CS & IT ENGINEERING

Data Structure

Tree

Chapter- 5

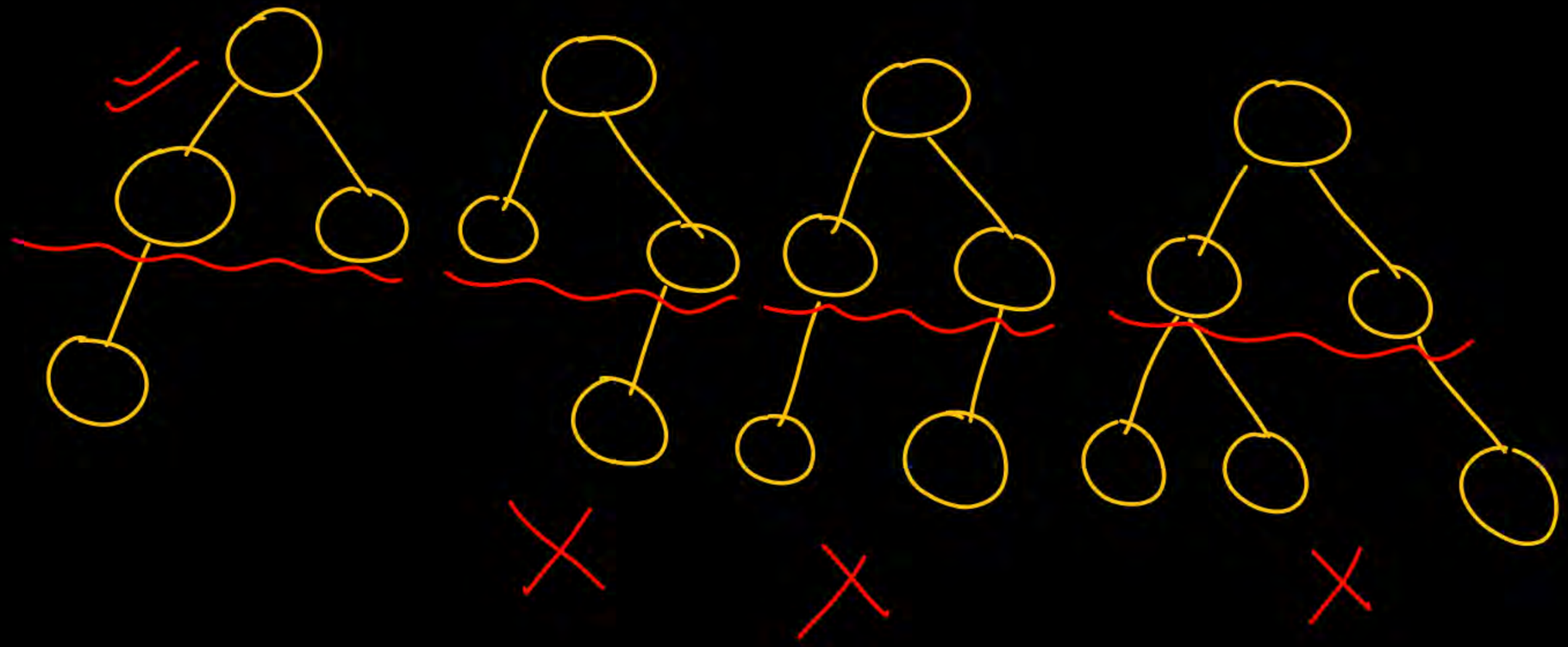
Lec- 08

By- Pankaj Sharma sir

Tree-VIII

Heap

Complete binary tree :

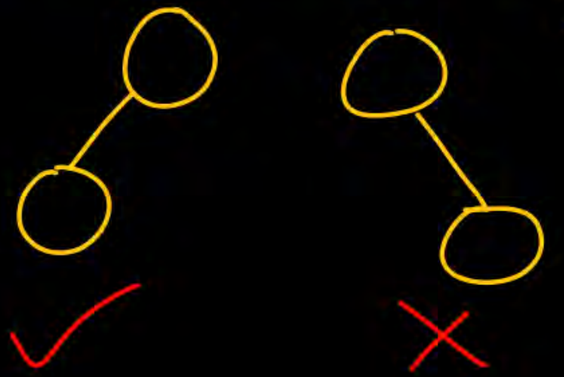


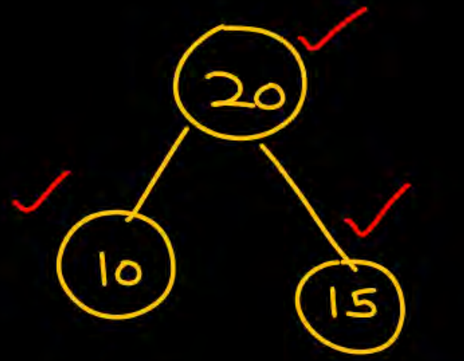
A CBT in which Every node follow the property: The value of the node is greater than its childrens.

A CBT in which every node follow the property: The value of node is smaller than its children.

