CS & IT ENGINEERING

Data Structure & Programming

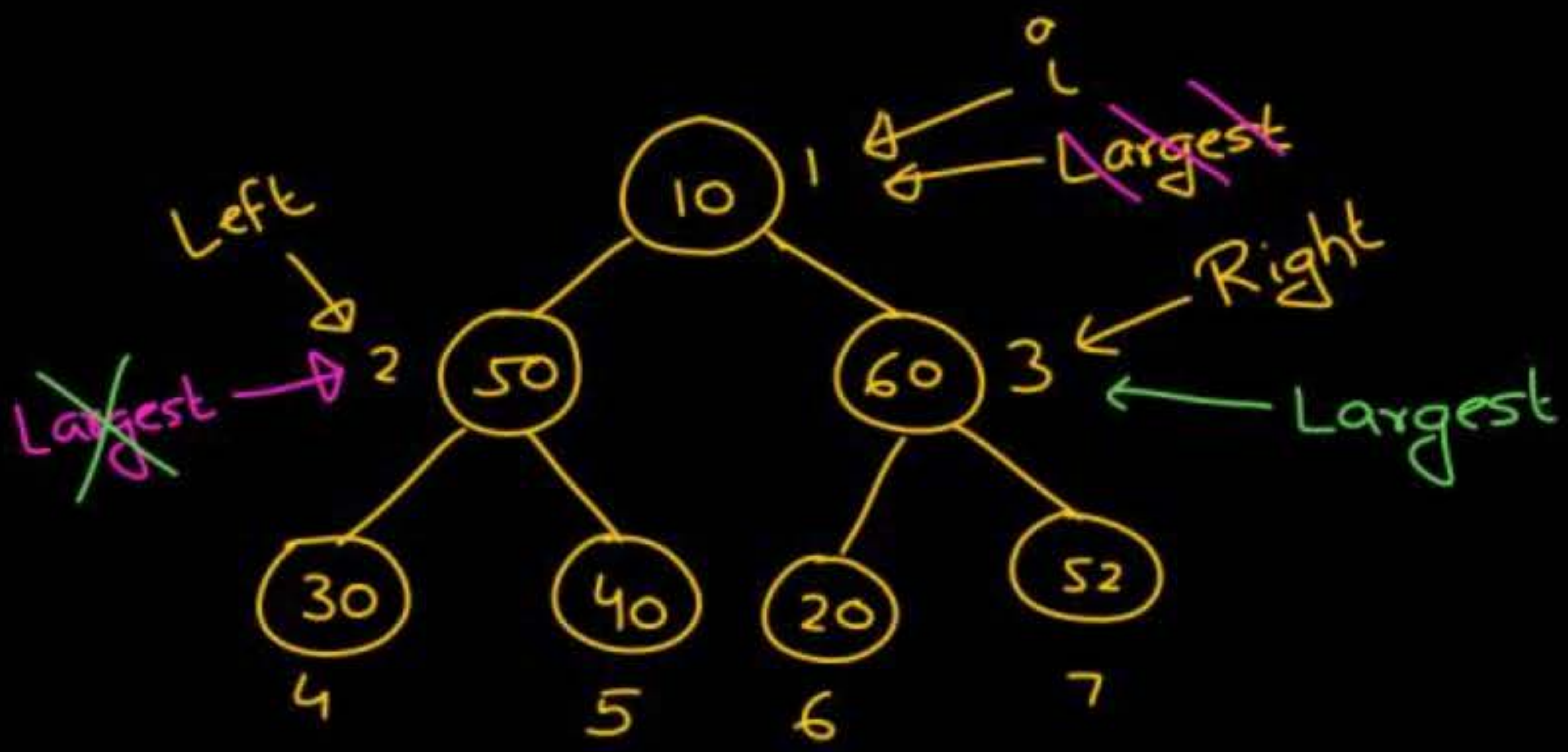
Tree

Lec- 08

By- Pankaj Sharma Sir

TOPICS TO BE COVERED

- Tree 08



Heapify at index 1

$\boxed{n = 7}$

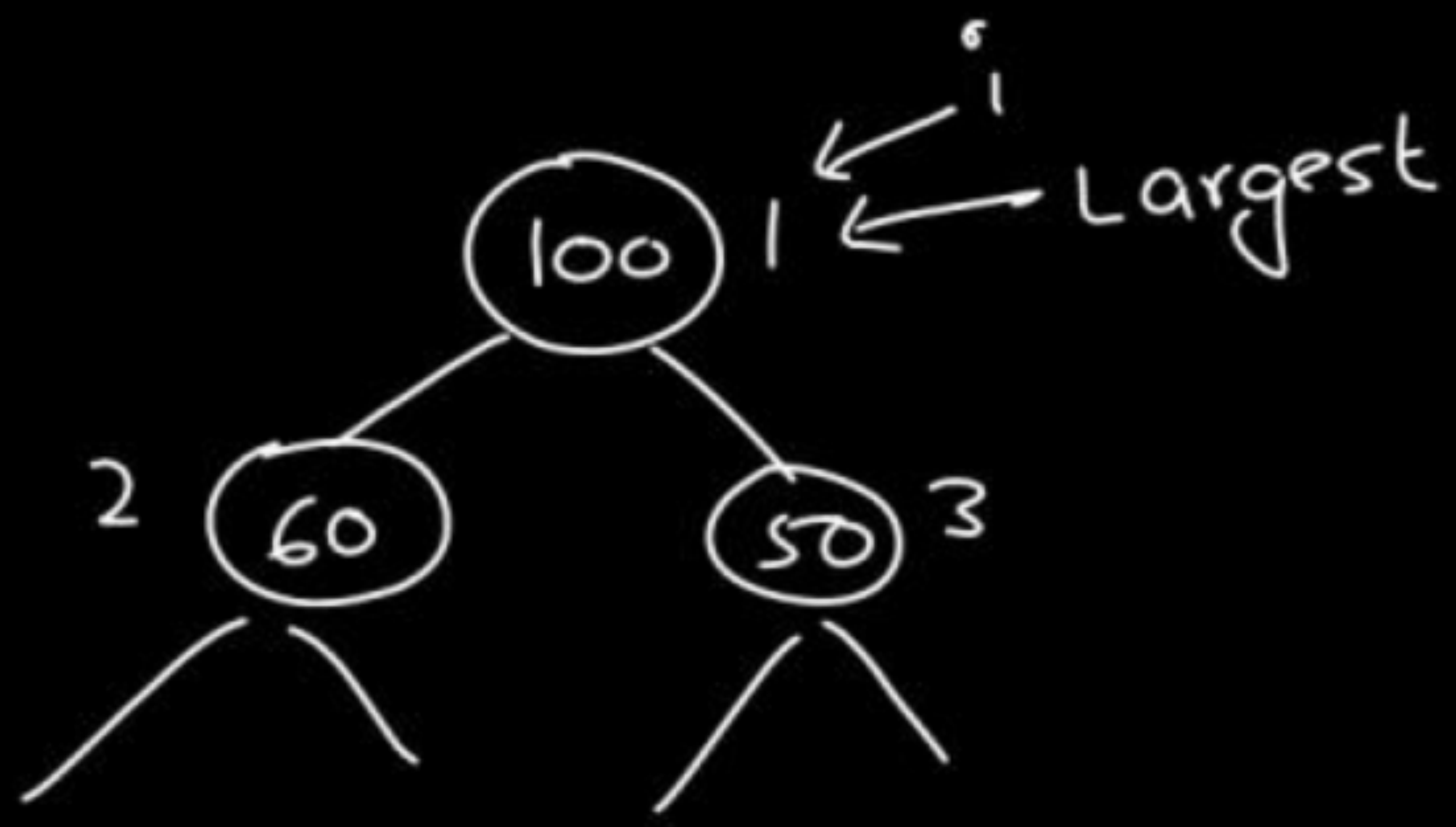
A						
10	50	60	30	40	20	52
1	2	3	4	5	6	7

Heapify(A, i, n)

1) Left = 2*i ; Right = 2*i + 1; Largest = i ;

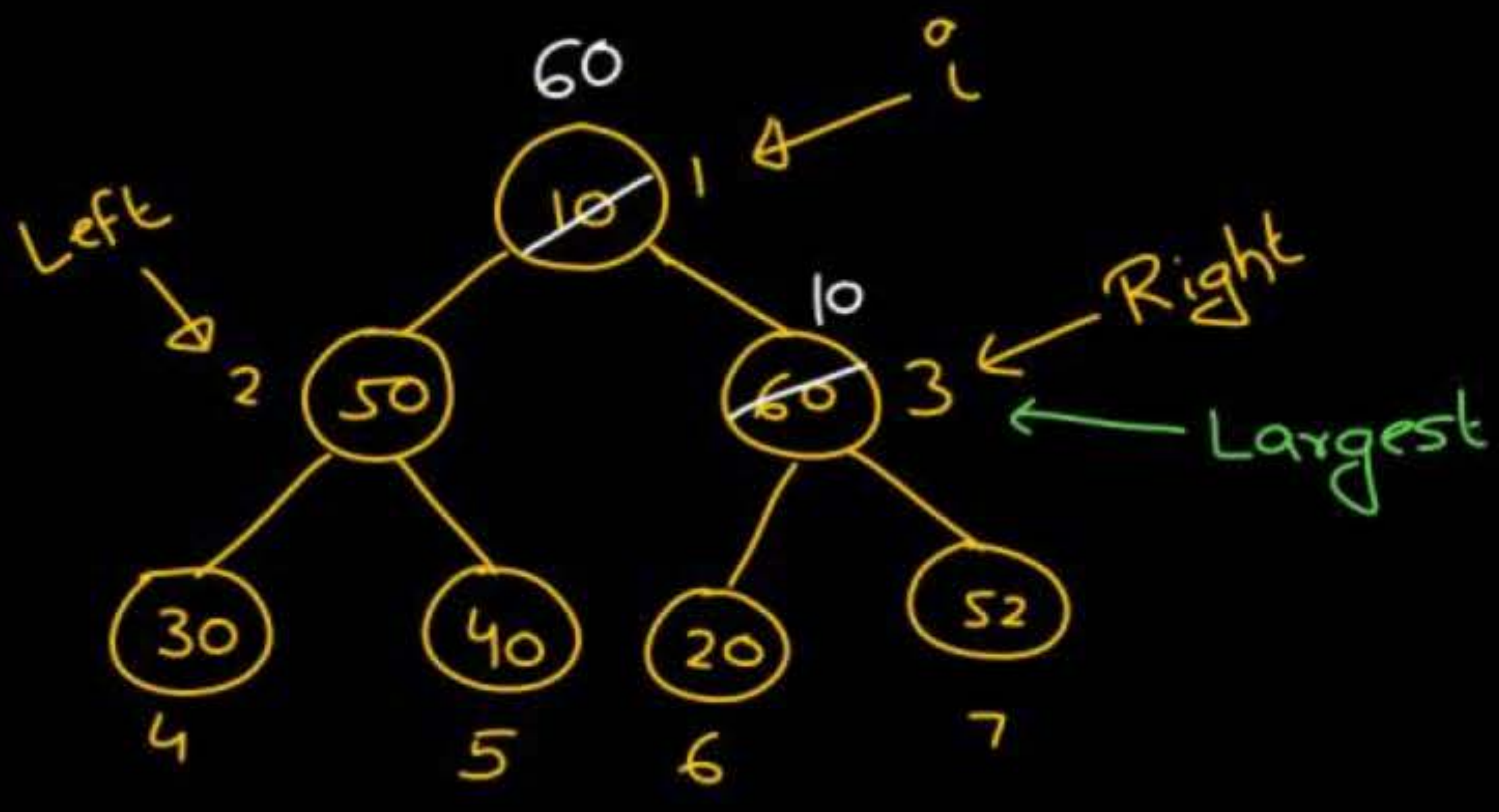
2) if Left <= n && A[Largest] < A[Left]
   Largest = Left ;

3) if Right <= n && A[Largest] < A[Right]
   Largest = Right ;



i == largest

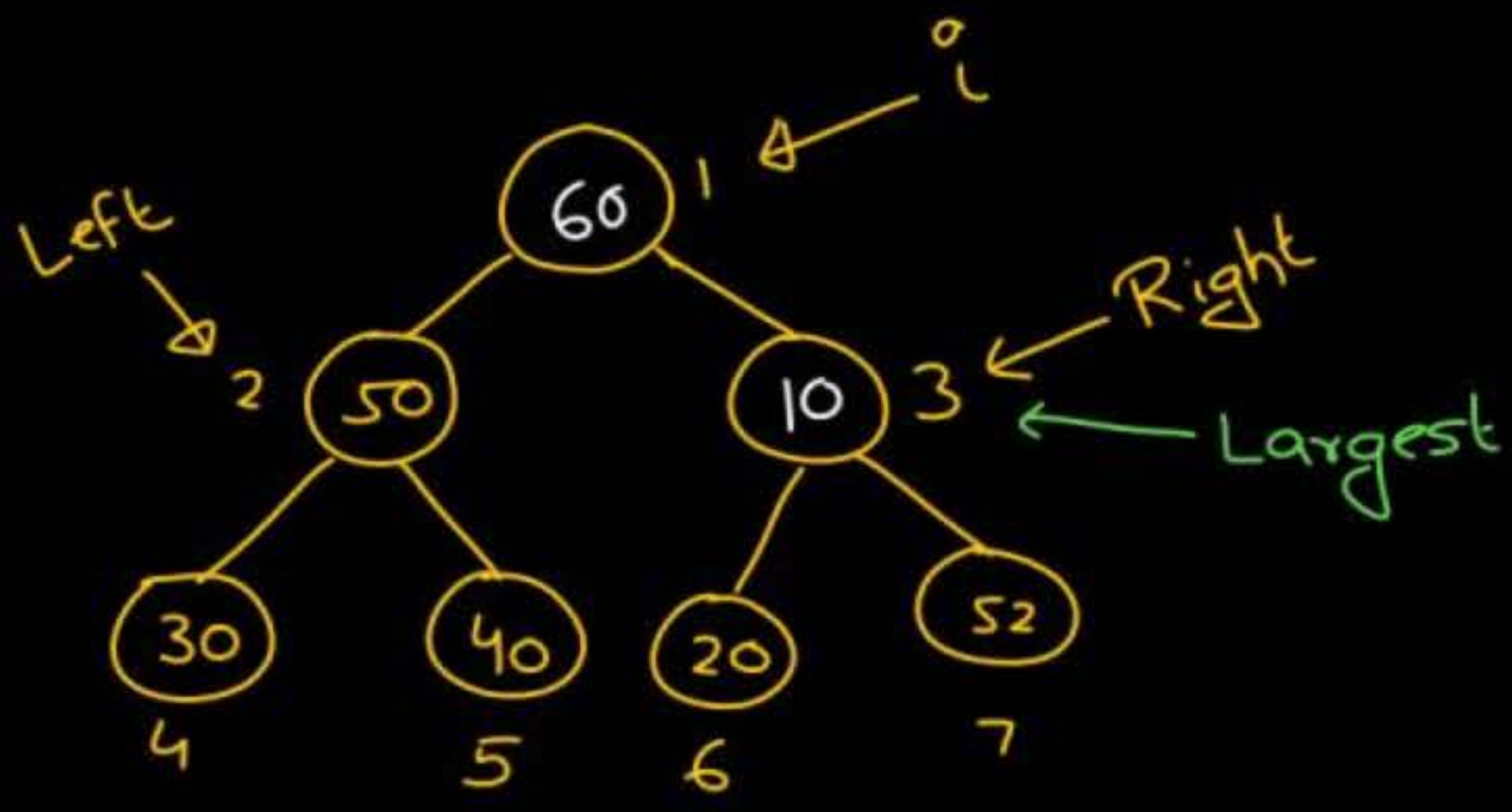
1.
2.
3



A

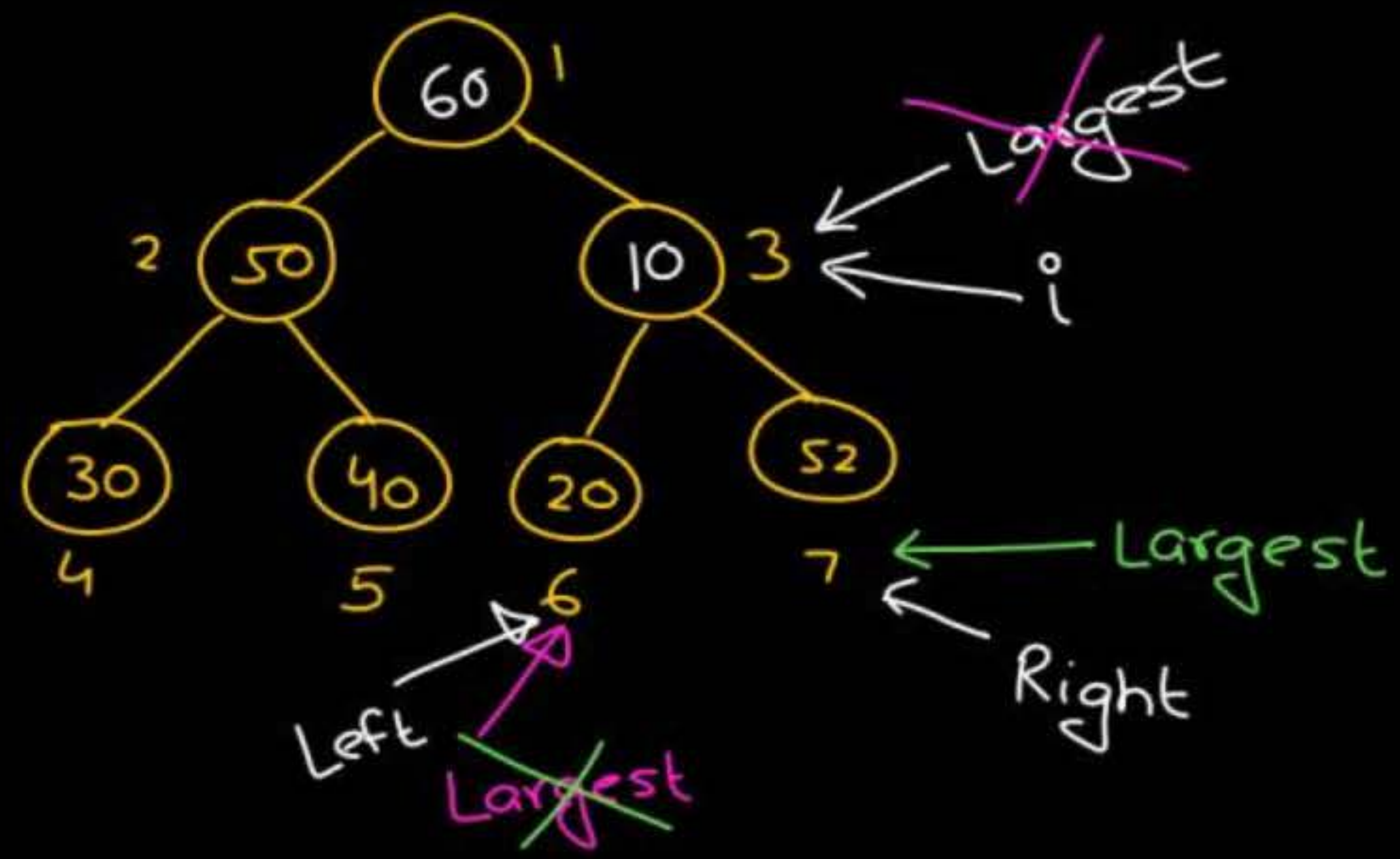
10	50	60	30	40	20	52
1	2	3	4	5	6	7

```
Heapify(A, i, n)

1) Left = 2*i ; Right = 2*i + 1; Largest = i ;

2) if Left <= n && A[Largest] < A[Left]
          Largest = Left ;

3) if Right <= n && A[Largest] < A[Right]
          Largest = Right ;

4) if (i != Largest) {
          swap(A[i], A[Largest]);

   }
```



A	10	50	60	30	40	20	52
	1	2	3	4	5	6	7

```
Heapify(A, i, n)

1) Left = 2*i ; Right = 2*i + 1; Largest = i;
2) if Left <= n && A[Largest] < A[Left]
        Largest = Left;
3) if Right <= n && A[Largest] < A[Right]
        Largest = Right;

4) if (i != Largest) {
        swap(A[i], A[Largest]);
        Heapify(A, Largest, n);
   }
```



