max heapify:

LST → max heap

RST → max heap

But parent node failed max heap property

Thus max heapify will make it max heap

1 200 ← heap property failed.

2 60 200

3 100

50 60 70 80 40

max heap

max heap.

Heap.

LC(1) = 2×1

RC(1) = 2×1+1

max heapify (A, i)

{

$l = 2i$;

$r = 2i+1$;

if ($l \le A.heapsize$ and $A[l] > A[i]$)

$largest = l$;

else $largest = i$;

if ($r \le A.heapsize$ and $A[r] > A[largest]$)

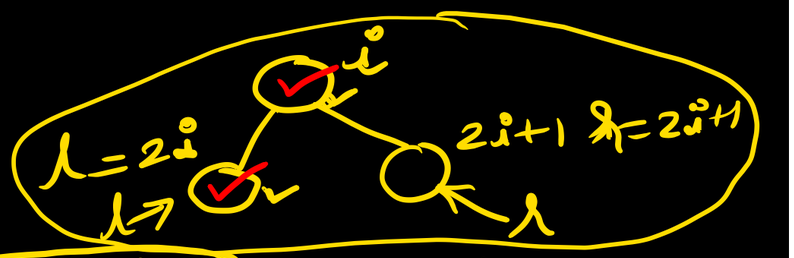
$largest = r$

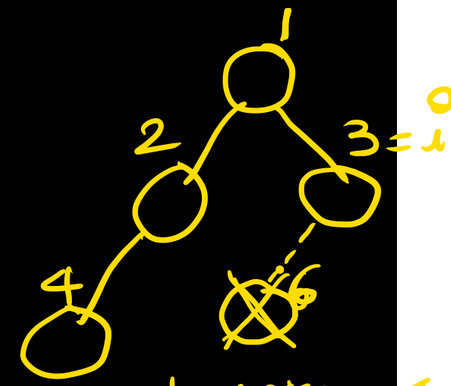
if ($largest \neq i$)

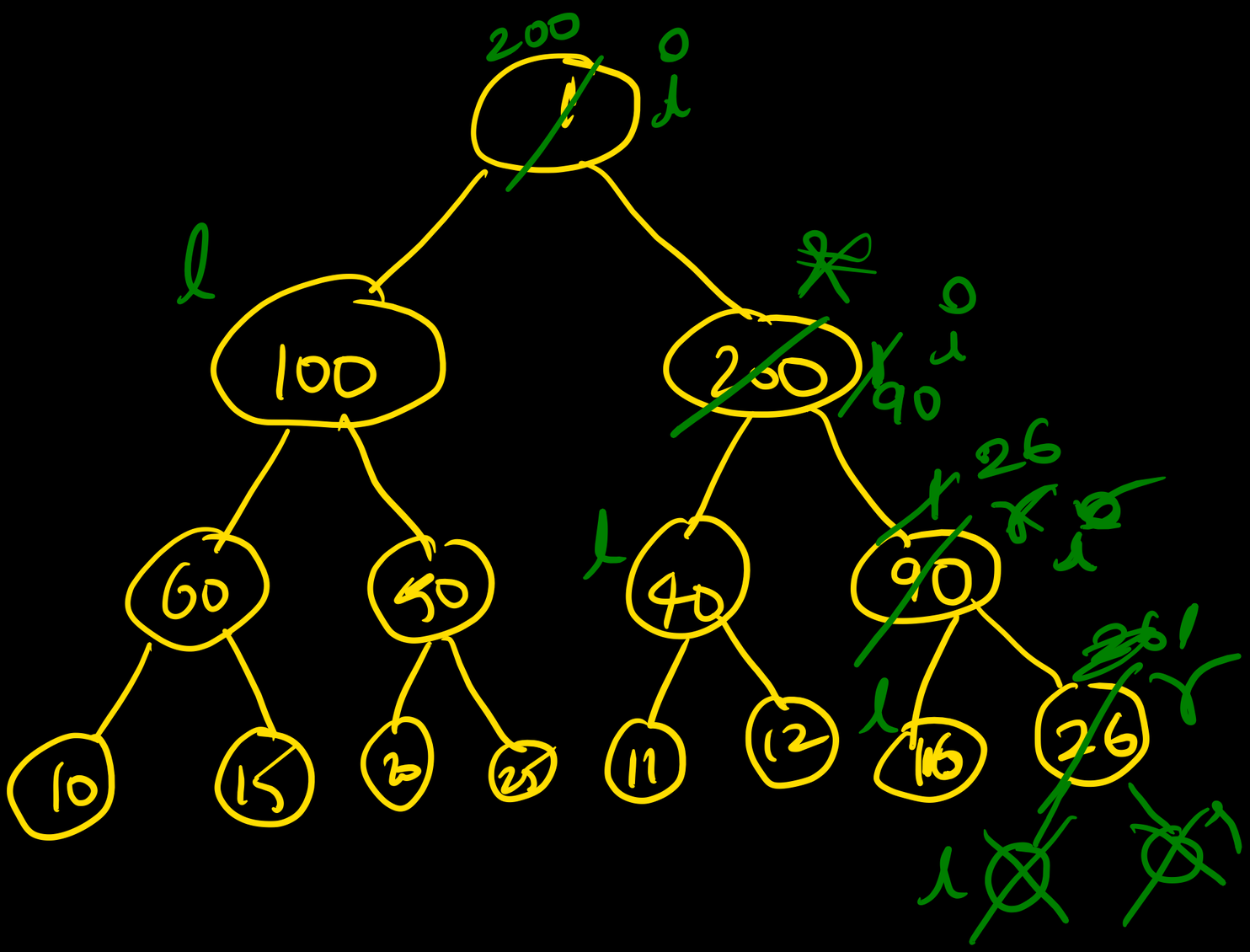
exchange $A[i]$ with $A[largest]$

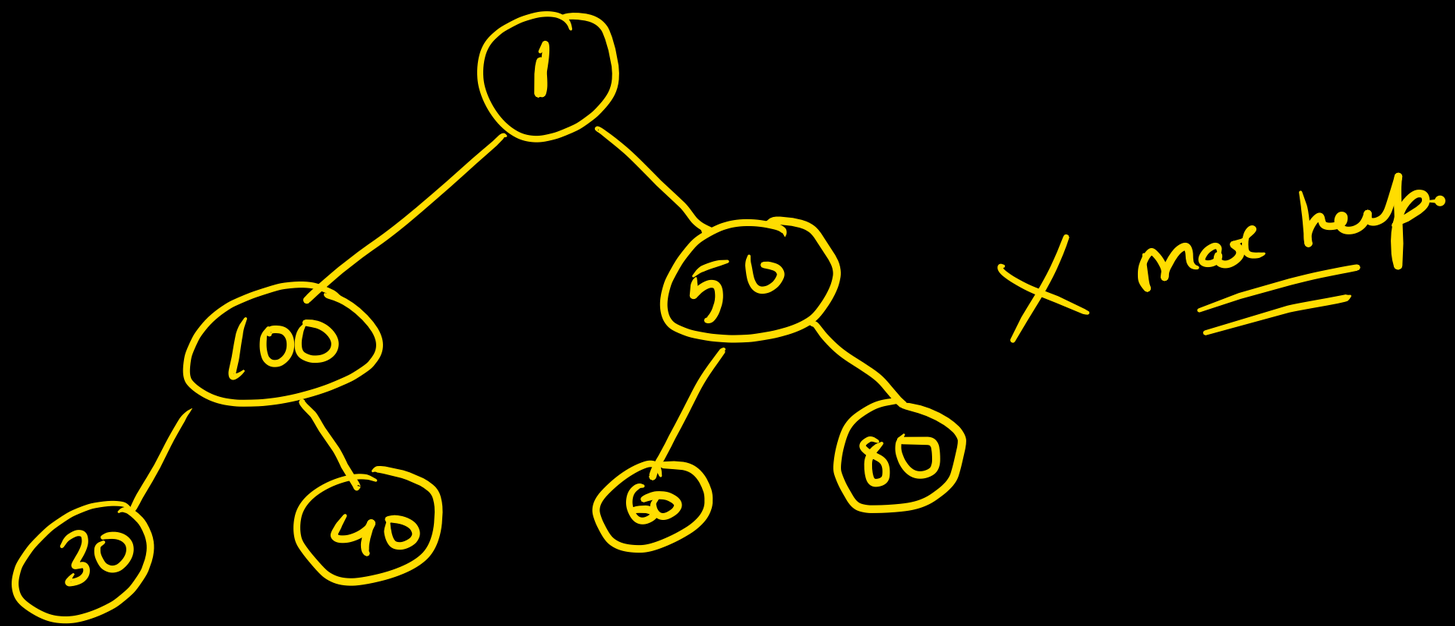
max heapify (A, largest)

