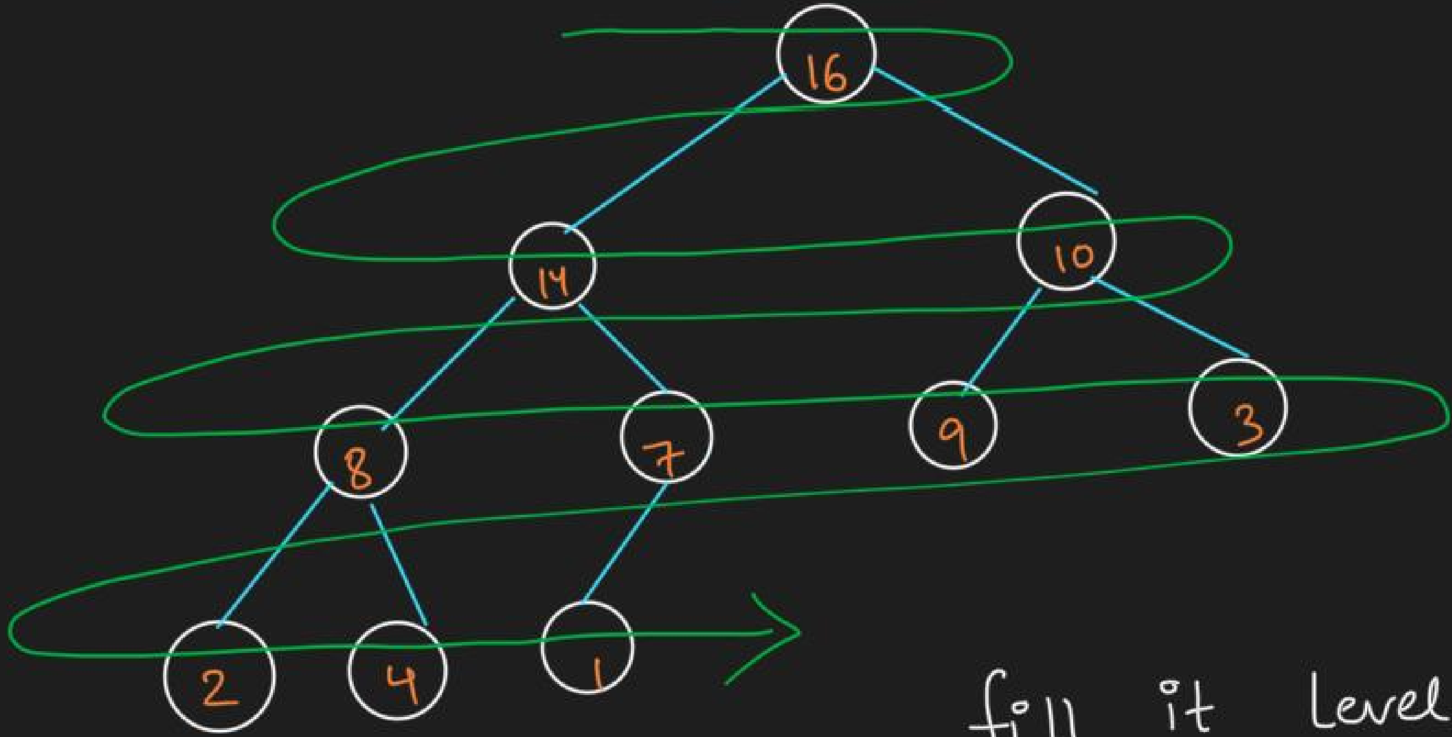
24 September 2020 at 9:14 PM — Shared

Heapsort

<u>Heap</u> : Almost complete Binary Tree



fill it level by level from left to right.