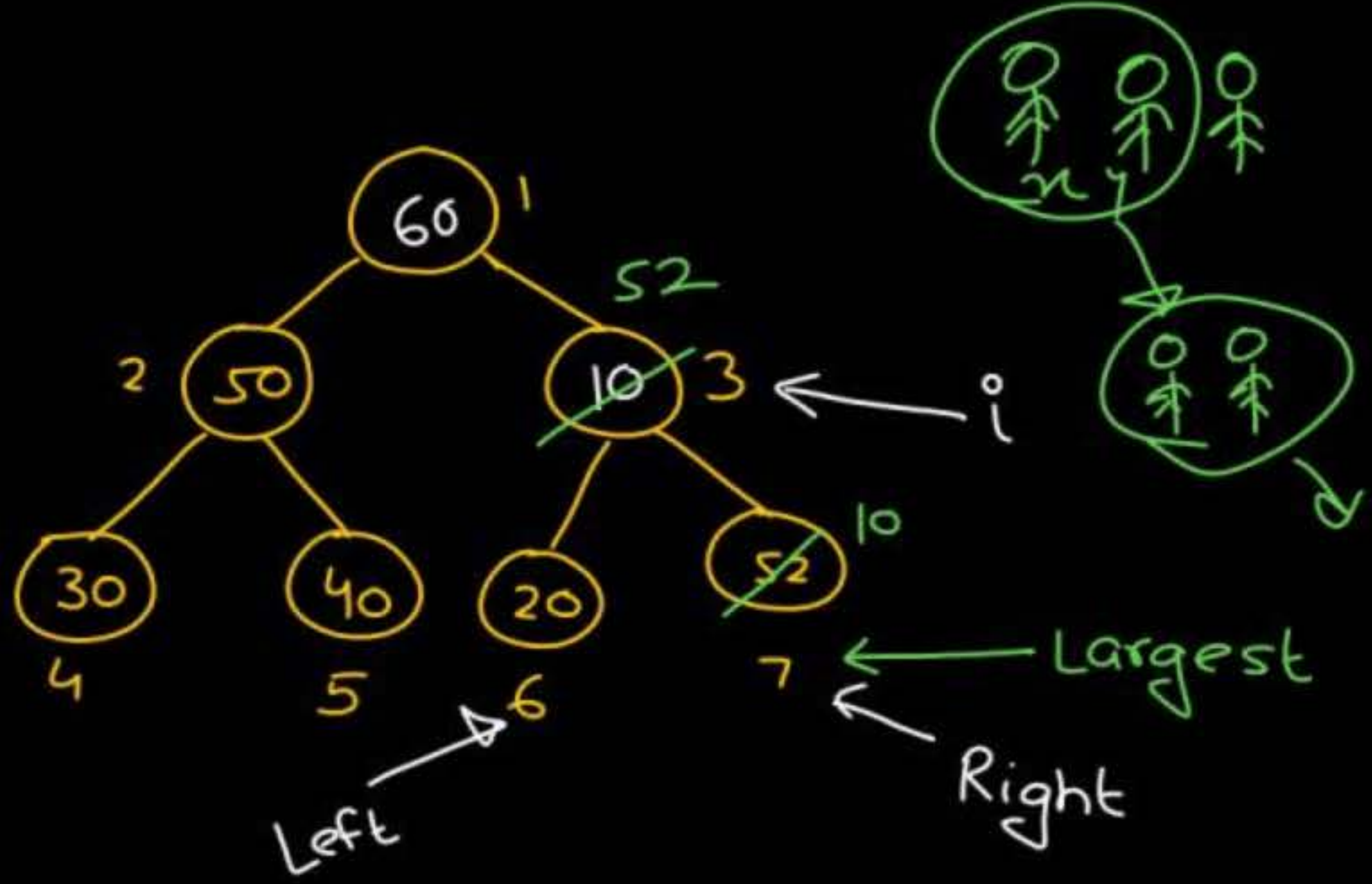
$n=7$

$6 <= 7$

A

10	50	60	30	40	20	52
1	2	3	4	5	6	7

```
Heapify(A, i, n)

1) Left = 2*i ; Right = 2*i + 1; Largest = i;

2) if Left <= n && A[Largest] < A[Left]
        Largest = Left;

3) if Right <= n && A[Largest] < A[Right]
        Largest = Right;

4) if (i != Largest) {
        swap(A[i], A[Largest]);
        Heapify(A, Largest, n);
   }
```

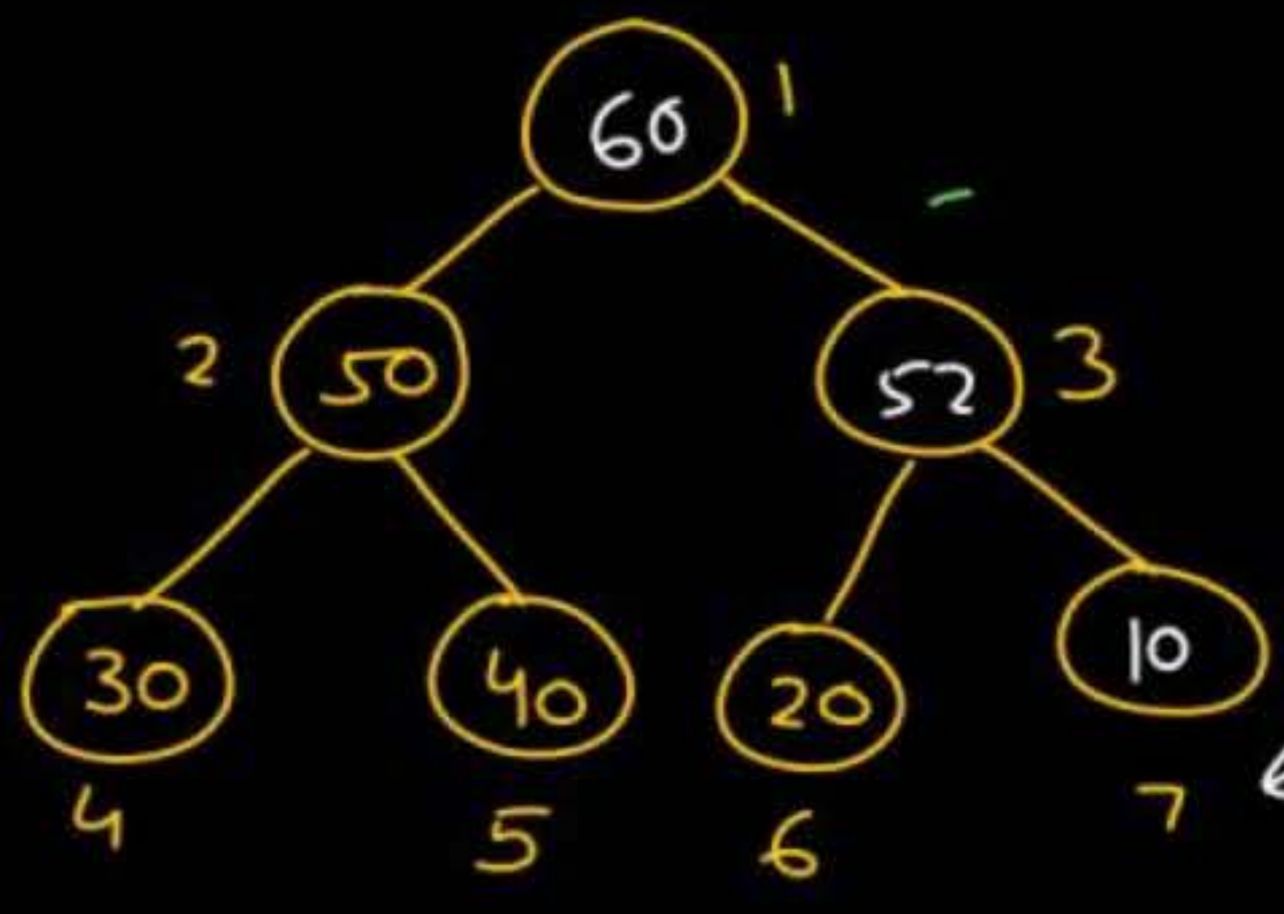


Heapify(A, ~~7~~, 7)

A

10	50	60	30	40	20	52
1	2	3	4	5	6	7

```
Heapify(A, i, n)

1) Left = 2*i; Right = 2*i+1; Largest = i;
2) if Left <= n && A[Largest] < A[Left]
       Largest = Left;
3) if Right <= n && A[Largest] < A[Right]
       Largest = Right;

4) if (i != Largest) {
       swap(A[i], A[Largest]);
       Heapify(A, Largest, n);
   }
```



i

$n = 7$

Heapify(A, 7, 7)

A						
10	50	60	30	40	20	52
1	2	3	4	5	6	7

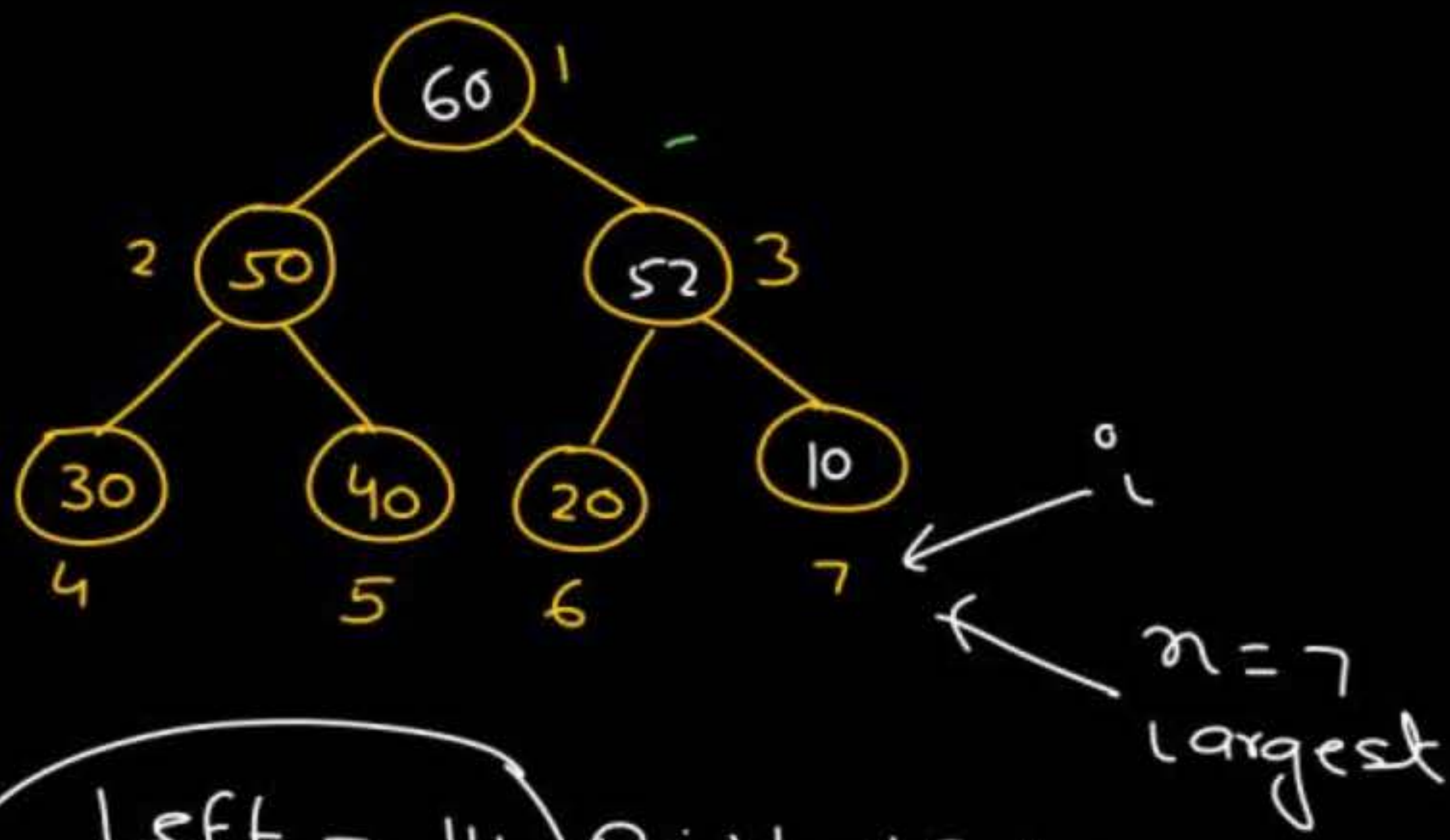
Heapify(A, i, n)

1) Left = 2*i ; Right = 2*i + 1; Largest = i;

2) if Left <= n && A[Largest] < A[Left]
   Largest = Left;

3) if Right <= n && A[Largest] < A[Right]
   Largest = Right;

4) if (i != Largest) {
   swap(A[i], A[Largest]);
   Heapify(A, Largest, n);
   }



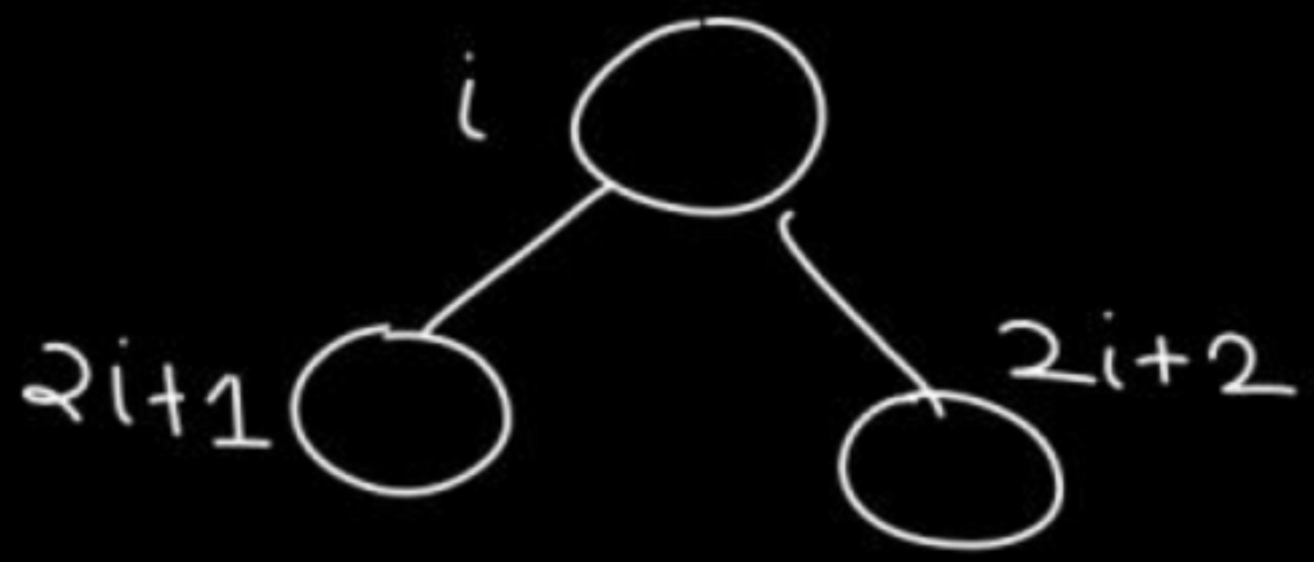
Left = 14, Right = 15, Largest

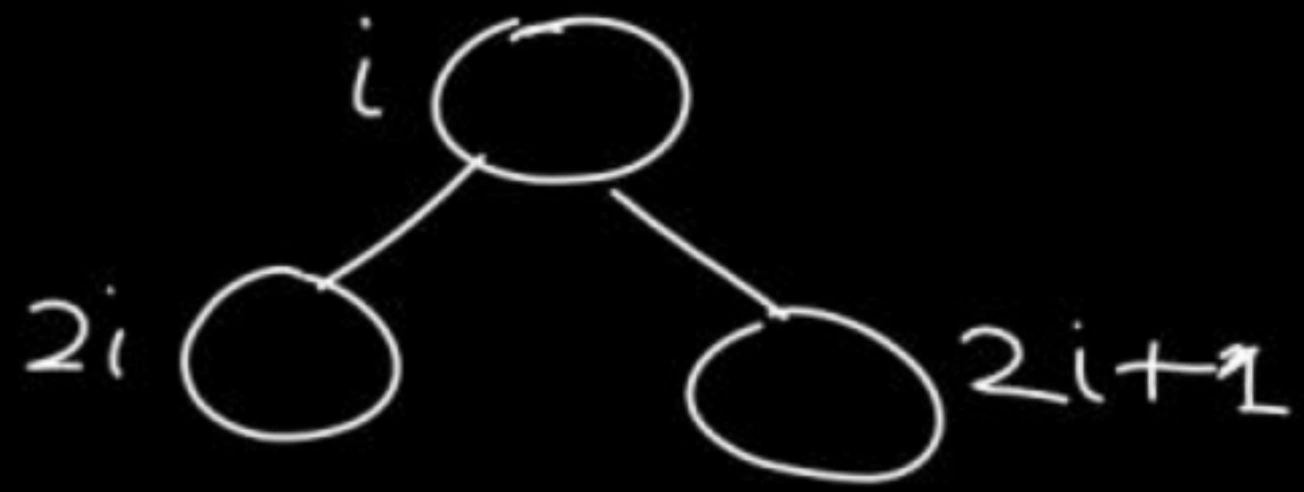
A

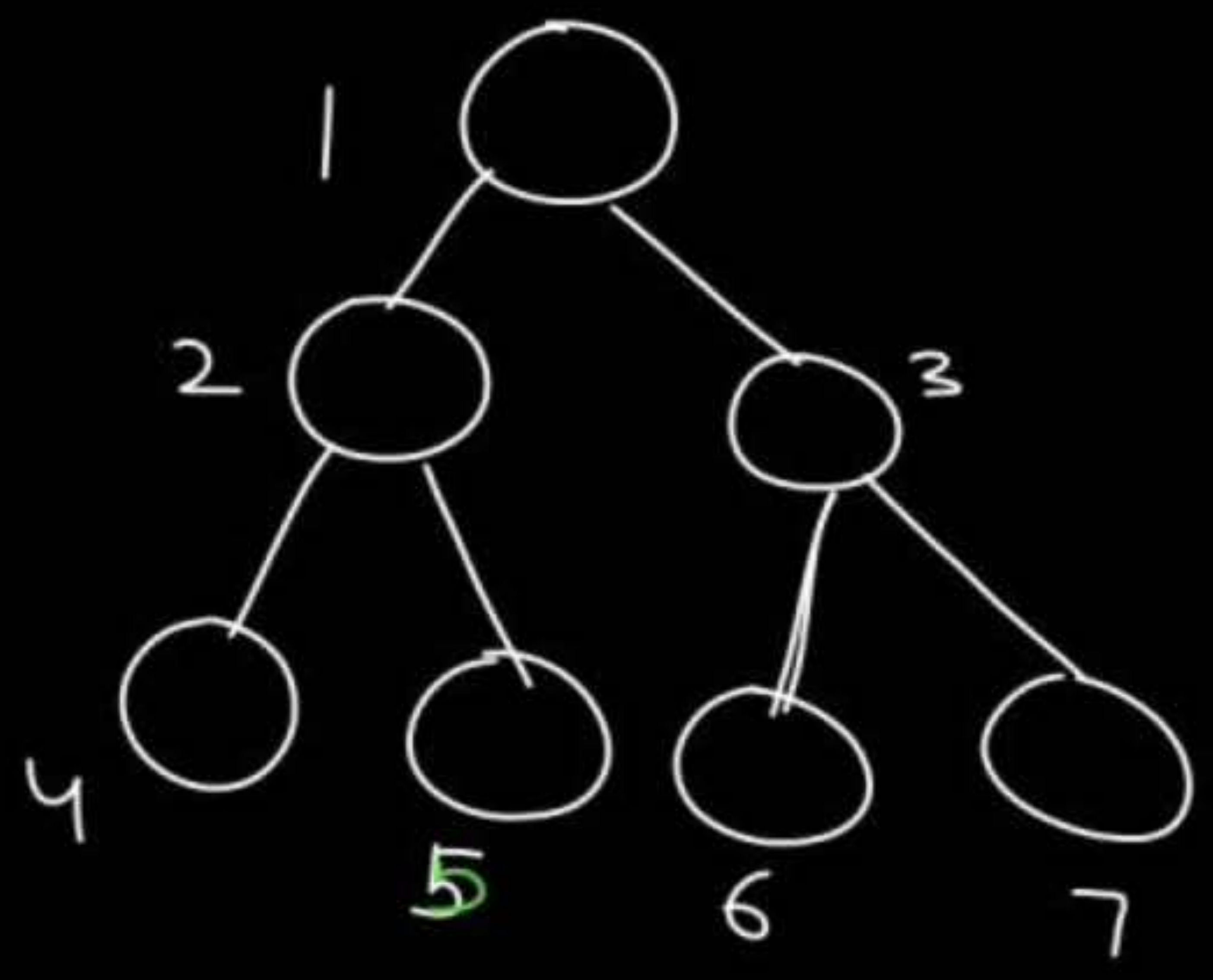
10	50	60	30	40	20	52
1	2	3	4	5	6	7

```
Heapify(A, i, n)

1) Left = 2*i ; Right = 2*i + 1; Largest = i;
2) if (Left <= n) && A[Largest] < A[Left]
        Largest = Left;
3) if (Right <= n) && A[Largest] < A[Right]
        Largest = Right;

4) if (i != Largest) {
        swap(A[i], A[Largest]);
        Heapify(A, Largest, n);
   }
```

Array rep.