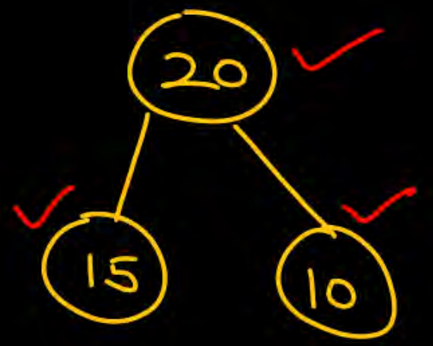
CBT ✓
Max-heap ✓
Min-heap ✓

Structure of a CBT with 2 nodes

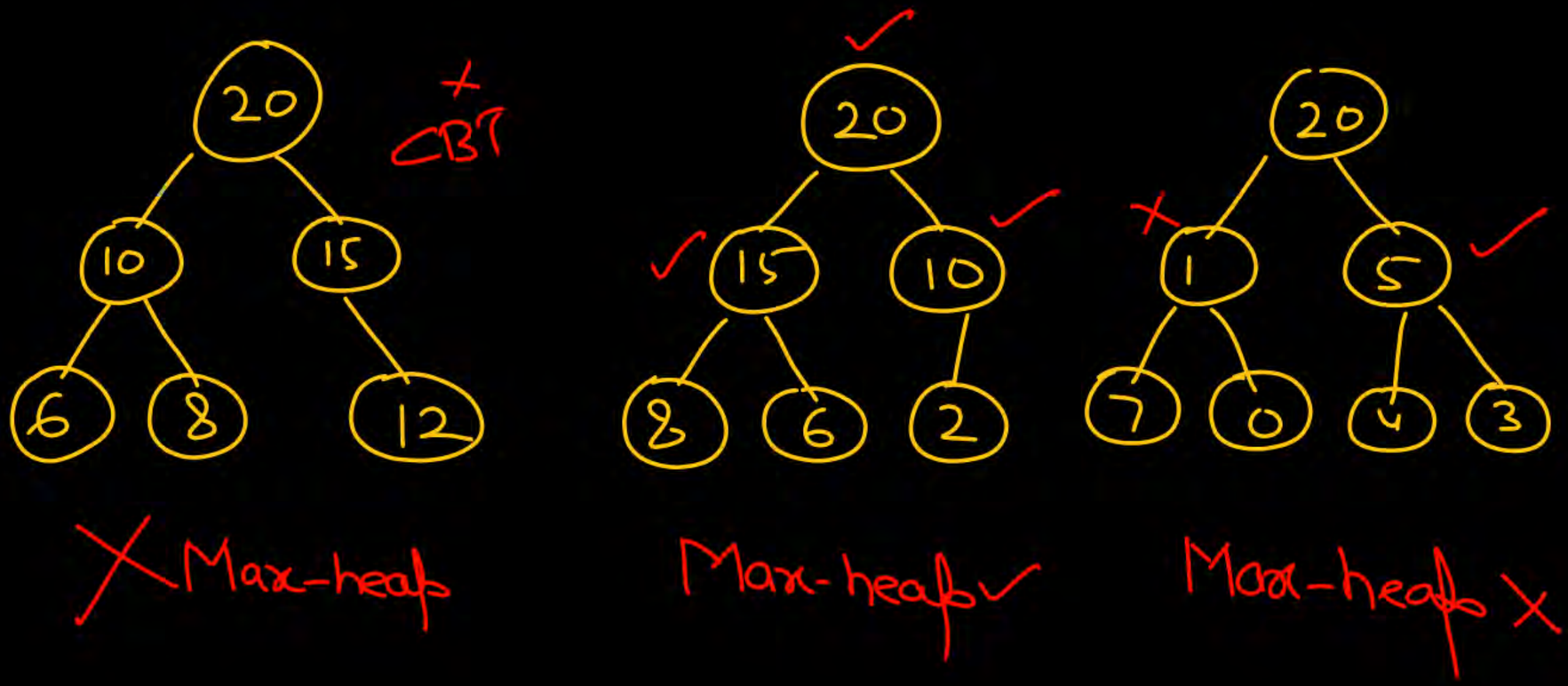




Max-heap ✓
Min-heap ✗



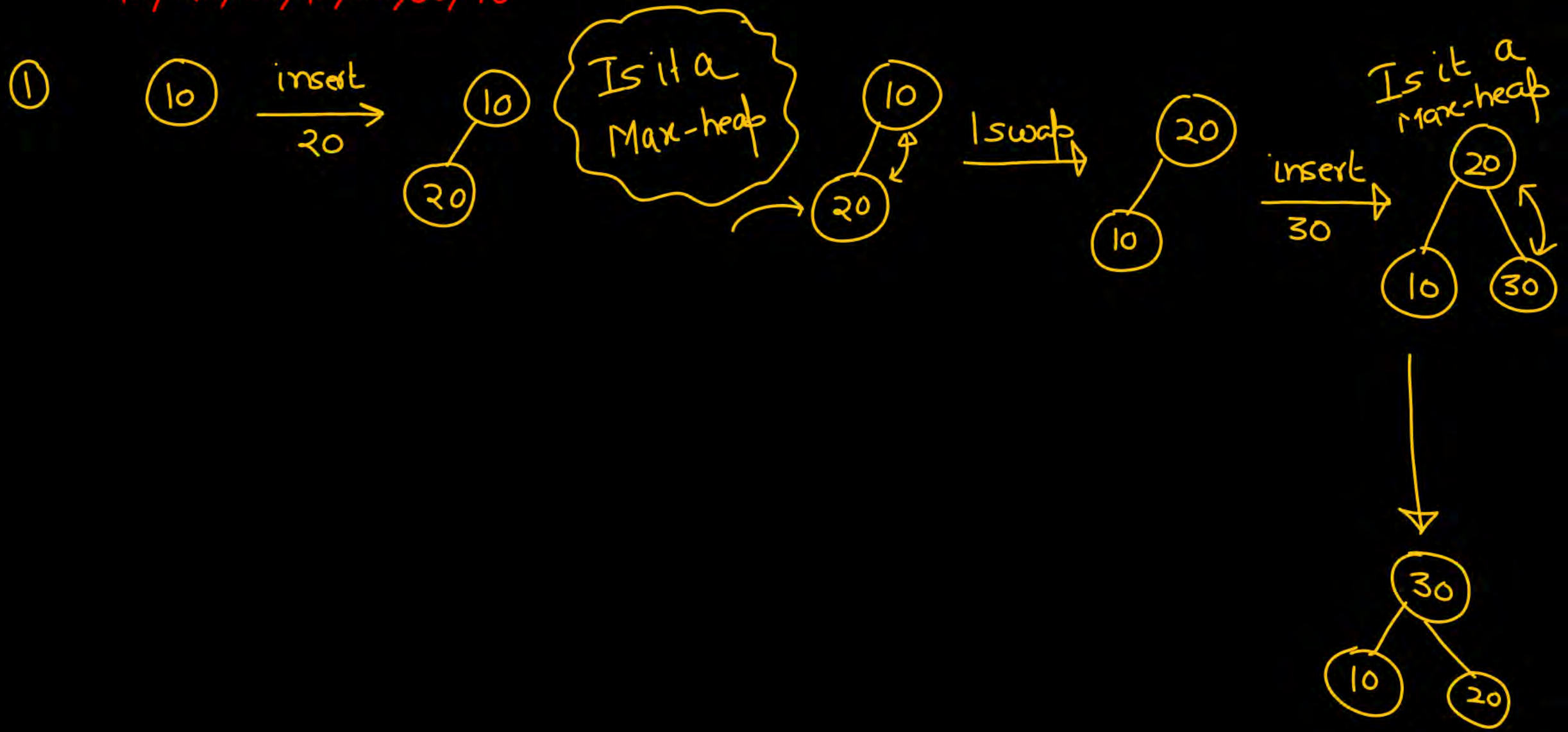
Max-heap ✓
Min-heap ✗


20
10
15
6
8
12
CBT
20
15
10
8
6
2
20
1
5
7
0
4
3
Max-heap
Max-heap
Max-heap

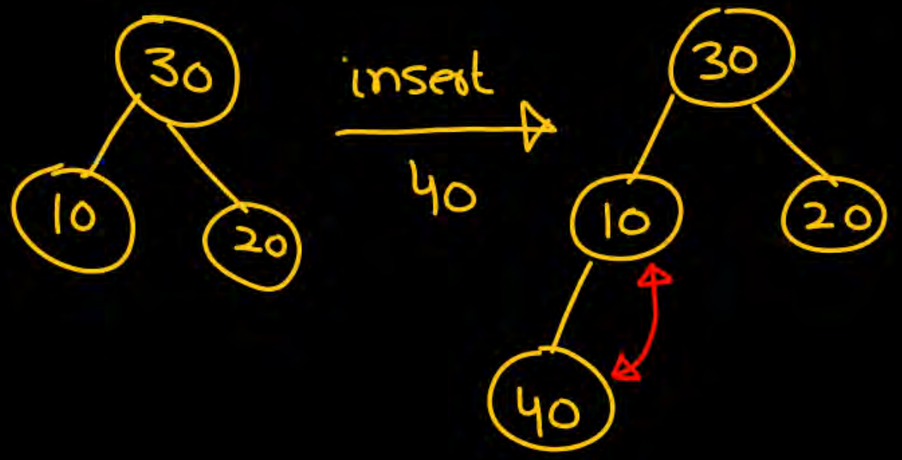
① Construction of heap by inserting keys one after another in a given order.

Const. a max-heap by inserting keys 10, 20, 30, 40, 50, 60, 70.

