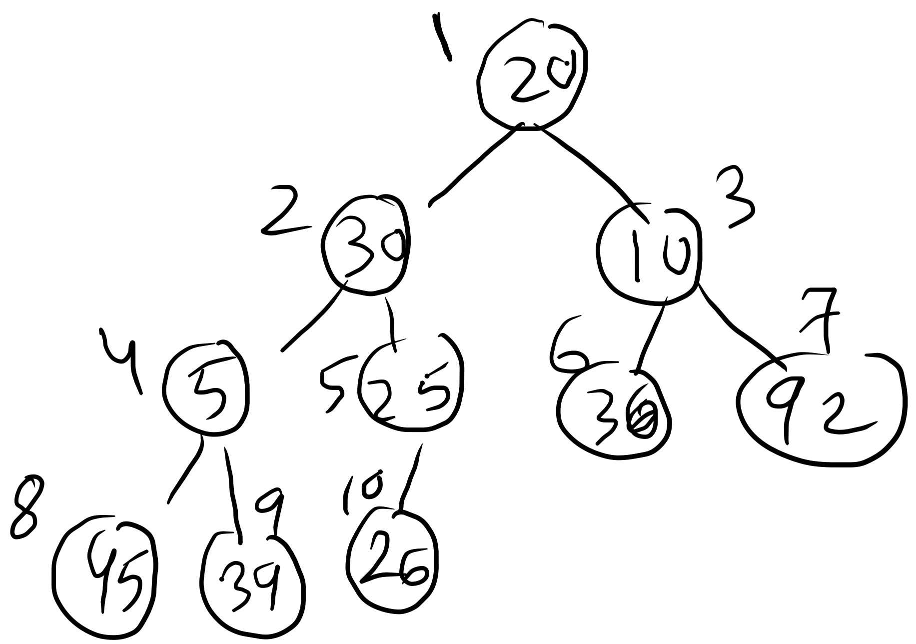
Heap

Heap data structure ← It is an array.

