CS & IT ENGINEERING

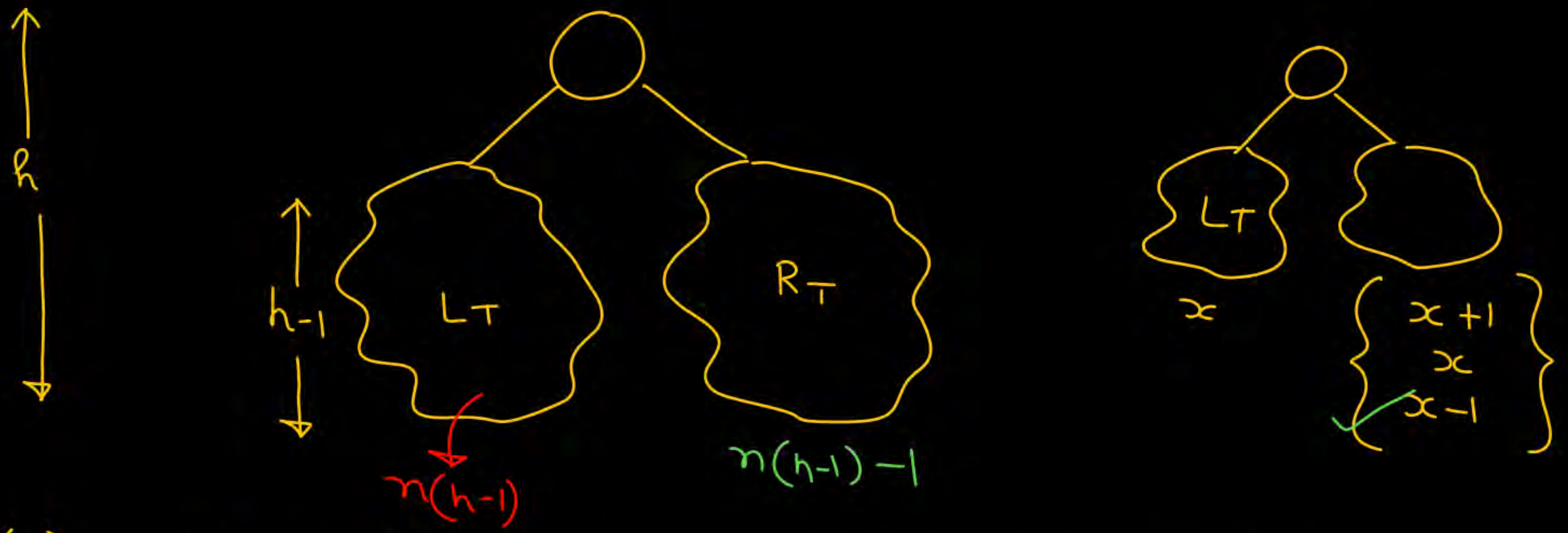
Data Structure & Programming

Tree

Lec- 07

By- Pankaj Sharma Sir


TOPICS TO
BE
COVERED
Tree 07



$n(h)$: Min no. of nodes in such given tree of h height.

$$n(h) = 1 + n(h-1) + n(h-1)-1$$

$$\boxed{n(h) = 2n(h-1)}$$

$n(h) = 2n(h-1)$

$n(1) = 2 \cdot n(0) = 2^1$

$n(2) = 2 \cdot n(1) = 2^2$

$\vdots$

$n(5) = 2^5 = 32$

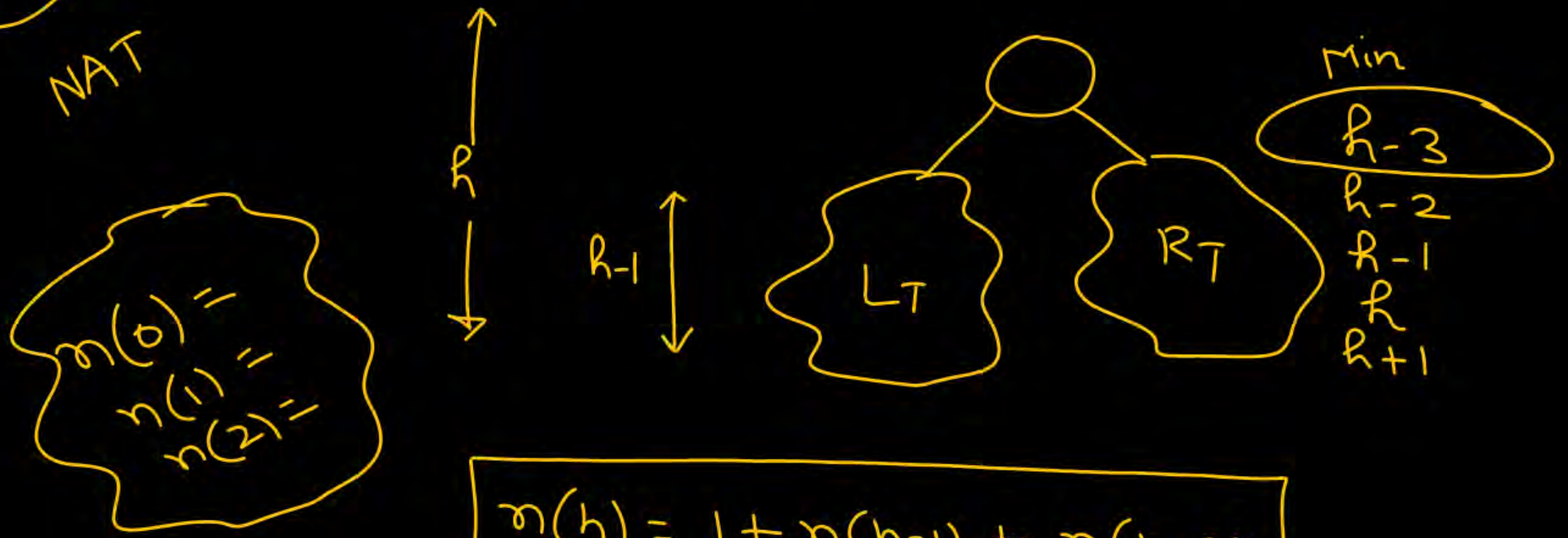
$n(0) = 1$

Every node ∴ diff of height of L_T and height of R_T is atmost 2

Min no. of nodes in such a binary tree of $h = 4$.