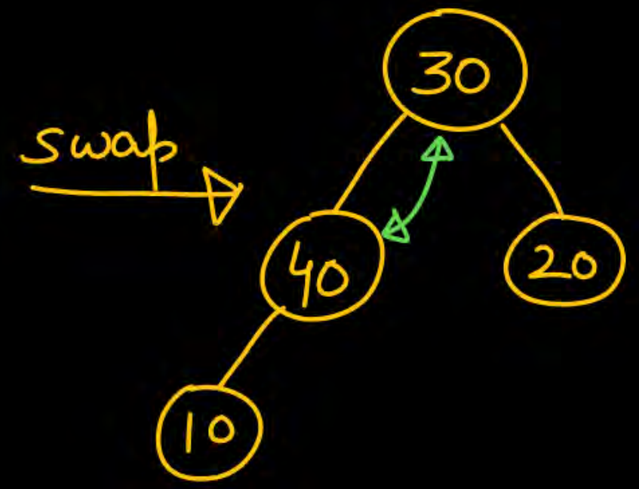
10, 20, 30, 40, 50, 60, 70



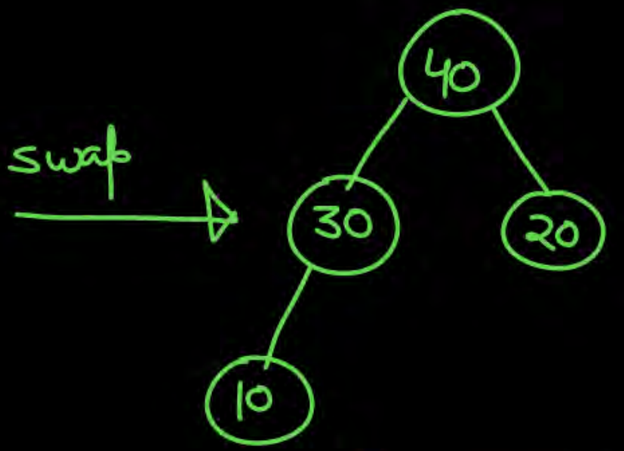
10, 20, 30, 40, 50, 60, 70

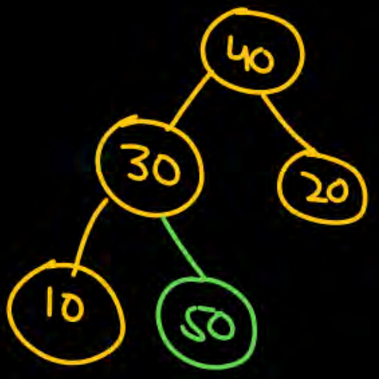


Is it a max-heap?
No



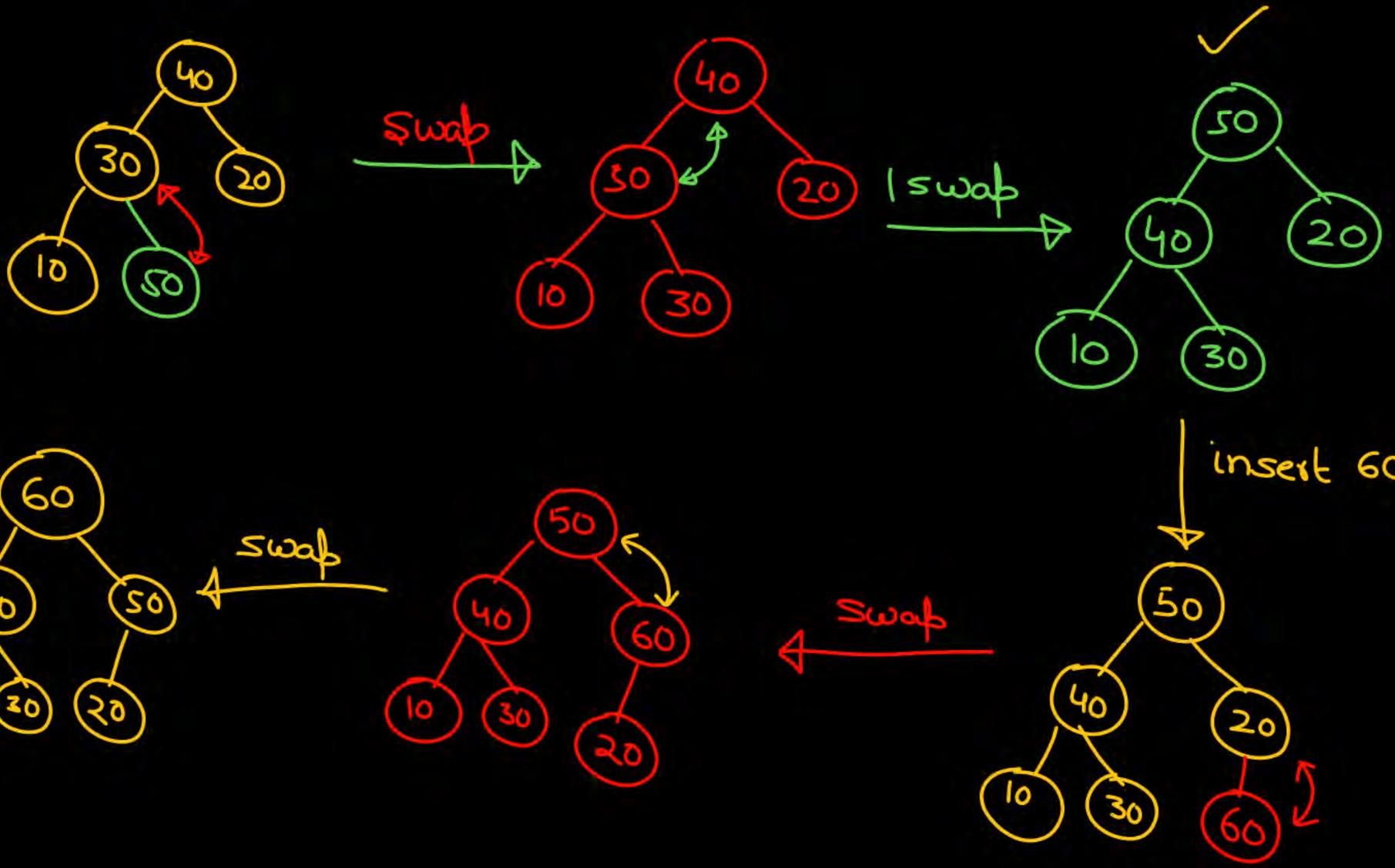
Is 40 at its correct position?