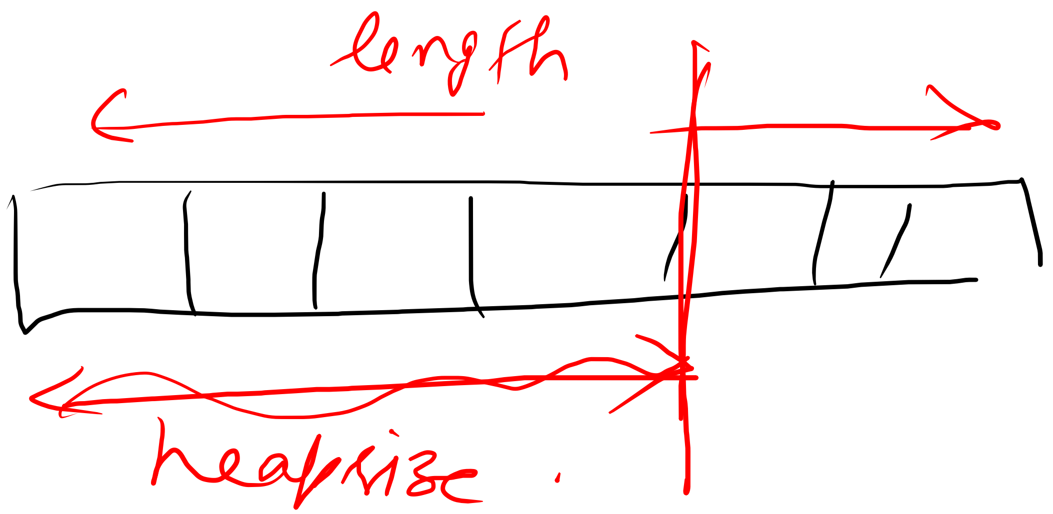
A

20	30	10	5	25	36	92	45	39	26
1	2	3	4	5	6	7	8	9	10

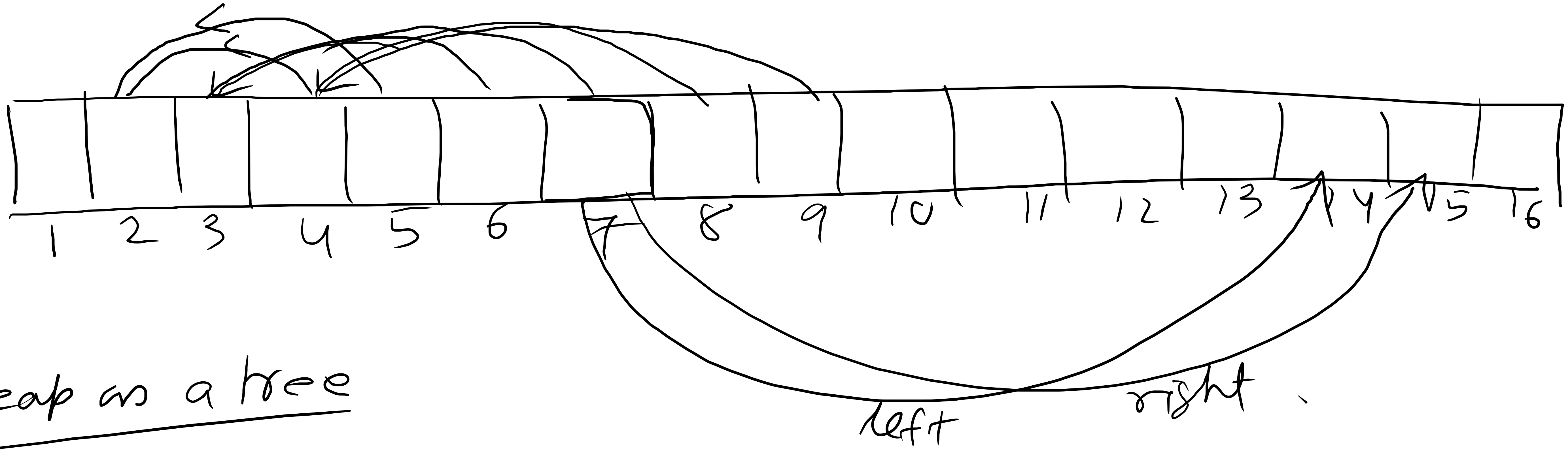


Two attributes of a heap

- length[A] ⟵ length of the array / heap



- heapsize[A] ⟵



Heap as a tree

$\text{Parent}(i) = \lfloor i/2 \rfloor$

$\text{left}(i) = 2i$

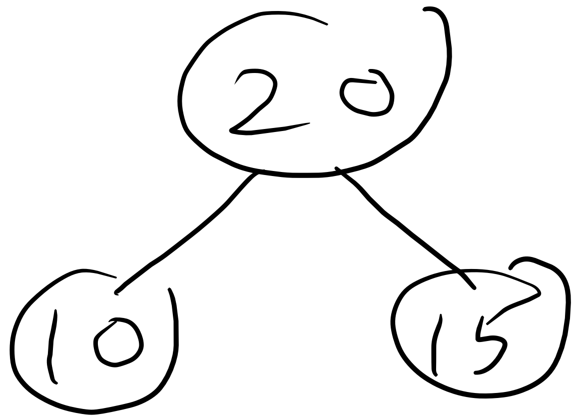
$\text{right}(i) = 2i+1$

```
Parent (i)
    return i/2
left(i)
    return 2i
right (i)
    return 2i+1
```