Expected

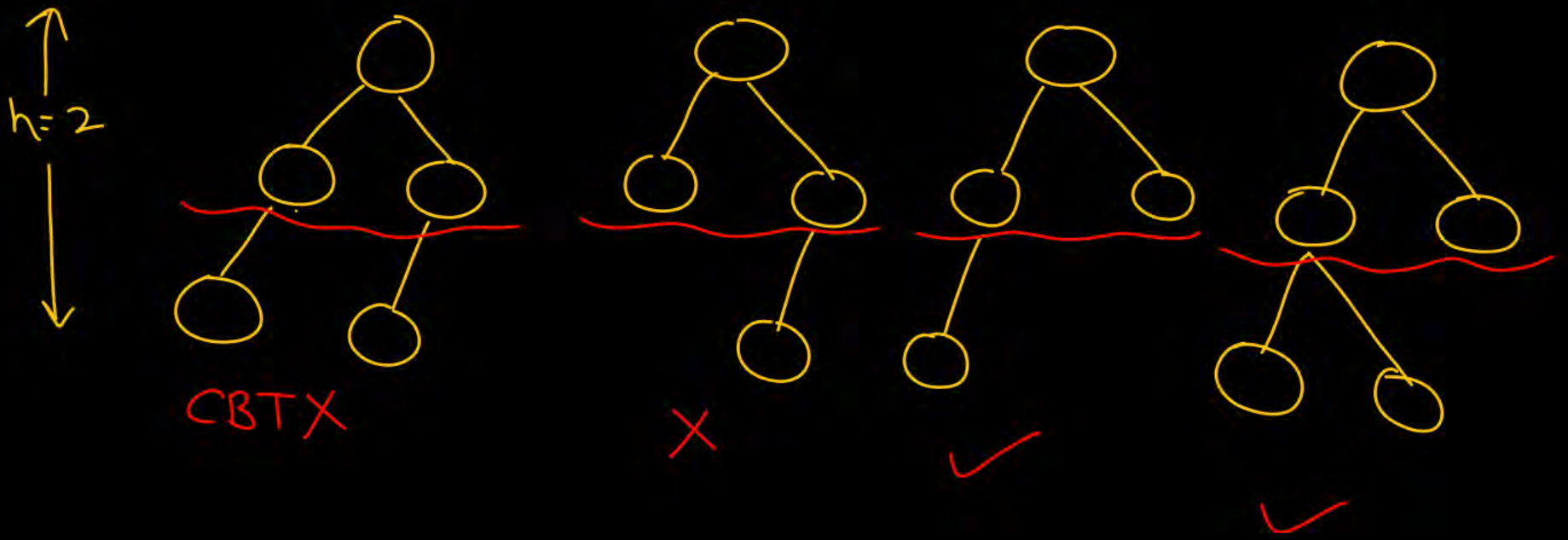
NAT



$$n(h) = 1 + n(h-1) + n(h-3)$$

Heap

CBT



Heap

Max-heap

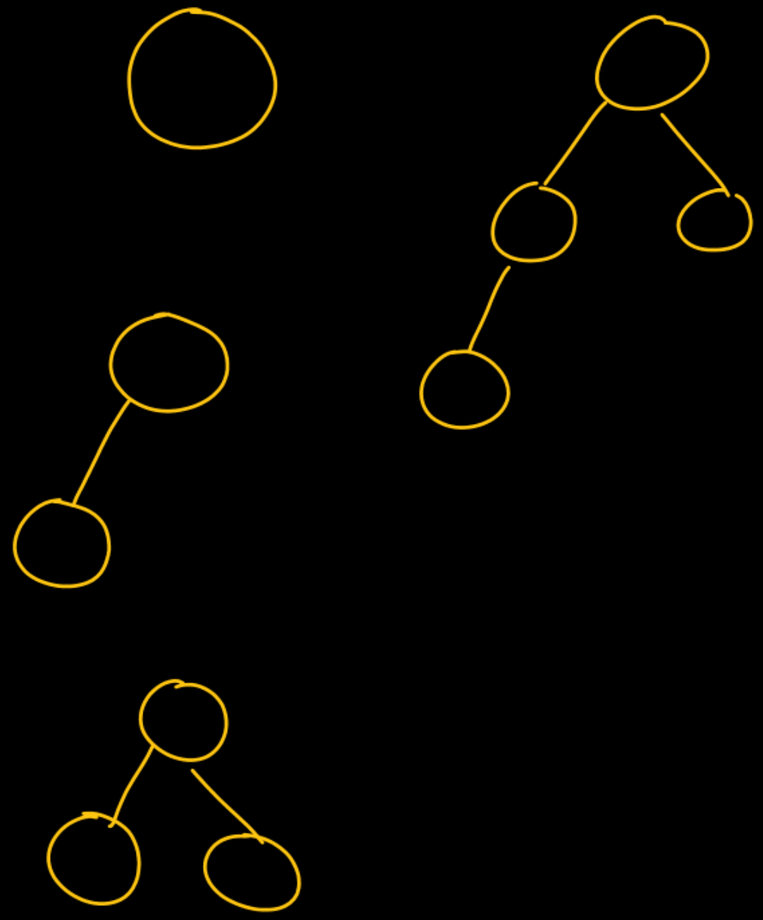
A **CBT** in which **every node** satisfies property:

The value of node is greater than its children.

Min-heap

A **CBT** in which **every node** satisfies the property:

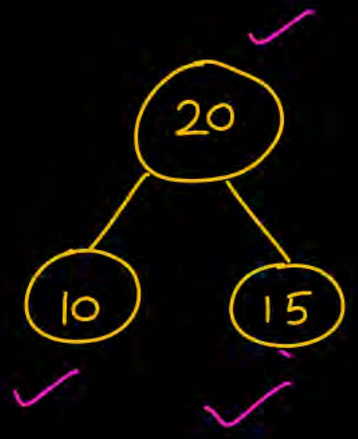
The value/key of node is smaller than its children.



CBT ✓

Max-heap ✓

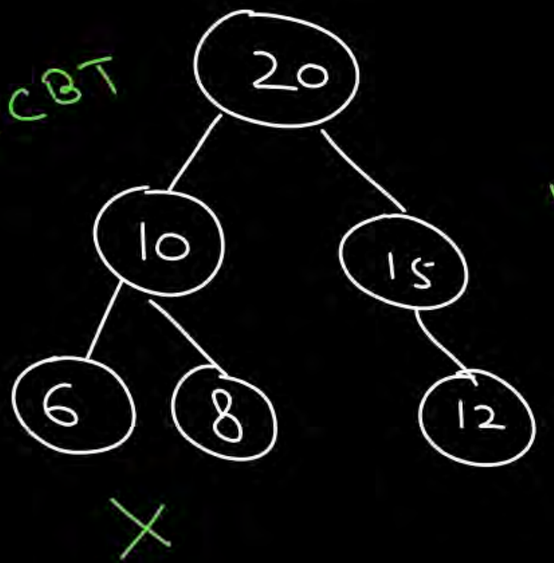
Min-heap ✓



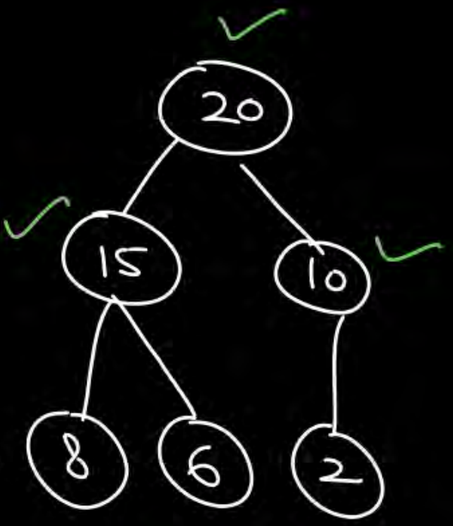
Max-heap ✓

Min-heap X

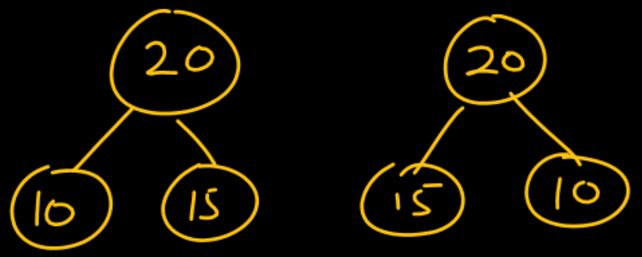
Not a CBT



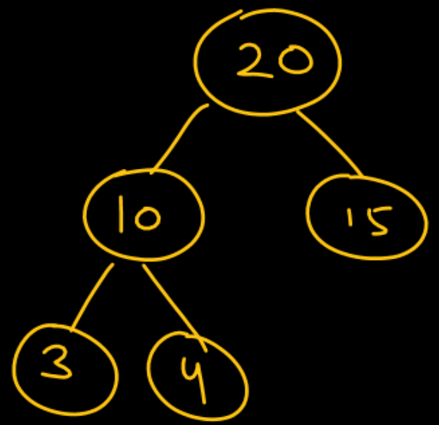
X

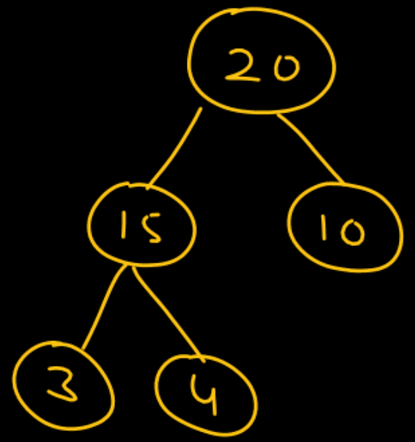


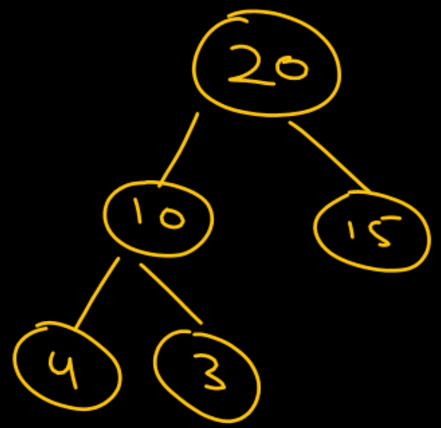
Max-heap

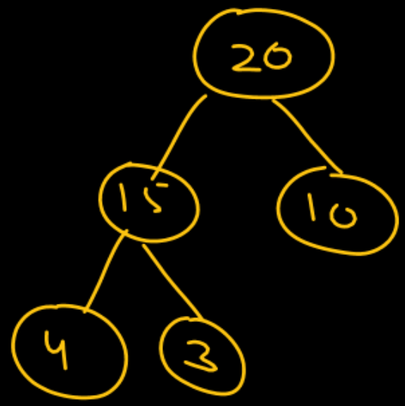

20
10
15

20
15
10


20
10
15
3
4


20
15
10
3
4


20
10
15
4
3

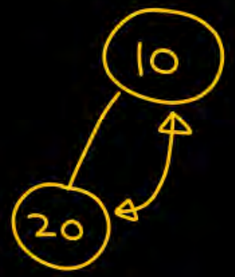

20
15
10
4
3

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

Construct Max-heap by inserting keys 10, 20, 30, 40, 50, 60, 70 in same order.

Insert 10

Insert 20

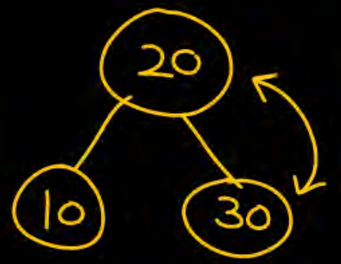


Is it a max-heap?