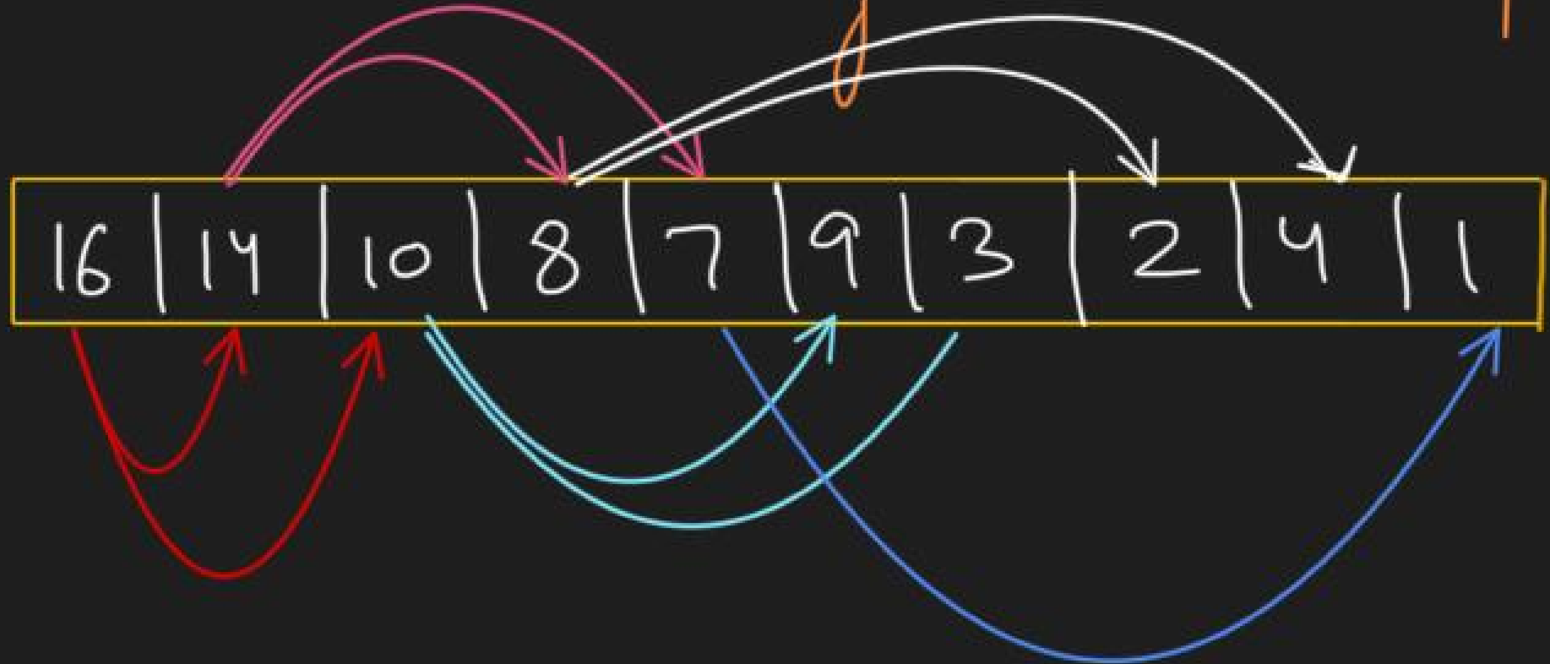
Almost Complete → left bushy

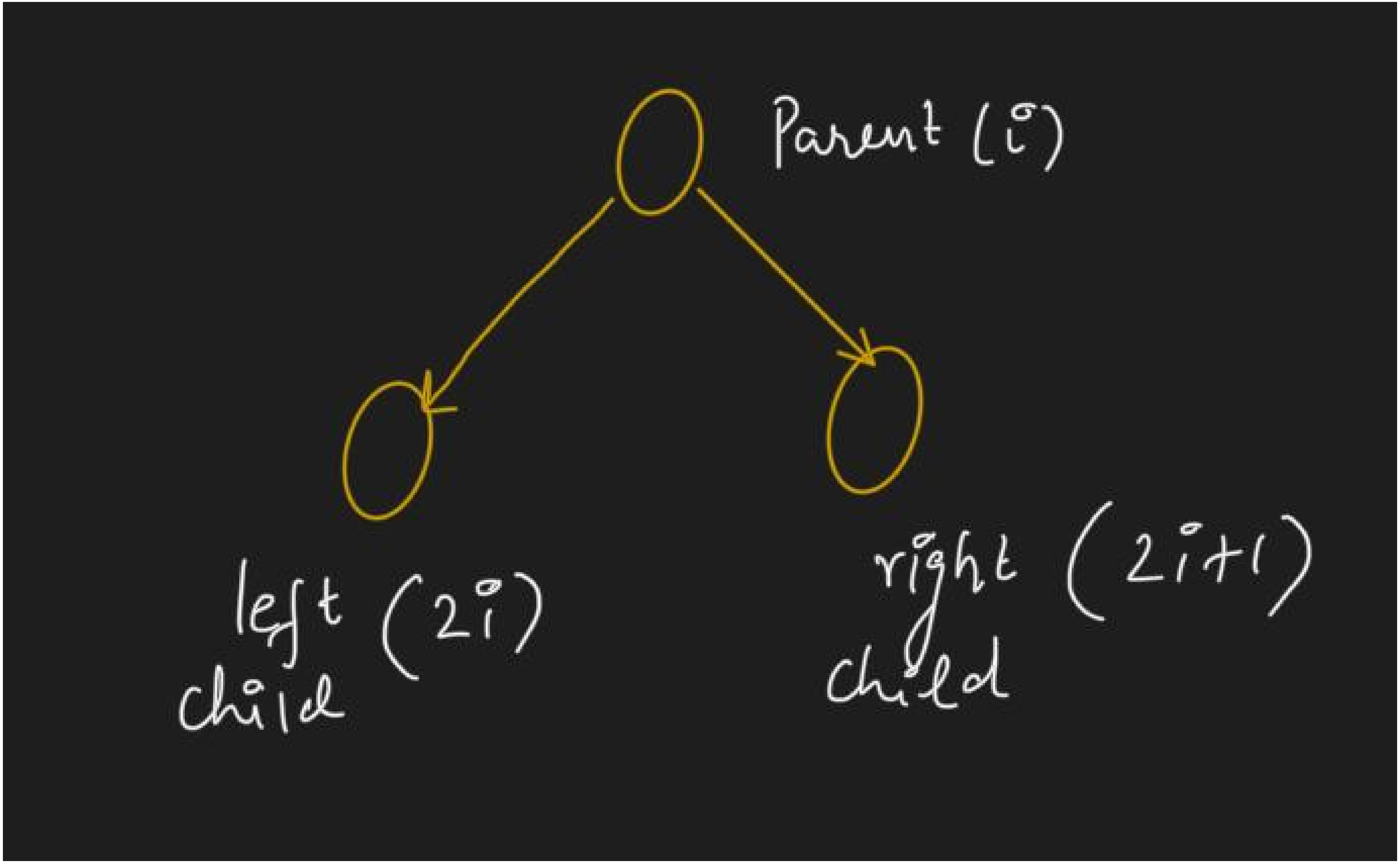
Binary tree → at-most 2 children

If you are at level k,
the nodes of level K-2 must