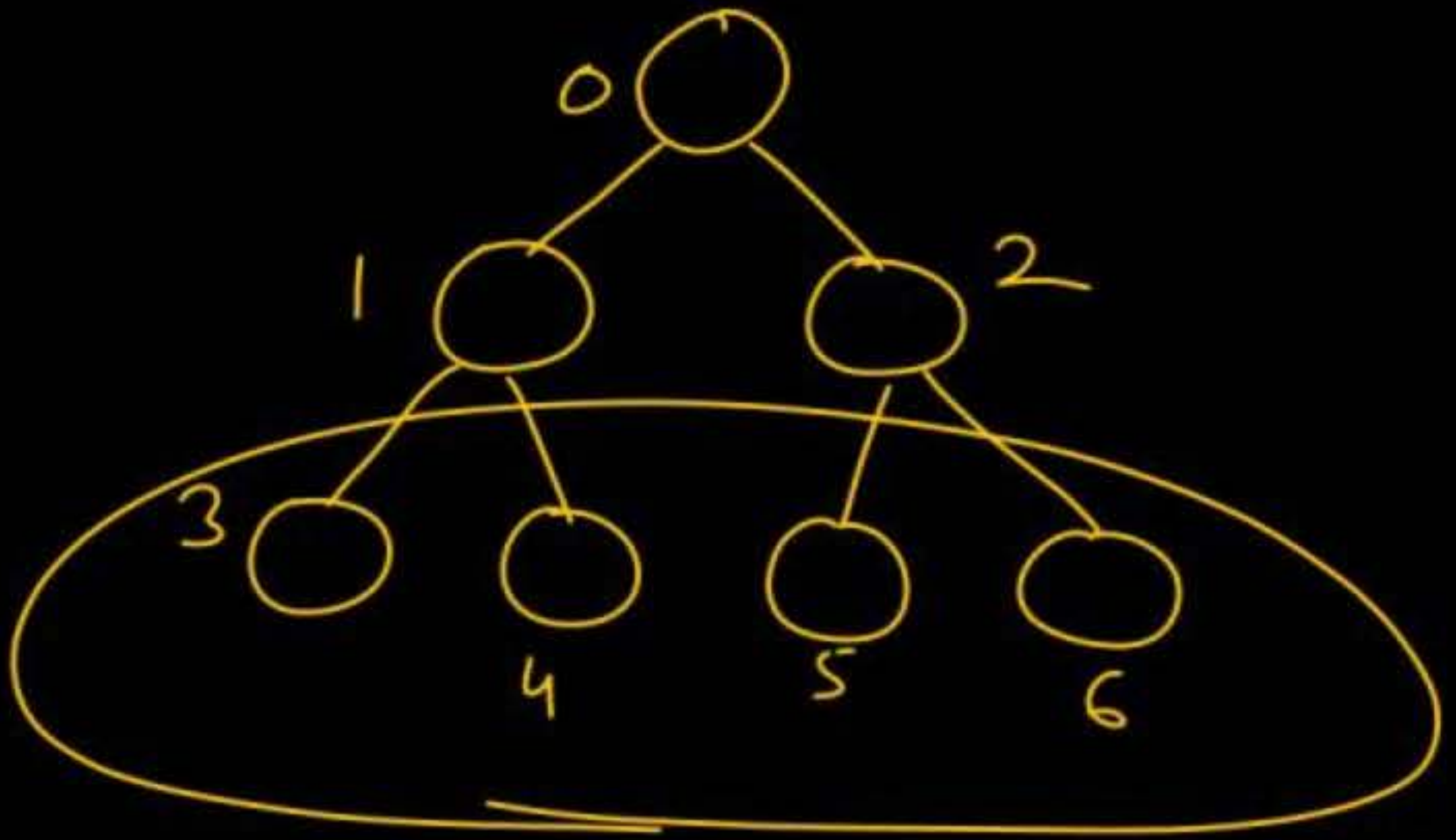




index

Node $\Rightarrow$ i

Parent $\Rightarrow$ $\left\lfloor \frac{i}{2} \right\rfloor$



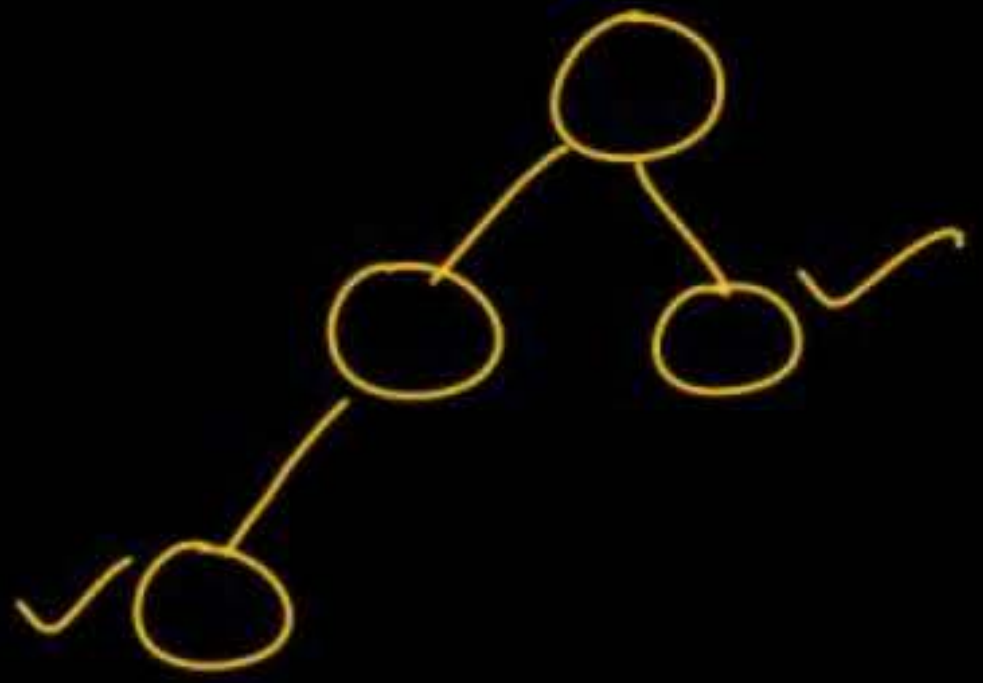
index

Node $\Rightarrow i$

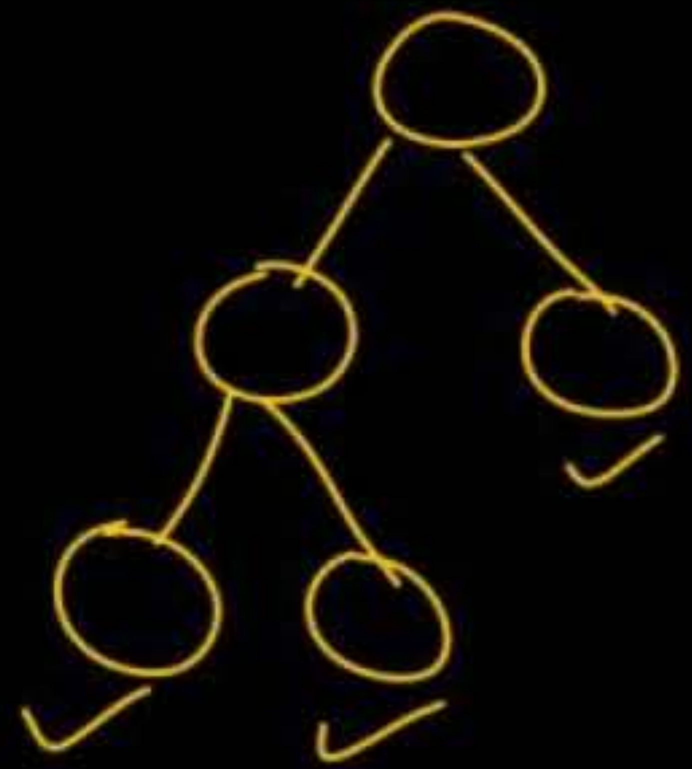
Par $\Rightarrow \left\lfloor \frac{(i-1)}{2} \right\rfloor$

No. of leaf nodes in a heap with n nodes

$\left\lceil \frac{7}{2} \right\rceil = 4 \qquad = \left\lceil \frac{n}{2} \right\rceil$



$$\left\lceil \frac{4}{2} \right\rceil = 2$$



$$\left\lceil \frac{5}{2} \right\rceil = \lceil 2.5 \rceil = 3$$

1) Const. of heap by inserting keys one after another in a given order

$\Rightarrow n \log n$.

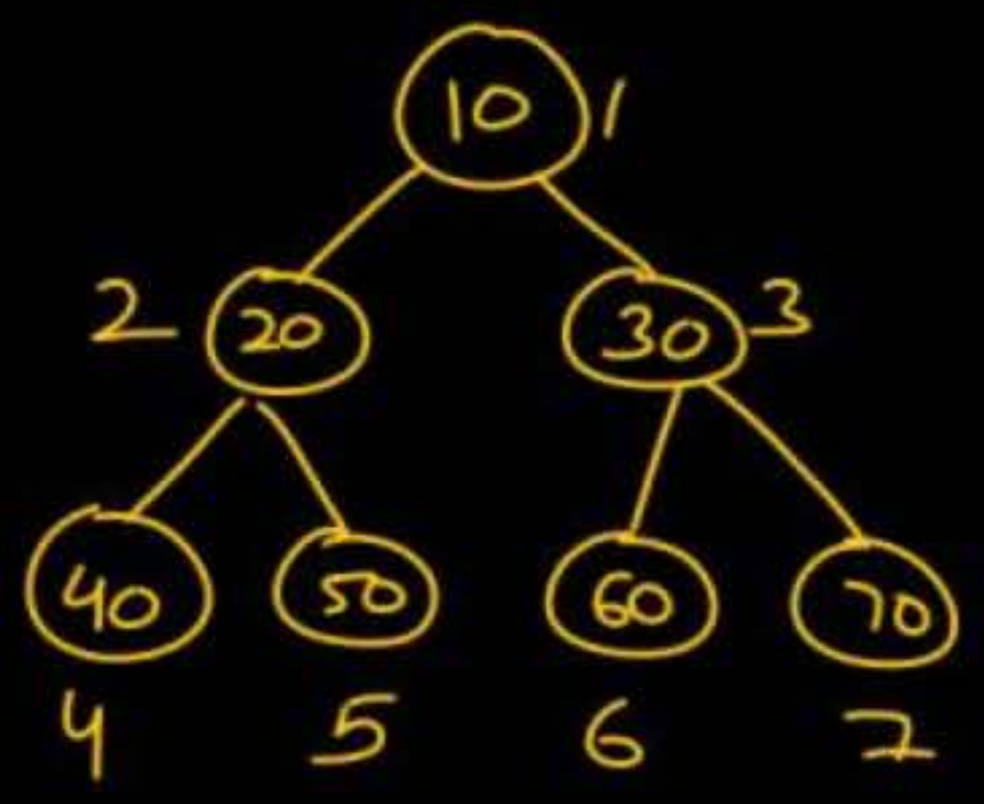
2) Build-Heap, Heapify-algo $\Rightarrow$?

for every internal node in reverse order

$\Rightarrow$ Heapify.

Given an array rep. a CBT

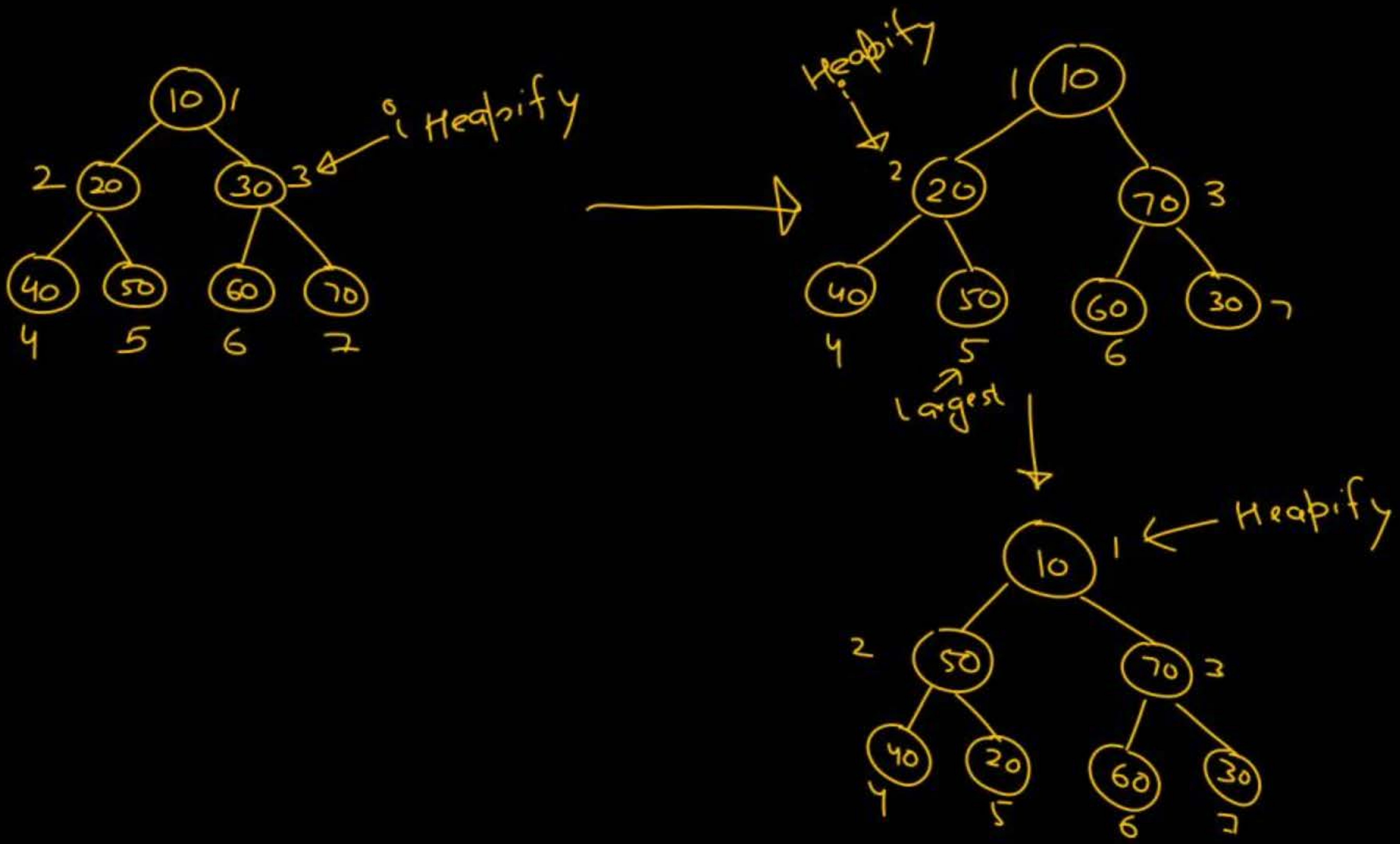
10, 20, 30, 40, 50, 60, 70 $\Longrightarrow$ Convert to max heap

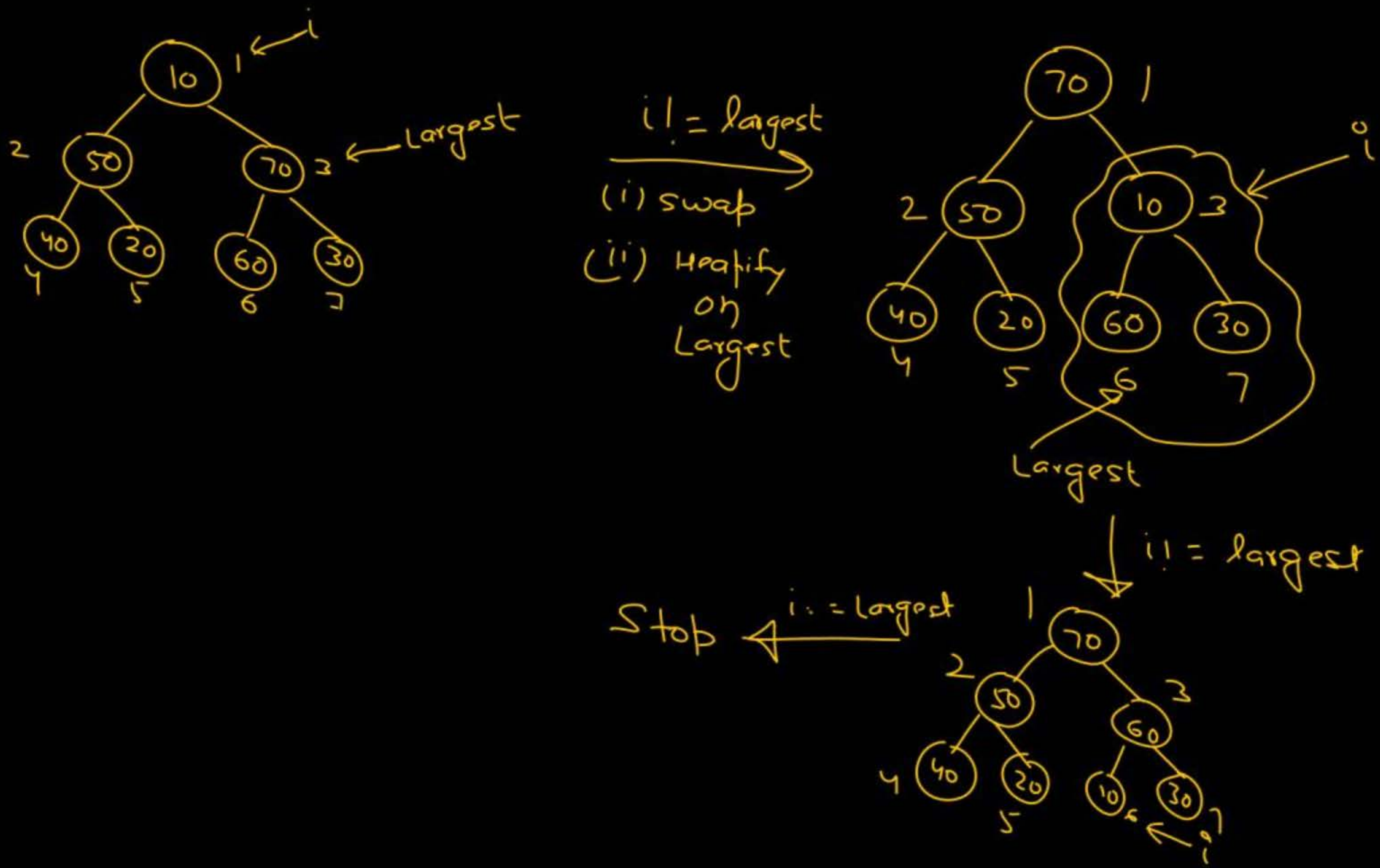


1 to $\lfloor \frac{n}{2} \rfloor \Rightarrow 1$ to 3

1, 2, 3

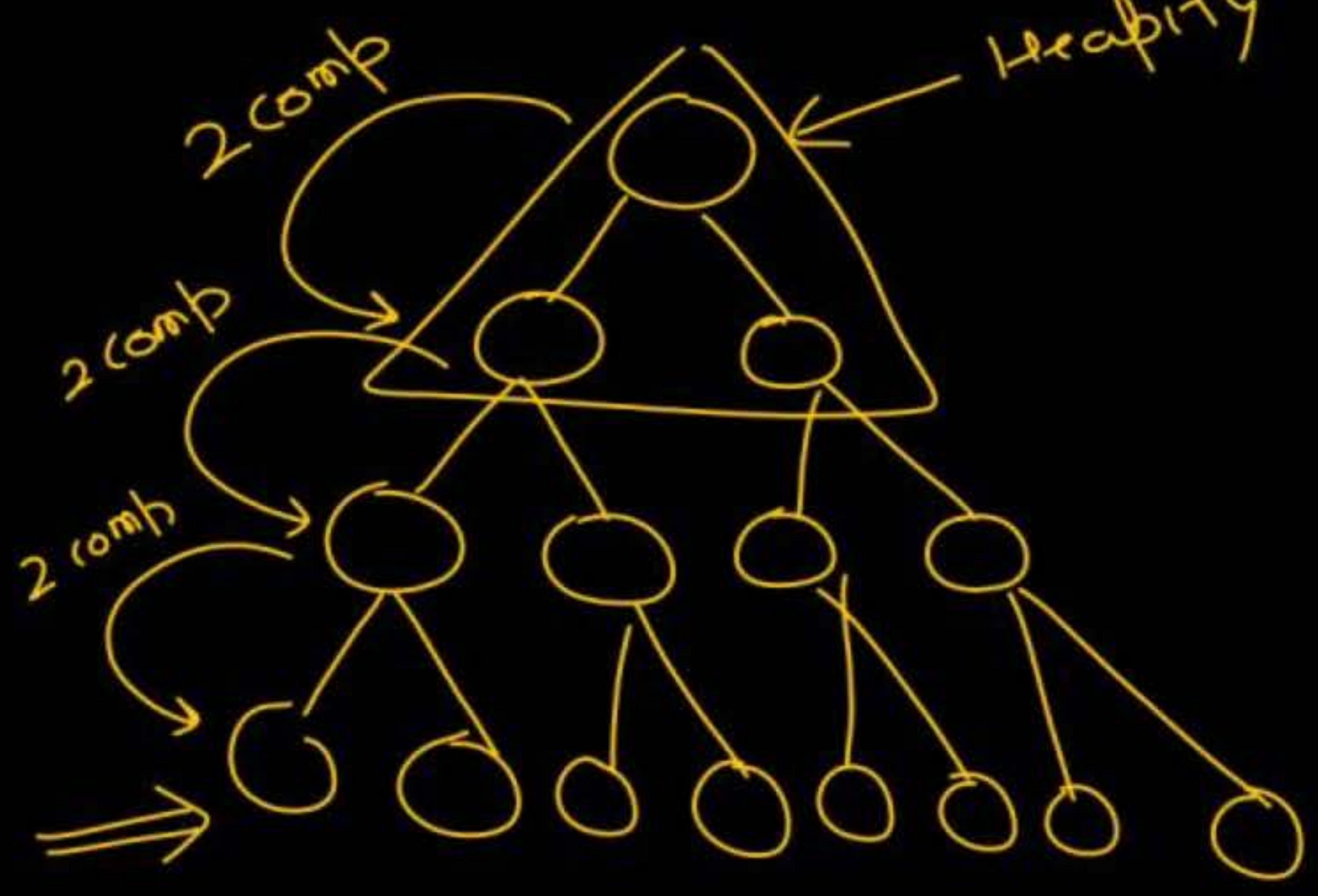
```
Build-Heap(A, n) {
    for ( i = ⌊n/2⌋ ; i >= 1 ; i-- )
        Heapify(A, i, n);
}
```


i Heapify
Heapify
Largest
Heapify



i
Largest
i != largest
(i) swap
(ii) Heapify on Largest
i
Largest
i != largest
Stop
i = largest
i