}

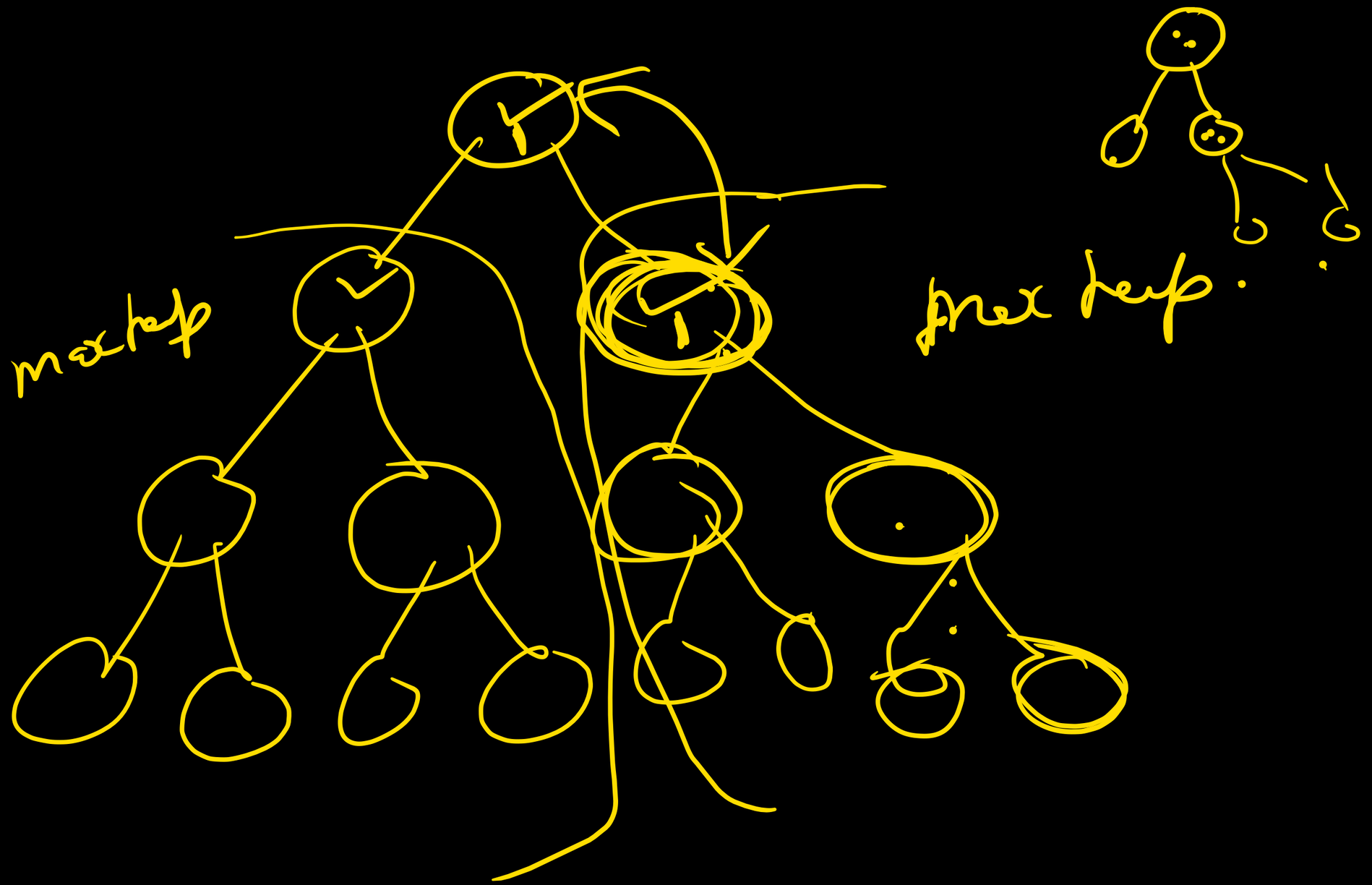


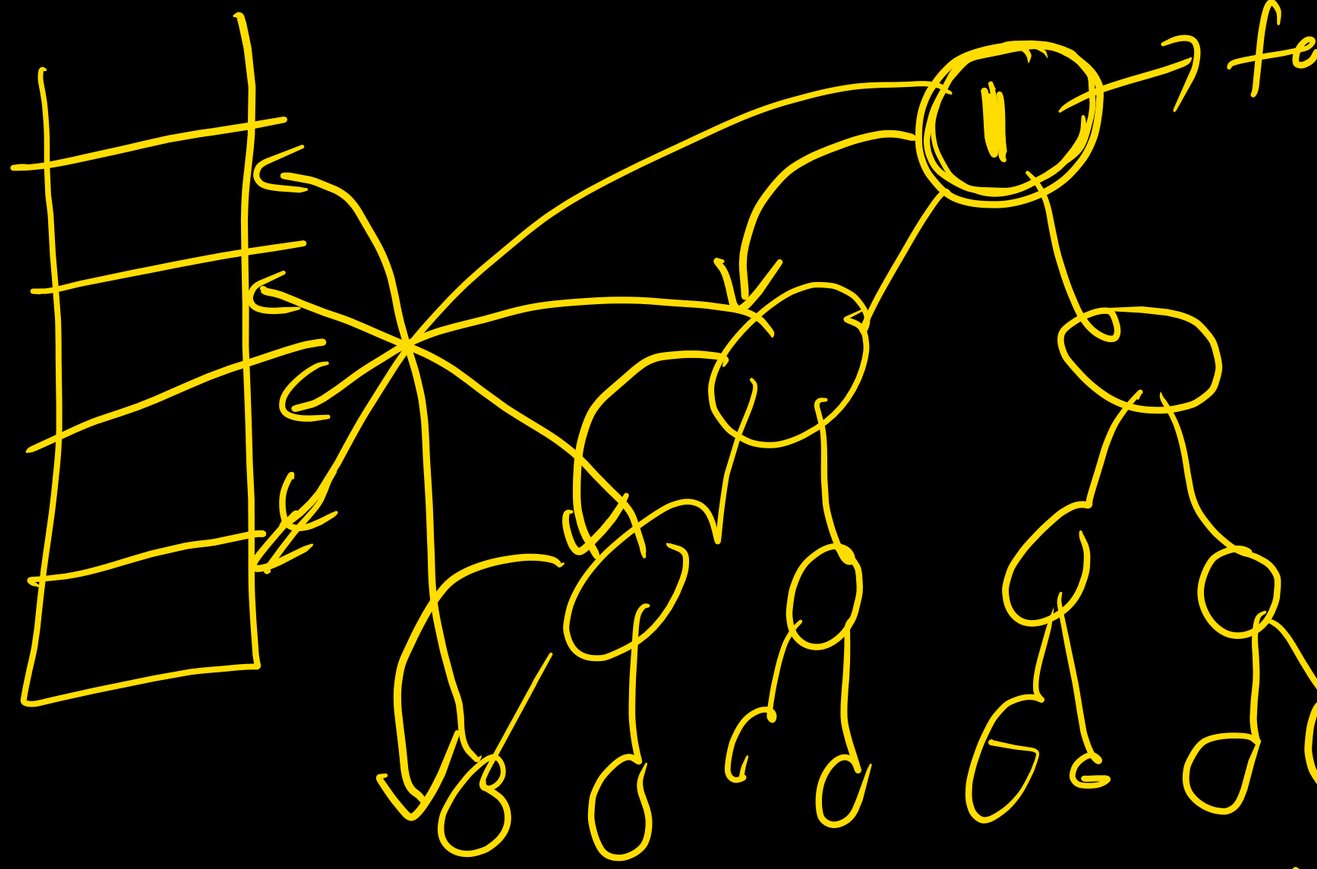



200
100
60
50
10
15
40
90
26



✗ max heap


mother
father



fail max heap property.

logn.

TC = O(logn)

SC = O(logn)

maxHeapify(A,1)

A(2)
A(4)
A(8)

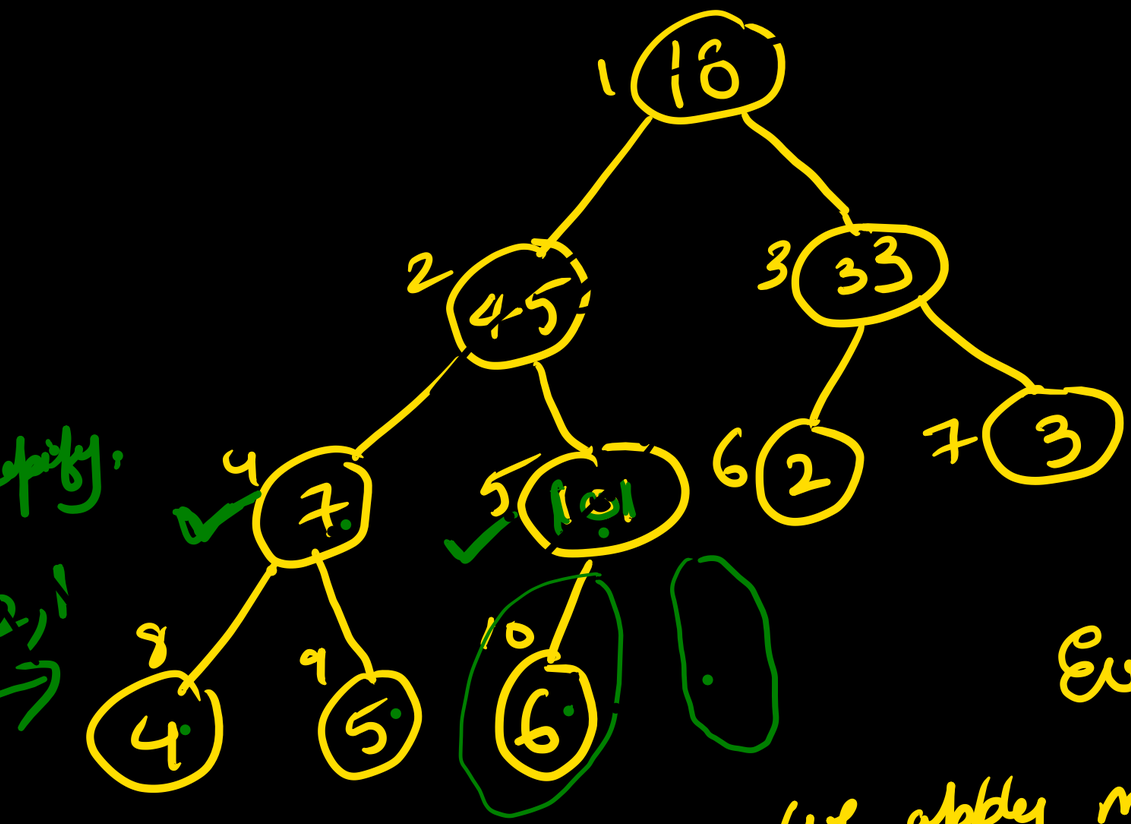
max ~~heapify~~ heapify.