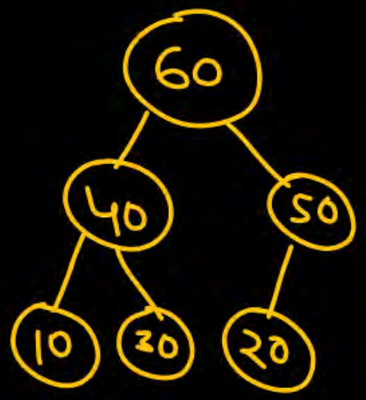
Is it a max-heap?
No

10, 20, 30, 40, 50, 60, 70

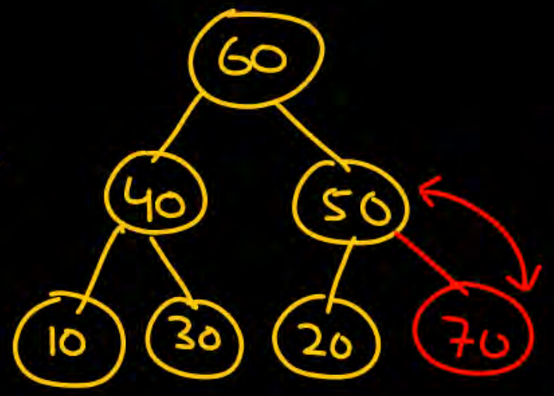


Is it a max-heap
No

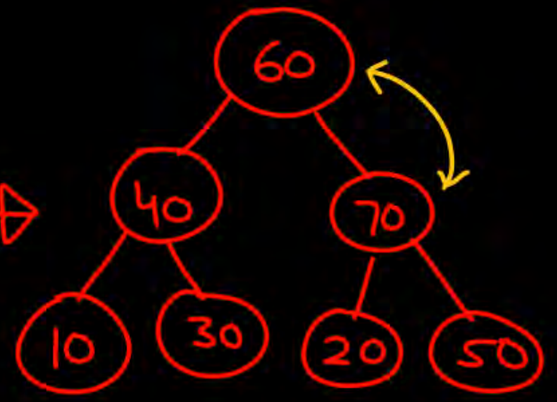
10, 20, 30, 40, 50, 60, 70



insert 70

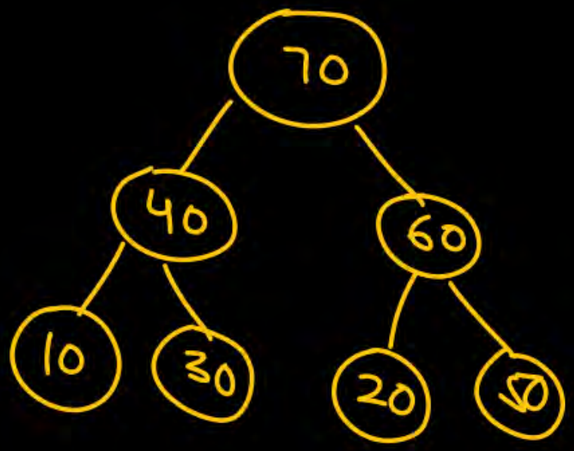


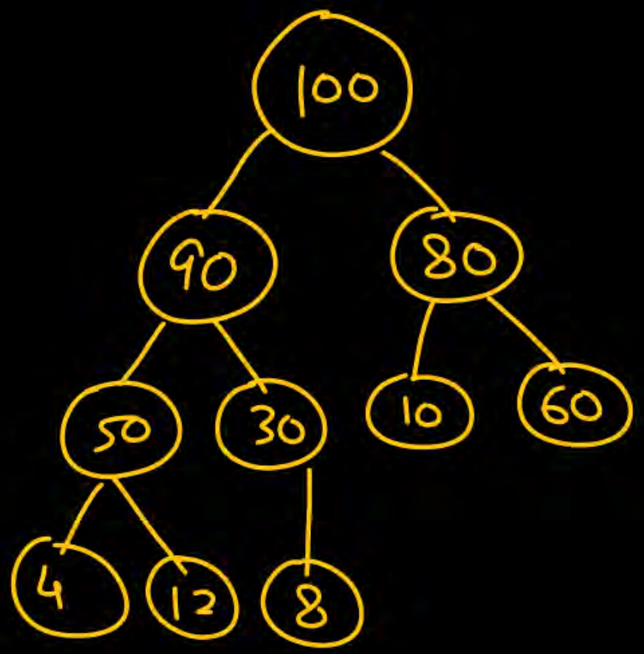
swap



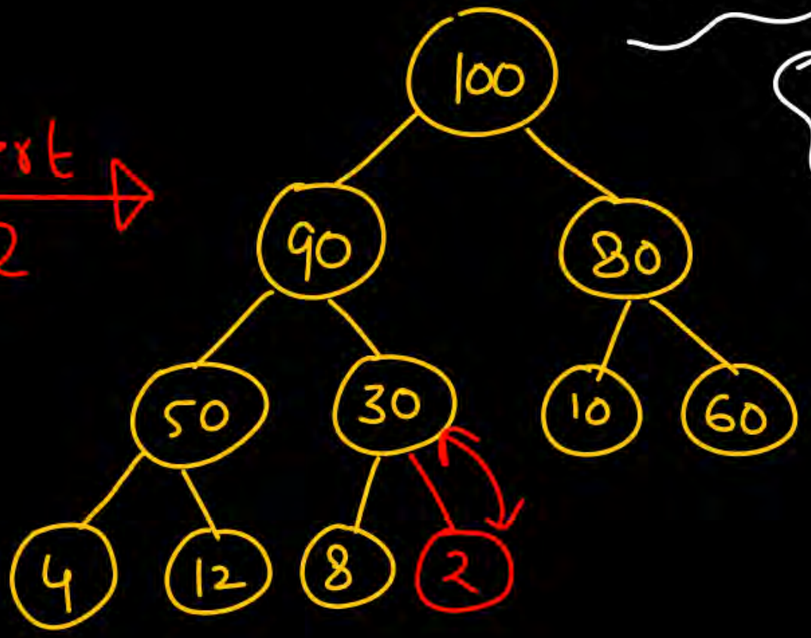
Is it a Max-heap ?