Build-Heap

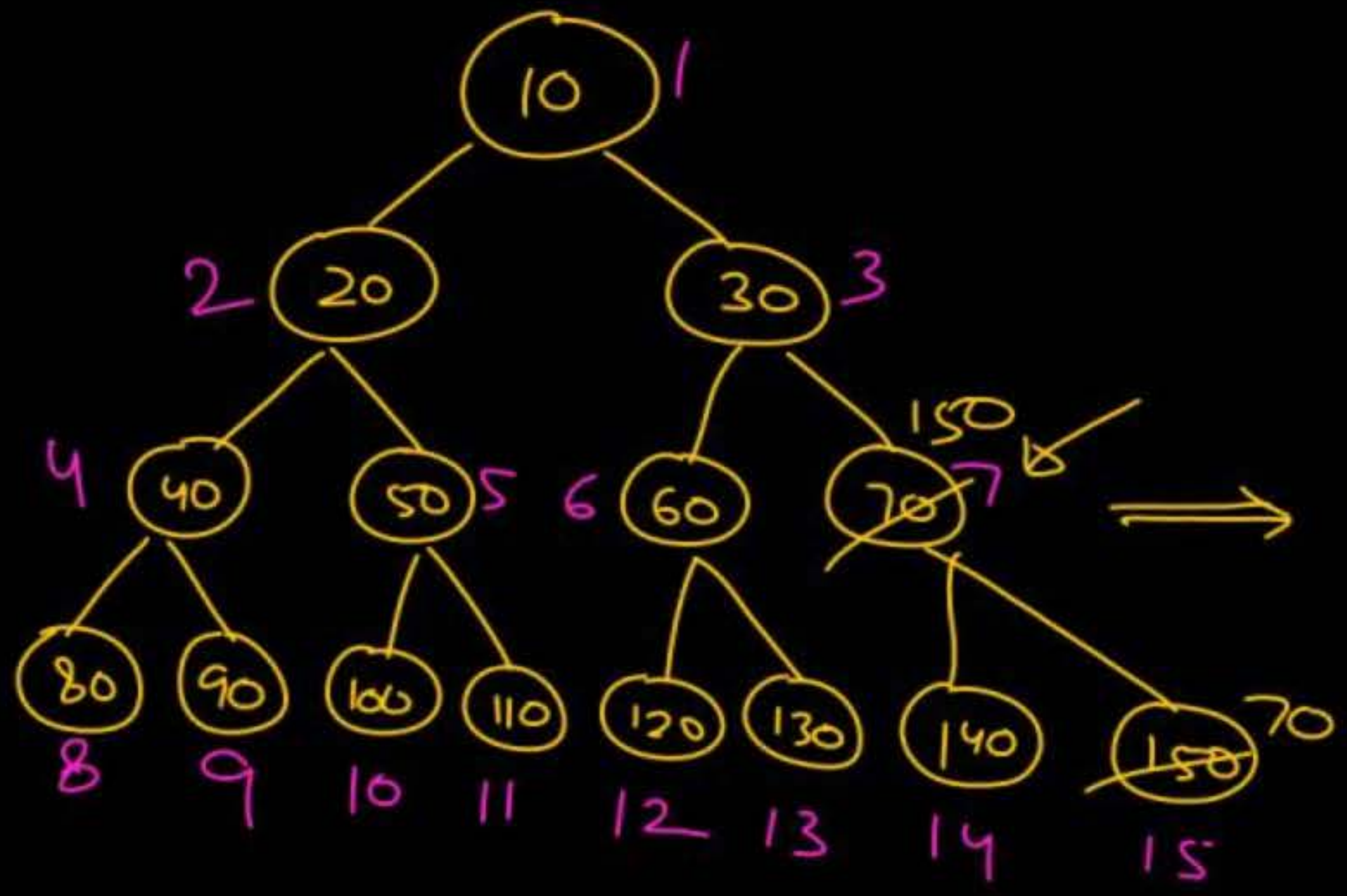


$$2 \times \log_2 n \Rightarrow O(\log_2 n)$$

~~Build-Heap~~

$$\cancel{\left\lceil \frac{n}{2} \right\rceil \times O(\log_2 n) = \frac{n}{2}\log_2 n}$$

$$\cancel{= n\log_2 n}$$

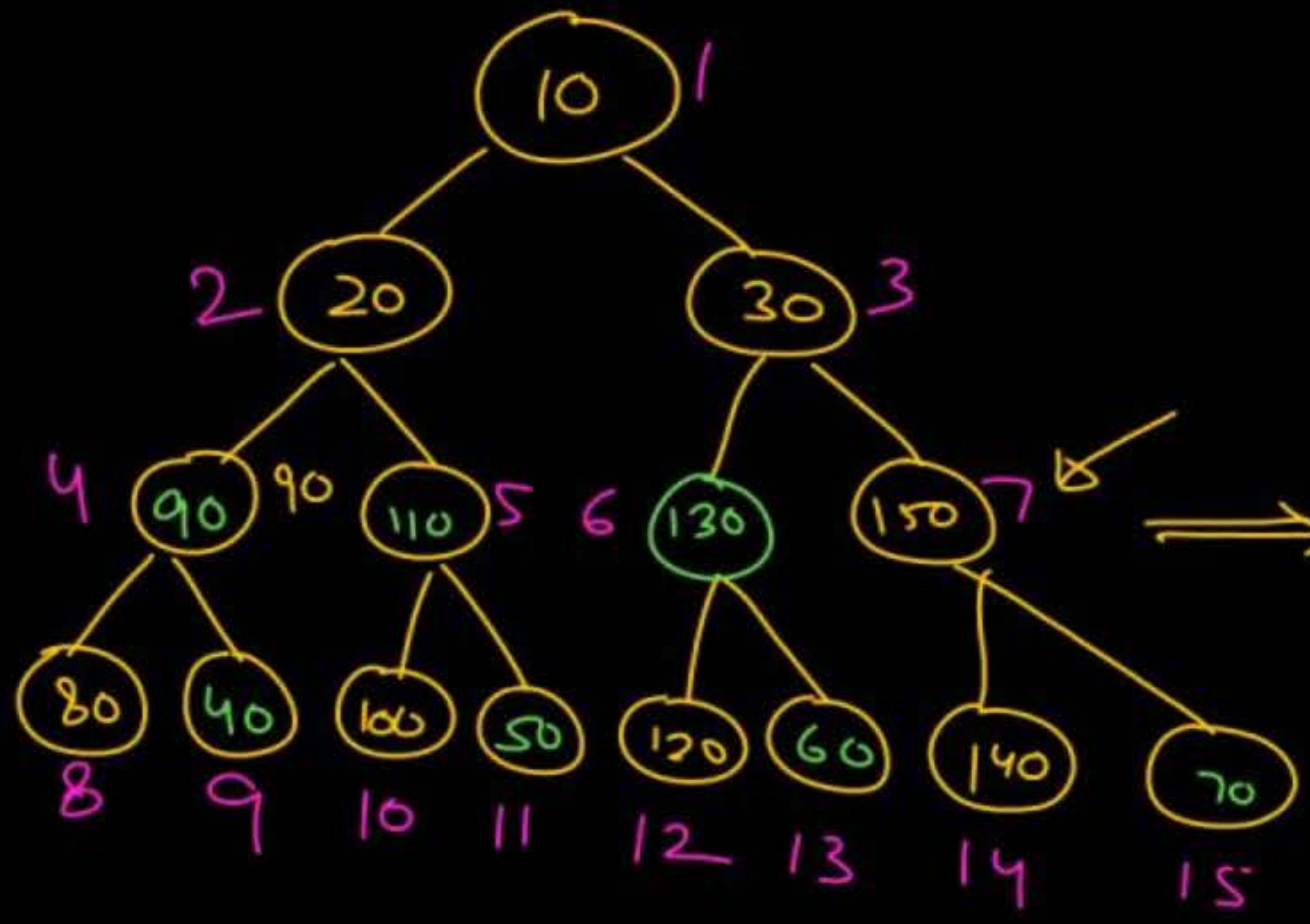


leaf

internal nodes

for $\Rightarrow$

$\Longrightarrow$ Every node at this level $\Rightarrow$ 2 compar

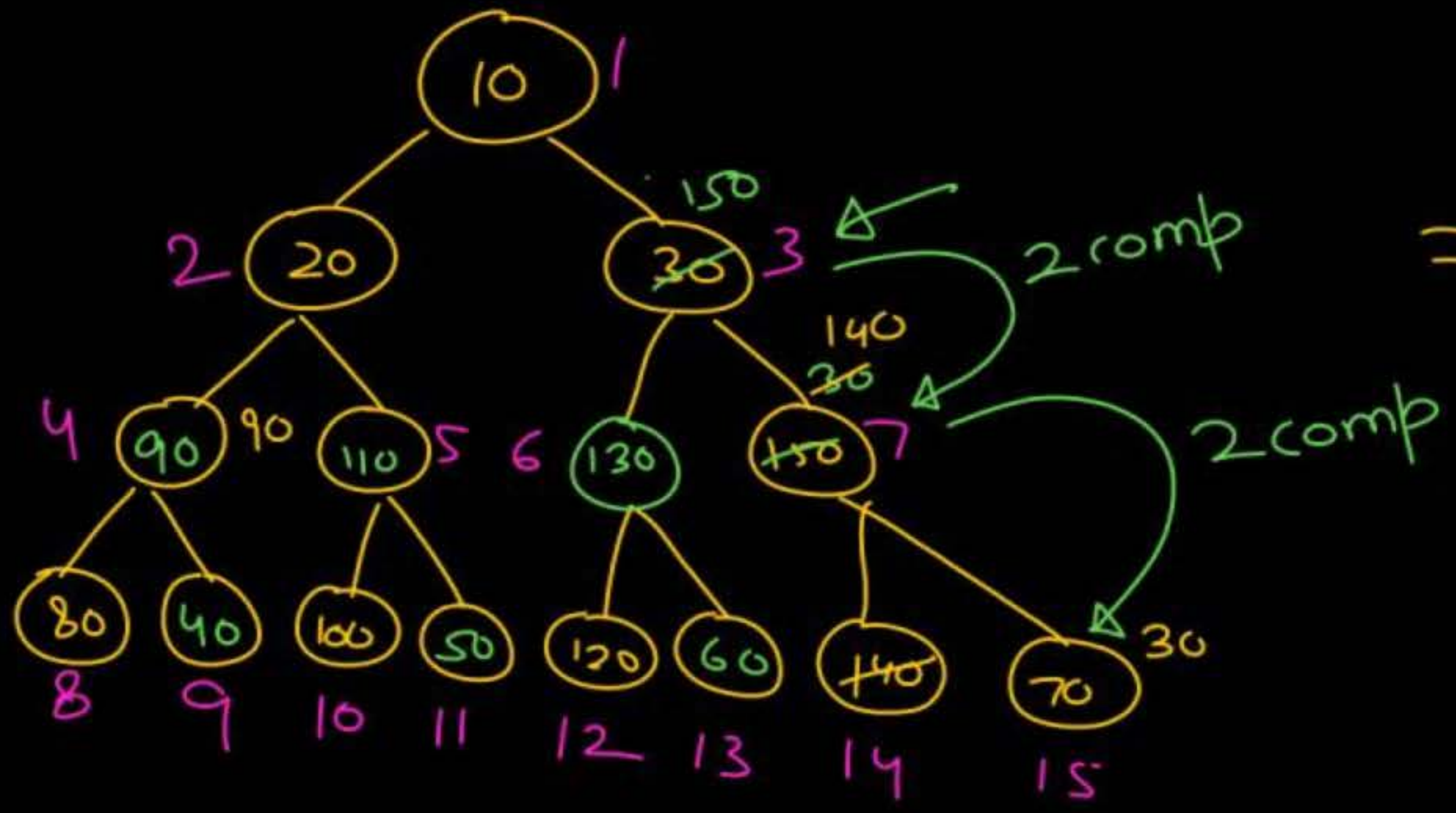


leaf

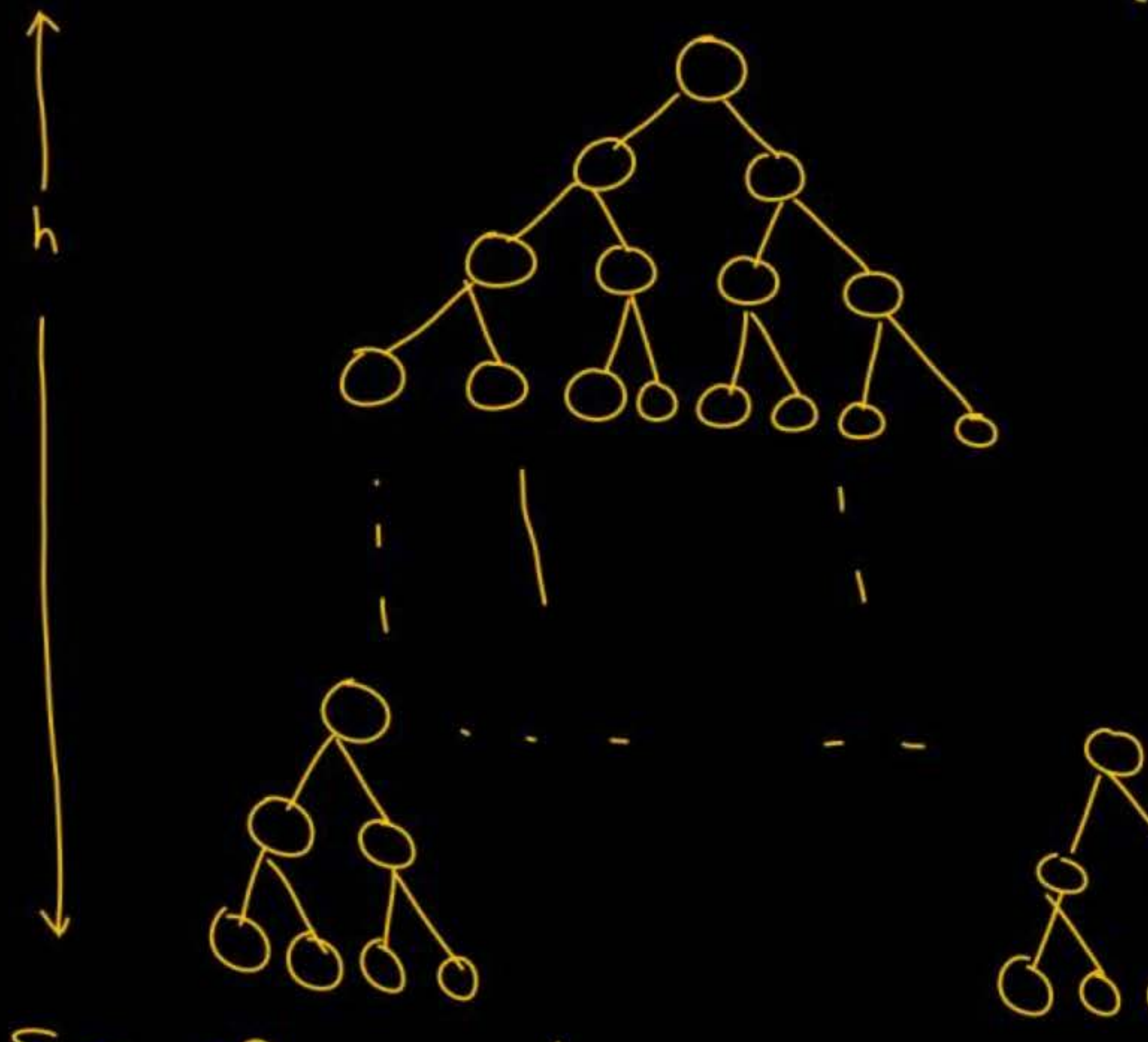
internal nodes

for $\Rightarrow$

Every node at this level $\Rightarrow$ 2 compar



$\Longrightarrow$ for Every node at this level = 2.2



level	# comp for each node	# nodes at this level
0	$2(h)$	2^0
1	$2(h-1)$	2^1
2	$2(h-2)$	2^2
3	$2(h-3)$	2^3
		$\vdots$
$h-2$	$2(2)$	2^{h-2}
$h-1$	$2\cdot 1$	2^{h-1}
h	0	2^h

$$S = 2^0 \times 2(h) + 2^1 \times 2(h-1) + 2^2 \times 2(h-2) + 2^3 \times 2(h-3) + \cdots + 2^{h-2} \times 2(2) + 2^{h-1} \times 2(1)$$

$$S = 2^0 \times 2(h) + 2^1 \times 2(h-1) + 2^2 \times 2(h-2) + 2^3 \times 2(h-3) + \cdots + 2^{h-2} \times 2(2) + 2^{h-1} \times 2(1)$$

$$S = 2\left[2^0(h) + 2^1(h-1) + 2^2(h-2) + 2^3(h-3) + \cdots + 2^{h-2}(2) + 2^{h-1}(1)\right]$$

$$\frac{S}{2} = 2^0(h) + 2^1(h-1) + 2^2(h-2) + 2^3(h-3) + \cdots + 2^{h-2}(2) + 2^{h-1}(1)$$

$$-S = -2^1(h) \mp 2^2(h-1) \mp 2^3(h-2) \mp \cdots \mp 2^{h-1}(2) \mp 2^h(1)$$

$$-\frac{S}{2} = 2^0(h) + 2^1(h-1-h) + 2^2(h-2-h+1) + 2^3(h-3-h+2) + \cdots + 2^{h-1}(1-2) - 2^h$$

$$-\frac{S}{2} = 2^0(h) + 2^1(h-1-h) + 2^2(h-2-h+1) + 2^3(h-3-h+2) + \cdots + 2^{h-1}(1-2) - 2^h$$