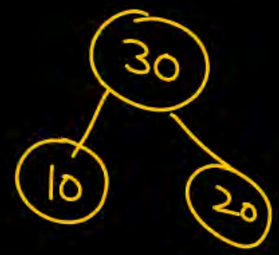
swap

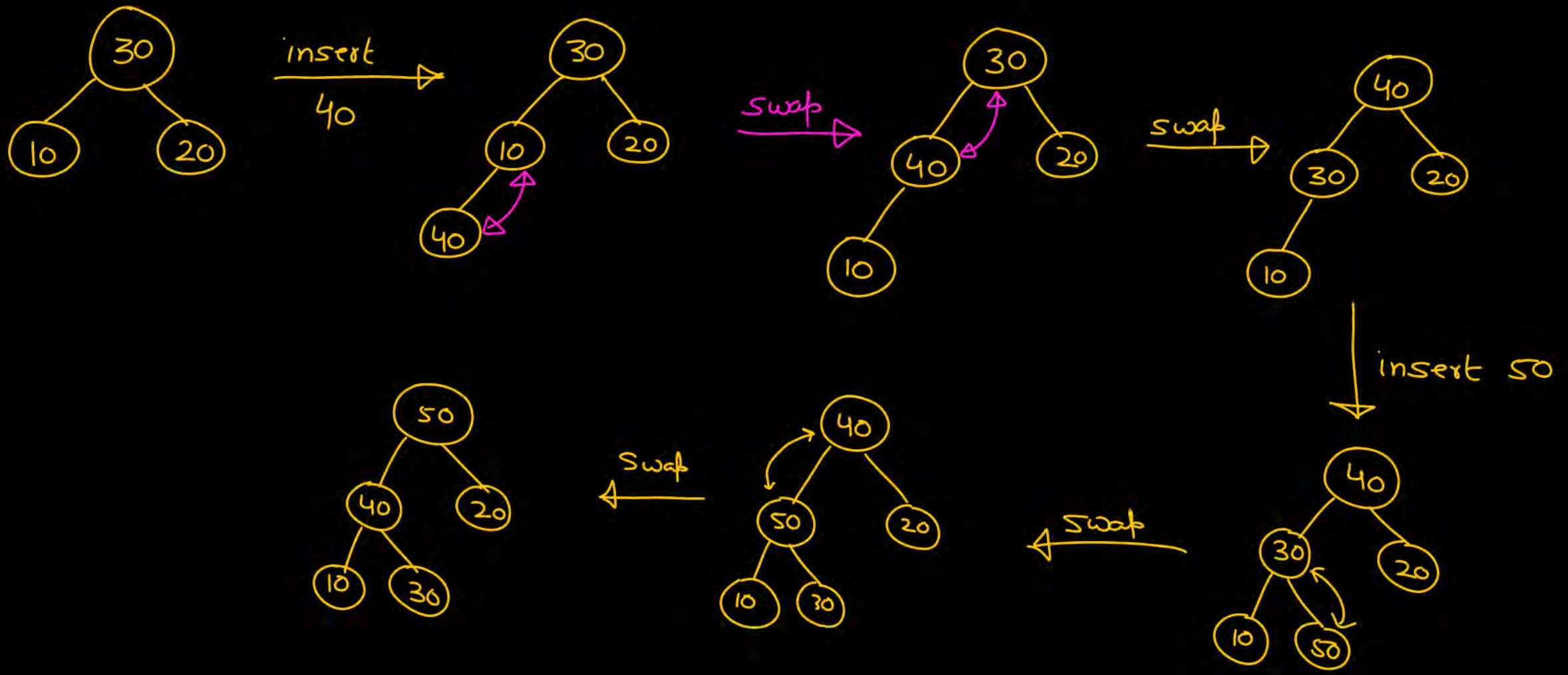
Insert

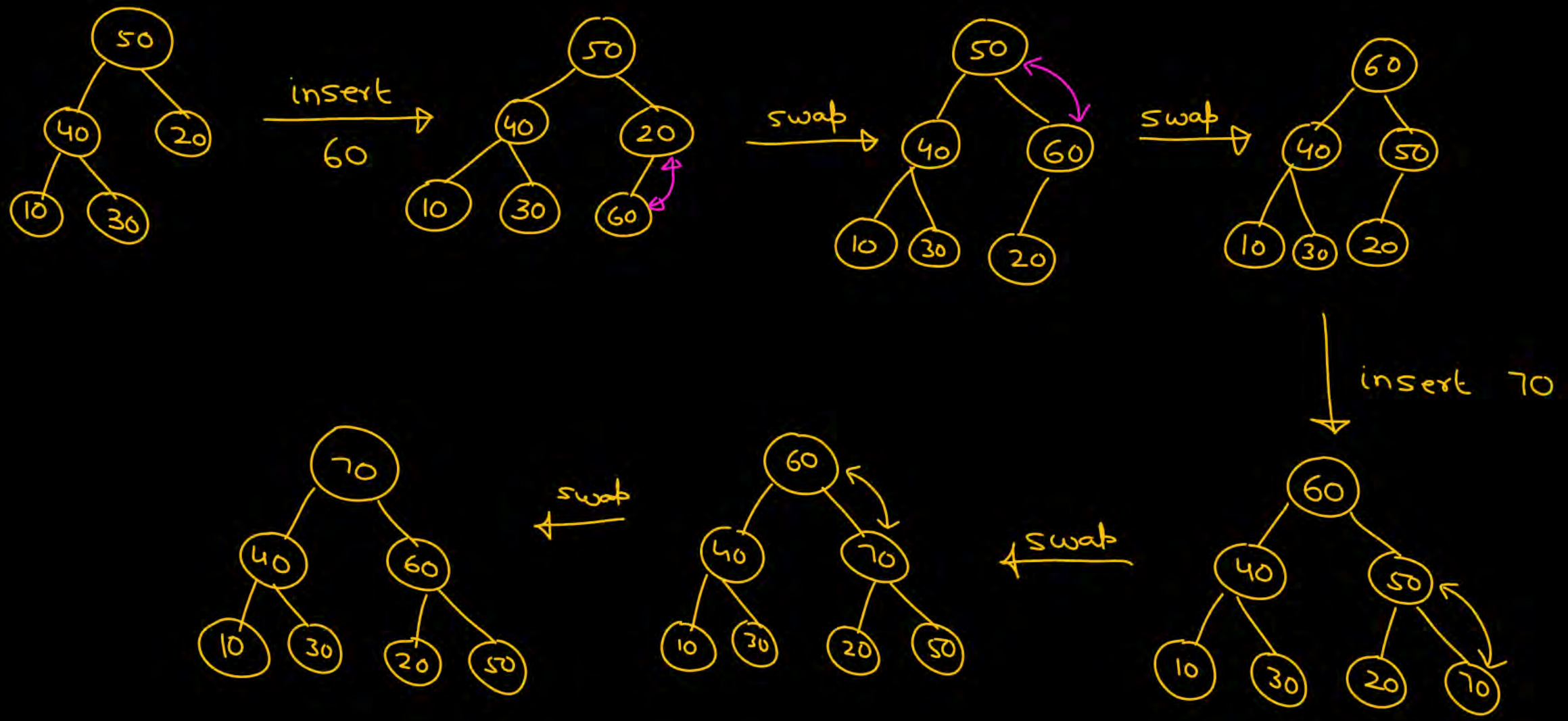
30



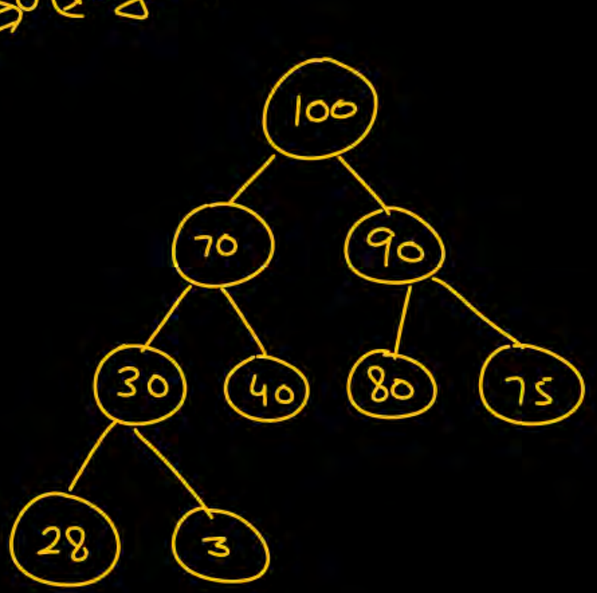
swap




30
10
20
insert
40
30
10
20
40
swap
30
40
20
10
swap
40
30
20
10
insert 50
40
30
20
10
50
swap
40
50
20
10
30
swap
50
40
20
10
30


50
40
20
10
30
insert
60
50
40
20
10
30
60
swap
50
40
60
10
30
20
swap
60
40
50
10
30
20
insert 70
60
40
50
10
30
20
70
swap
60
40
70
10
30
20
50
swap
70
40
60
10
30
20
50