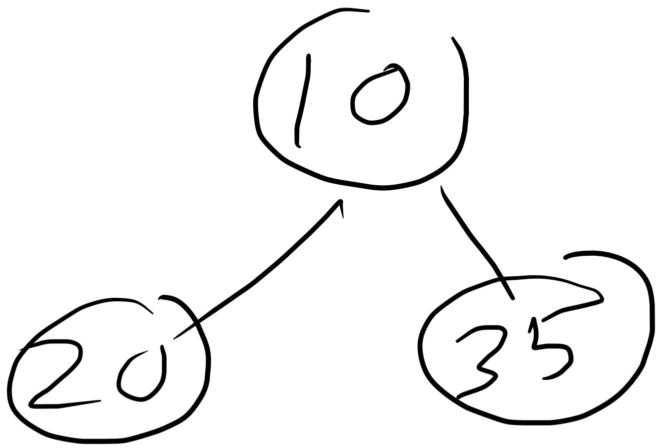
Heap properties

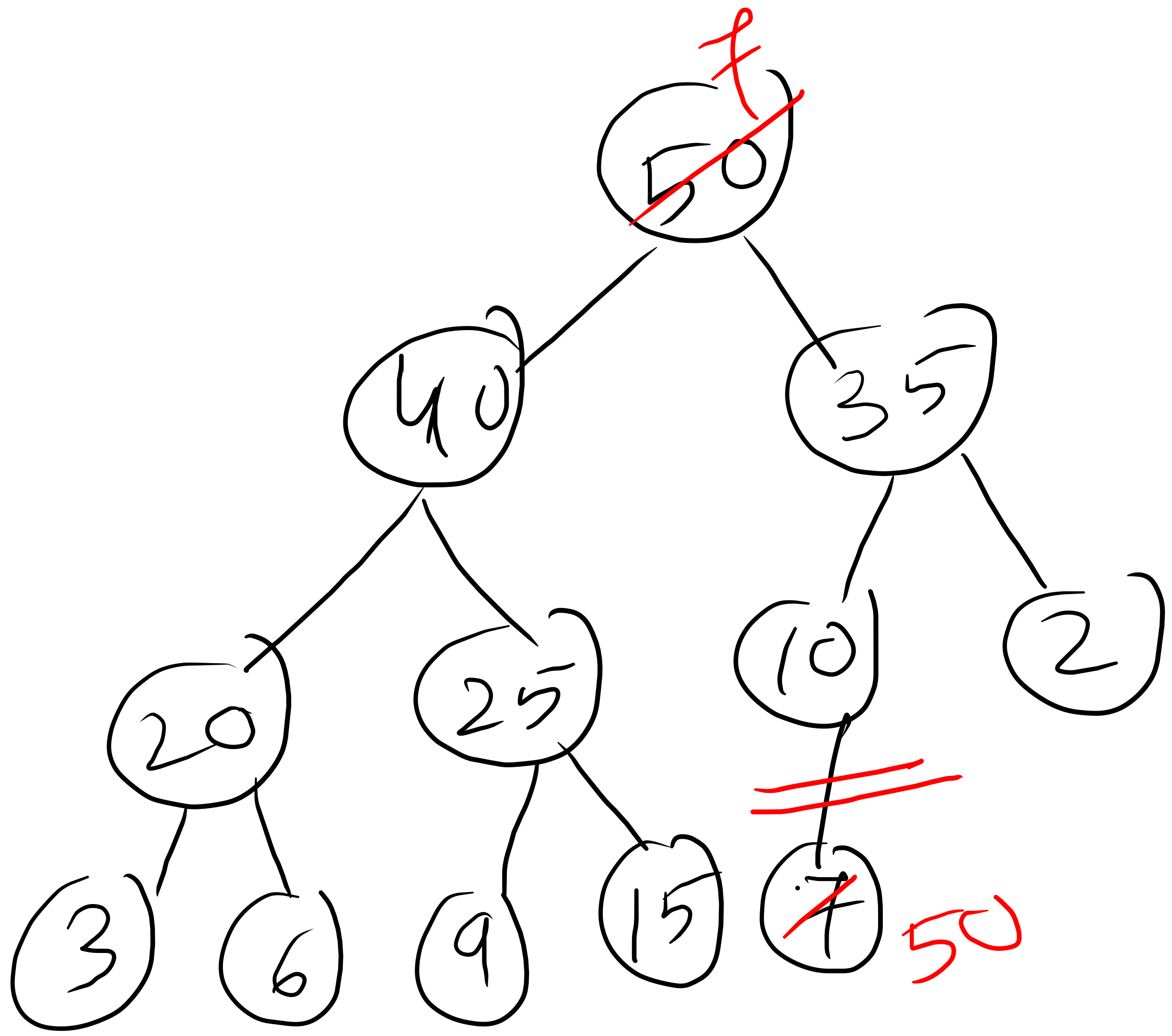
Max-heap property:

key value of the root of any subtree > key value of its children.