swap



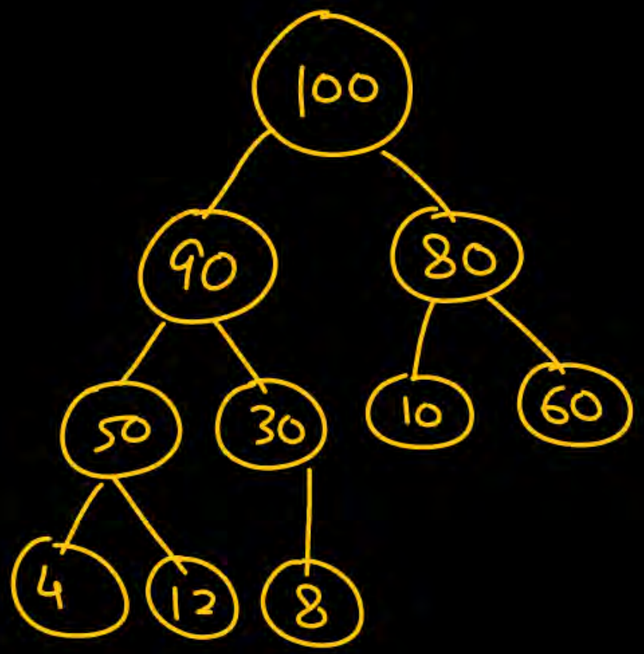


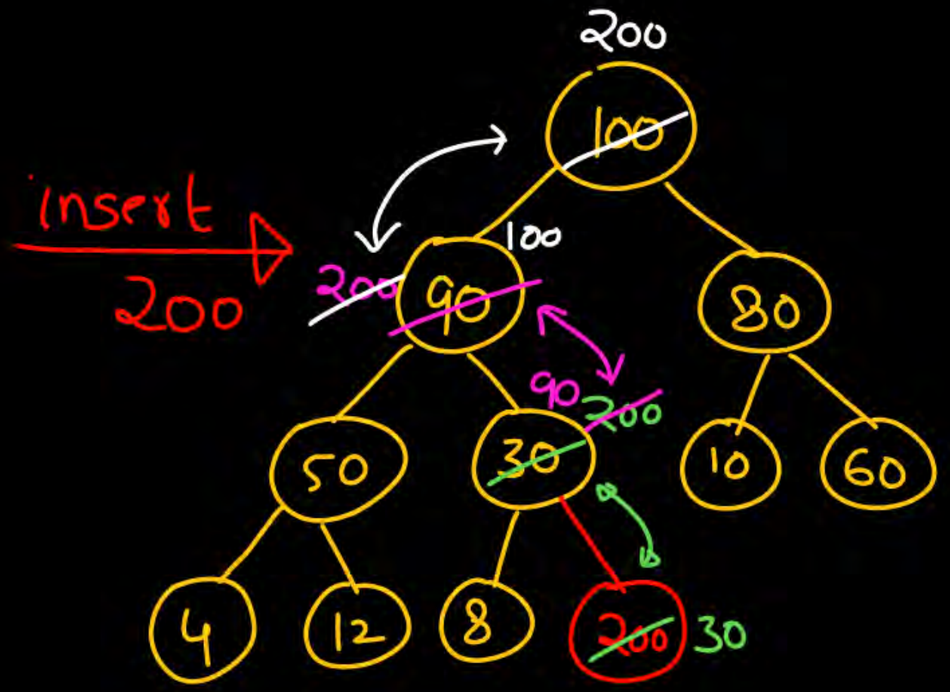
insert 2



best case

insert a key into an existing heap

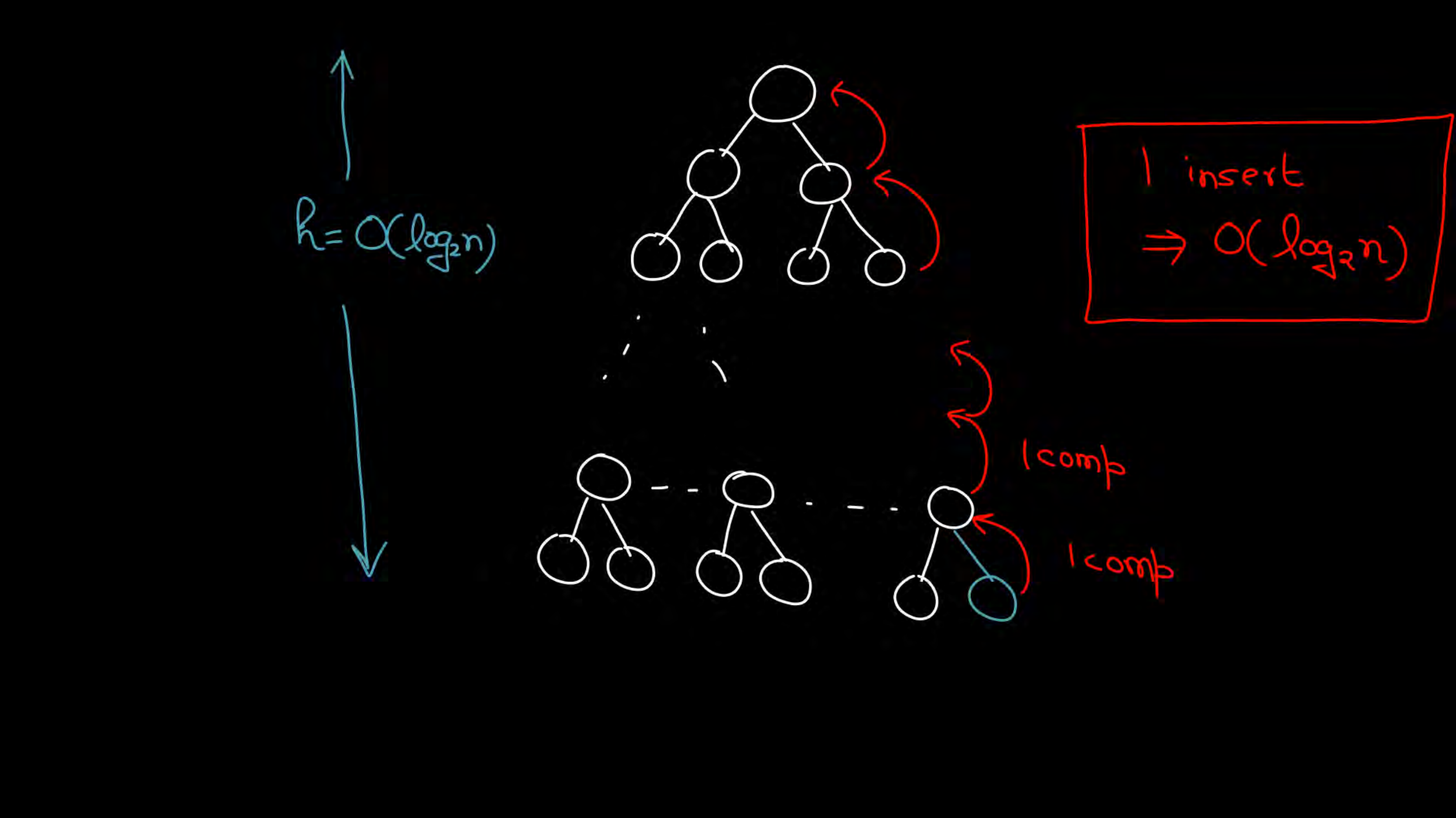




worst case

CBT

$h = O(\log_2 n)$



1 insert
$\Rightarrow O(\log_2 n)$