$$-\frac{S}{2} = h - 2^1 - 2^2 - 2^3 \cdots - 2^h$$

$$-\frac{S}{2} = h - (2^1 + 2^2 + \cdots 2^h)$$

$$-\frac{S}{2} = h - \frac{2^1(2^h-1)}{2-1} \qquad \frac{a(r^n-1)}{r-1}$$

$$-\frac{S}{2} = h - (2^{h+1}-2)$$

$$\frac{S}{2} = 2^{h+1} - 2 - h$$

$$\boxed{\frac{S}{2} = 2^{h+1} - 2 - h}$$

$$\boxed{n_{max} = 2^{h+1} - 1}$$

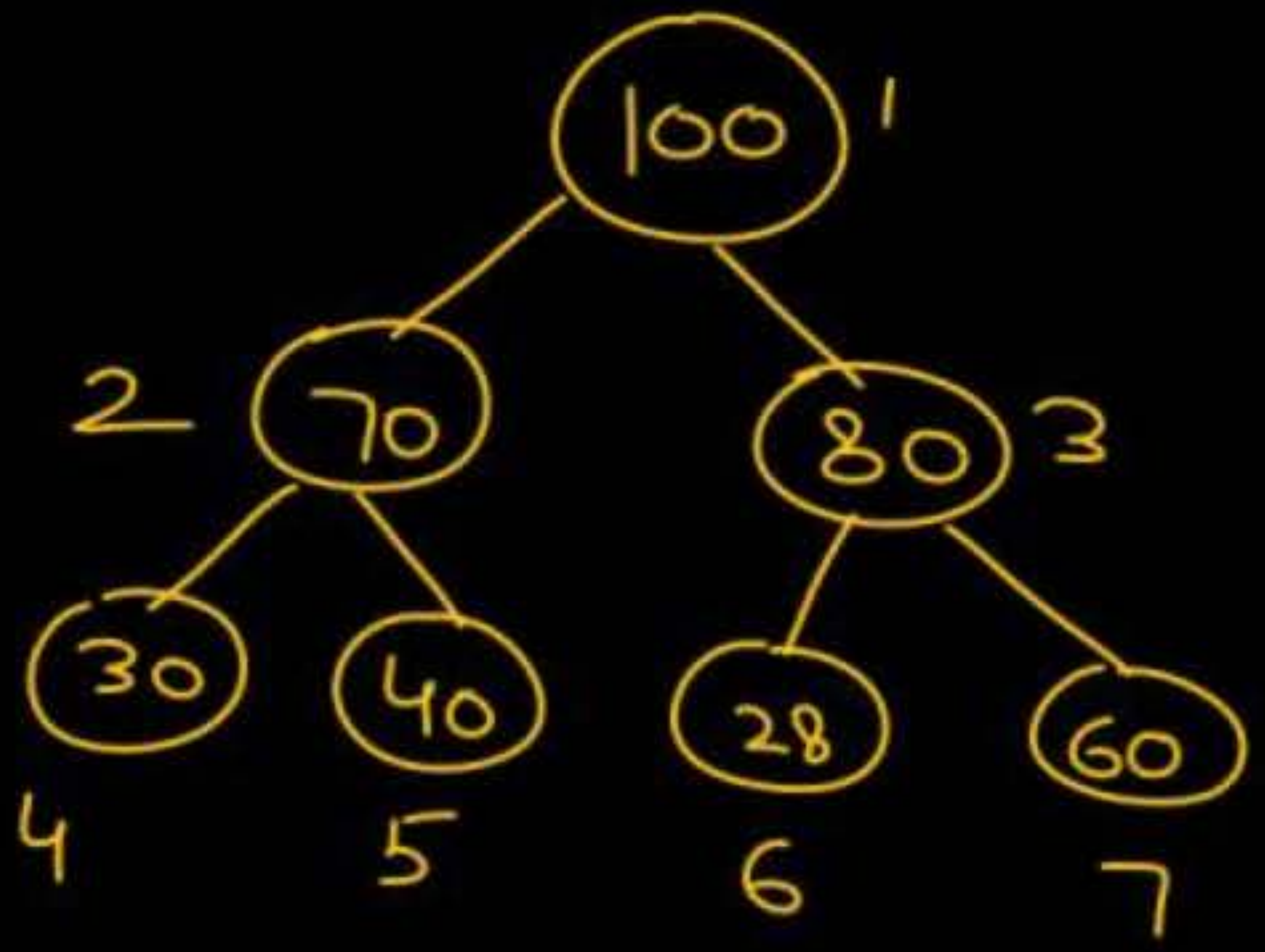
$$\frac{S}{2} = n+1-2-\log_2 n$$

$$\frac{S}{2} = n-1-\log_2 n$$

$$S = 2n - 2 - 2\log_2 n$$

$$S = O(n)$$

Max-Heap

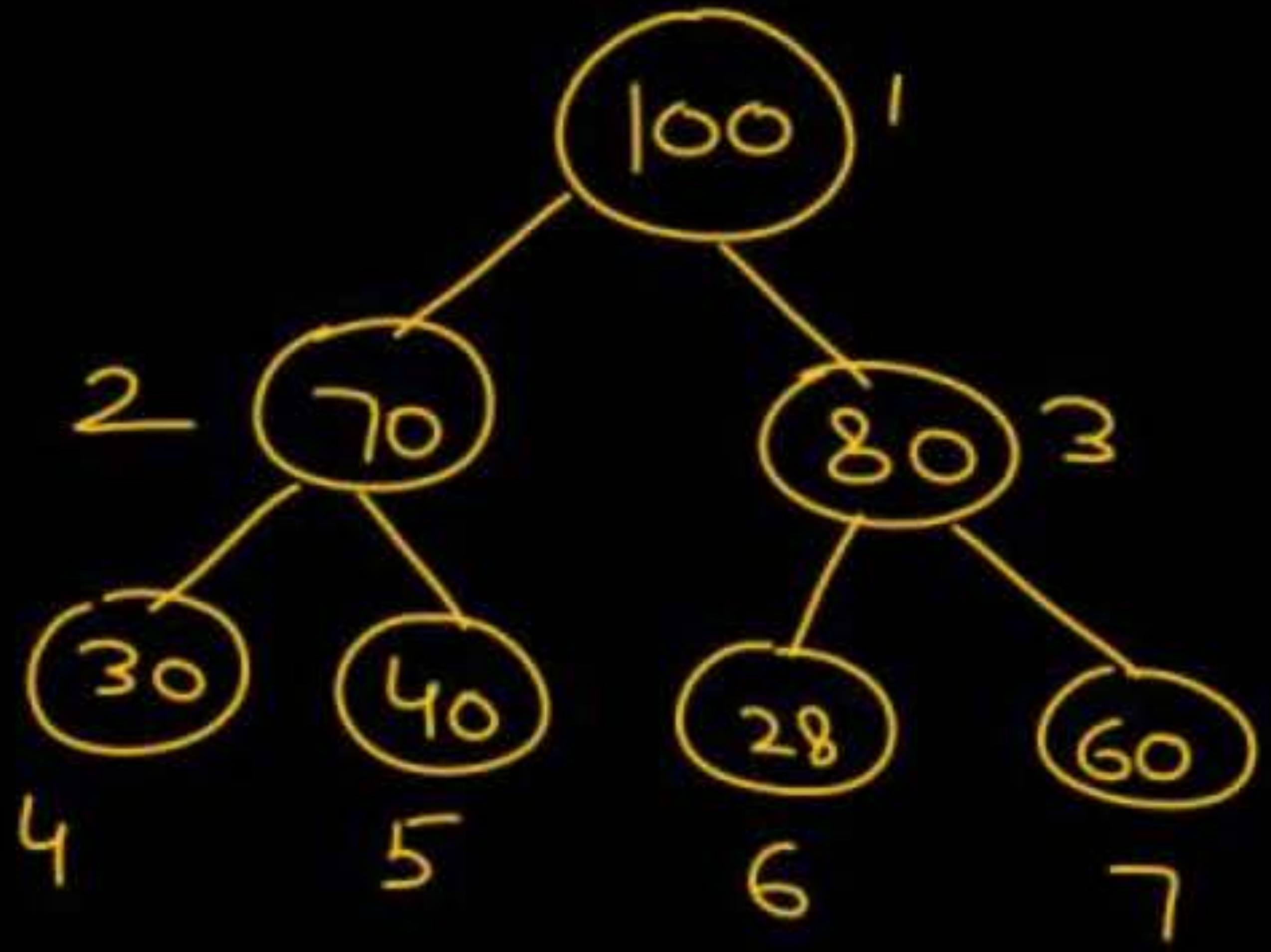


1	2	3	4	5	6	7
100	70	80	30	40	28	60

return A[1]

O(1), constant

Max-Heap

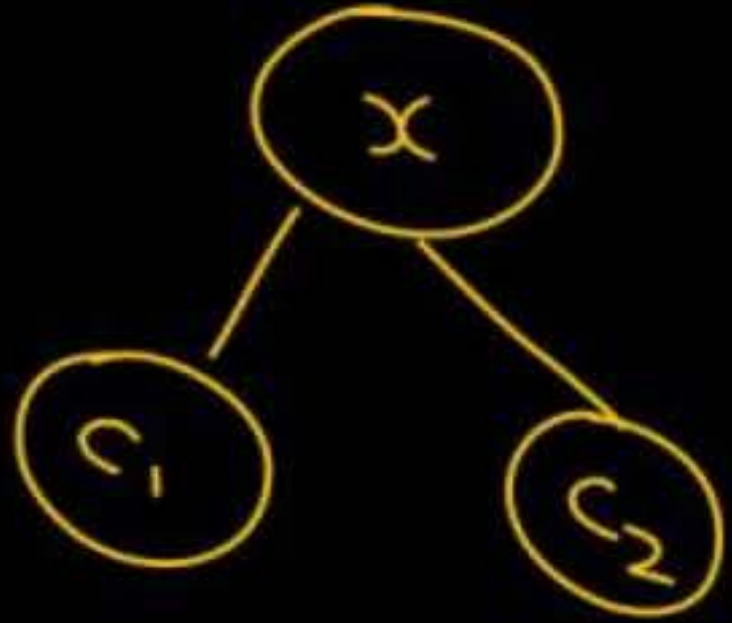


1	2	3	4	5	6	7
100	70	80	30	40	28	60

Find_min → can be some leaf node

No. of leaf node $= \lceil \frac{n}{2} \rceil = O(n)$

$\boxed{x > c_1, c_2}$



Minimum can be c_1 or c_2

Minimum among these 3

comp.

10 element $\Rightarrow$ 9

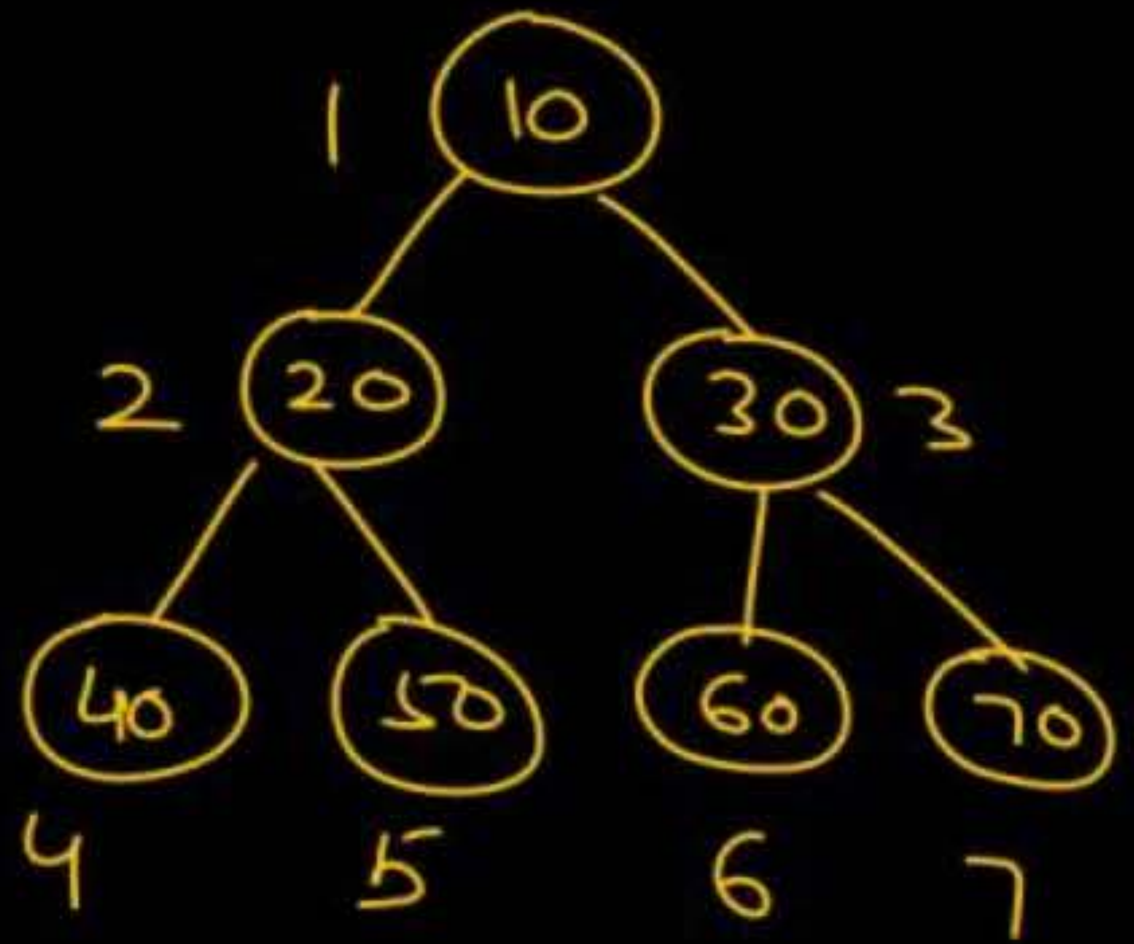
n ele $\Rightarrow$ $n-1$

$\lceil \frac{n}{2} \rceil$ elem $\Rightarrow$ $\lceil \frac{n}{2} \rceil - 1 \Rightarrow O(n)$

Max-heap

$$\boxed{\begin{array}{l} \text{Find\_Min} \Rightarrow O(n) \\ \text{Find\_Max} \Rightarrow O(1) \end{array}}$$

Min-heap



	1	2	3	4	5	6	7
A	10	20	30	40	50	60	70

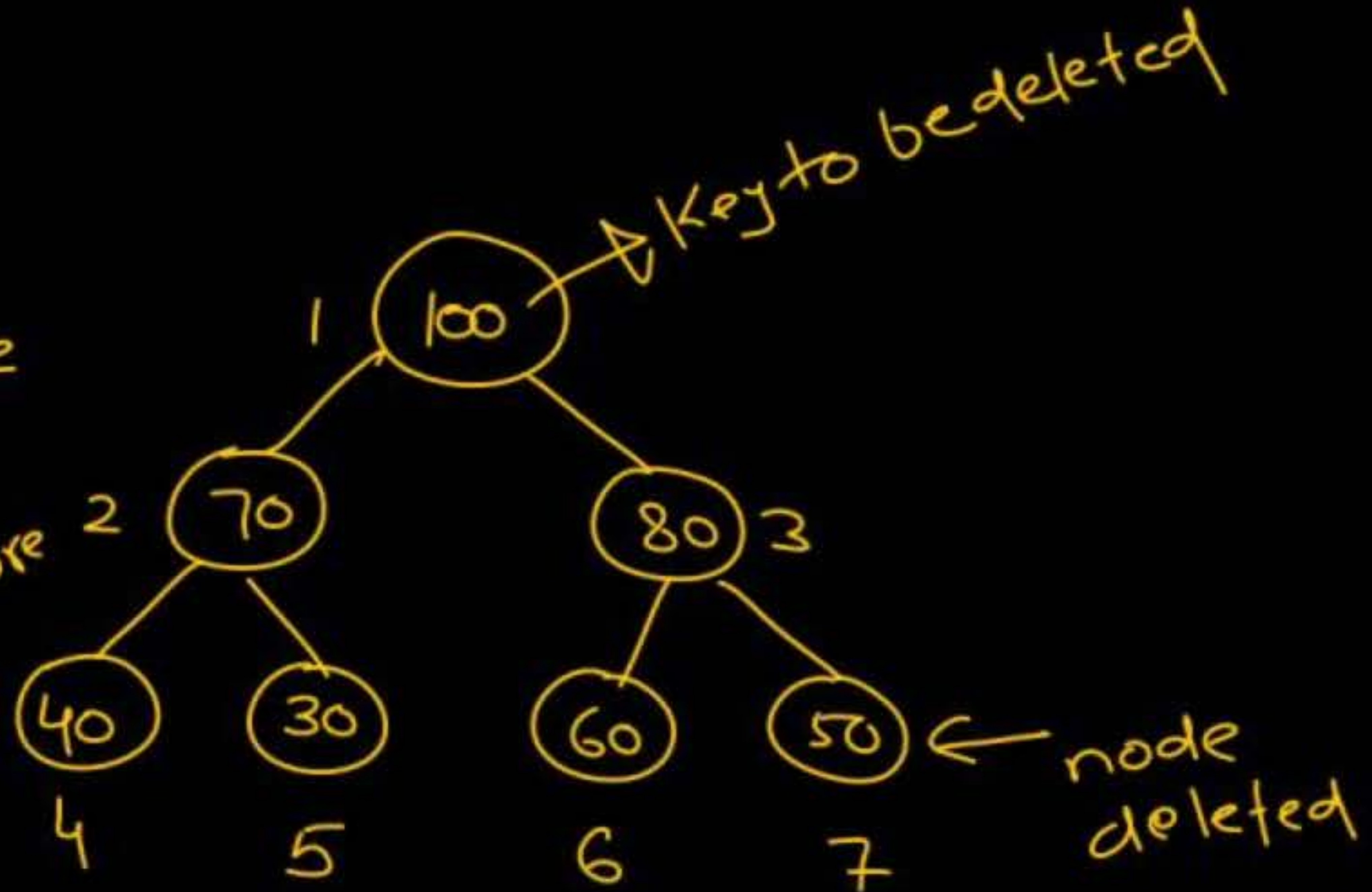
Find-Min $\Rightarrow$ return $A[1] \Rightarrow O(1)$