be completely full

Children must be lower than parent in case of min-heap.

Root is minimum element in case of min heap.



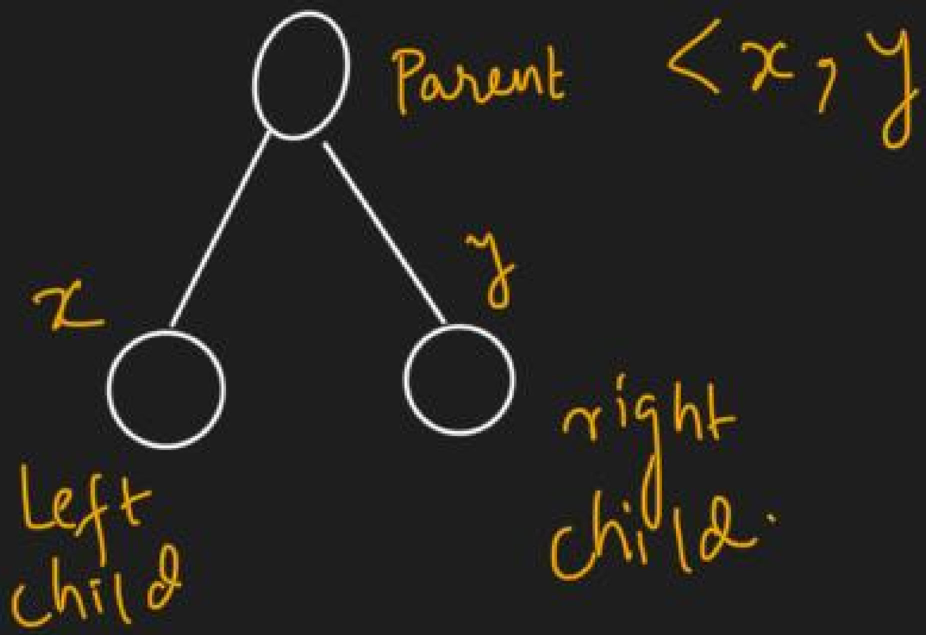

Parent (i)
left child (2i)
right child (2i+1)