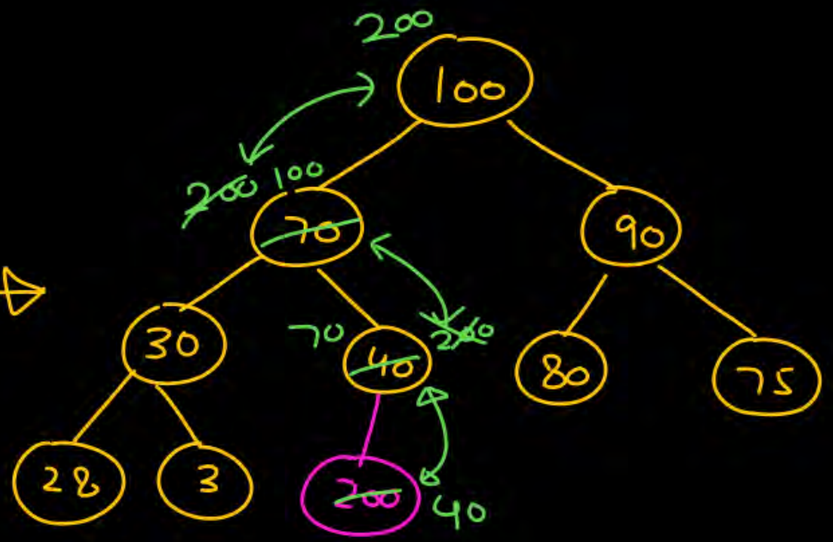
$n \Rightarrow O(\log_2 n)$



Heap

insert

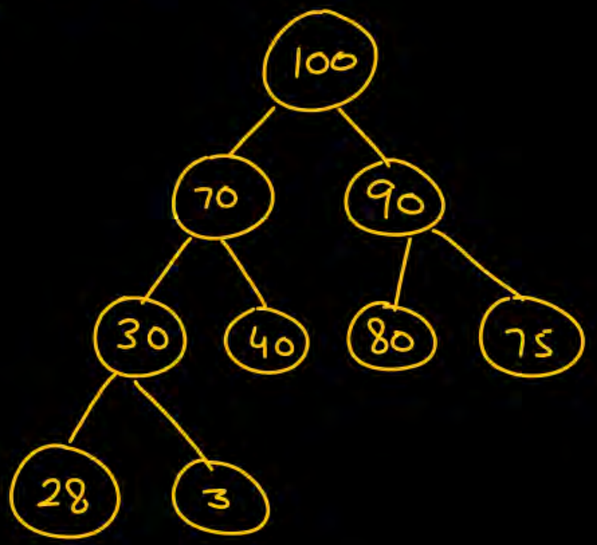
200



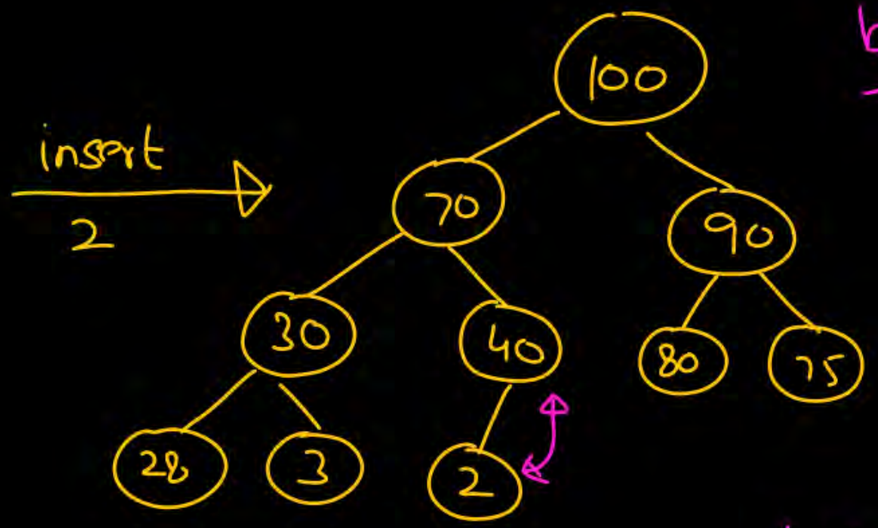
$n \log n$

insertion in heap with n nodes $\longrightarrow O(\log_2 n)$

$n \Rightarrow O(\log_2 n)$



Heap

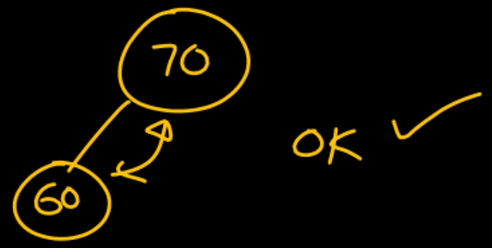


best case

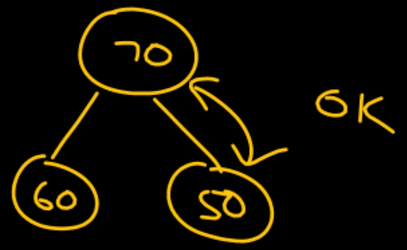
1 comparision

$\Rightarrow$ constant time

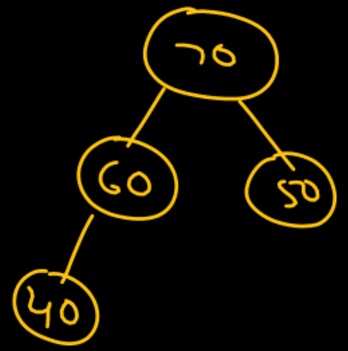
Const. max. heap by inserting keys 70, 60, 50, 40, 30, 20, 10 in given order.



OK ✓



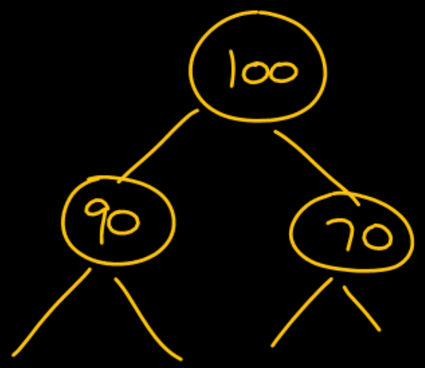
OK

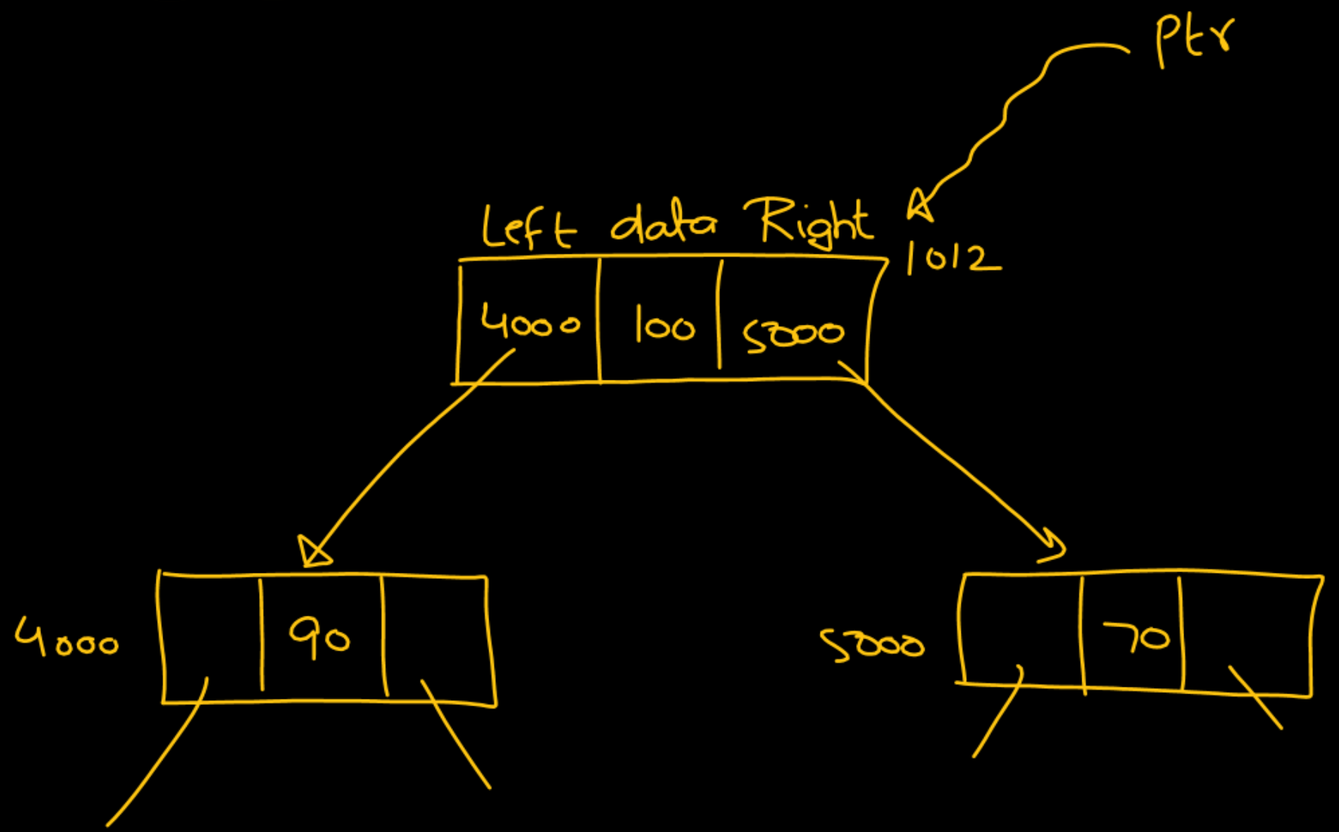


1. Build-Heap
2. Heapify Algo
3. Given an array rep. CBT, convert it into max-heap.

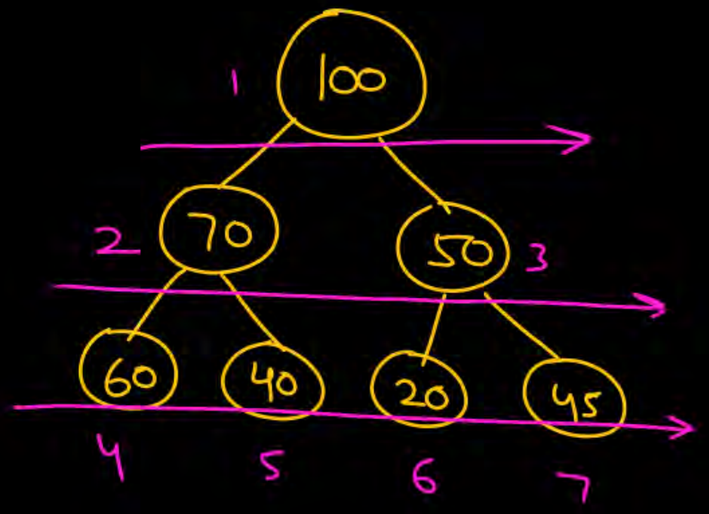
Tree representation

L.L.