(Q) How to build max heap from scratch.

array

18	45	33	7	101	2	3	4	5	6
1	2	3	4	5	6	7	8	9	10

Almost complete binary tree



Apply max heapify.

5, 4, 3, 2, 1

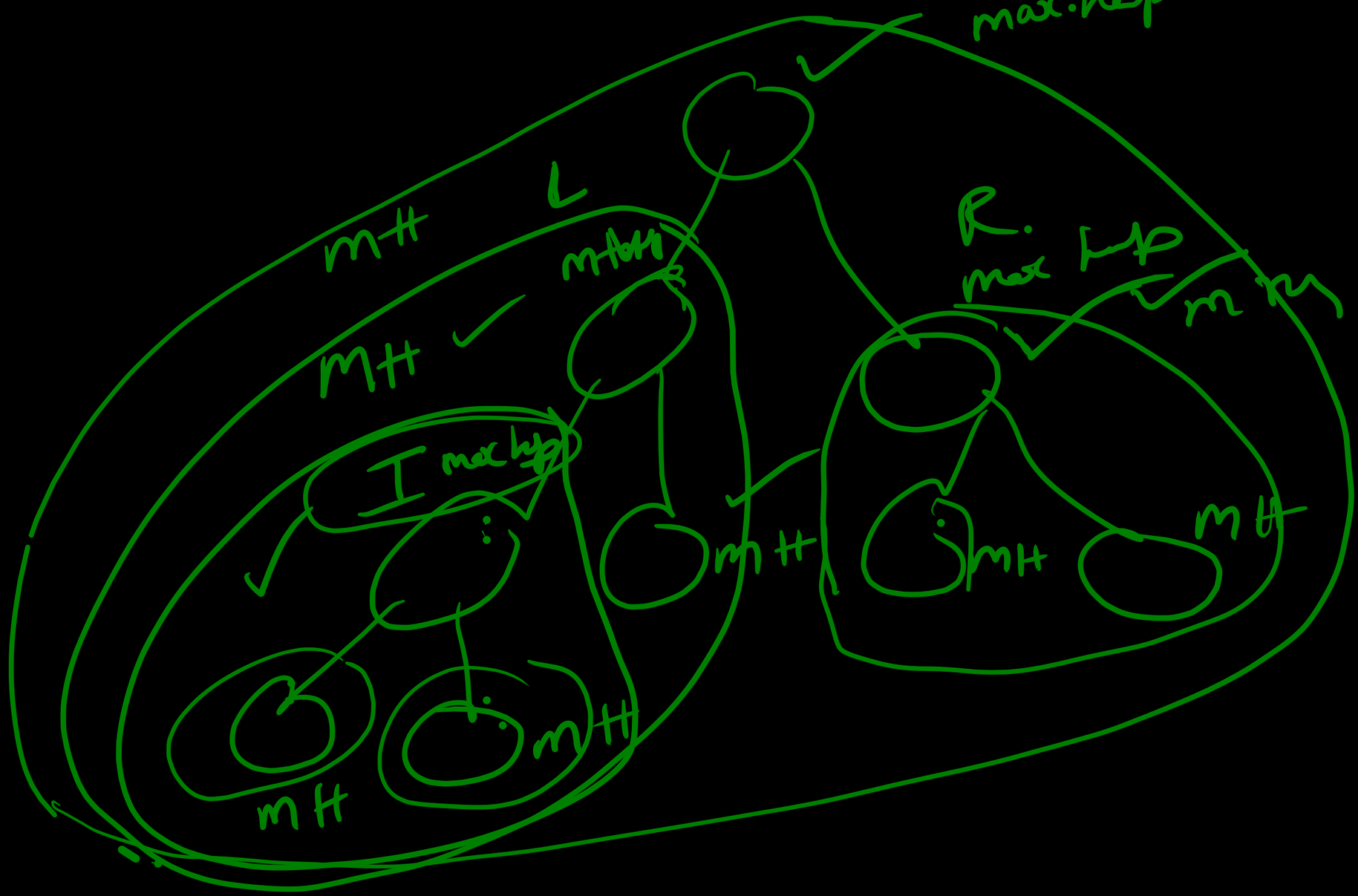
leaves = $\left\lfloor \frac{A.length}{2} \right\rfloor + 1$ to n

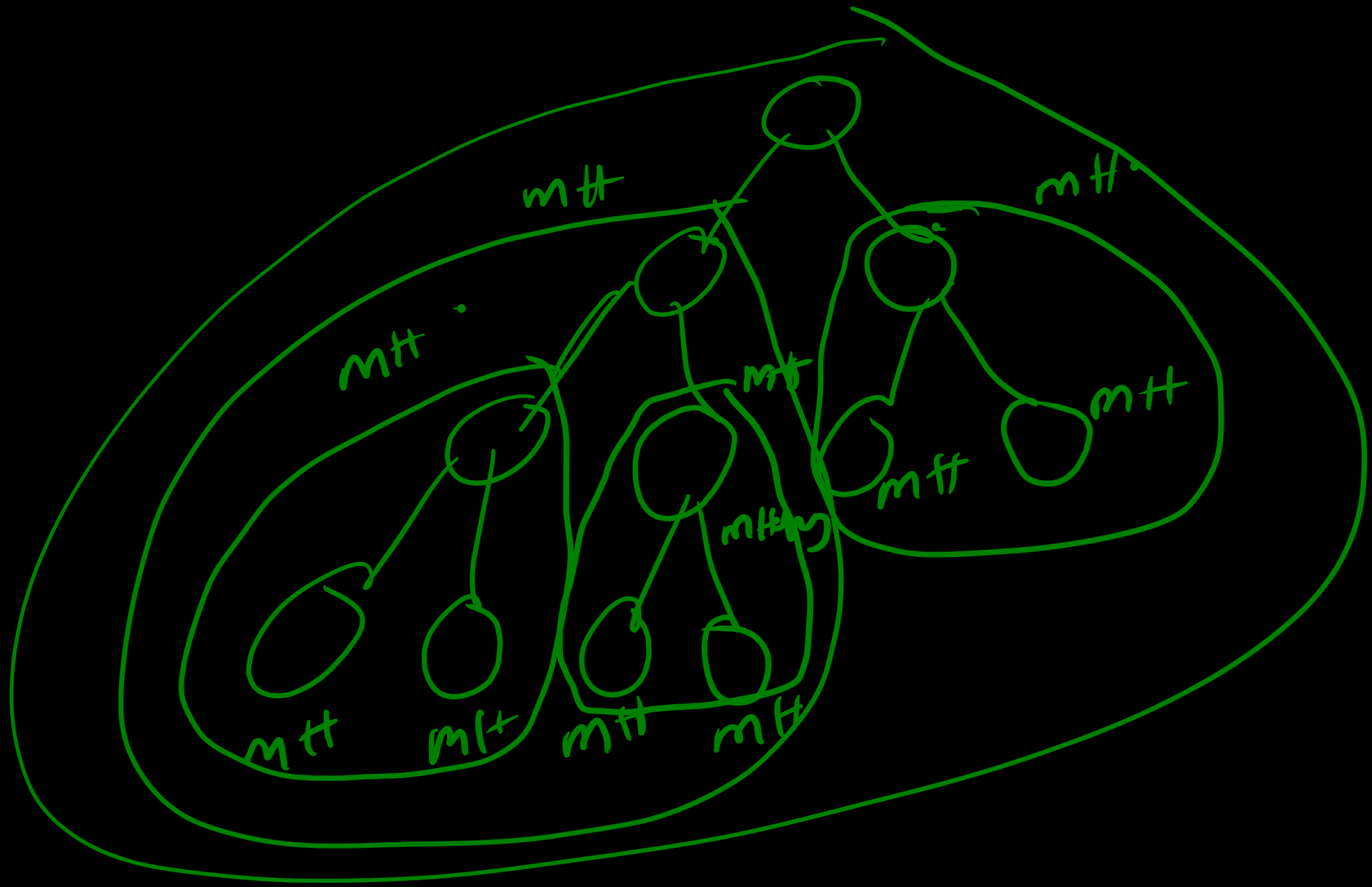
= $\left\lfloor \frac{10}{2} \right\rfloor + 1$ to 10