Find-Max $\Rightarrow O(n)$

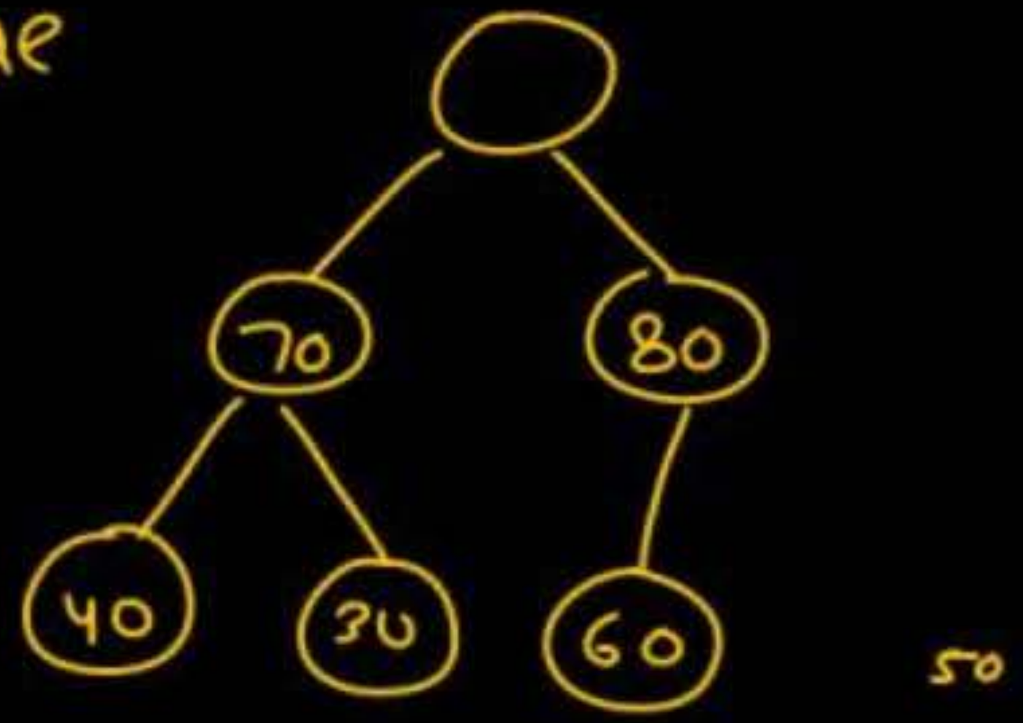
Search in heap : worst case : $O(n)$

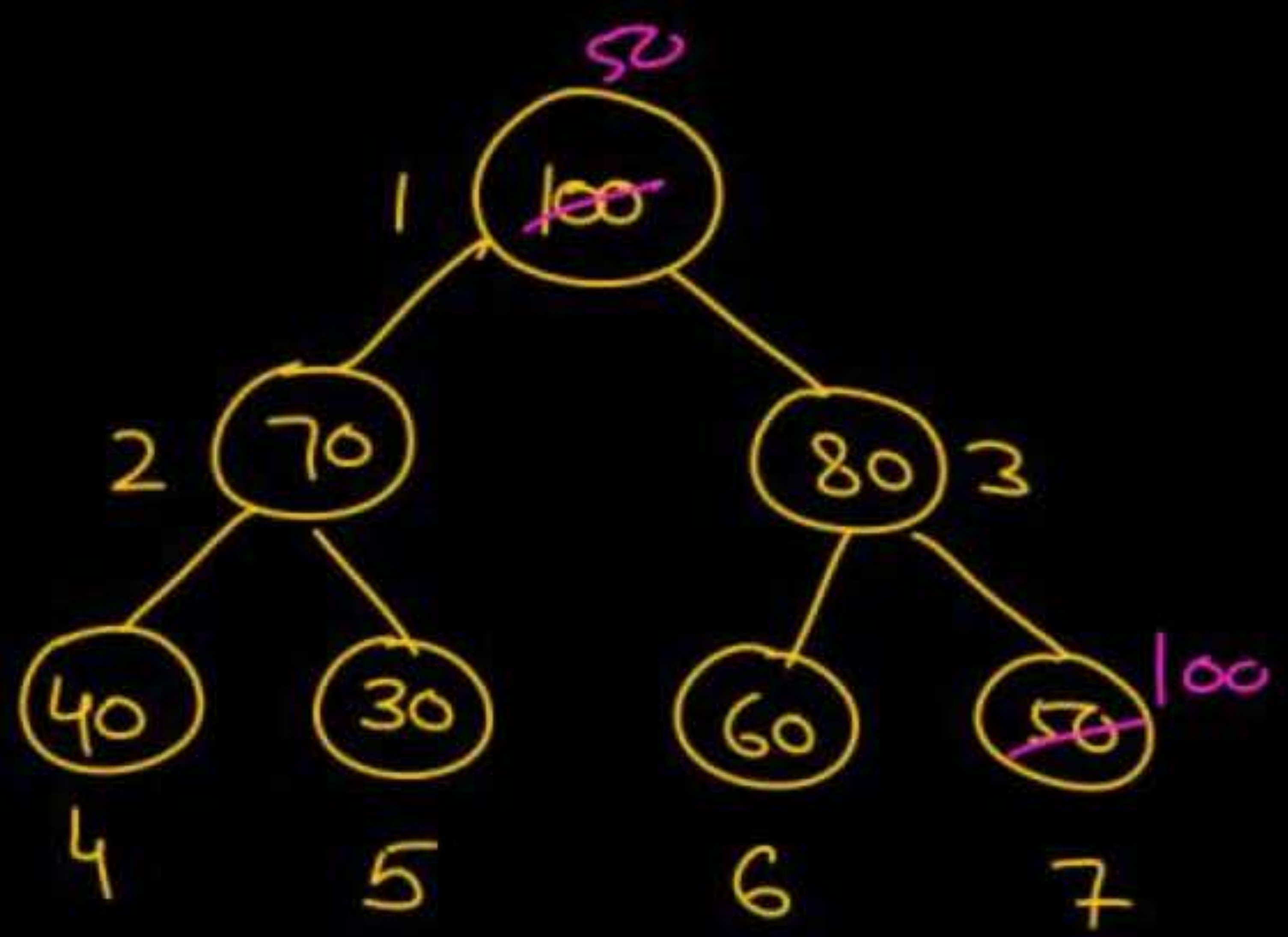
Insert : $O(\log_2 n)$

7 node
का
structure



6 node





$n = 7$

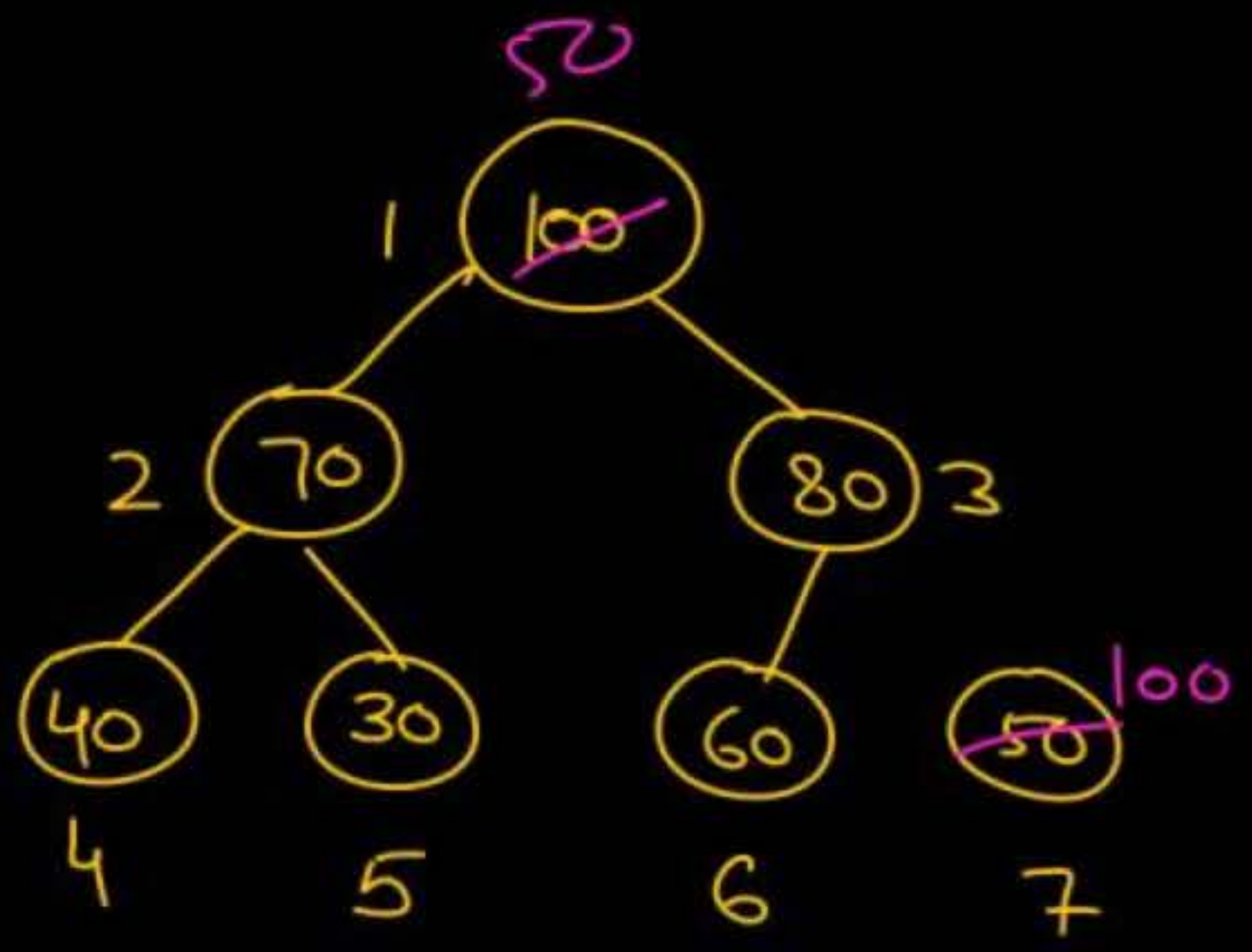
$$A[1] \longleftrightarrow A[n]$$

$$\boxed{n = n-1}$$

100	70	80	40	30	60	50
1	2	3	4	5	6	7

$n = 6$

50	70	80	40	30	60	100
1	2	3	4	5	6	7



$n=7$

$$A[1] \longleftrightarrow A[n]$$

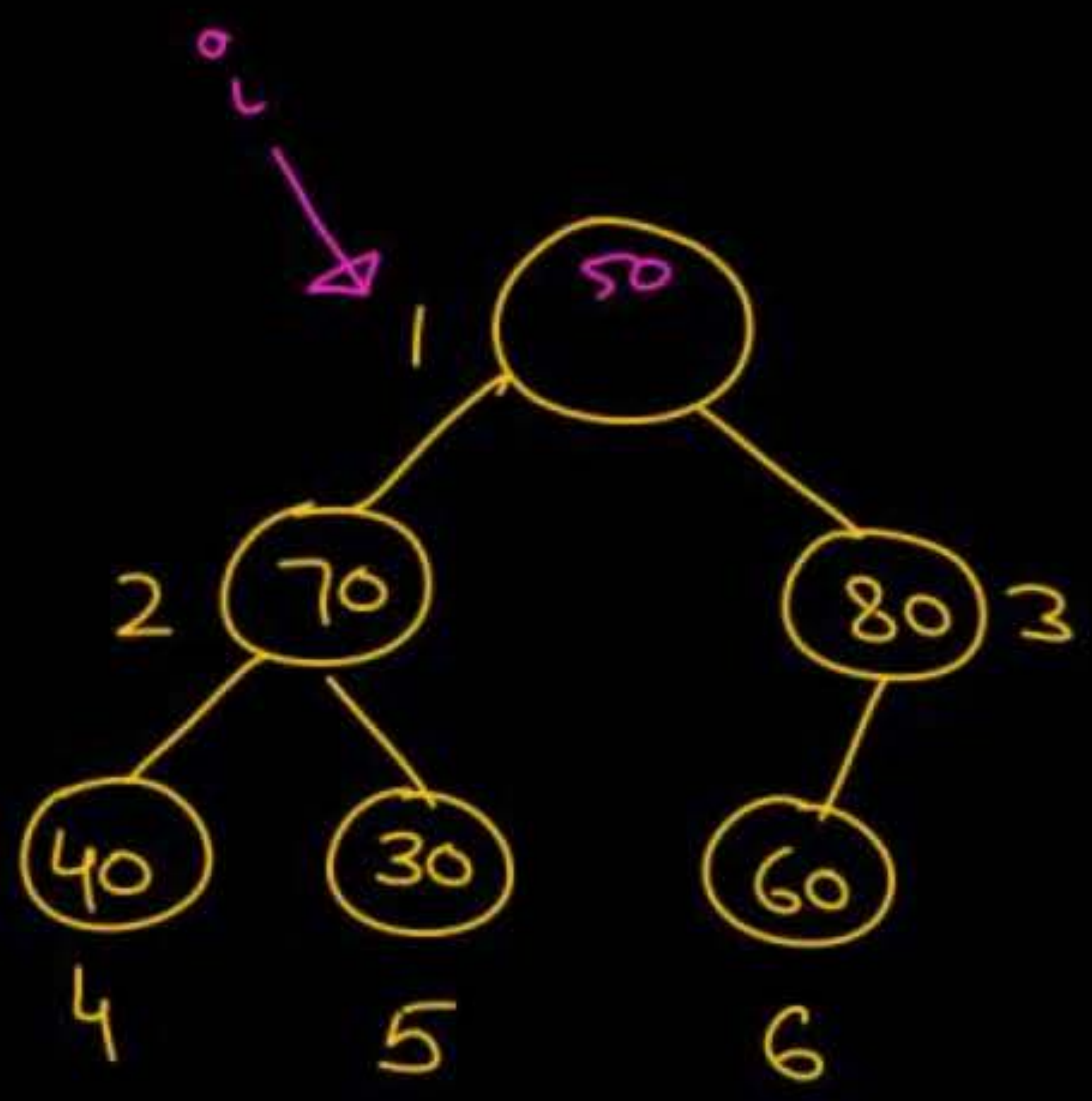
$$\boxed{n = n-1}$$

100	70	80	40	30	60	50
1	2	3	4	5	6	7

$n=6$

50	70	80	40	30	60	100
1	2	3	4	5	6	7

$n = 6$



$A[1] \longleftrightarrow A[n]$

नीचे (leaf) $\longrightarrow$ root

small $\longrightarrow$ root] Heapify

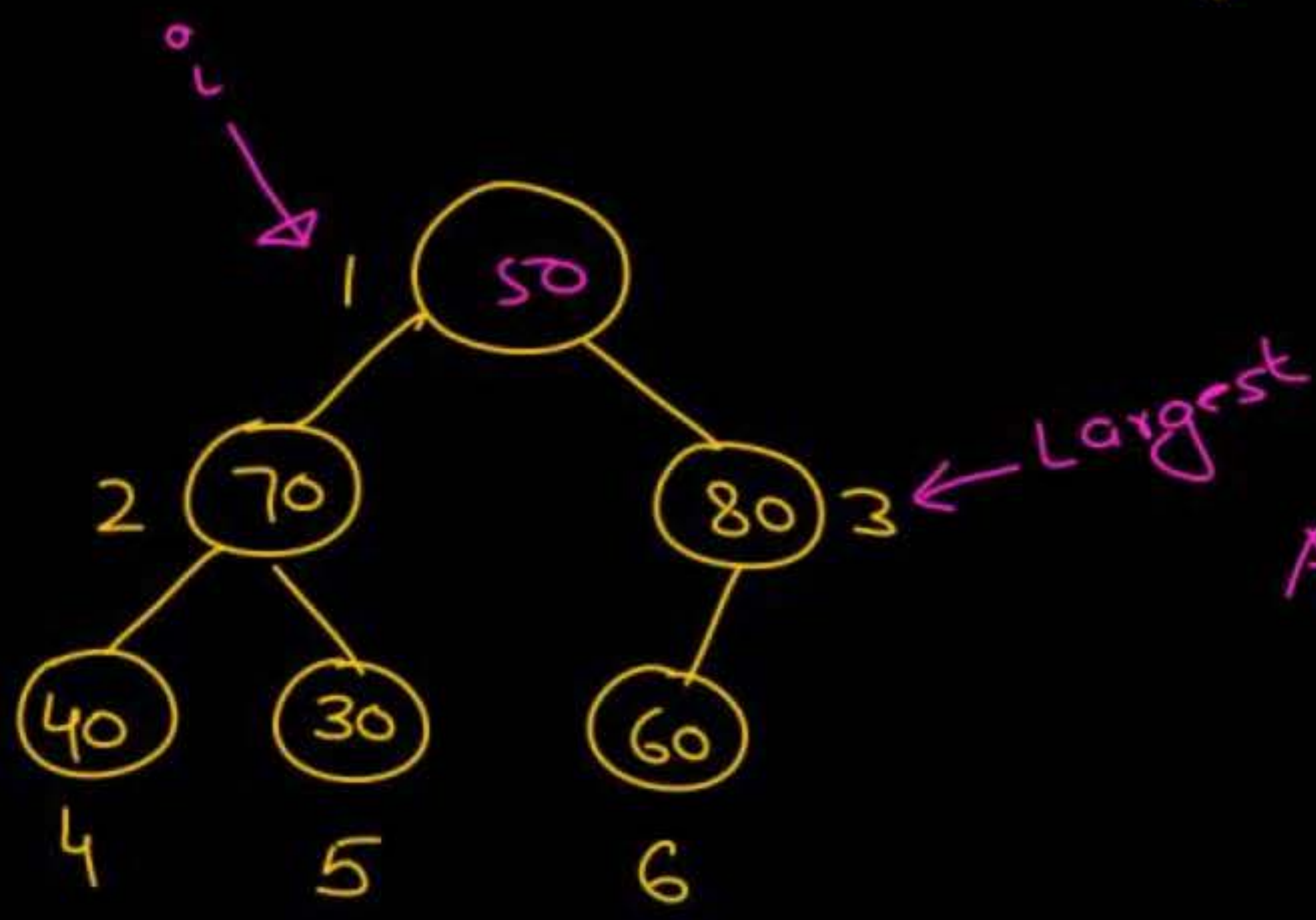
$A[1] \Longleftrightarrow A[n]$

$n = n-1$

Heapify(A, 1, n)

50	70	80	40	30	60	100
1	2	3	4	5	6	7

$n = 6$



$A[1] \leftrightarrow A[n]$

छोटी (leaf) $\rightarrow$ root

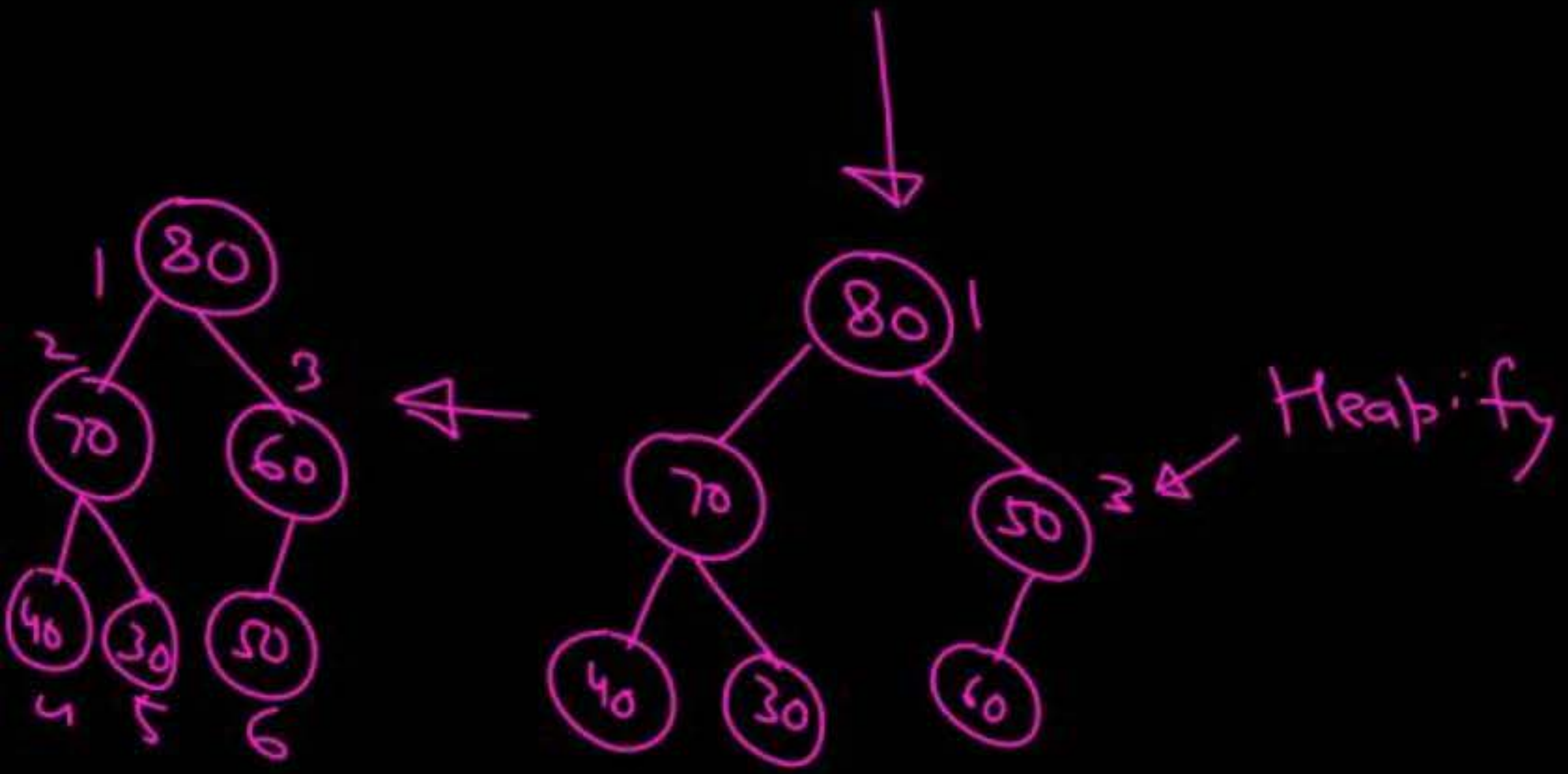
small $\rightarrow$ root } Heapify

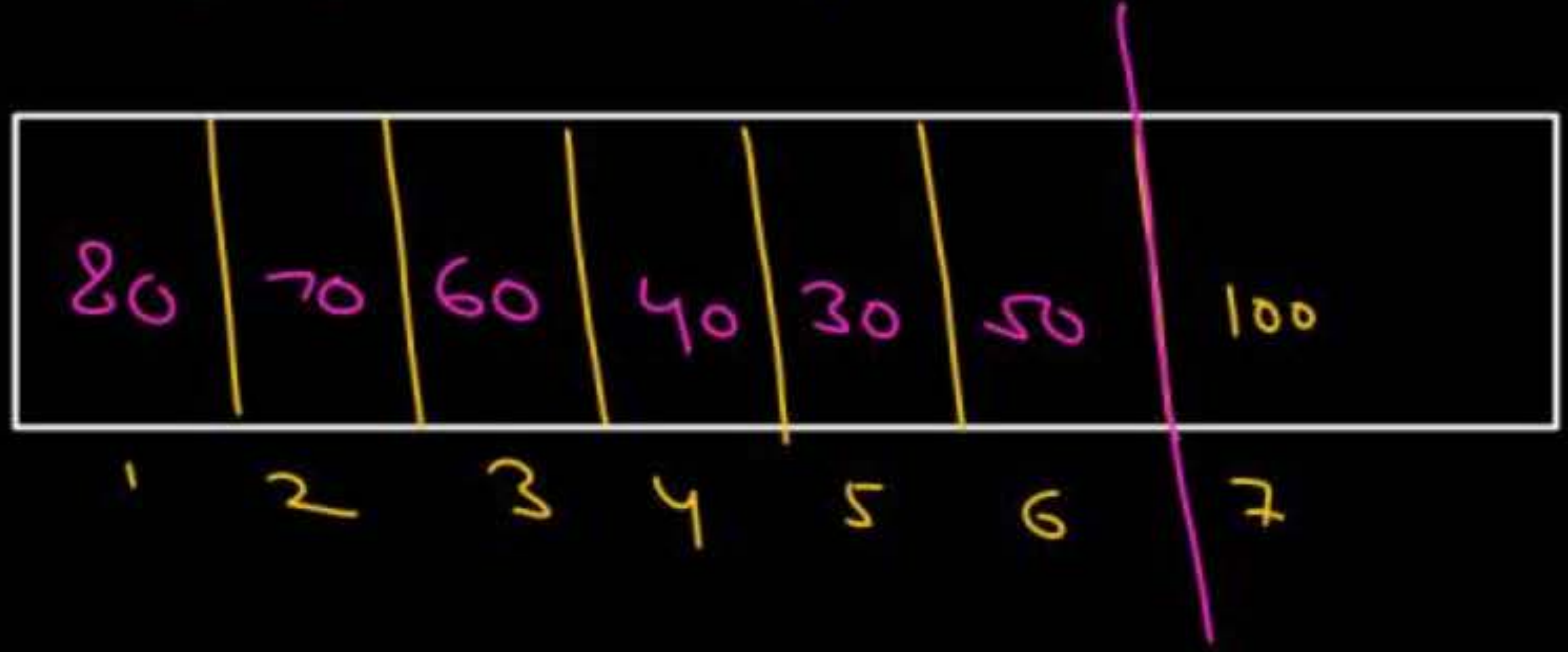
const.

$A[1] \leftrightarrow A[n] \rightarrow$ const.