min-heap property

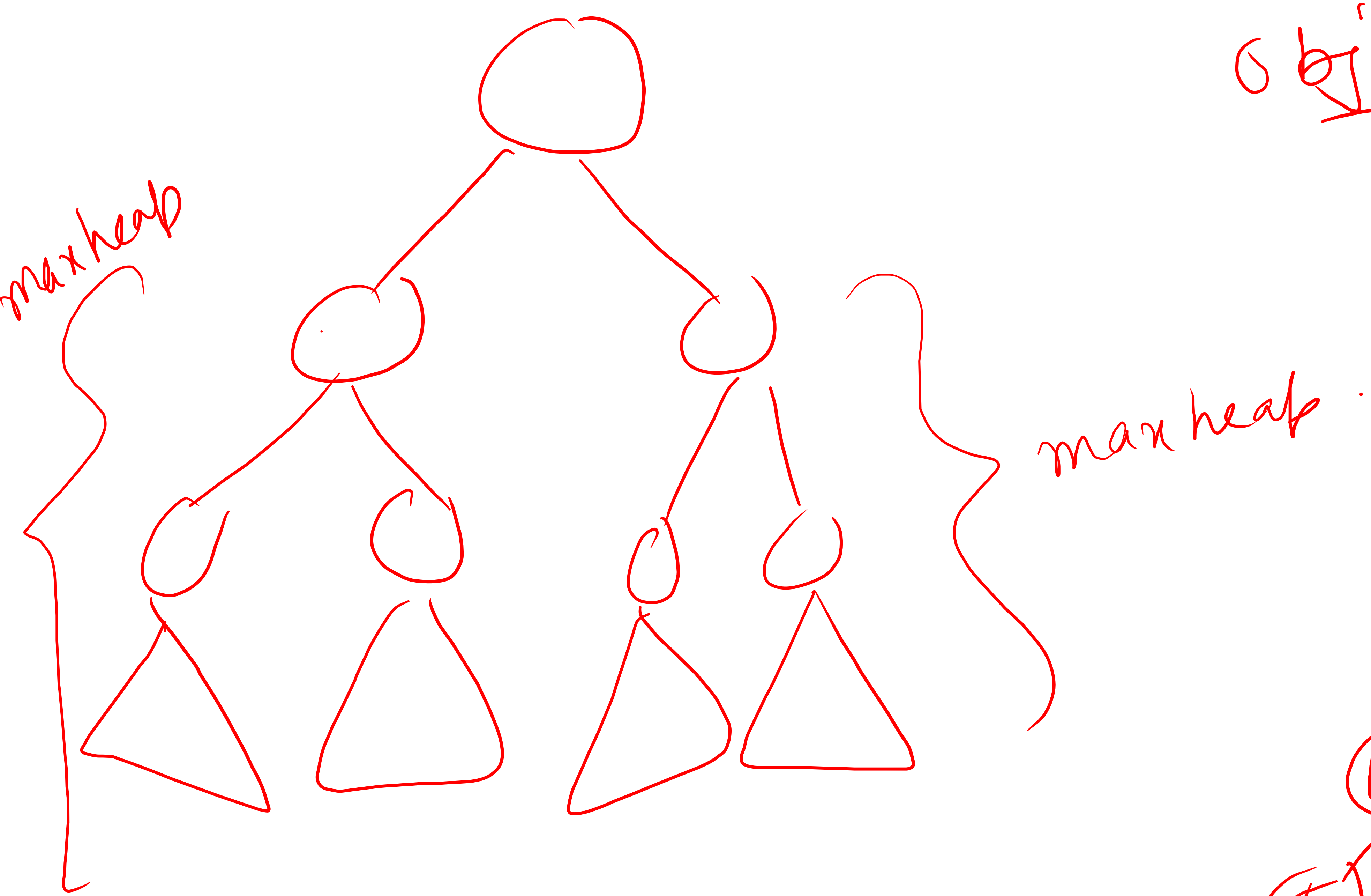


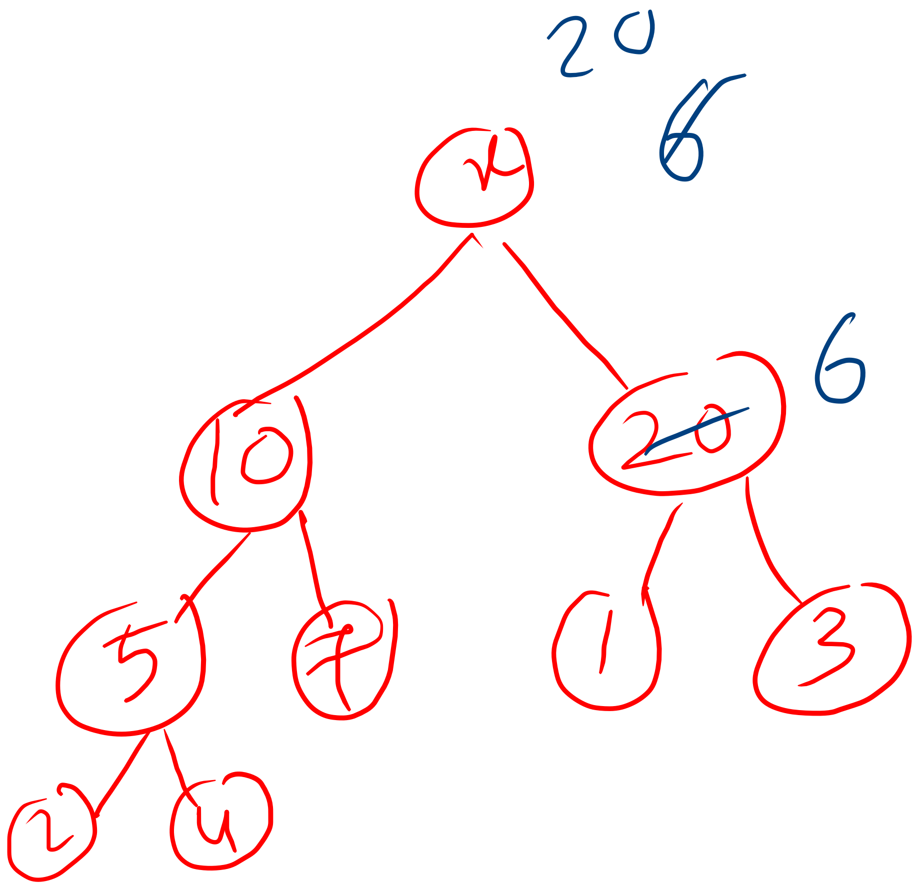


7
50
40
35
20
25
10
2
3
6
9
15
7
50

Objective

make the whole tree max-heap