= 6 to 10

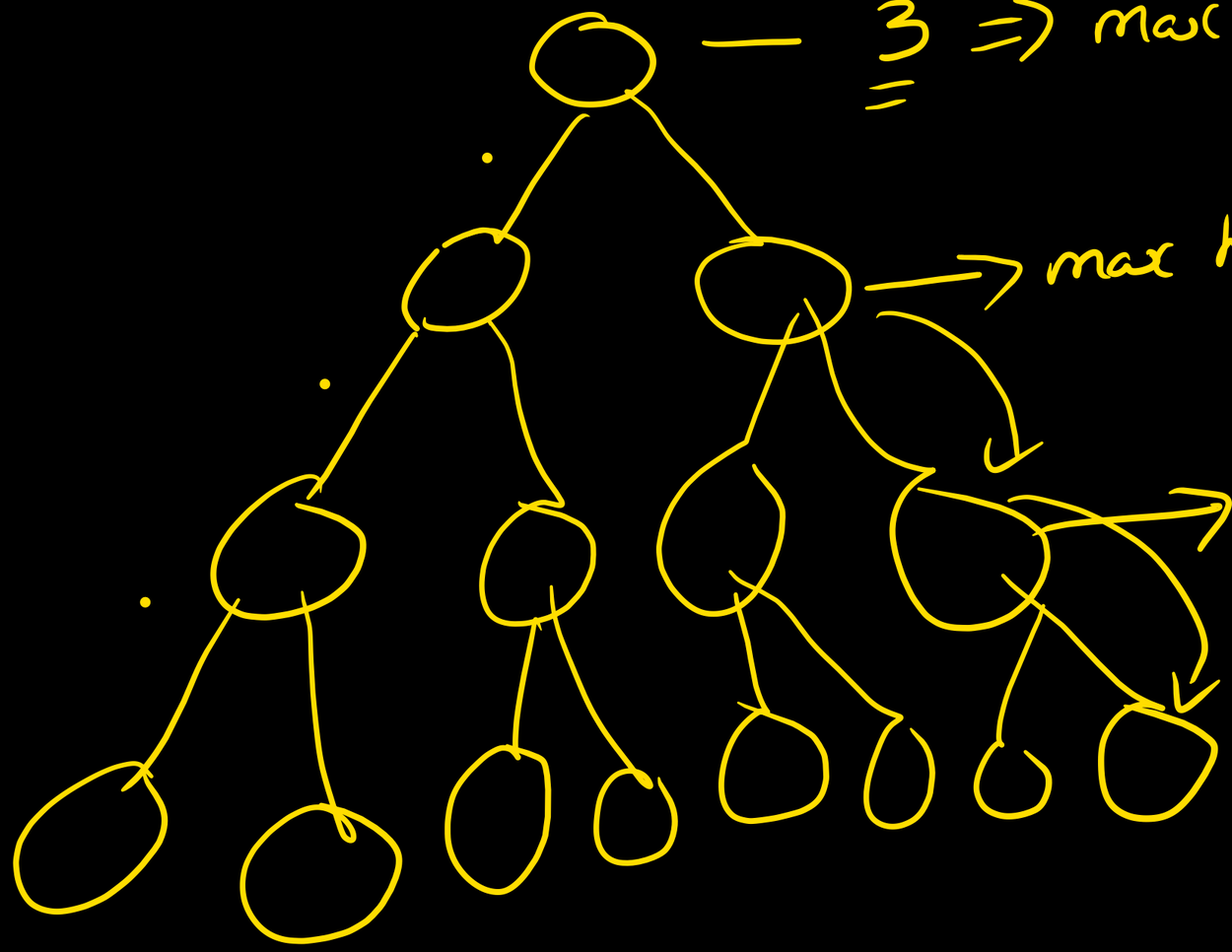
Every leaf is a max heap.

we apply max heapify on non non leaves from end


max.heap.
L
R.
max heap
max heap
max heap
max heap
max heap
max heap
max heap
max heap

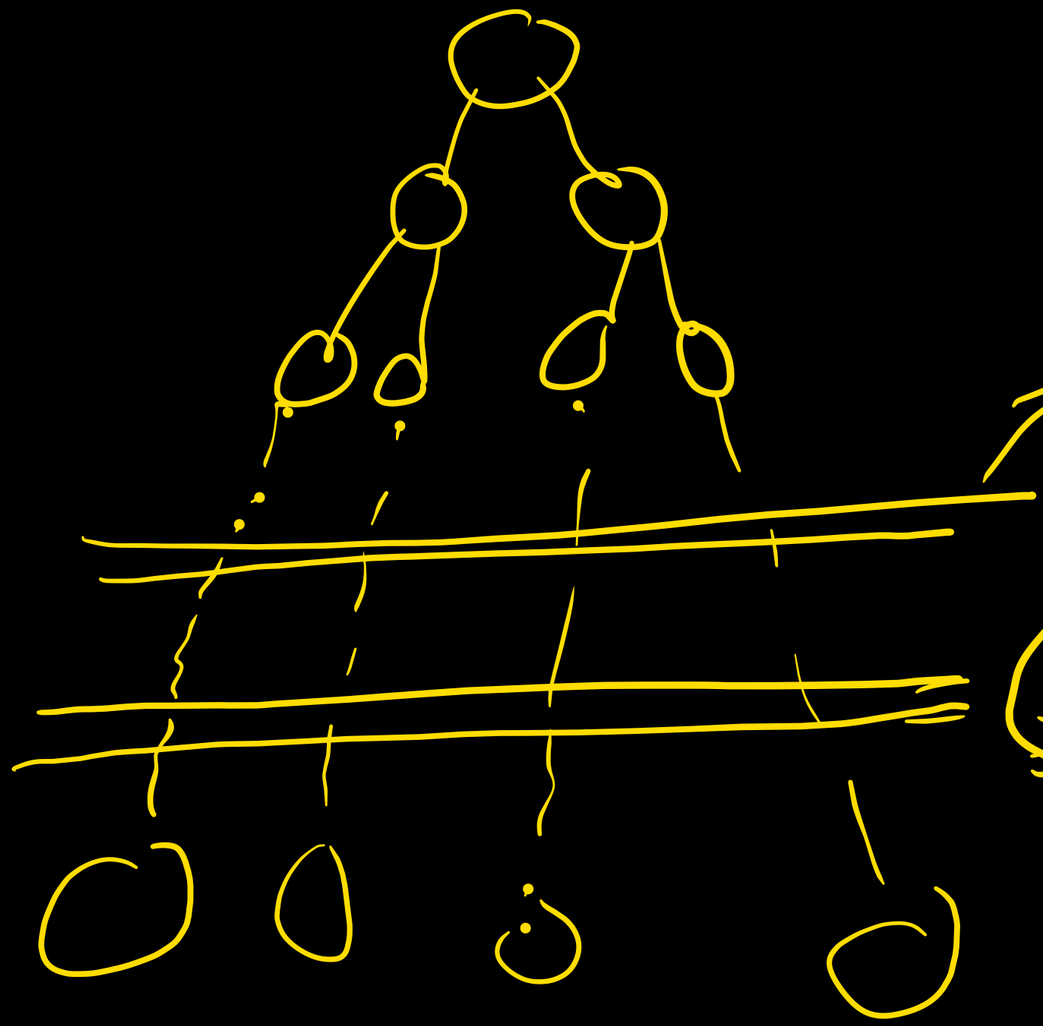


You are
wrong..


3 ⇒ max heapify = O(h) = O(3)
max heapify O(h) = O(2)
max heapify — O(n) = O(1)