Binary heap

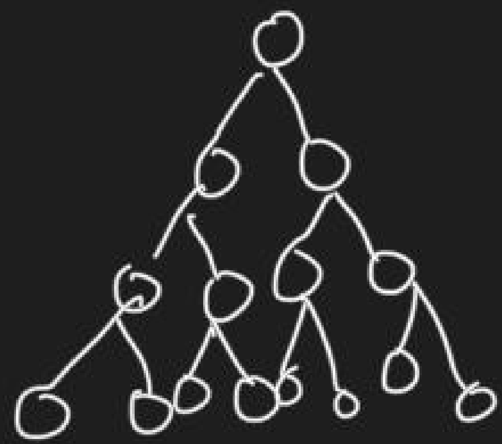
min heap

max heap

* min-heap property



Height of Heap $O(\log n)$



almost Complete Binary tree

Max-heapify : $O(\log n)$

Build max-heap : $O(n)$

↓

Beautiful proof

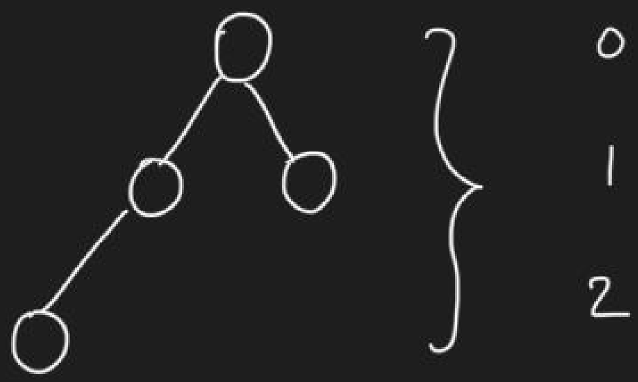
Heap-sort $O(n * \log n)$

In place - sorting Algorithm

Extract-min: $O(\log n)$ for min-heap

Extract-max $O(\log n)$ for max heap

Q min & max no of elements in a heap of height h



min elements

height 2 : 2^h

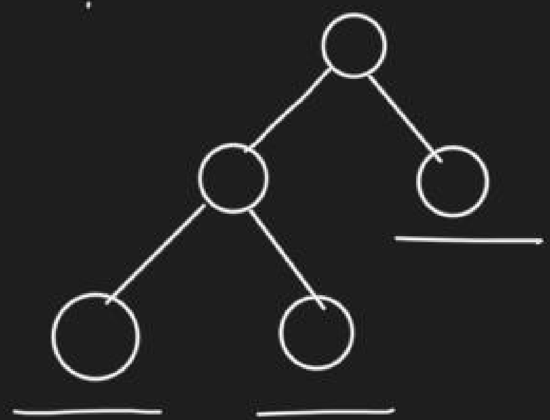
max elements : $2^{h+1} - 1$

$$2^h \leq x \leq 2^{h+1} - 1$$

$$x \in [2^h, 2^{h+1} - 1]$$

no of elements in heap of height h.

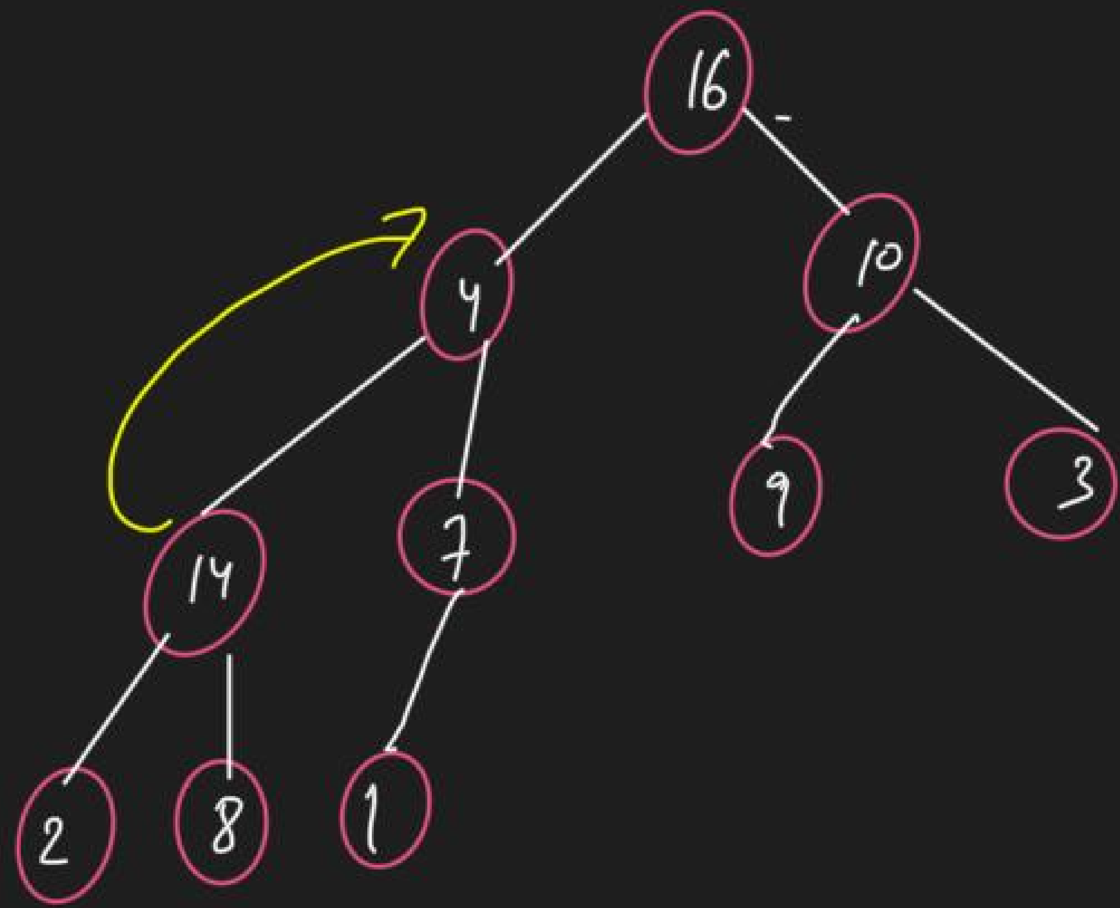
Q In a max-heap, all elements are distinct.
Where does the minimum element reside?