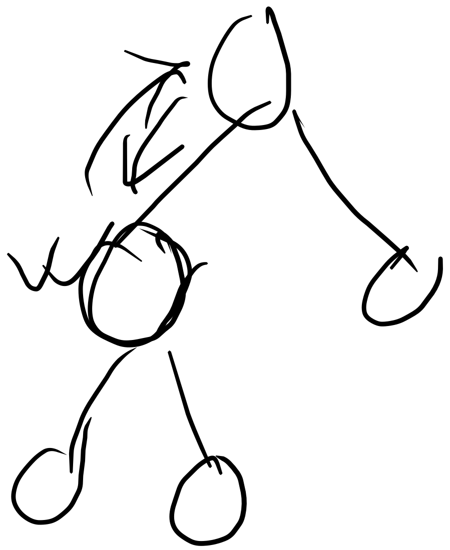


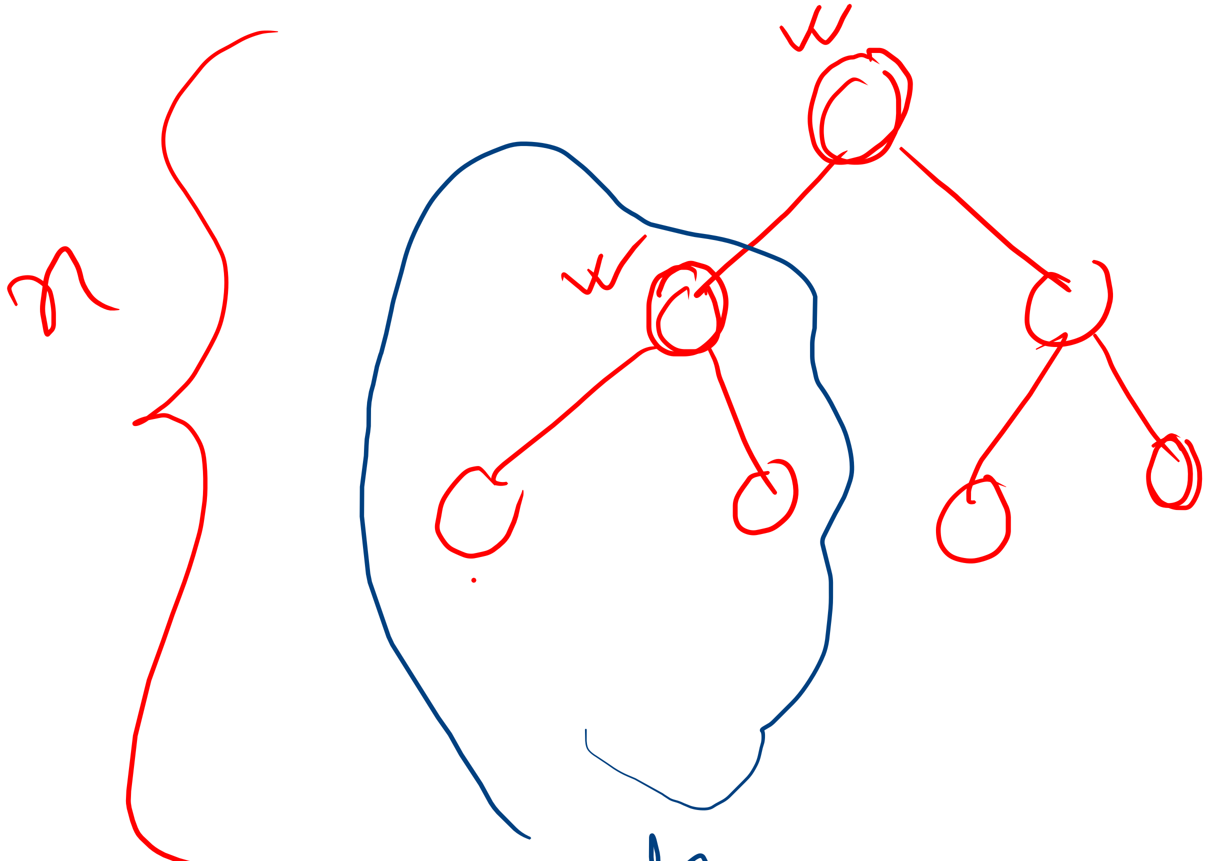
```
max-heapify (A, i)
l = left(i)
r = right(i)
if l ≤ heapsize(A) and A[i] > A[l]
        largest = i
  else
        largest = l
if r ≤ heapsize(A) and A[largest] < A[r]
        largest = r
if i ≠ largest
    swap A[i] and A[largest]
    max-heapify (A, largest)
```