Build max heapify

$= O(h) =$



height $= h$

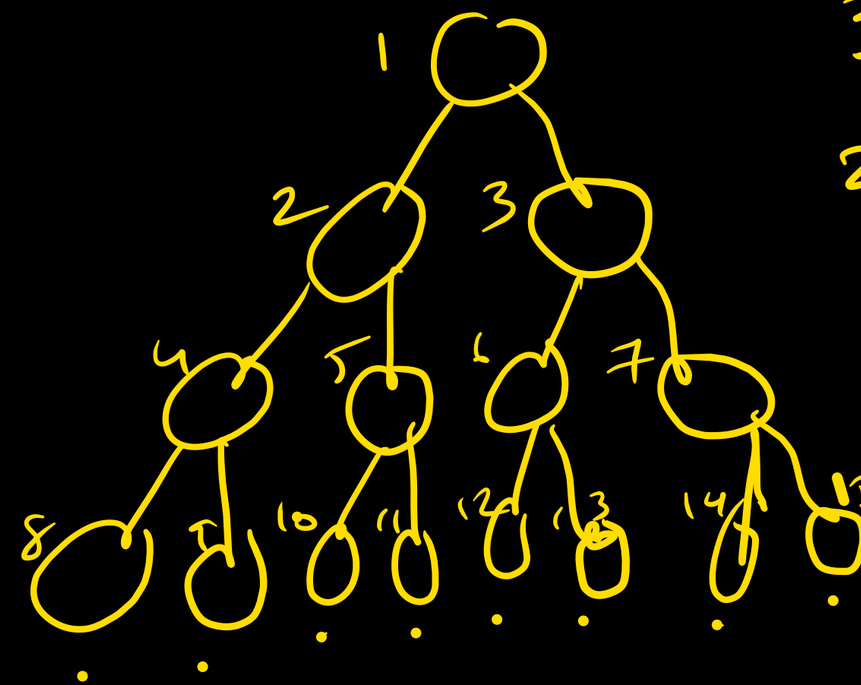
$K * O(h)$ ✓

$1 * O(h-1)$ ✓

$n \times \log n$

↓

max heapify is always $\log n$



$3-1$

$2-2$

$1-4$

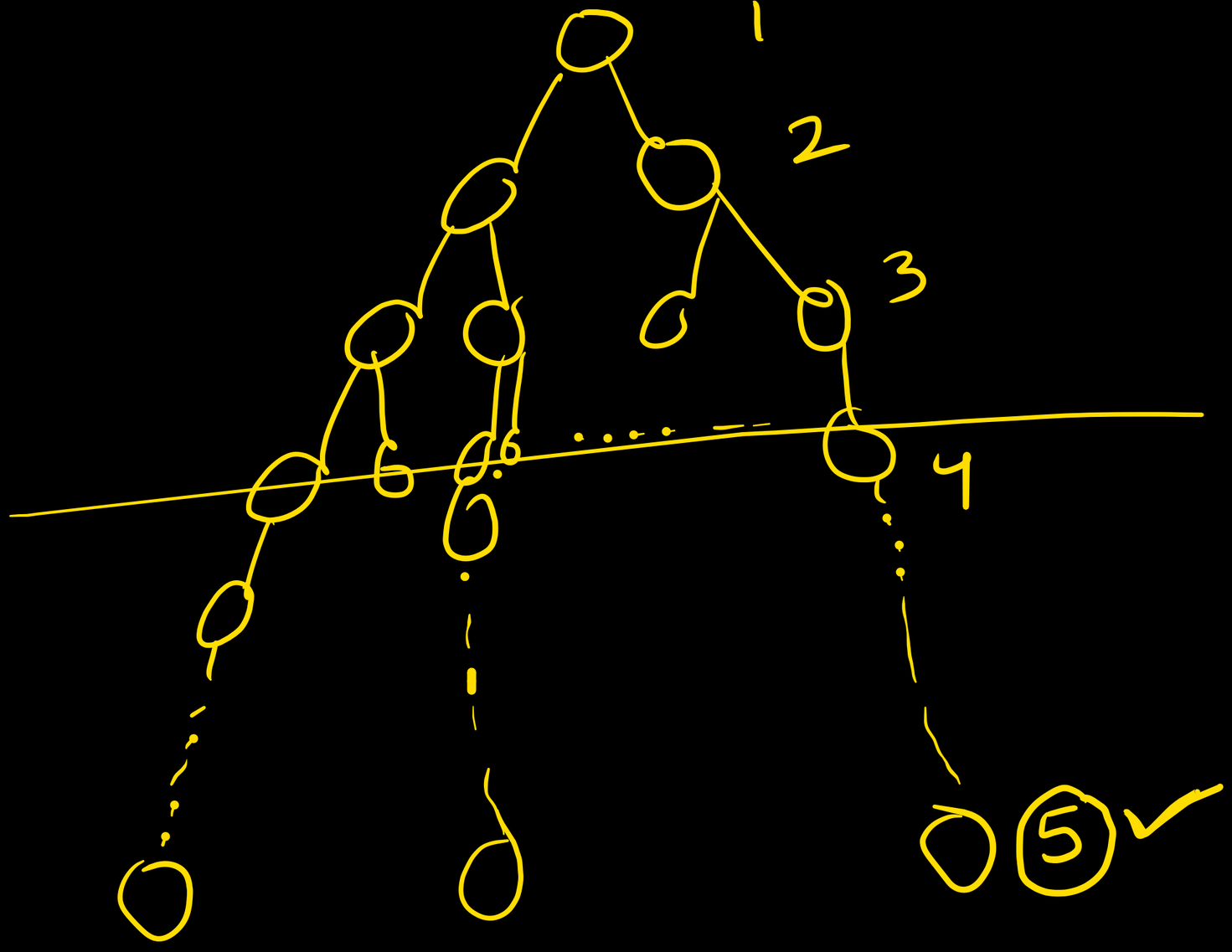
$0 \leq 8$

number of nodes which are present at level h in a complete BT

$$= \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

$$\left\lceil \frac{15}{2^{0+1}} \right\rceil = 8$$

$$\left\lceil \frac{15}{2^{1+1}} \right\rceil = 4$$

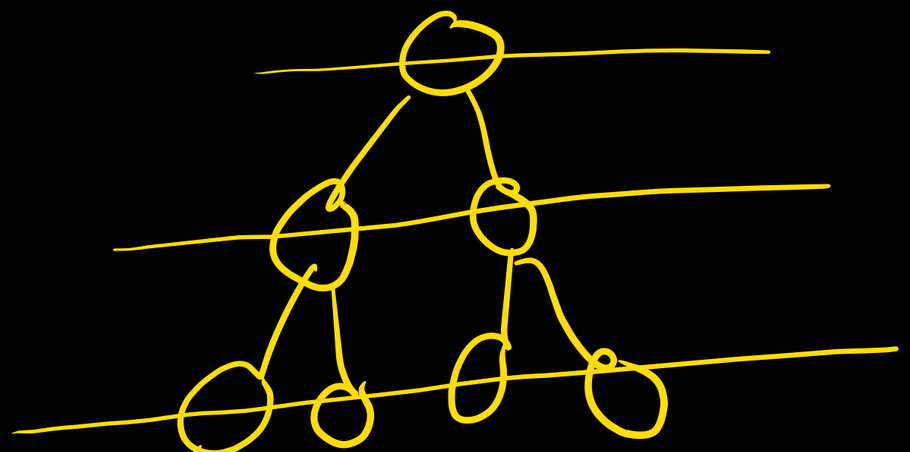


height $h := \left(\frac{n}{2^{h+1}}\right) O(h)$

total number of levels

$\log n$

height → 0 to $\log n$



$\log n$.

6

5

no of levels

nodes per level

max heapify.

nodes

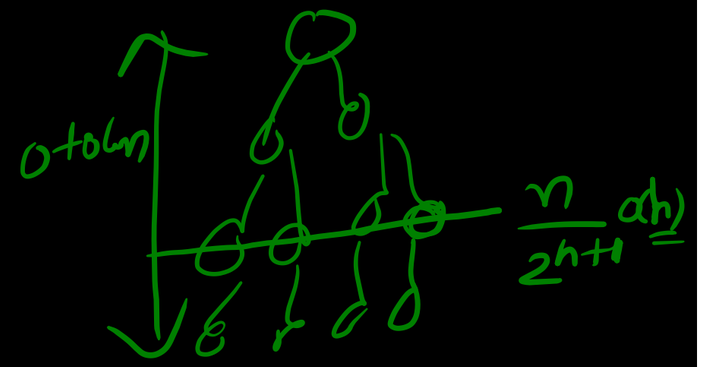
$\log n$

$\sum_{h=0}$

2^{h+1}

$O(h)$

Total work in CT $= \sum_{h=0}^{\log n} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$ ✓



leafs are included

$= \sum_{h=0}^{\log n} \left\lceil \frac{n}{2^{h+1}} \right\rceil (ch)$

$= \frac{cn}{2} \sum_{h=0}^{\log n} \left(\frac{h}{2^h}\right)$

$\leq \frac{cn}{2} \sum_{h=0}^{\infty} \frac{h}{2^h} = O(n)$ ✓

$\sum_{h=0}^{\infty} \frac{h}{2^h} \to 2$ Proof in CLRS Appendix A.8

max heapify $O(\log n)$

$\downarrow$

h ✓

$h = O(\log n)$.

Time taken to ~~but~~ build a max heap = $O(n)$

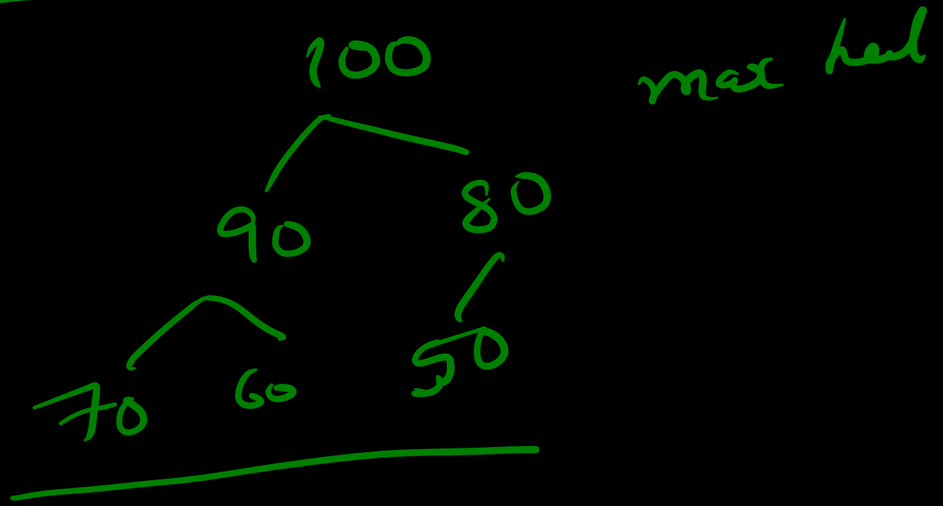
If we build max heap one by one element $\Rightarrow$
$= O(n \log n)$

If we build max heap with all elements
at a time $= O(n)$

max heap need be sorted array.

But Every sorted array is a min/max heap.