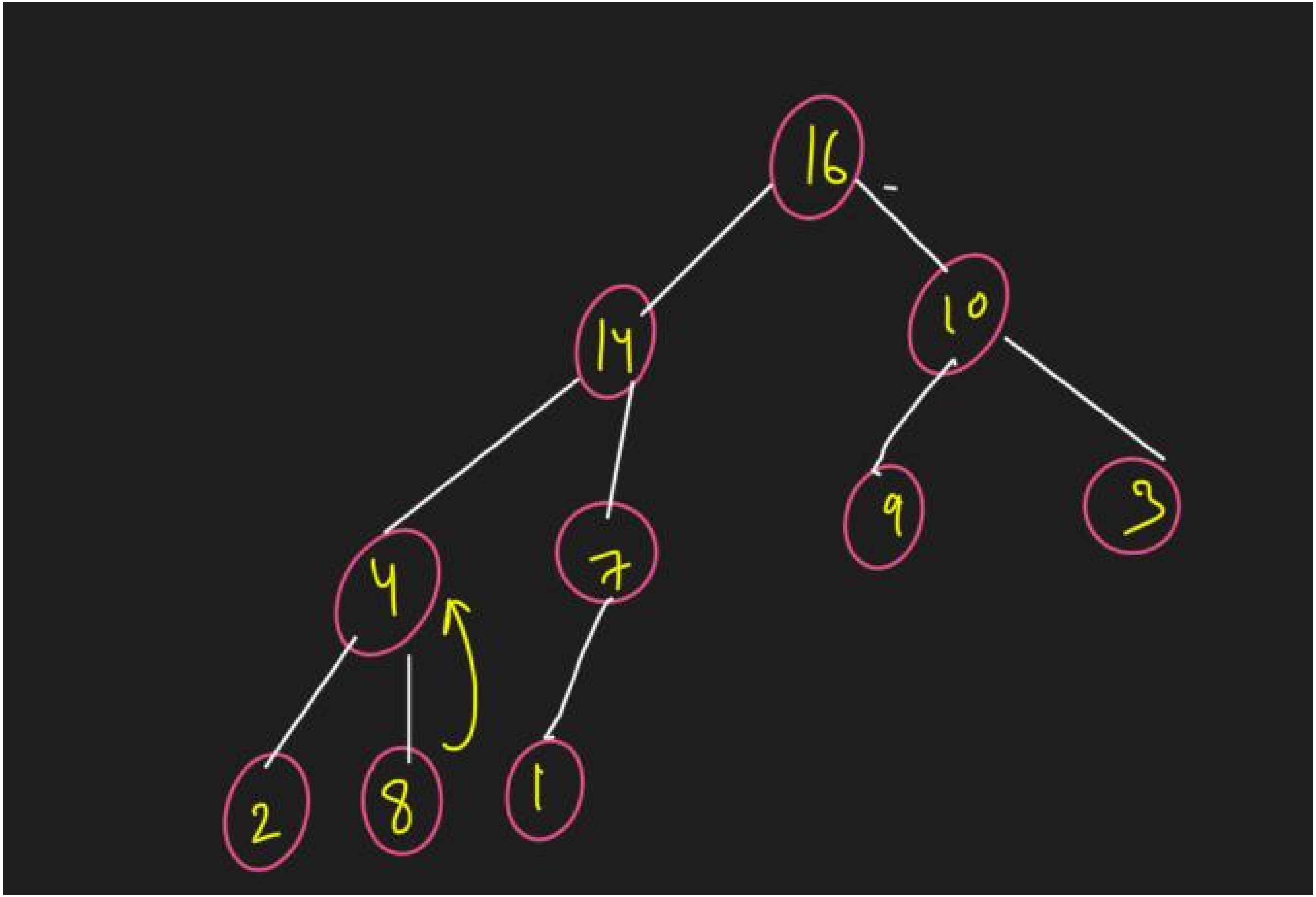


The minimum element can reside in last level & second last level only.