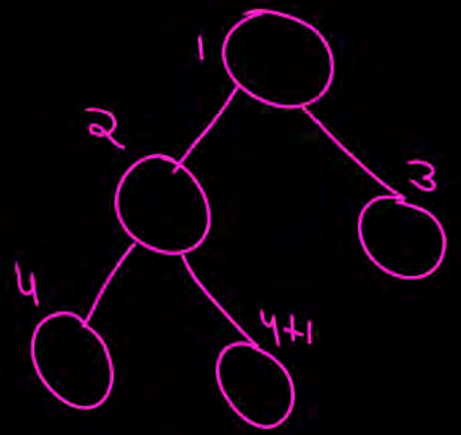


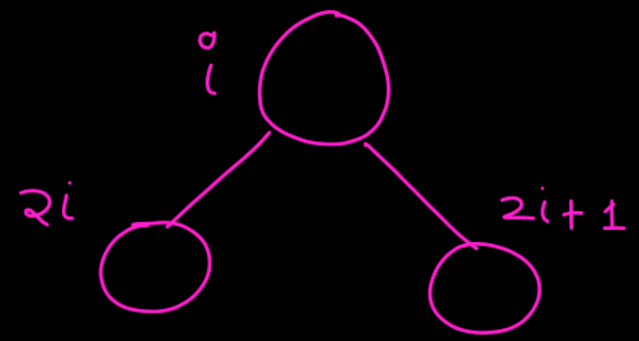
CBT ⟹ Array representation



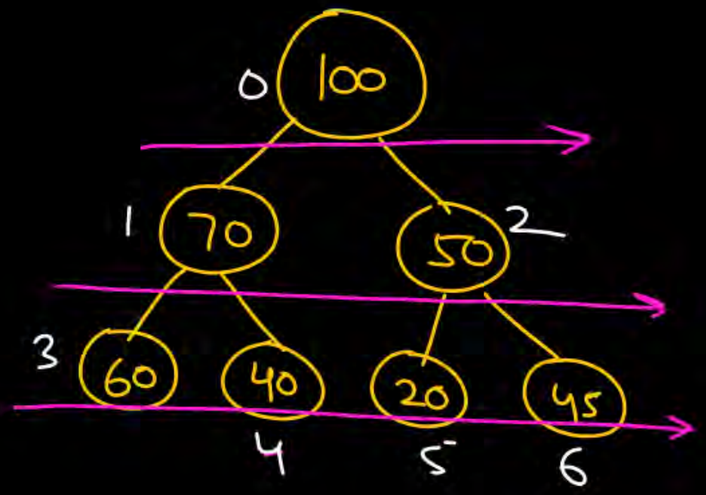
100	70	50	60	40	20	45
1	2	3	4	5	6	7

index of node 100 ⟹ 1
" " " 70 ⟹ 2




i
2i
2i+1

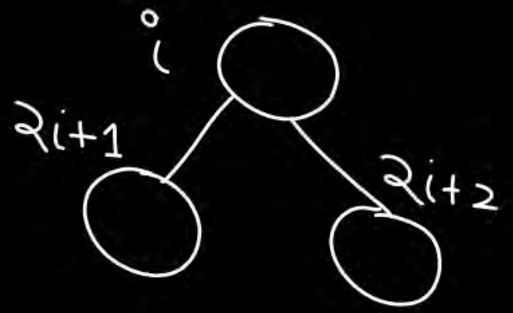
Prog Imp.



100	70	50	60	40	20	45
0	1	2	3	4	5	6

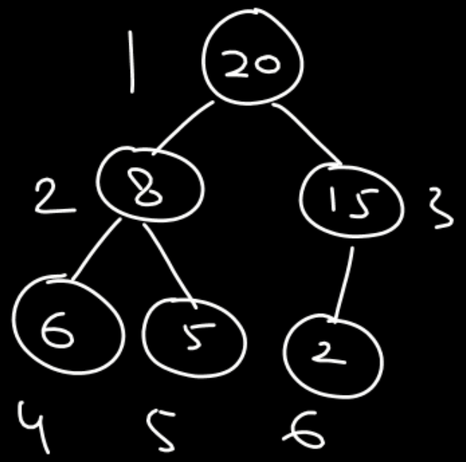
index of node 100 ⇒ 0

" " " 70 ⇒ 1



Given an array rep. of a CBT 20, 8, 15, 6, 5, 2

Is it a max-heap?

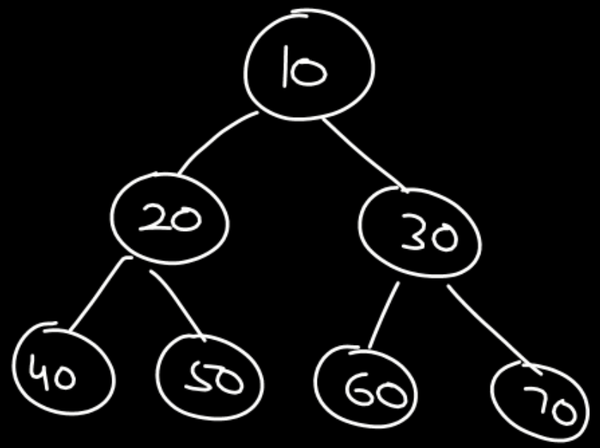


20	8	15	6	5	2
1	2	3	4	5	6

Given an array rep. of a CBT as 10, 20, 30, 40, 50, 60, 70 ?

Is it a max-heap?

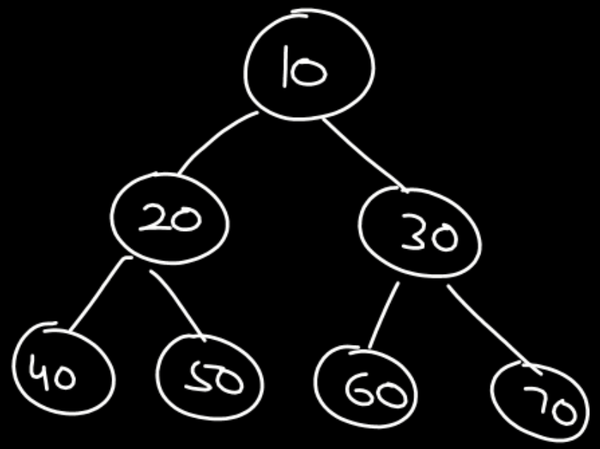
→ NO



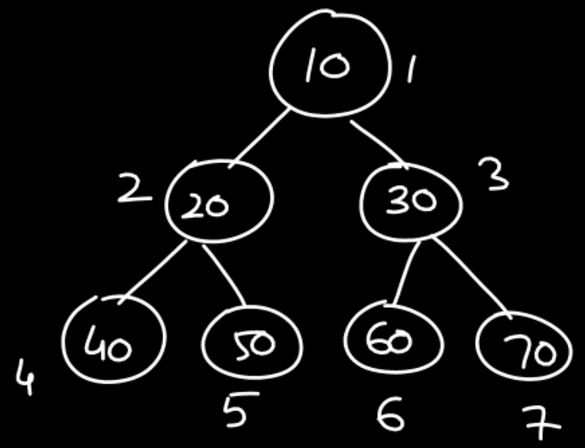
10	20	30	40	50	60	70
1	2	3	4	5	6	7

Given an array rep. of a CBT as 10, 20, 30, 40, 50, 60, 70 ?

Convert it to a max-heap.



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

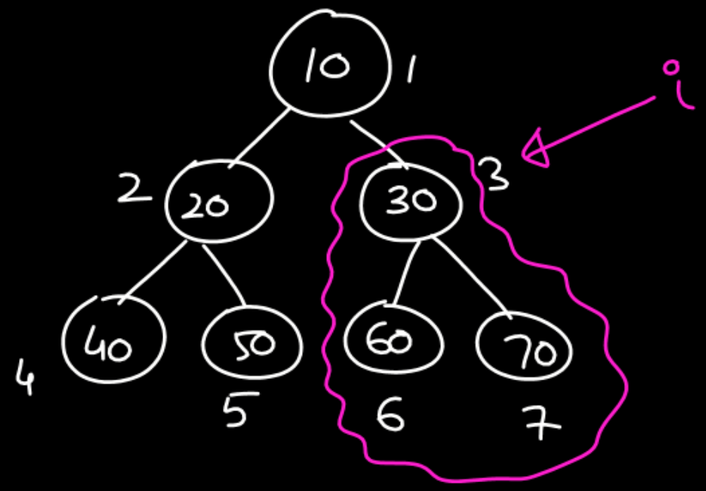


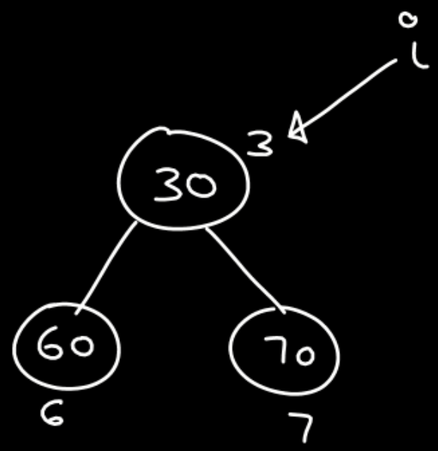
Every leaf node always satisfies max-heap property