100 90 80 70 60 50



max heap

TC → Build max heap = O(n)

SC → Build max heap =

↓

Calls max heapify, one after other

↓

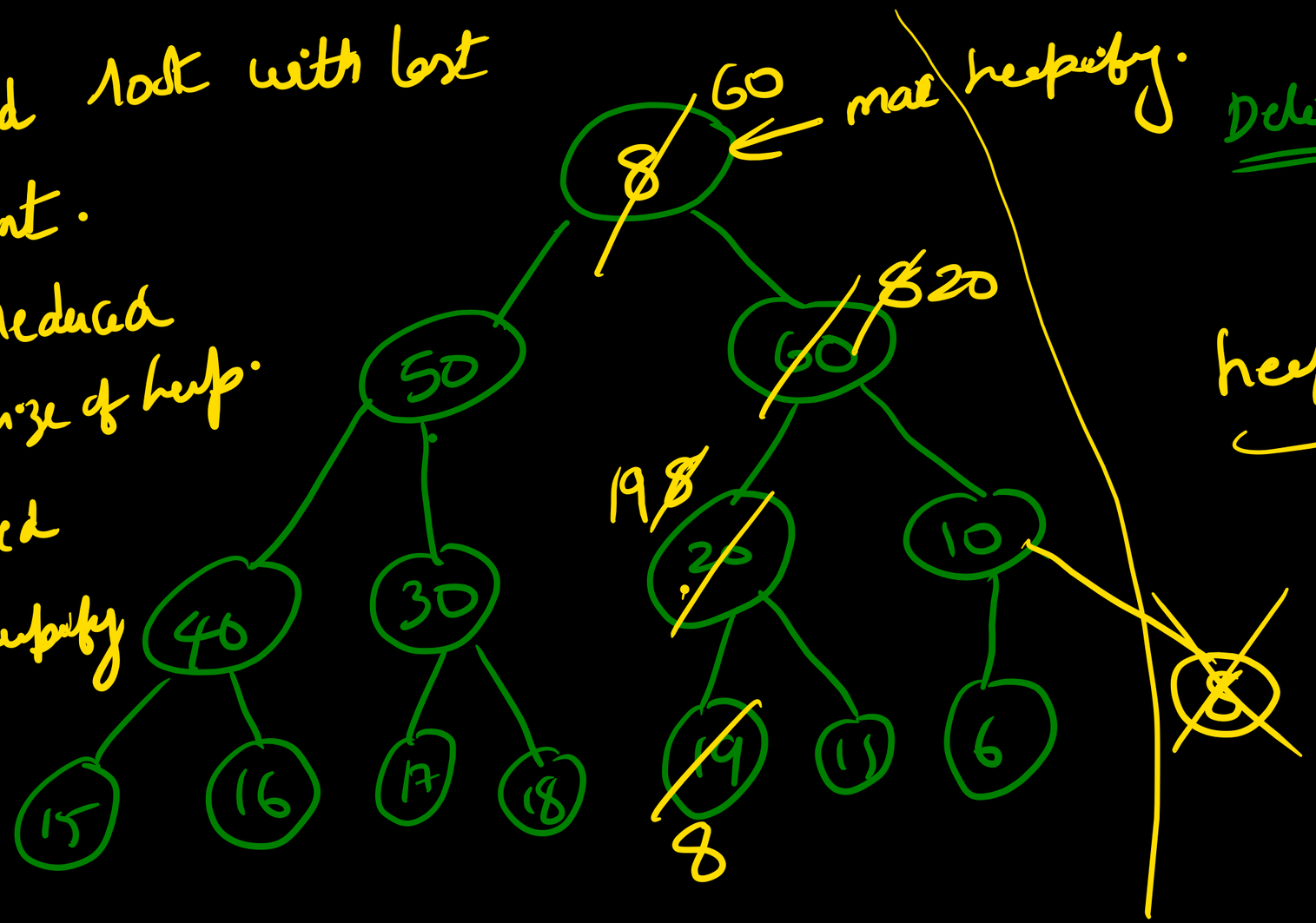
O(log n) SC.

∴ at any time only one stack is required with max size O(log n)

→ Replaced root with last element.

→ & reduced size of heap.

→ applied max heapify



max heapify.

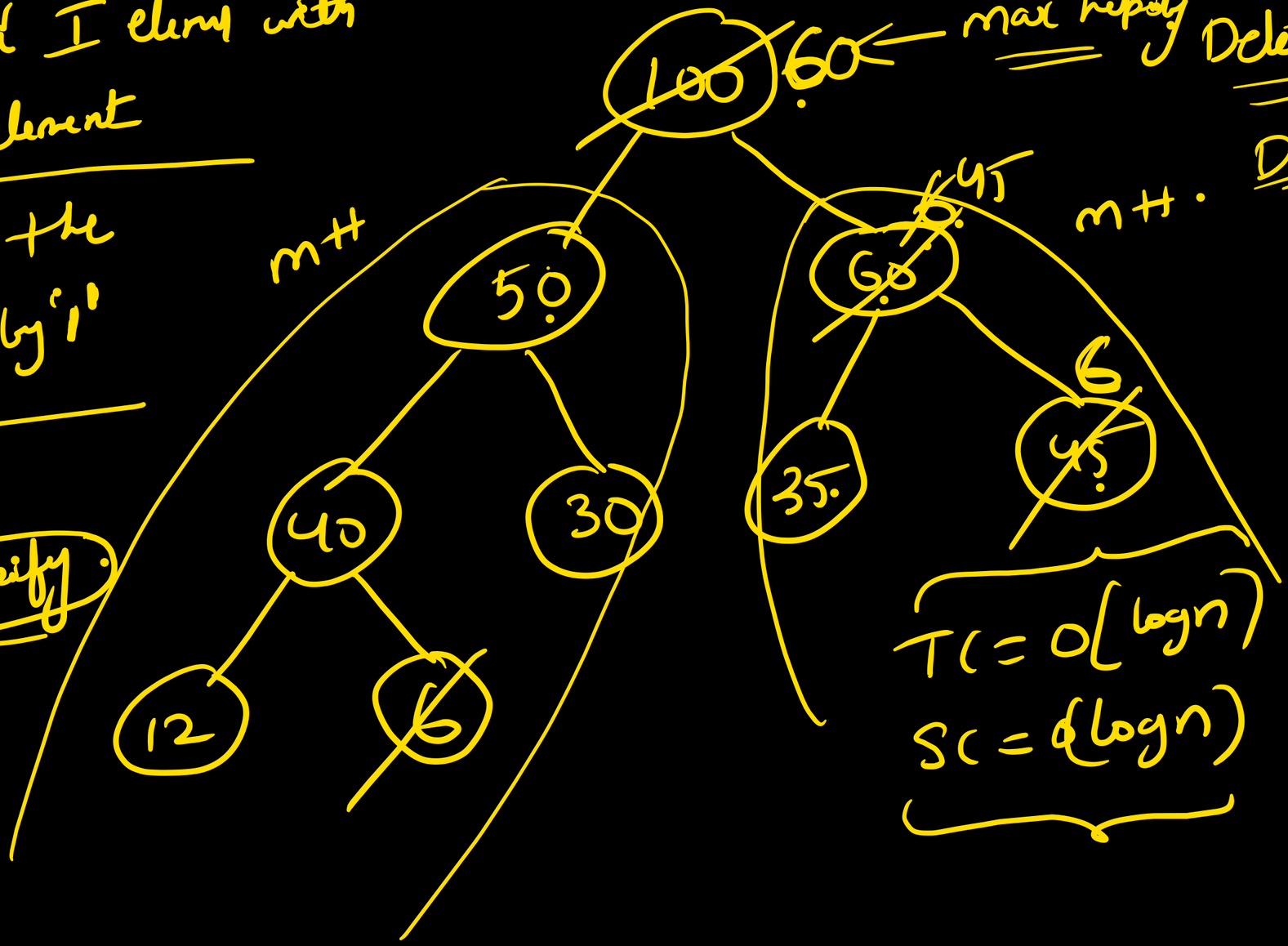
Delete max.

heap size − −

Replaced I elem with last element

reduce the heap size by '1'

apply max heapify



max heapify

Delete max

Delete min

min heap.

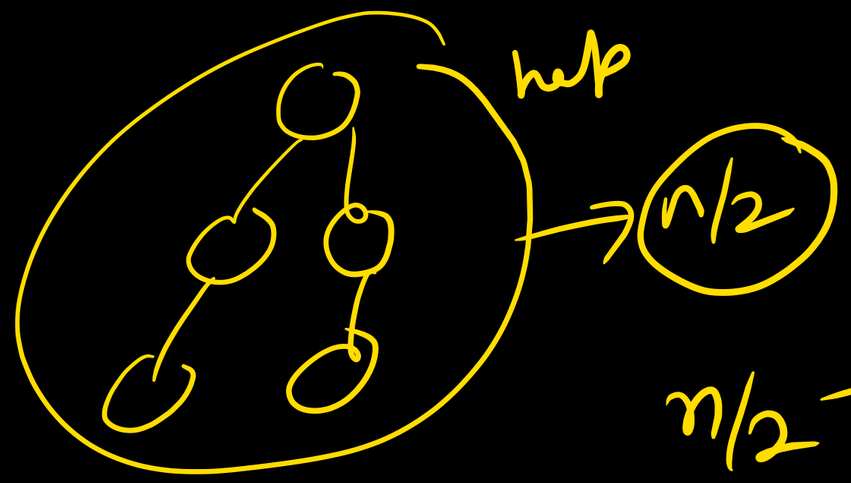
Build min heap

min heapify

Delete min.