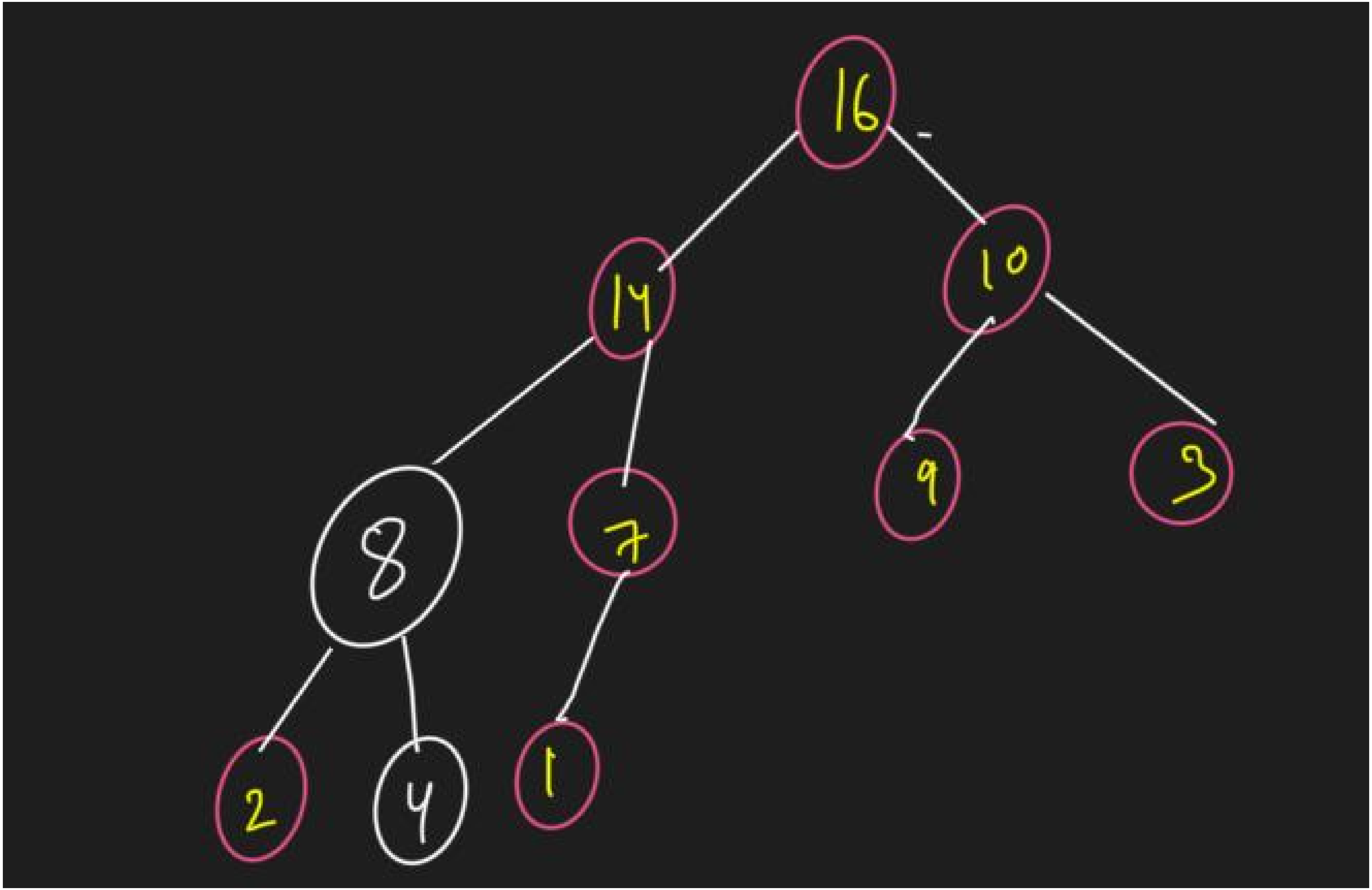
Max heapify

16, 4, 10, 14, 7, 9, 3, 2, 8, 1





16
14
10
4
7
9
3
2
8
1



16
14
10
8
7
9
3
2
4
1

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

master method Case II

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

Why max- heapify takes $O(n)$?

Build Max-heapify (A)

{

A. heap size = A. length

for $i = \left\lfloor \frac{A.len}{2} \right\rfloor$ to 1

max heapify (A, i)

i - - ;

}

Whenever max-heapify is called on a node, the two sub-trees of that node are both max-heaps.

$$\underbrace{\frac{1}{2} + \frac{2}{2^2} + \frac{3}{3^2} + \cdots + \frac{h}{2^h}}_{\log n \text{ times.}}$$