index of internal nodes = 1, 2, 3

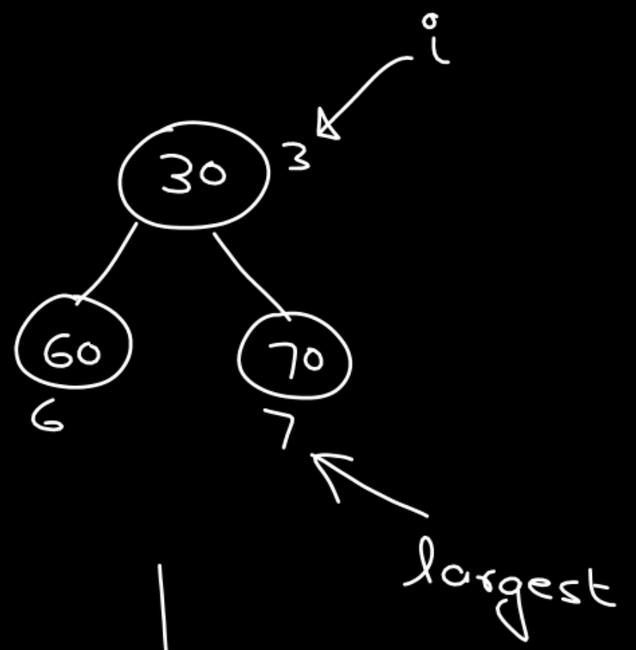
1 to $\lfloor \frac{n}{2} \rfloor$

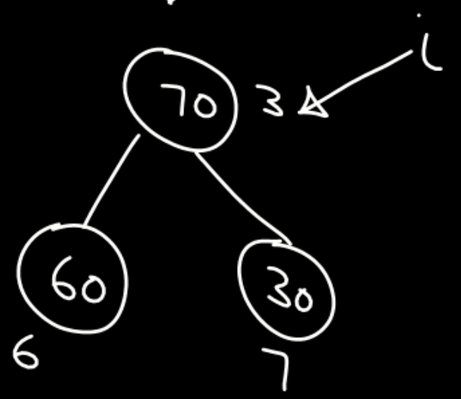
4,5,6,7

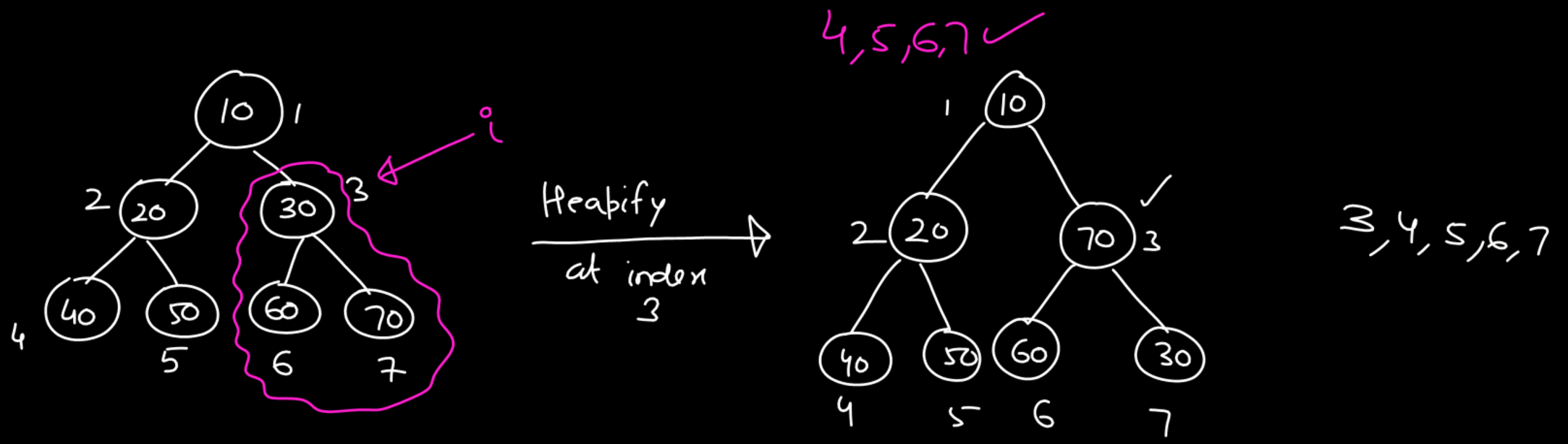

10
1
i
2
20
30
3
40
50
60
70
4
5
6
7

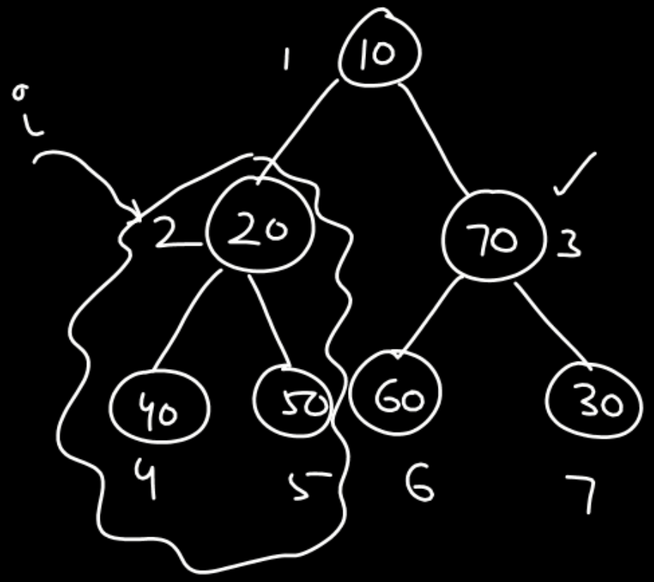

i
3
30
60
70
6
7

2 comp.

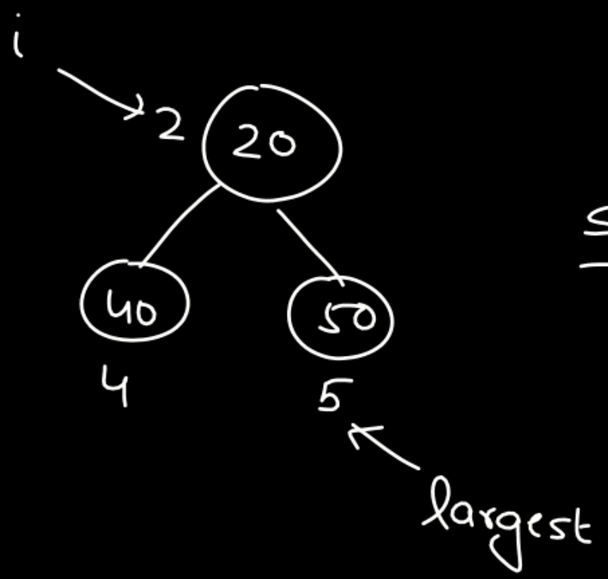

i
3
30
60
70
6
7
largest

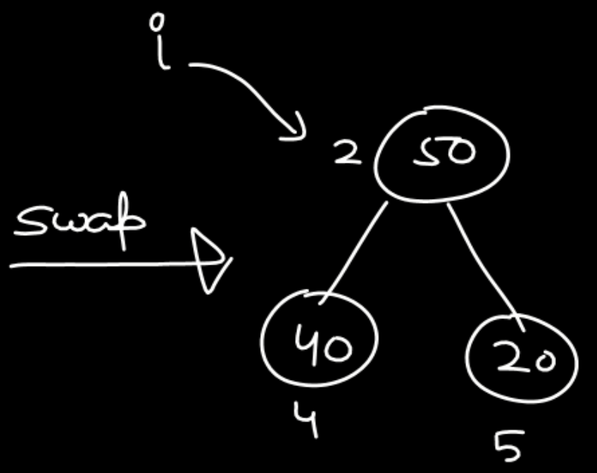

i
3
70
60
30
6
7


4,5,6,7
10
1
i
2
20
30
3
40
50
60
70
4
5
6
7
Heapify
at index
3
1
10
2
20
70
3
40
50
60
30
4
5
6
7
3,4,5,6,7