Heap Const. by inserting n keys - - —

Construction of Heap by inserting keys one after another in a given order

$$= n \times \log_2 n$$

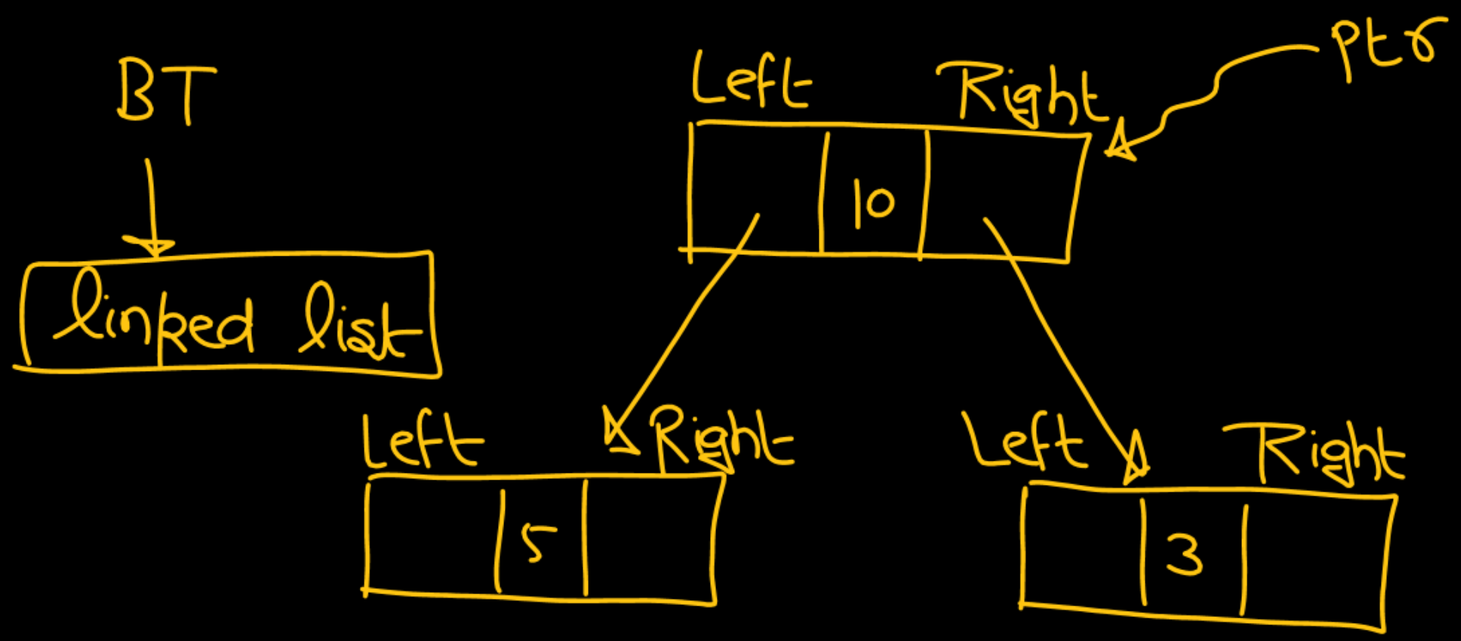
$$= O(n \cdot \log_2 n)$$

① Build-heap method

② Heapify algo.