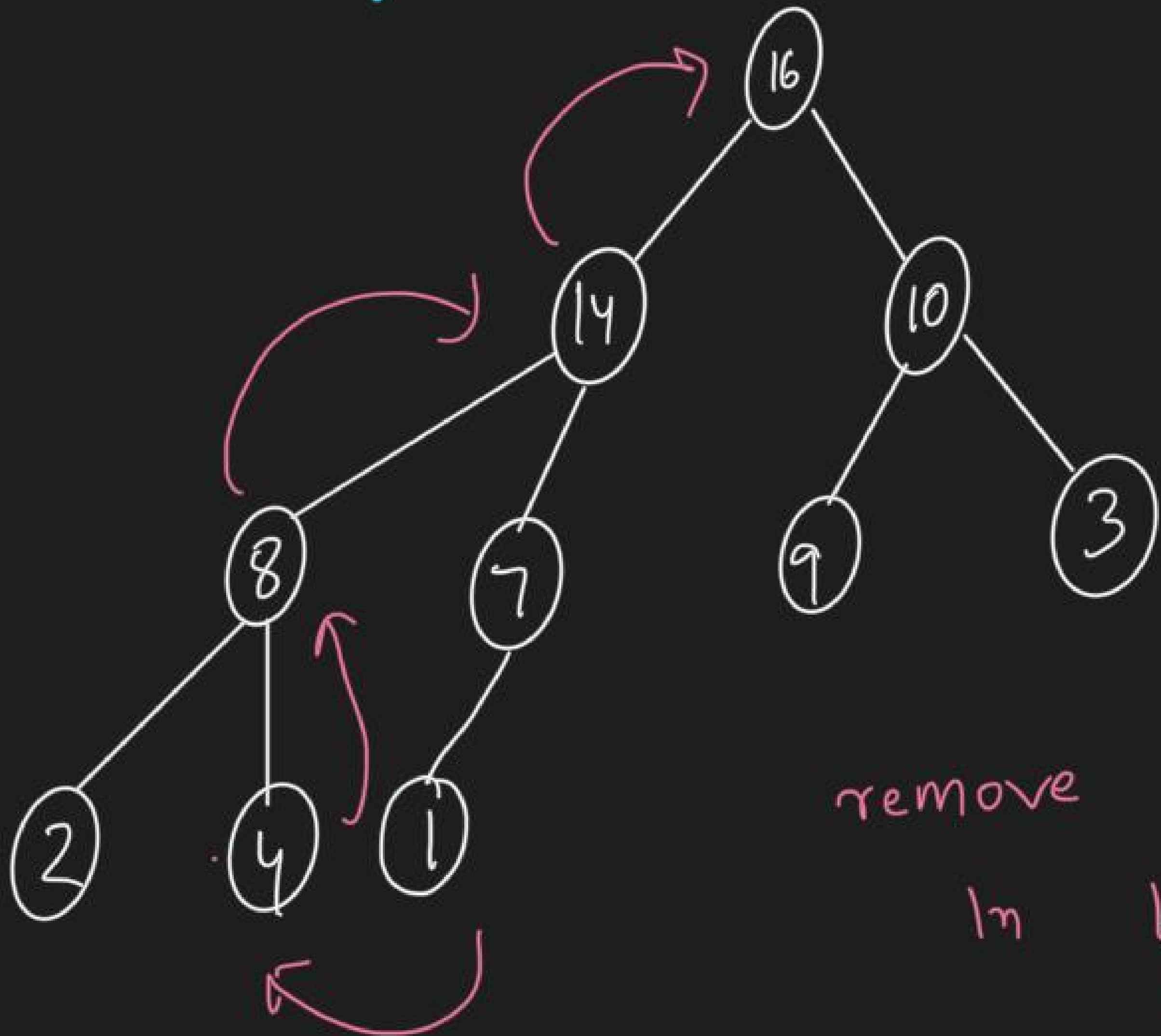
$$\sum \frac{h}{2^h} = \frac{1/2}{\left(1-\frac{1}{2}\right)^2} = 2$$