T.C = O(logn)

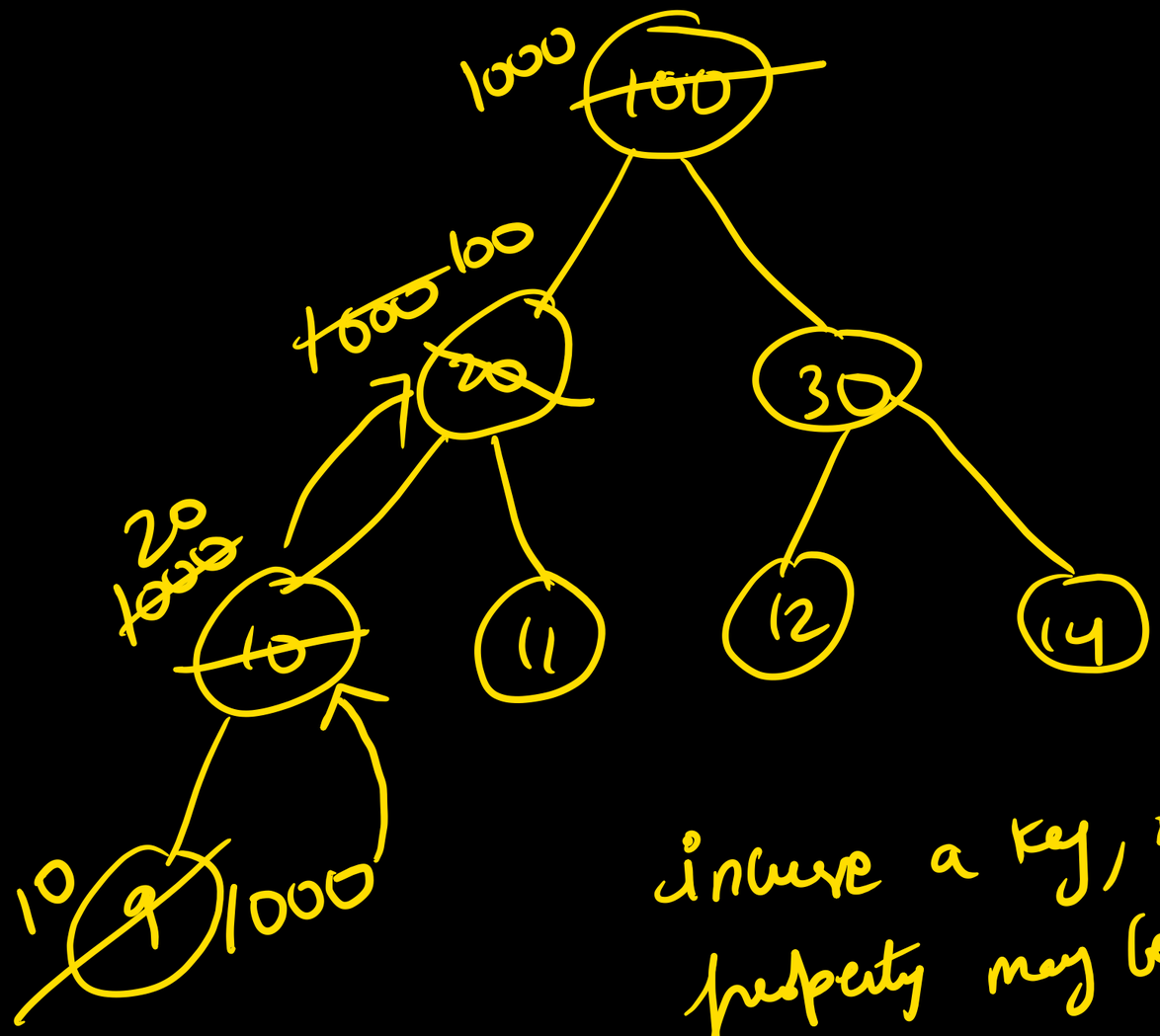
S.C = O(logn)



$n/2 \Rightarrow \log n$

$O(\log n) \times \left(\frac{n}{2}\right)$

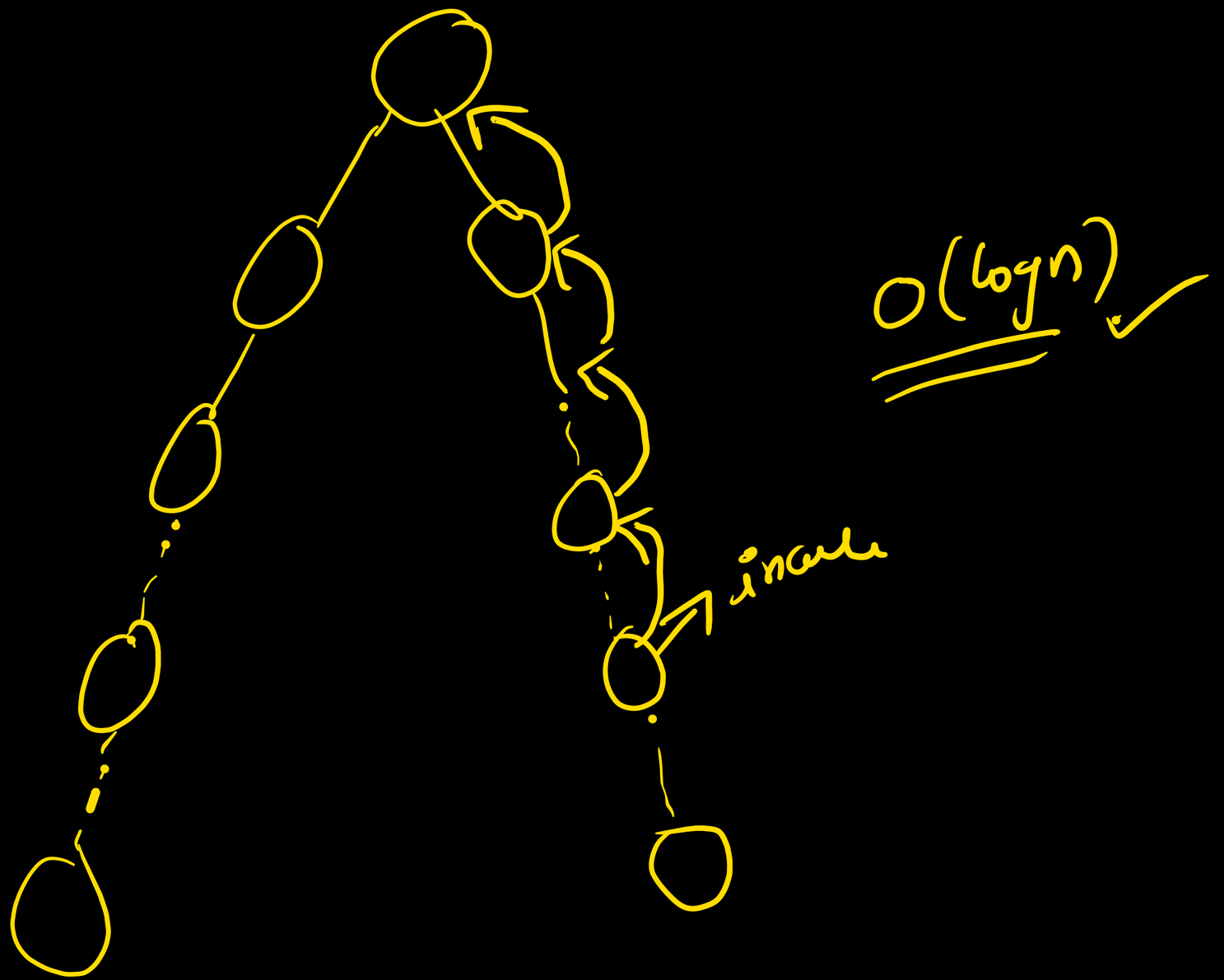
$O(n \log n)$.



increase (key) → element.

9 → 1000

increase a key, max heap property may be violated


O(logn)
single

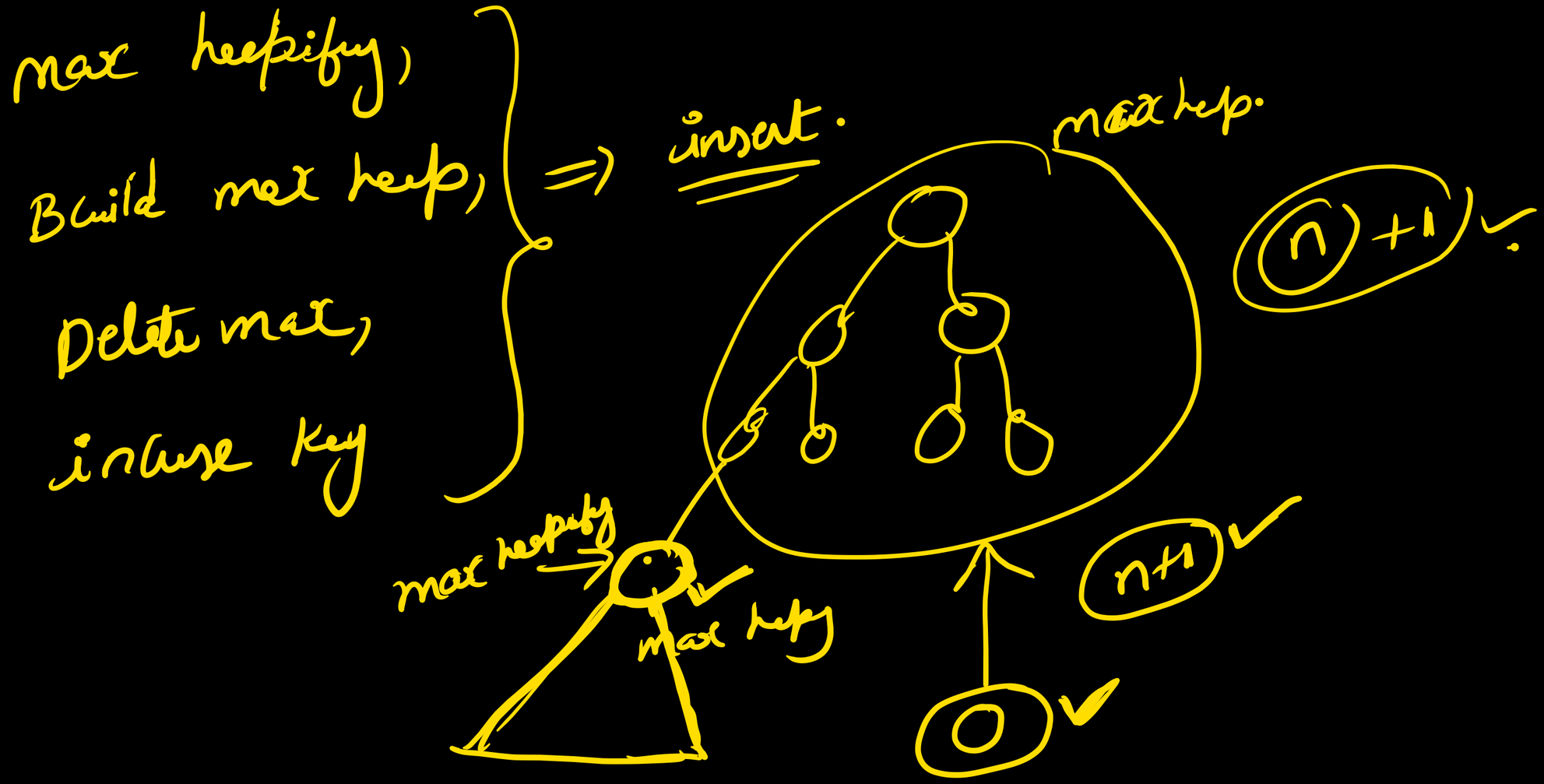
```
Heap_increase_key( A, i, key)
{  if (key < A[i])
       error.
   A[i] = key;
   while( i > 1 and A[i/2] < A[i]
   {
      Exchange ( A[i] and A[i/2])
      i = i/2
   }
}
```