③ Given a array representing a CBT, convert it to a max-heap

CBT

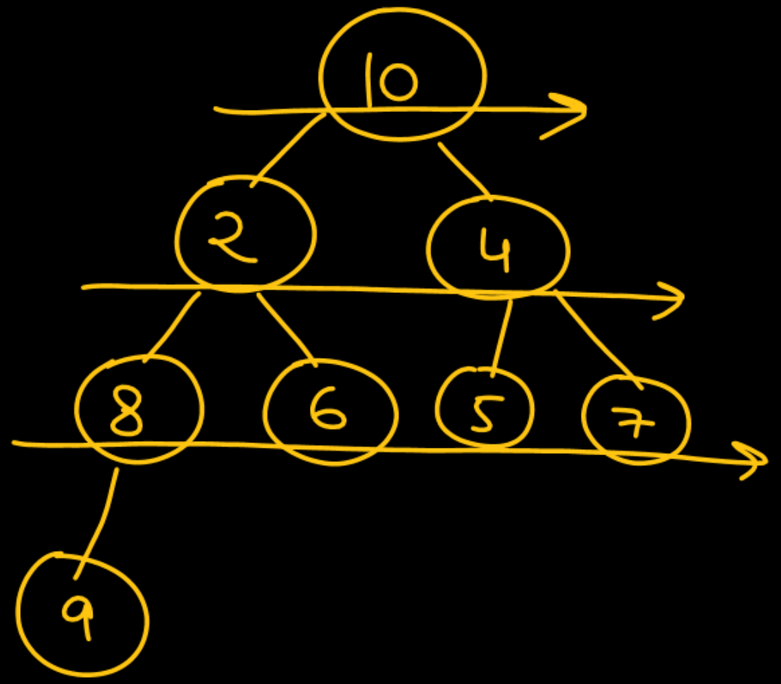


data

Left child

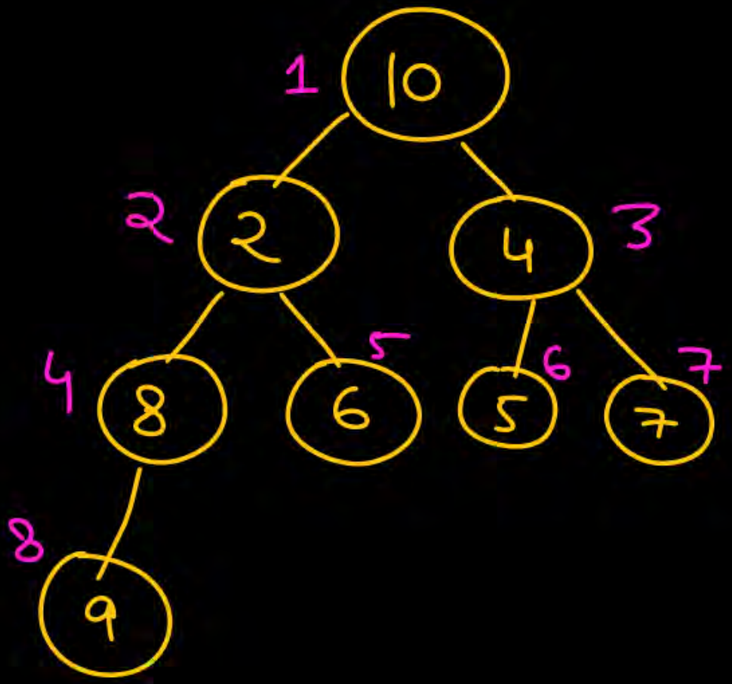
Right child

Array

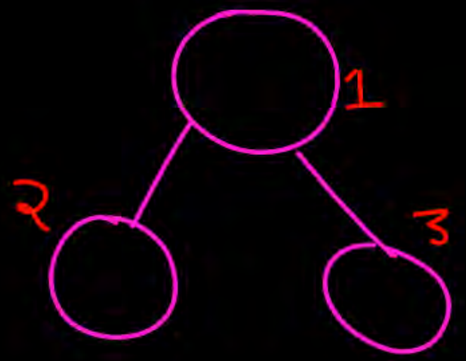


10	2	4	8	6	5	7	9
1	2	3	4	5	6	7	8

Array



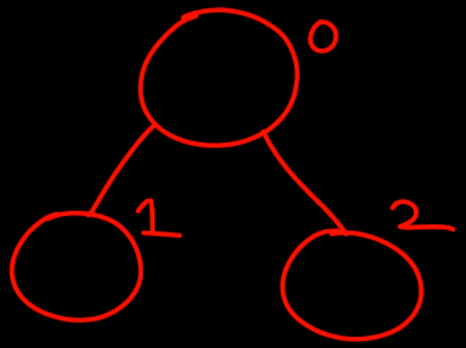
10	2	4	8	6	5	7	9
1	2	3	4	5	6	7	8