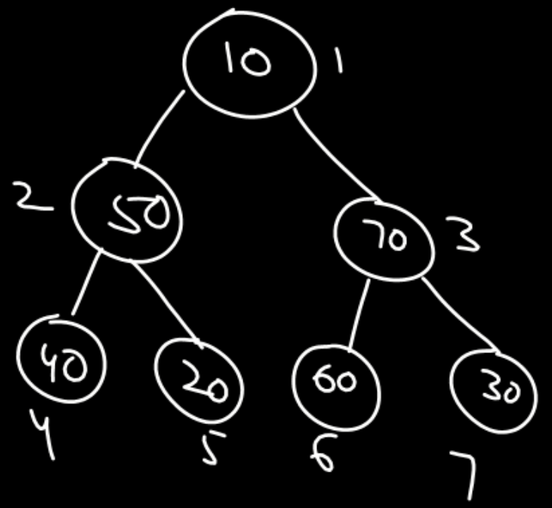
4,5,6,7 ✓



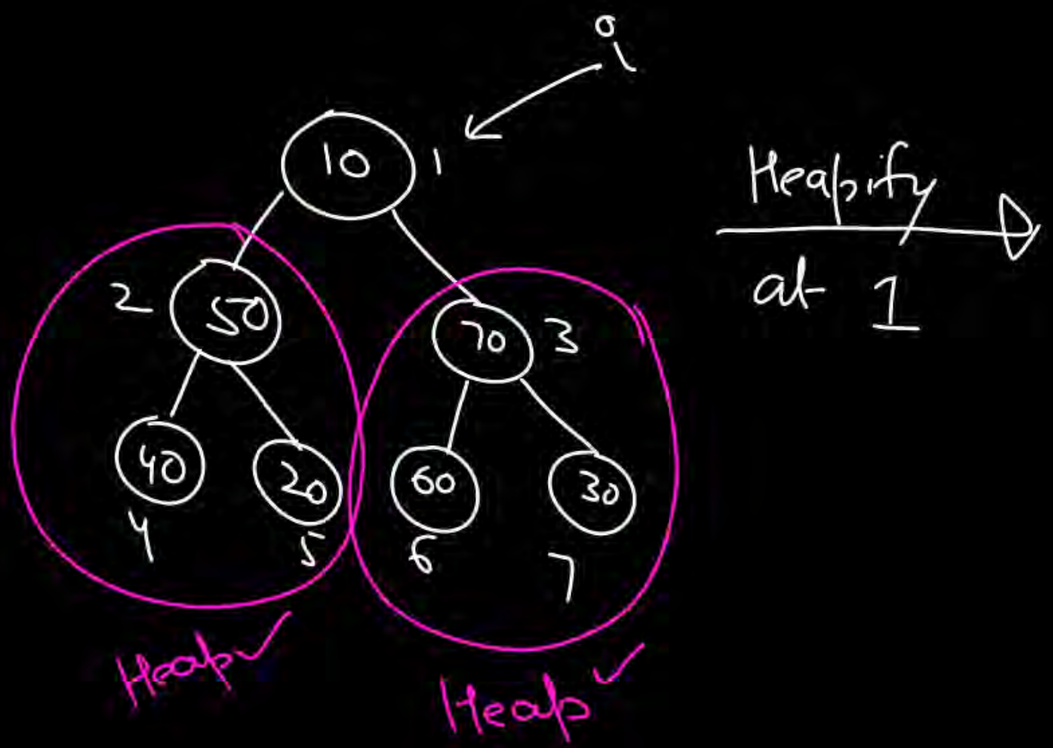


In 2 comparision

Heapify at index 2

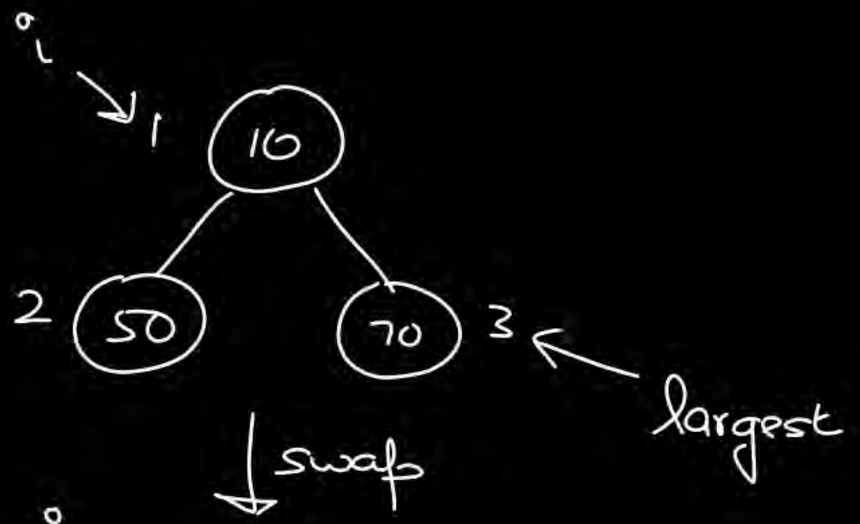


2,3,4,5,6,7 ✓

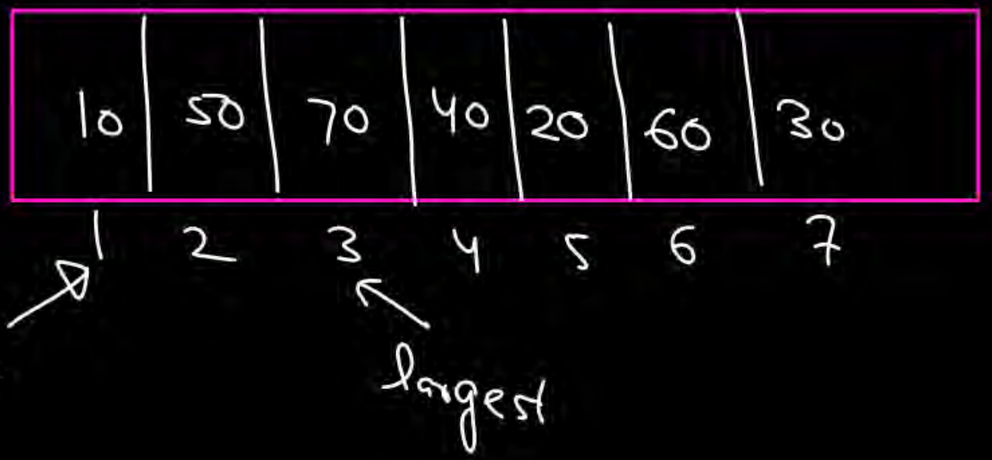


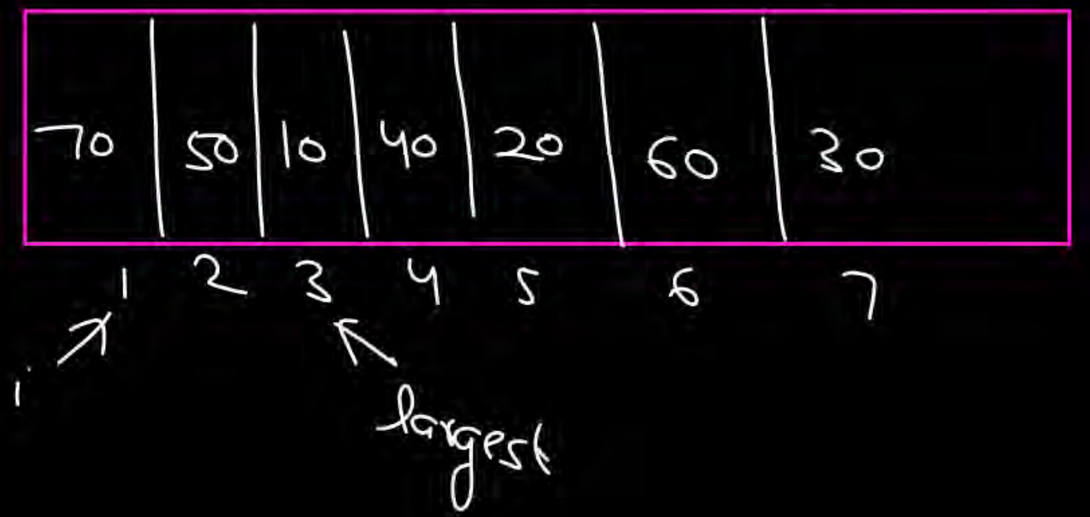
Heapify at 1

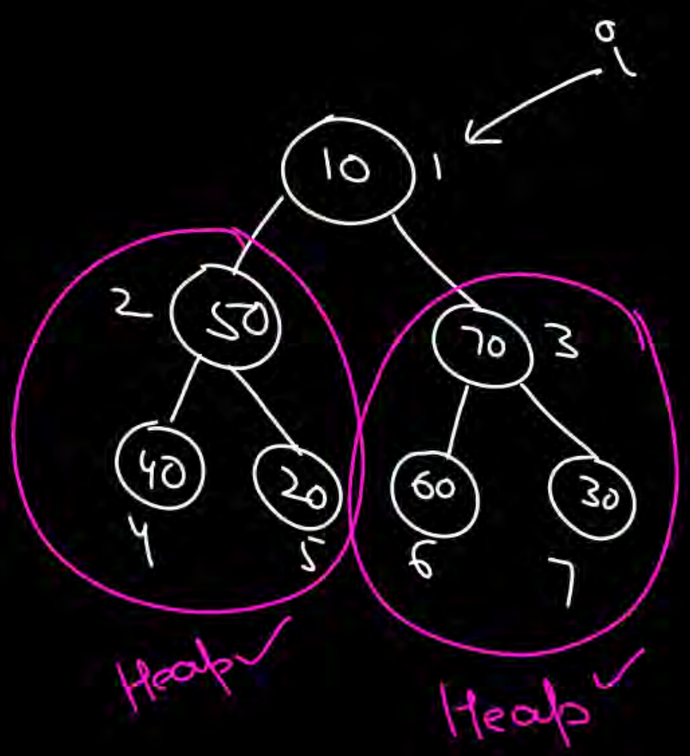
2 comp.