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

Worst-case Running time of Heapsort

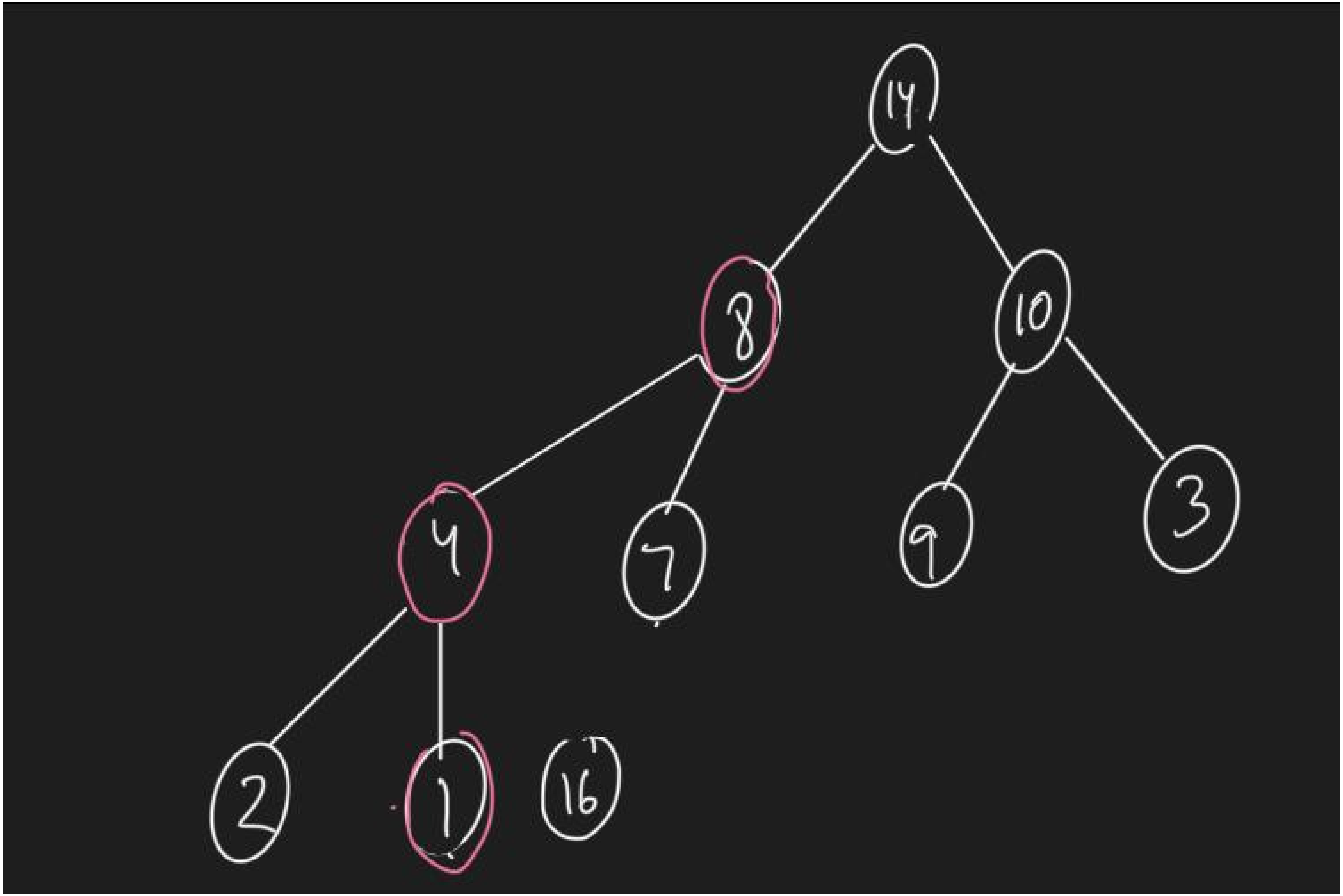
$= \Omega(n \log n)$

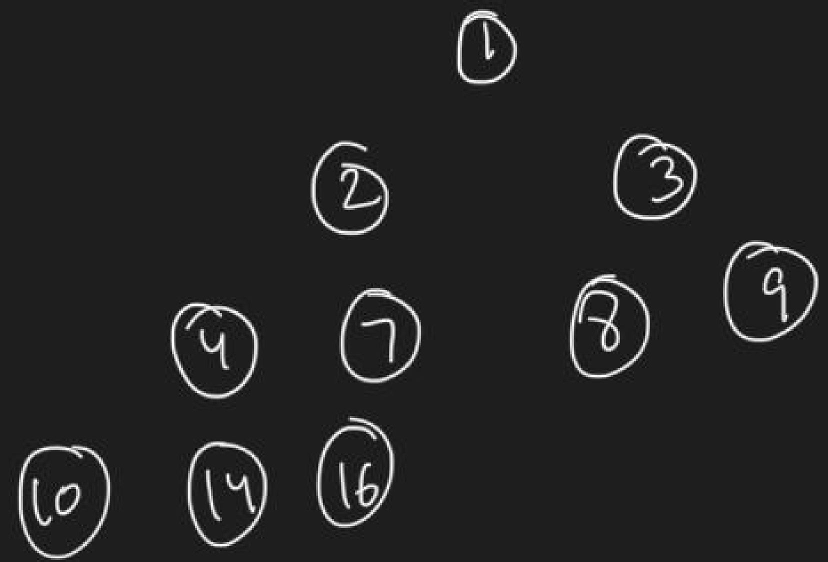
min-heapify   16, 14, 10, 8, 7, 9, 3, 2, 4, 1



remove Root

In logn +1 steps.


14
8
10
4
7
9
3
2
1
16



After 10 iterations we got this.

$n = 10.$

at each step we did $\log n$ work.

at n steps.

$n \log n$ work