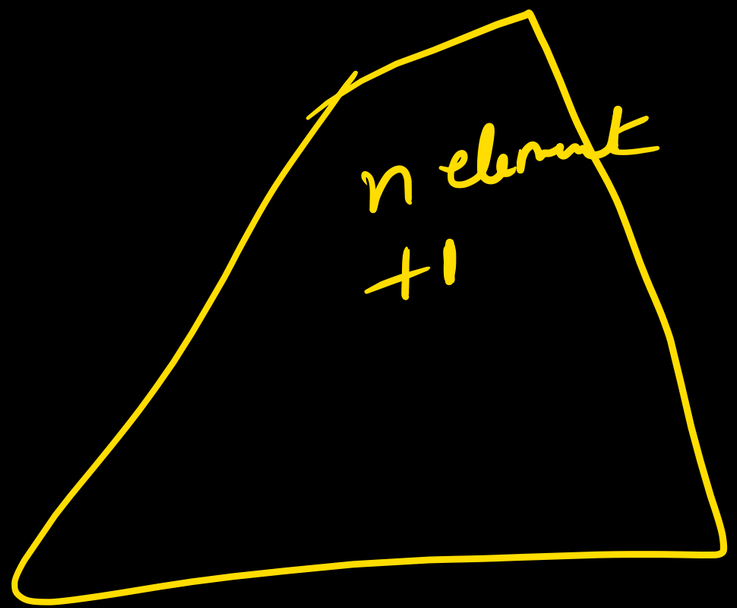
TC = O(log n)
SC = O(1). ✓

```
Heap_ extract_max (A)
{  if (A.heapsize <1)
       pf "error"
   max = A[1];
   A[1] = A[A.heapsize]
   A.heapsize = A.heapsize -1
   max heapify (A, 1)
   return max
}
```

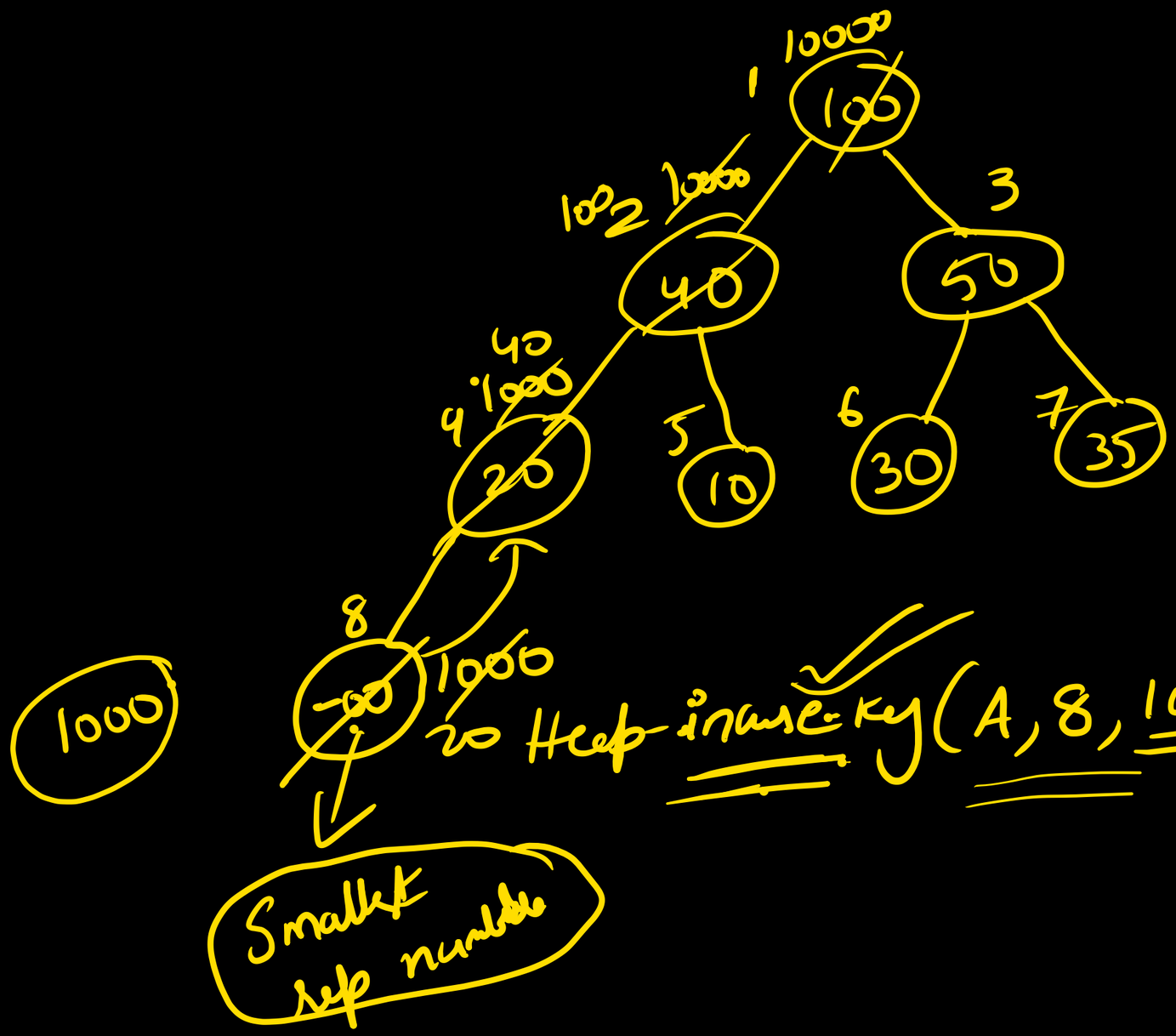
max heapify,
Build max heap,
Delete max,
increase key

⇒ insert.





Build heap(n+1) = O(~~log n~~)



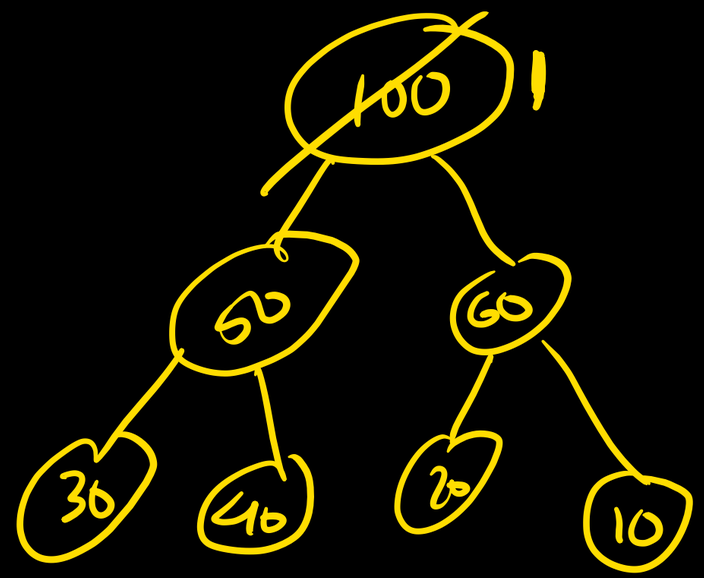
$-\infty \rightarrow$ Smallest no possible.

$T(C) = \log n.$

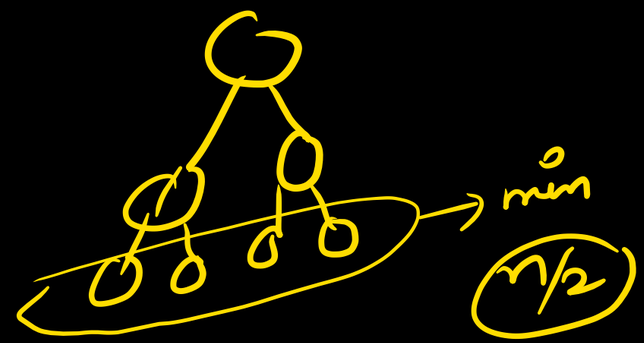
$SC = O(1)$

```
insert ( A, x)
{
    a  A[heapsize+1] = -∞;
       Heap_insert_key ( A, heapsize+1, x)
       heapsize = heapsize+1;
}
```

14



Call heapify.



O(n)

max heap:

Find max	Delete mk	insert	increase key	decrease key
O(1)	O(log n)	O(log n)	O(log)	O(log n)

Find min
O(n) ✓

Search for a random number
O(n)

Delete a random

Delete a random number

(1) Find the random numb → O(n)

(2) Replace it with last node and call max heapify O(nlogn)

O(n)