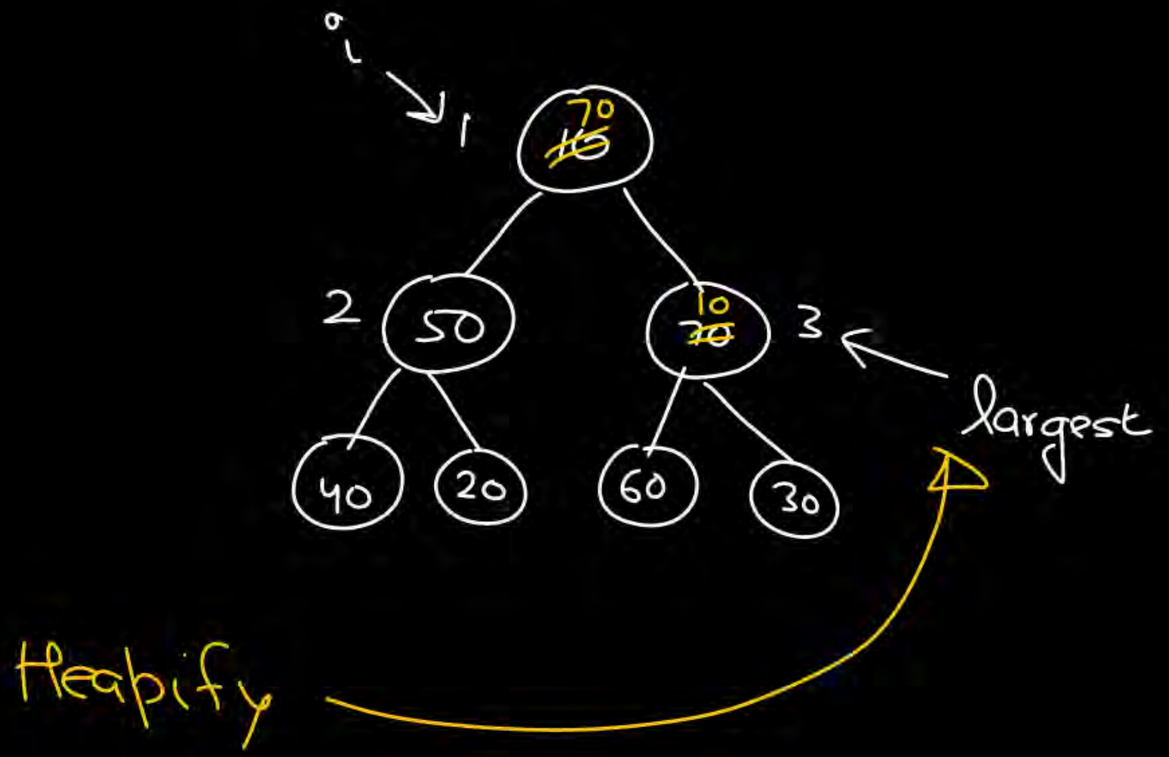


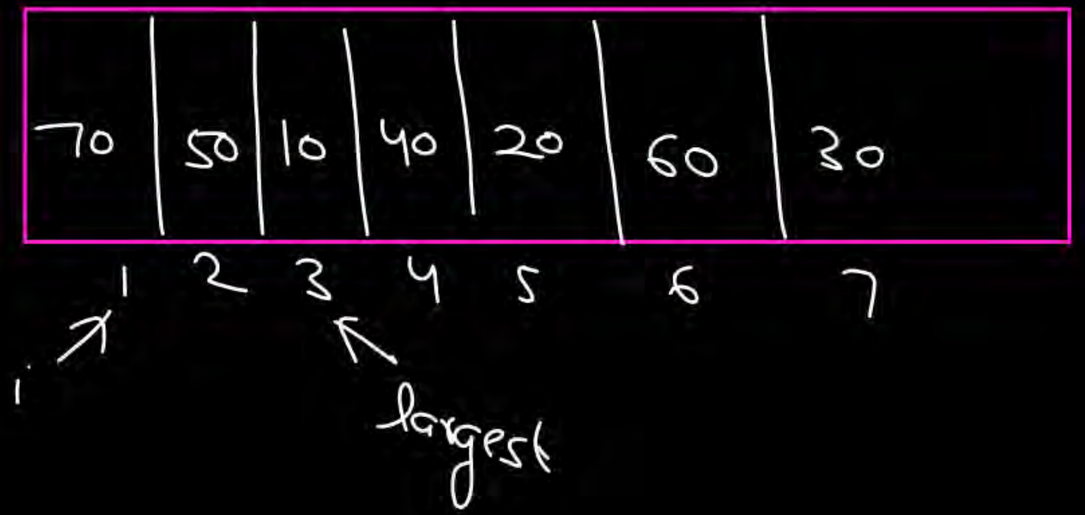
swap

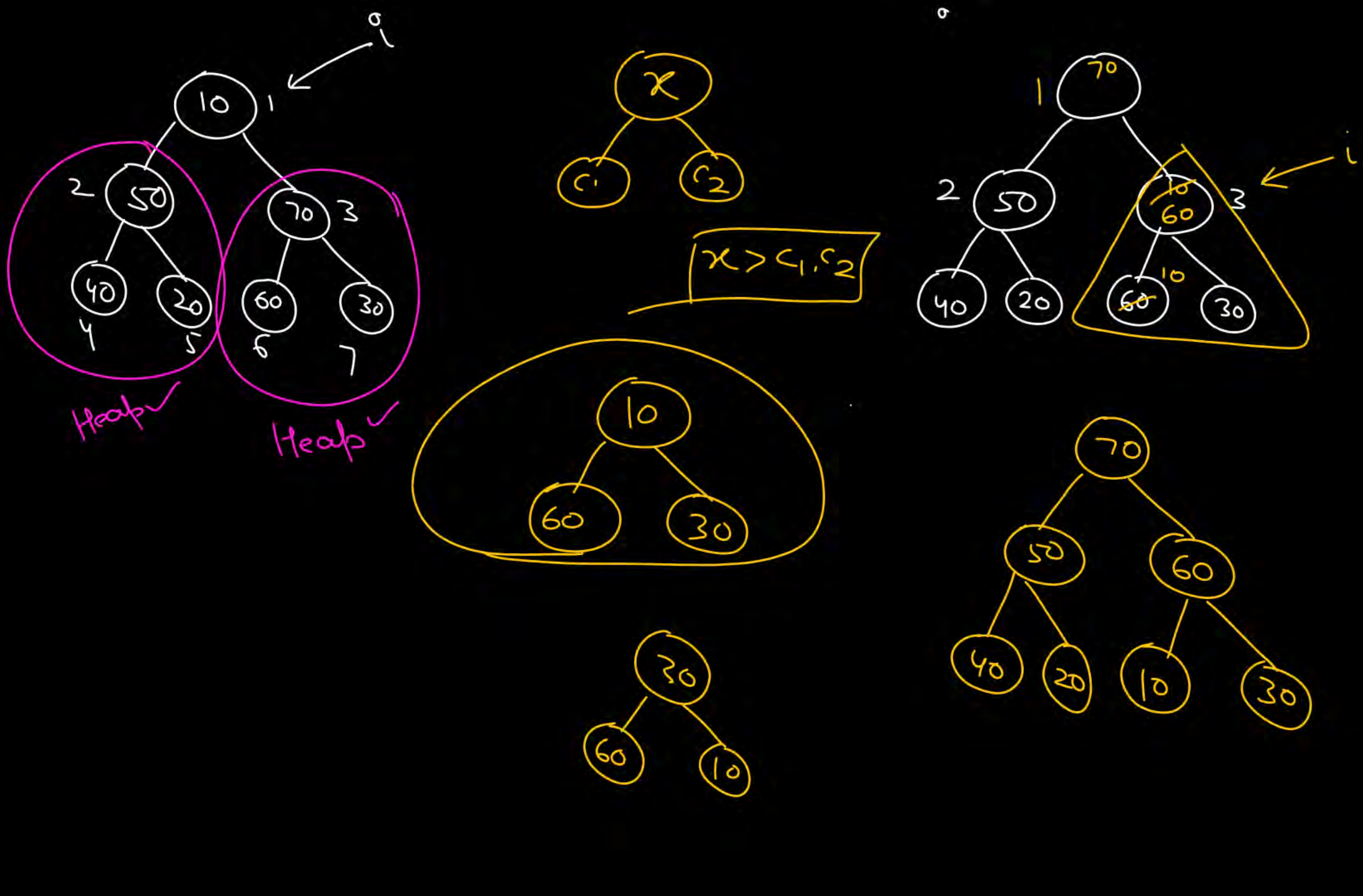


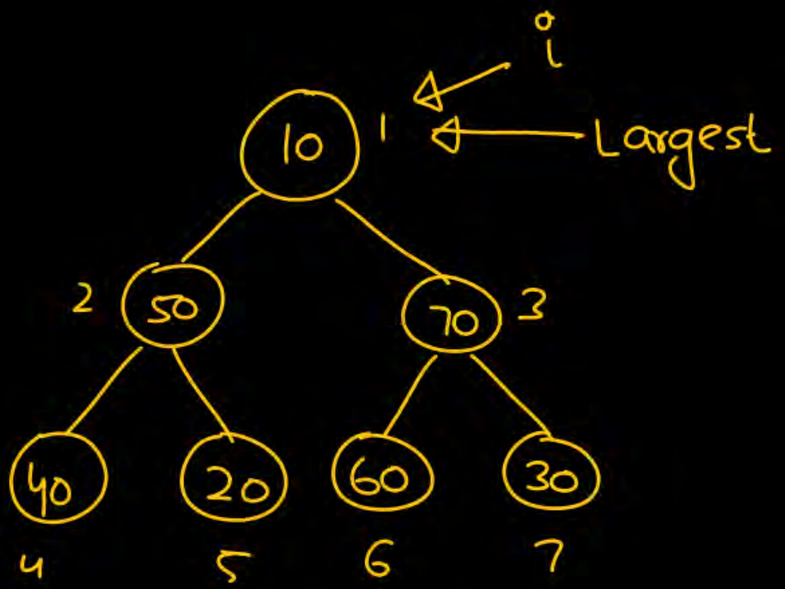



10
1
2
50
70
3
40
20
60
30
4
5
6
7
Heap
Heap


70
10
2
50
10
20
3
40
20
60
30
largest
Heapify


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


i
10
1
2
50
70
3
40
20
60
30
4
5
6
7
Heap
Heap
x
c1
c2
x > c1, c2
i
70
1
2
50
10
60
3
40
20
60
10
30
10
60
30
70
50
60
40
20
10
30
20
60
10



$n = 7$

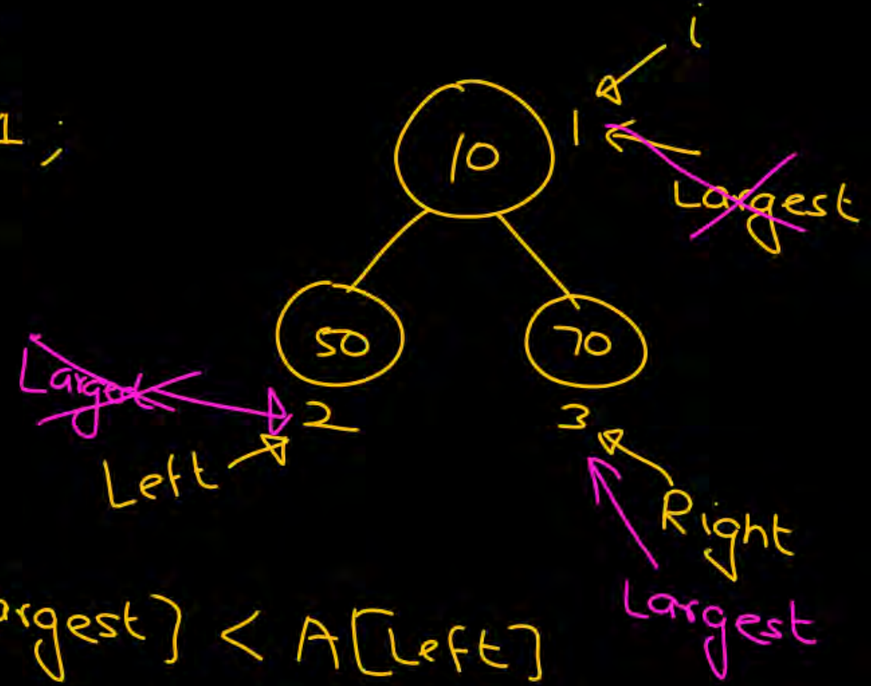
1 7

Heapify (A, i, n)

1) Left = 2*i ;

Right = 2*i + 1 ;

Largest = i ;



A[Largest] < A[Left]

Largest = Left

A[Largest] < A[Right]

Largest = Right


THANK YOU GW
SOLDIERS !