— $T(n)$

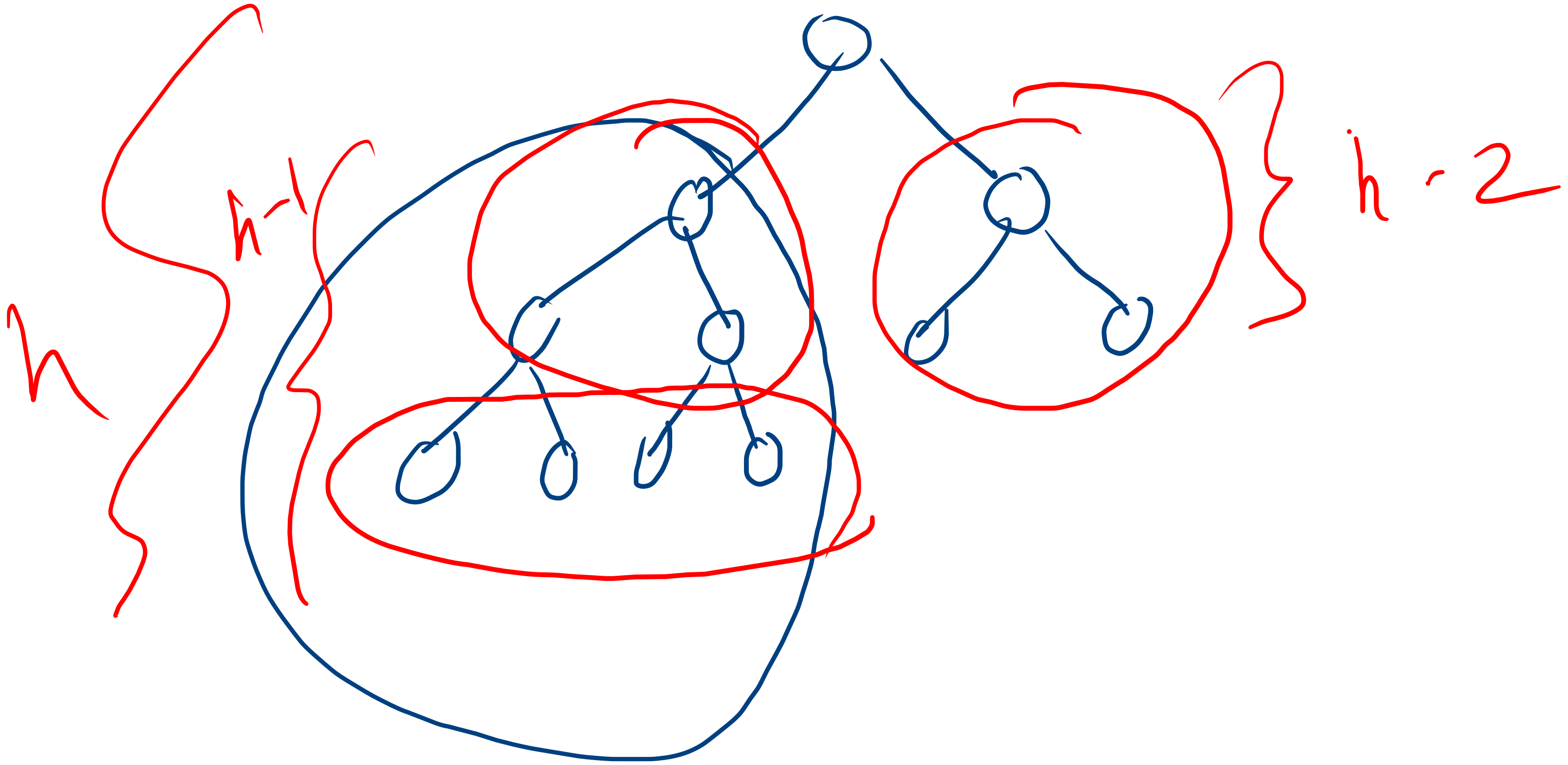
constant

$T(2n/3)$





estimate



$$h - 2^{h+1} - 1$$

$$\frac{2n}{3}$$

Recurrence relation

$$T(n) = T\left(\frac{2n}{3}\right) + \theta(1)$$

← use substitution method