$n = n-1 \rightarrow$ const.

$\text{Heapify}(A, 1, n) \rightarrow \log_2 n$





Max-Heap

Deletion : $O(\log_2 n)$

Insertion : $O(\log_2 n)$

Search : $O(n)$

Find_Max : $O(1)$

Find_Min : $O(n)$

Extract_Max : $O(\log_2 n)$

Find Max

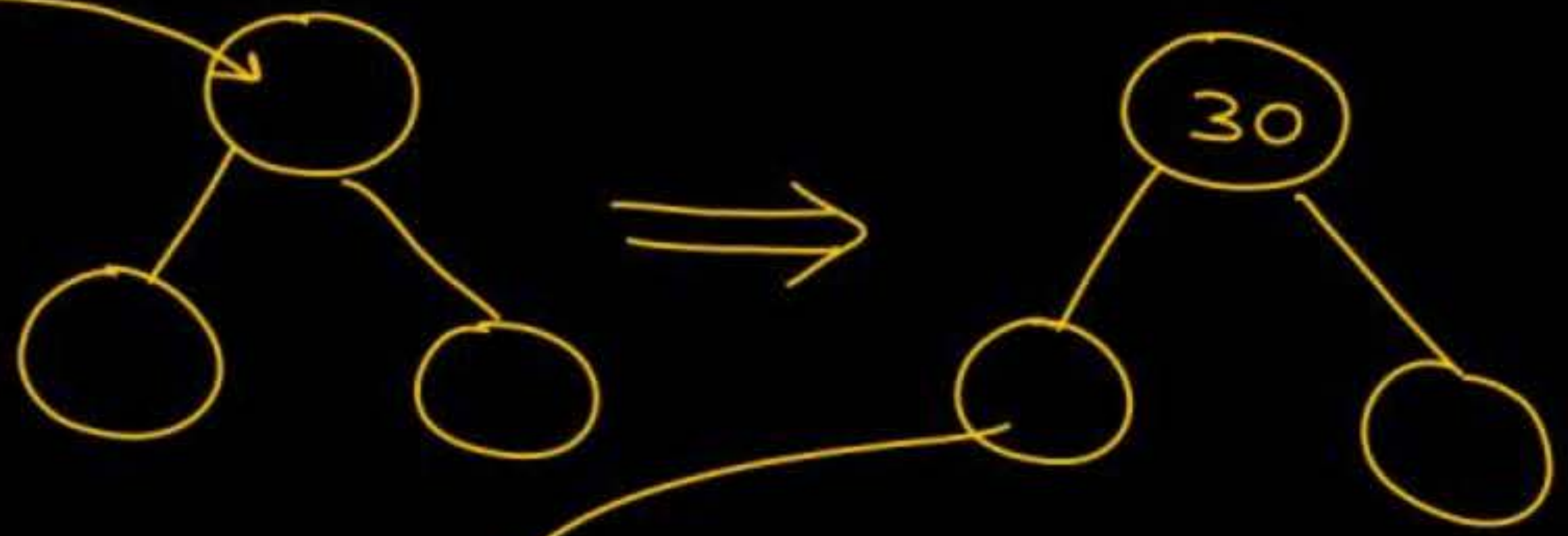
vs

Extract_Max $\rightarrow O(\log_2 n)$

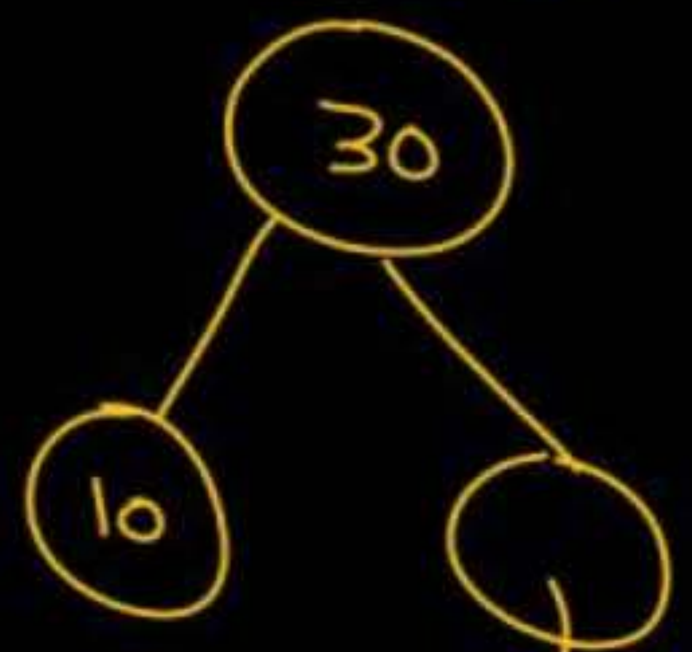
Q. How many max-heap can be possible with 3 distinct keys 10, 20, 30?

10, 20, 30

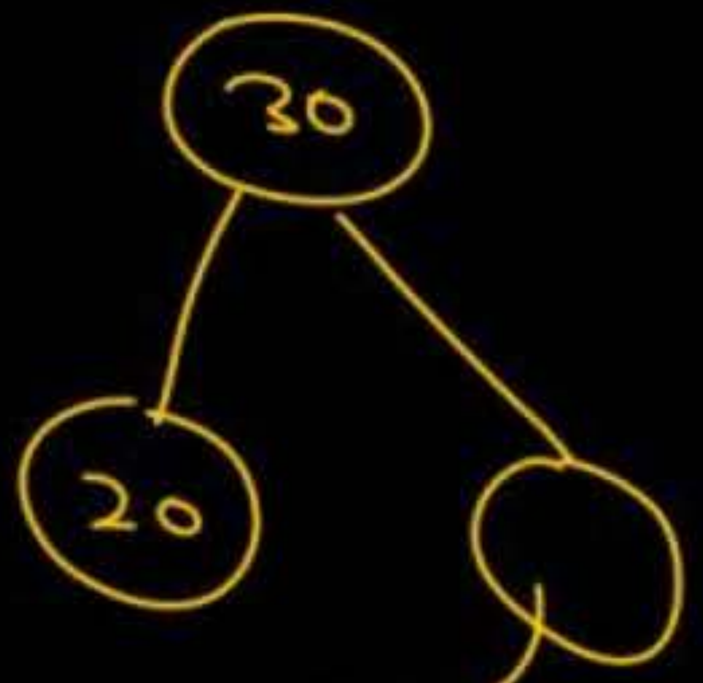
Maximum



1st way



2nd way

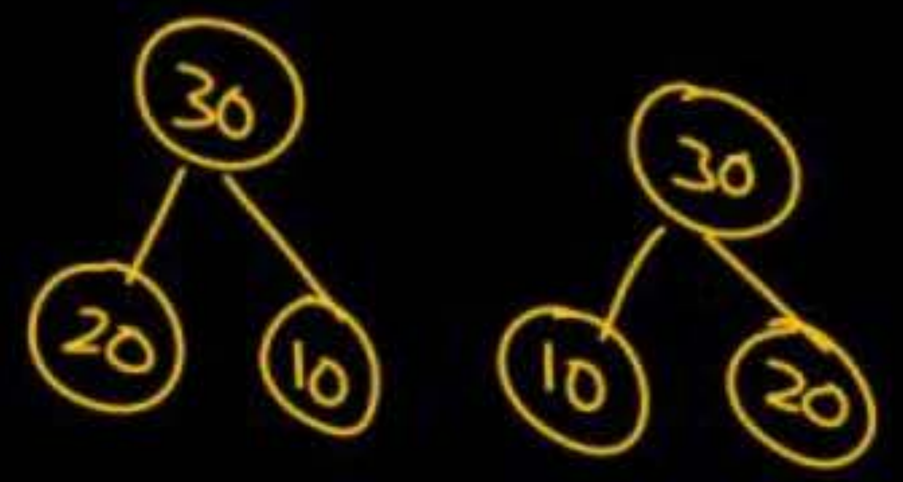


1 choice

rem. keys = 10, 20

कोई एक (Any one)

$^2C_1$



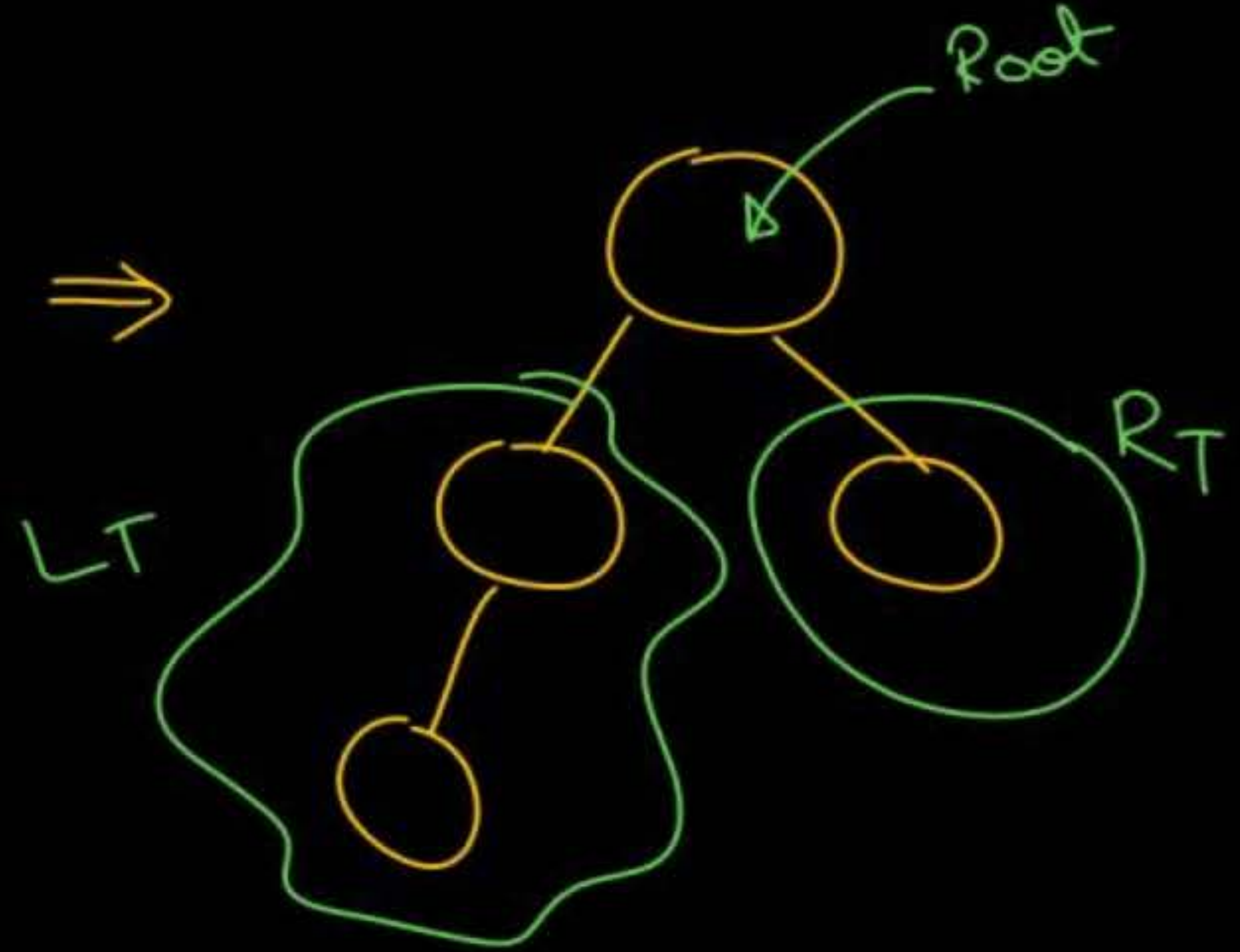
# of Max-heap

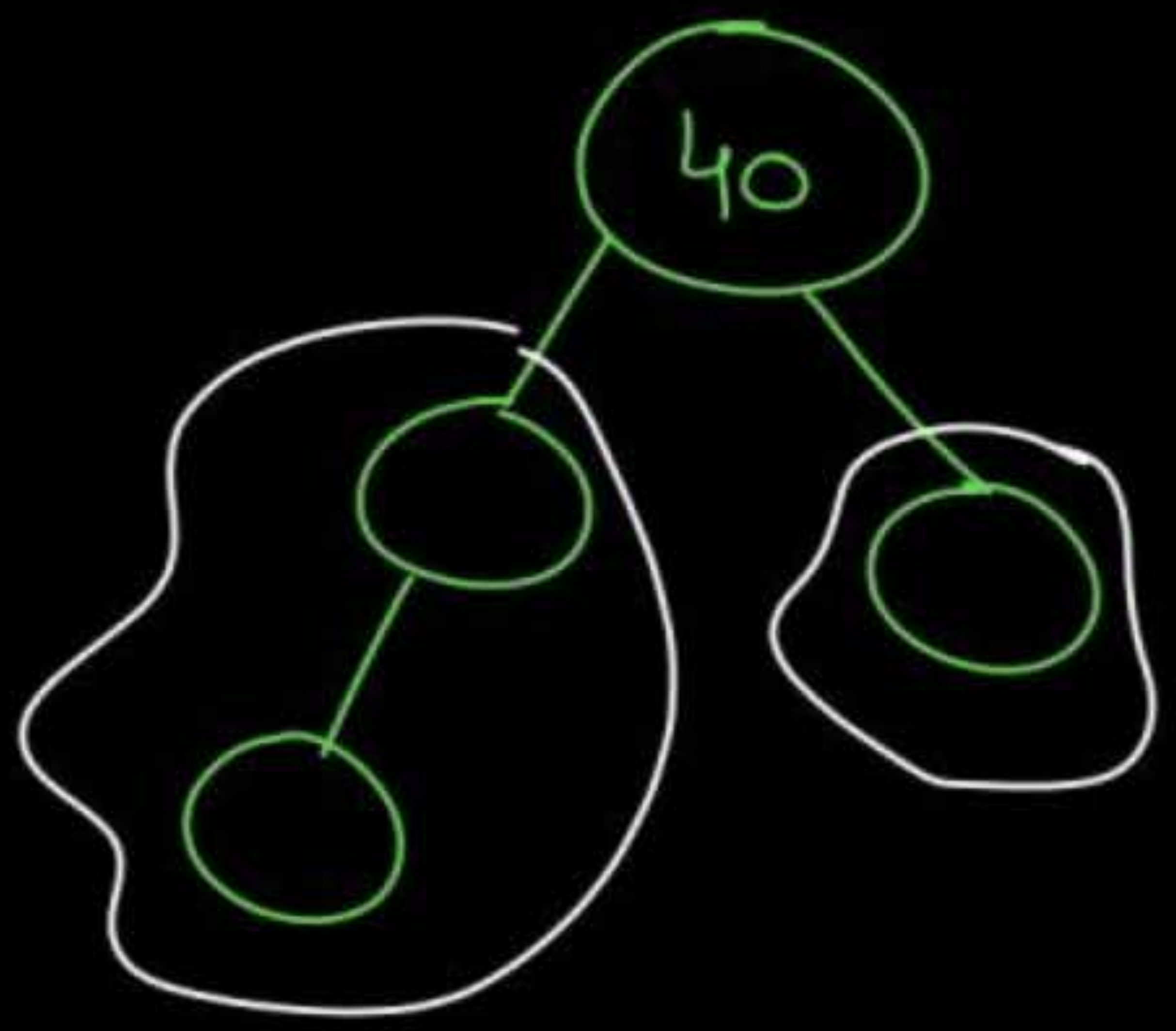
$n=3$ 2

$n=4$, 10,20,30,40

choice for Root $\Rightarrow$ 1 (maximum)

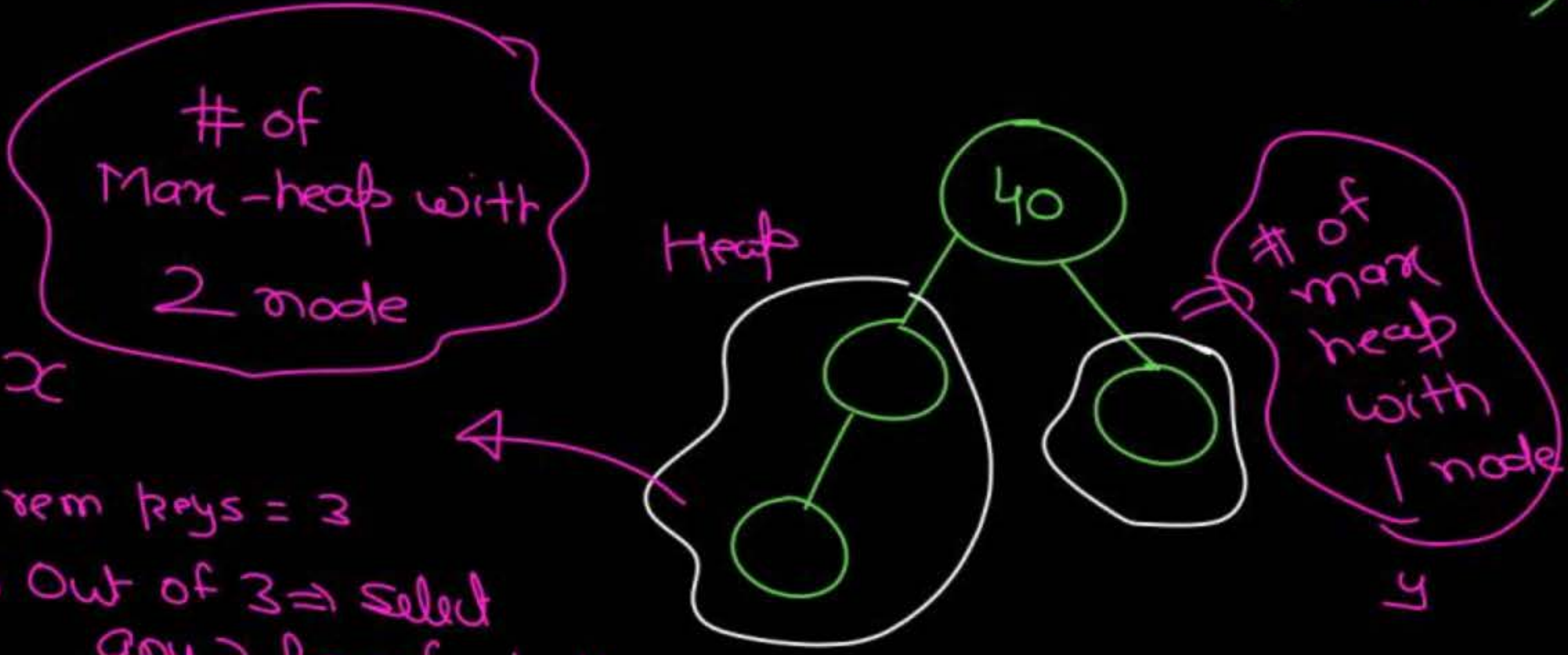
4 node structure $\Rightarrow$





$$^3C_2 \times \left(\begin{array}{c} \text{\# No. of max heap} \\ \text{with 2 dist. keys} \end{array} \right) \times \left(\begin{array}{c} \text{\# No. of max heaps with} \\ \text{1 keys} \end{array} \right)$$

choice for Root ⟹ 1 (maximum)