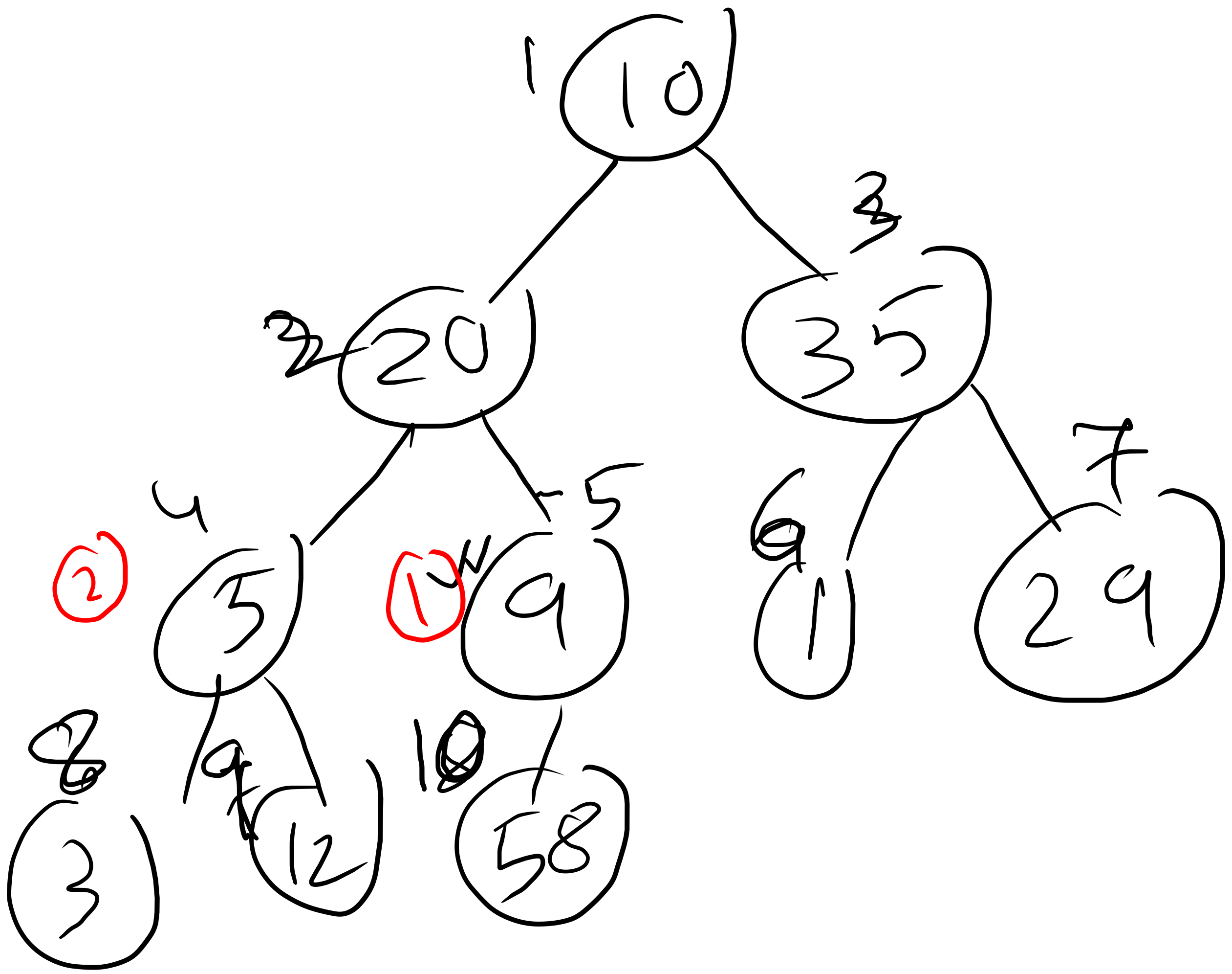
Guess $T(n) = O(n)$

H.W

tight bound

Guess $T(n) = O(\log n)$

10	20	35	5	9	1	29	3	12	58
1	2	3	4	5	6	7	8	9	10



Build max-heap(A)

for $i = \frac{n}{2}$ down to 1

maxheapify (A, i) $O(\lg n)$

Total time $O(n \lg n)$