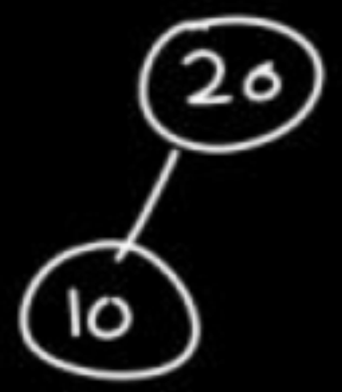
$x \times y$

$^3C_2 \times x \times y$

rem keys = 3

(i) Out of 3 ⟹ select any 2 keys for Left sub-tree

$n = 2$ 10, 20 $\Rightarrow$ 1

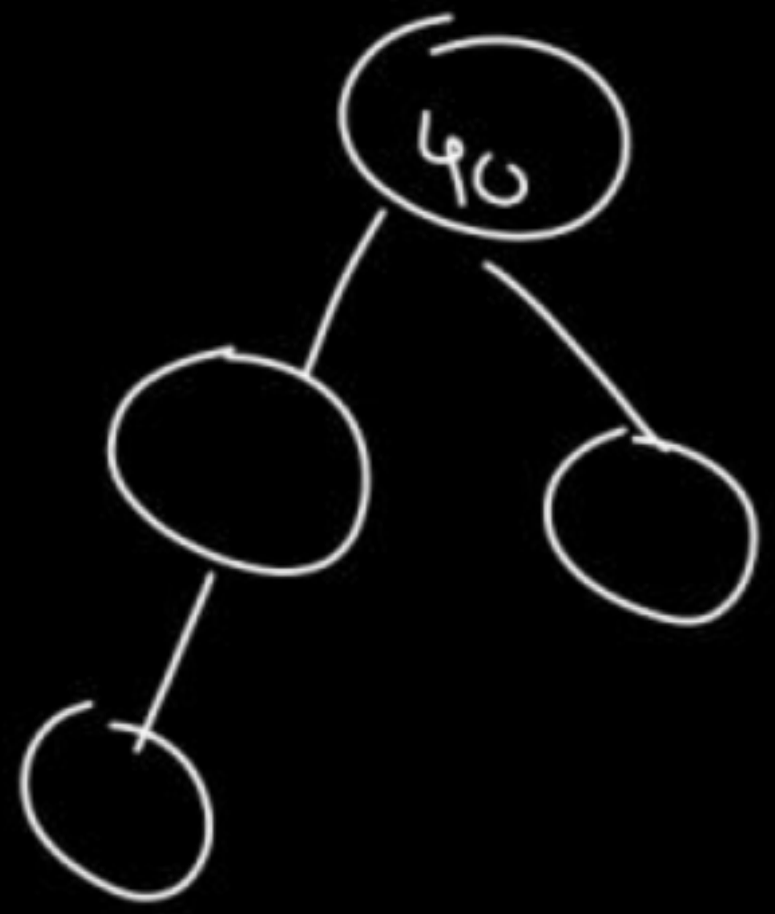


$n = 3$ 10, 20, 30

$n = 4$ 10, 20, 30, 40

(i) Root $\Rightarrow$ 40

(ii) L_T (2 keys) $\Rightarrow$



Rem keys : 10, 20, 30

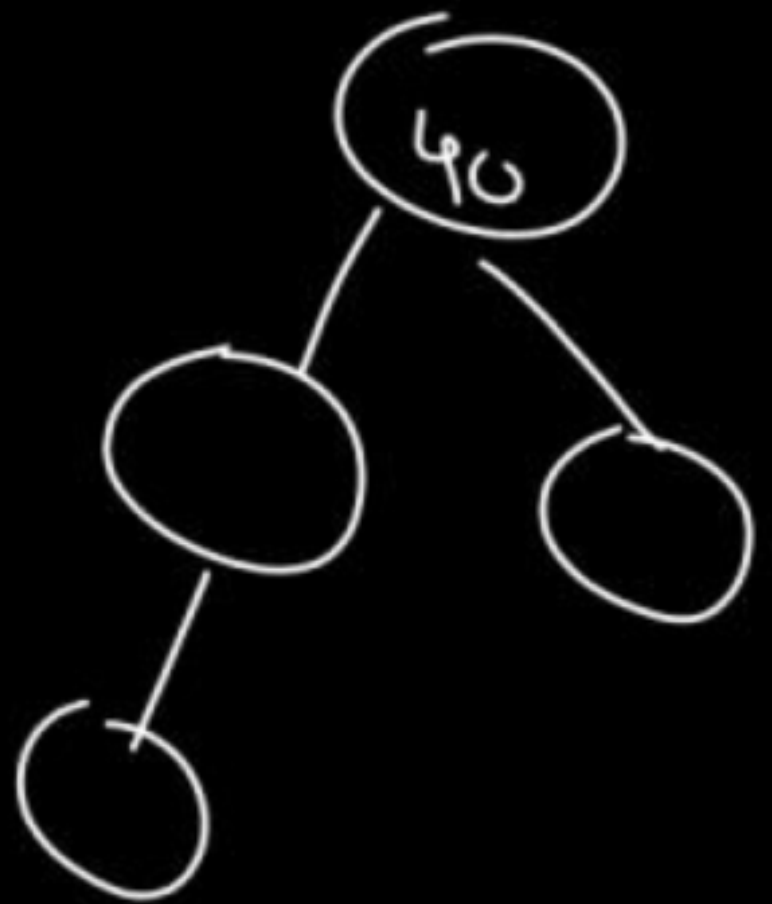
out of 3 $\Rightarrow$ select any 2

$^3C_2 = 3$

$n=4$ 10, 20, 30, 40

(i) Root $\Rightarrow$ 40

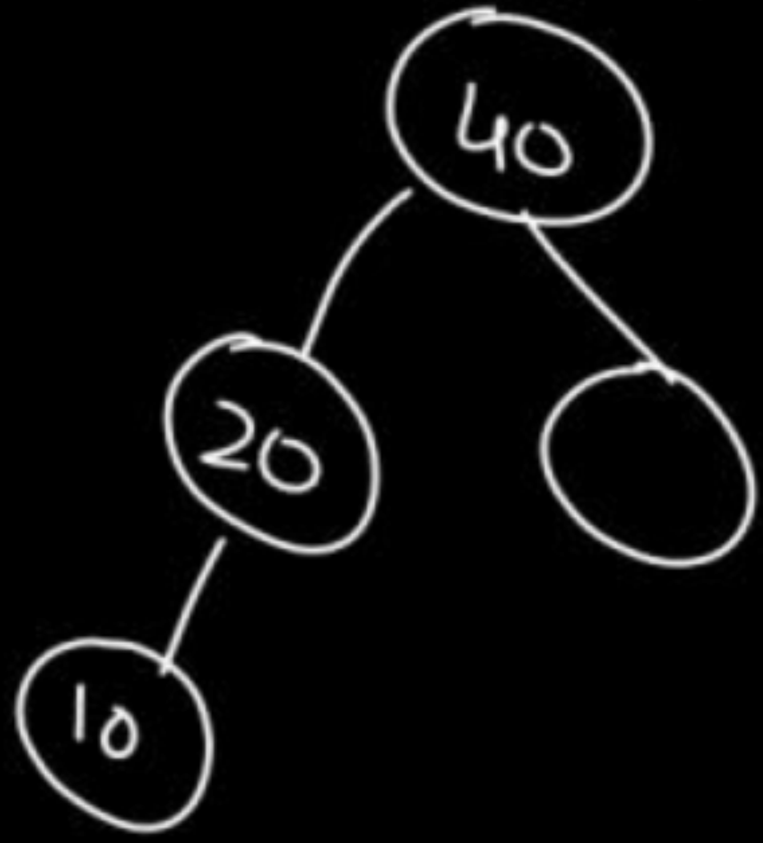
(ii) L_T (2 keys) $\Rightarrow$ Rem keys : 10, 20, 30

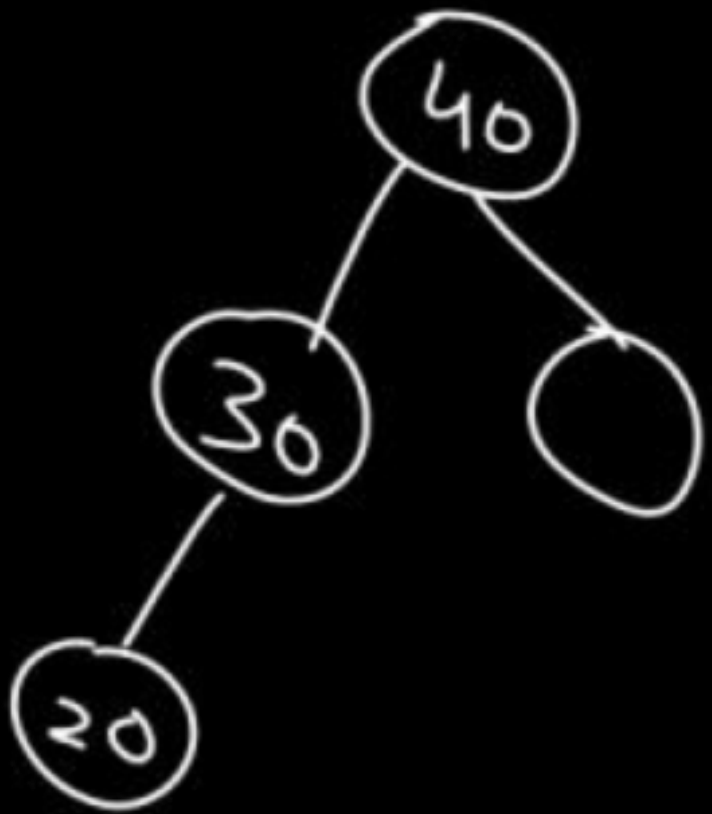


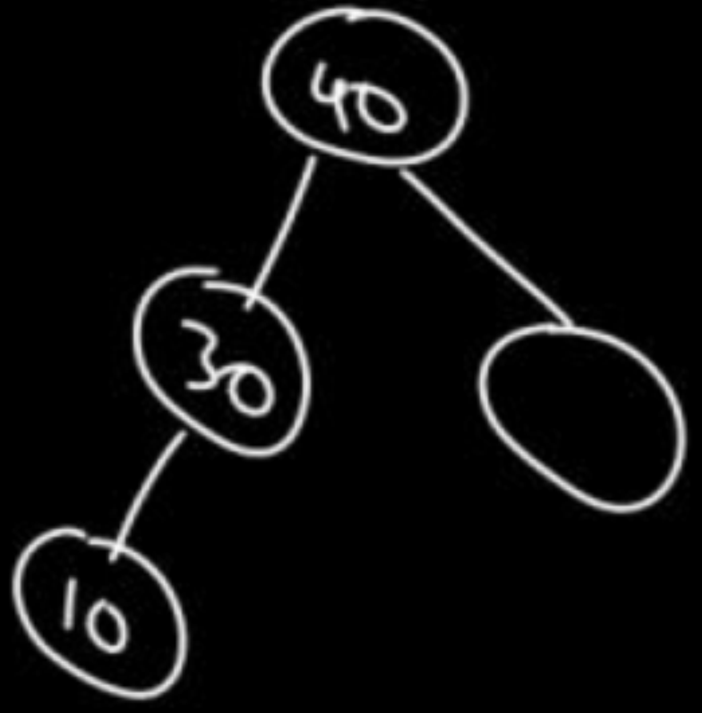
out of 3 $\Rightarrow$ select any 2

$^3C_2 = 3$

(i) 10, 20 (ii) 20, 30 (iii) 10, 30

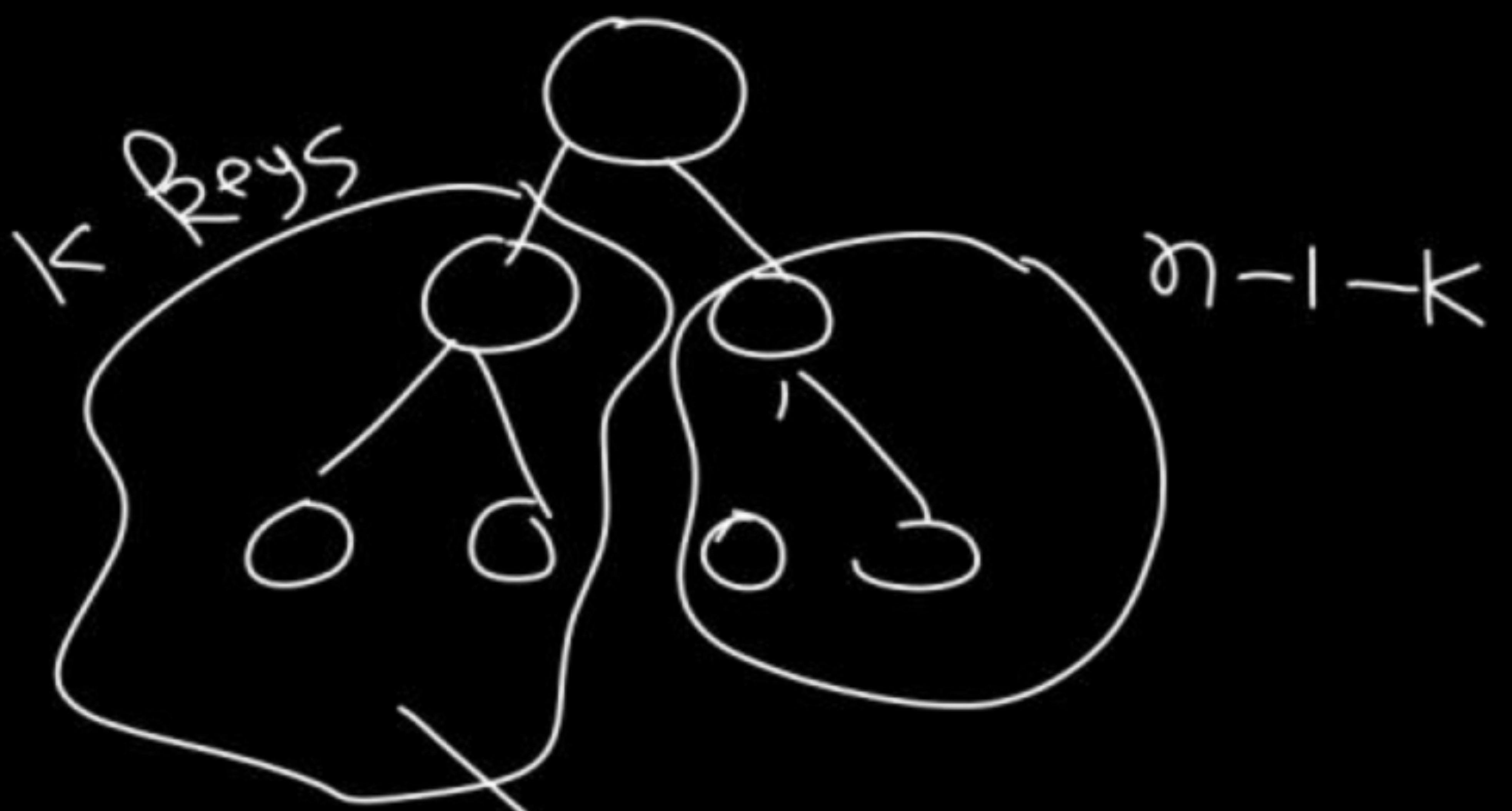






10, 20, 30, 40

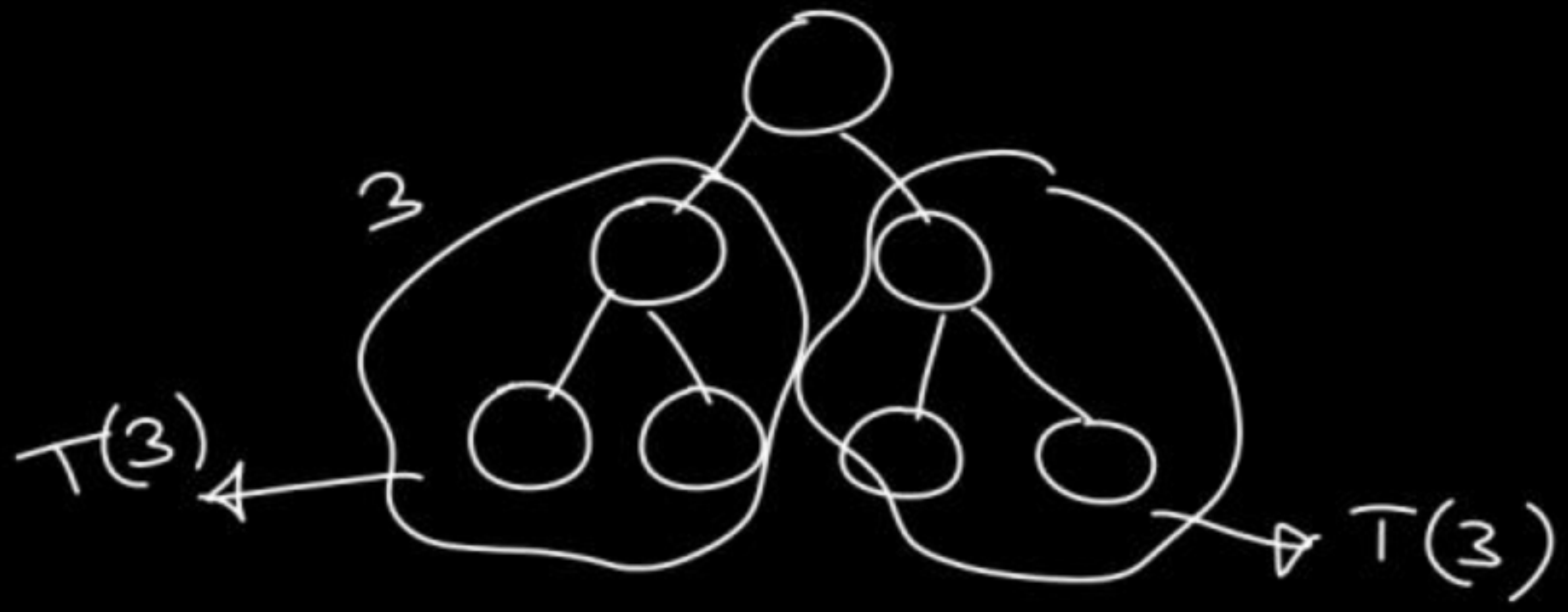
$T(n)$: no. of max-heap with n-distinct keys



$$T(n) = 1 \times {}^{n-1}C_k \times T(k) \times T(n-k-1)$$

selecting any k keys from $(n-1)$
for L_T

$n = 7$ (7 distinct keys)

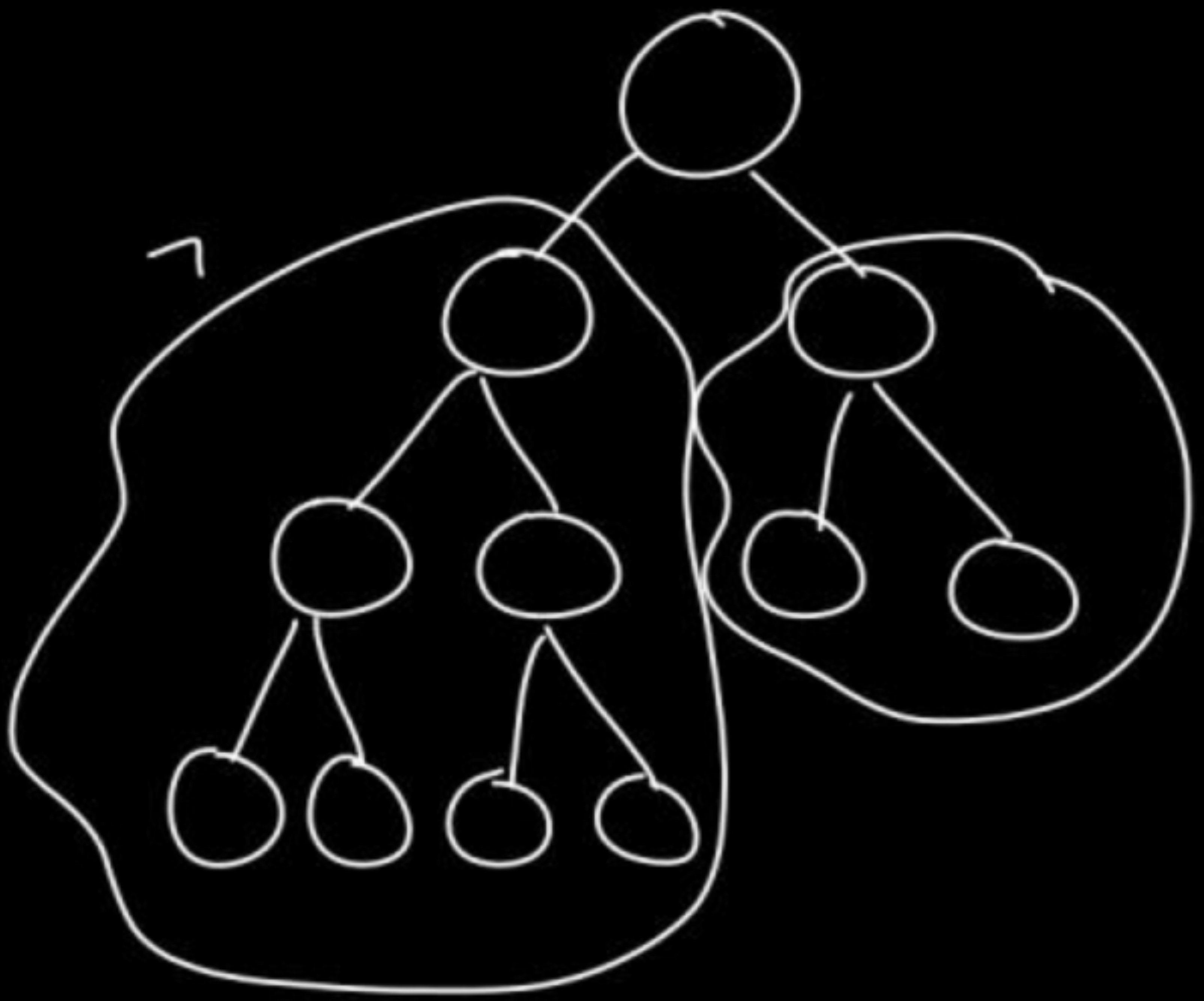


$$T(7) = 1 \times {}^{6}C_{3} \times T(3) \times T(3)$$

$$= 1 \times {}^{6}C_{3} \times 2 \times 2$$

$$= \frac{6!}{3!\,3!} \times 2 \times 2 = \frac{\cancel{6} \times 5 \times 4 \times \cancel{3!}}{\cancel{3!}\,\cancel{3!}} \times 2 \times 2 = 80$$

$n = 11$ distinct keys $\longrightarrow$ Max heap



$$T(11) = 1 \times {}^{10}C_7 \times T(7) \times T(3)$$

$$\boxed{T(11) = {}^{10}C_7 \times 80 \times 2}$$


THANK YOU GW
SOLDIERS !