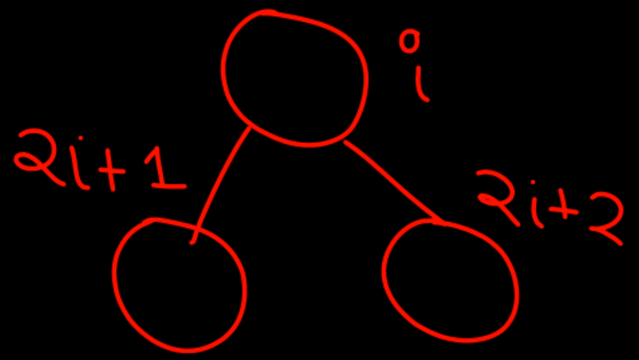


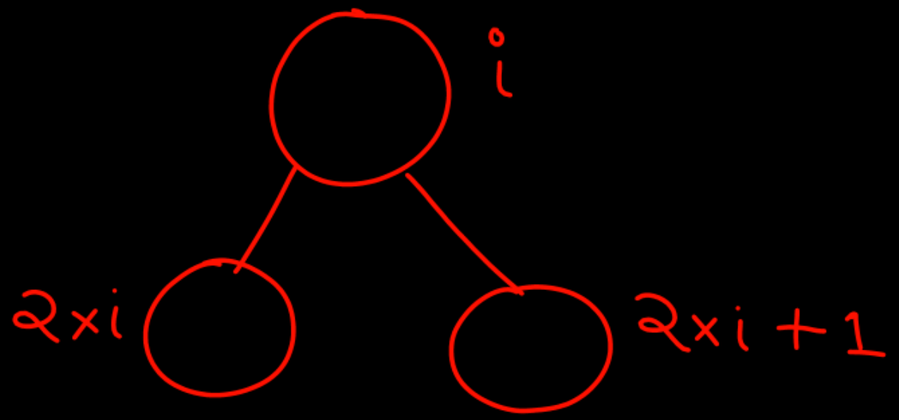
index of node with value 10 = 1

" " " " " 2 = 2

Prog:



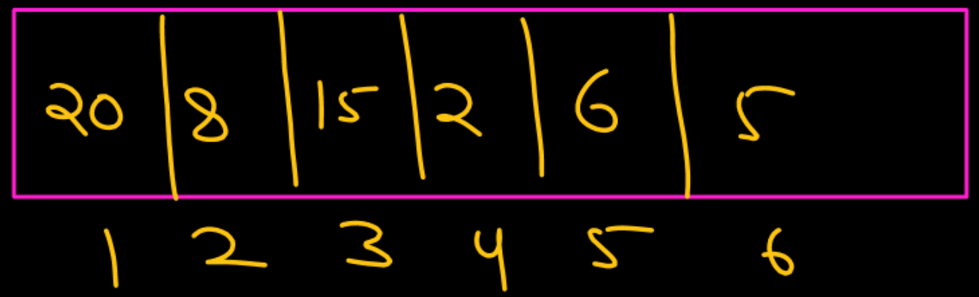


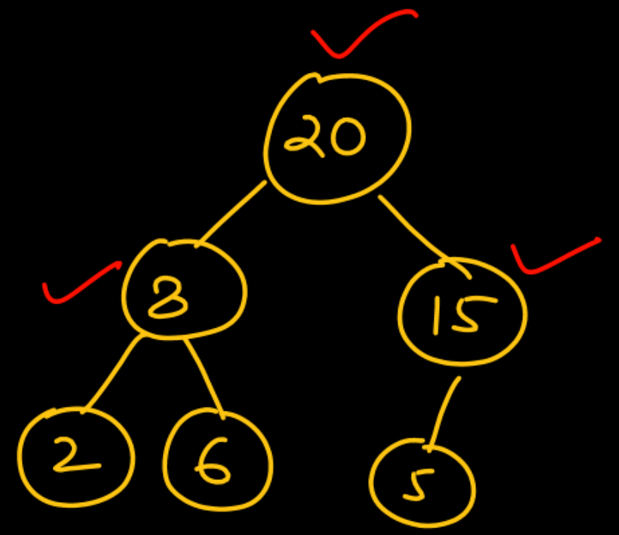


Given an array rep. a CBT :

20, 8, 15, 2, 6, 5

Is it rep. a max-heap

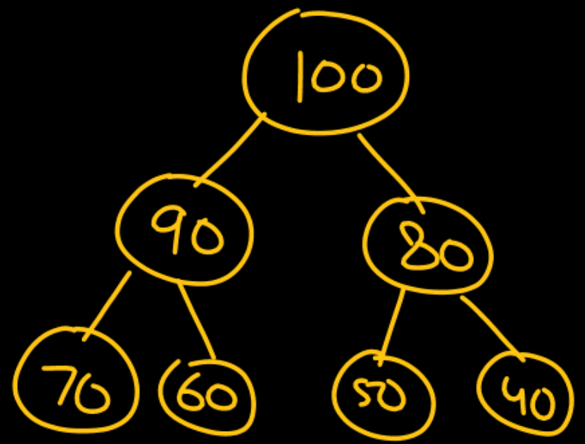




Yes

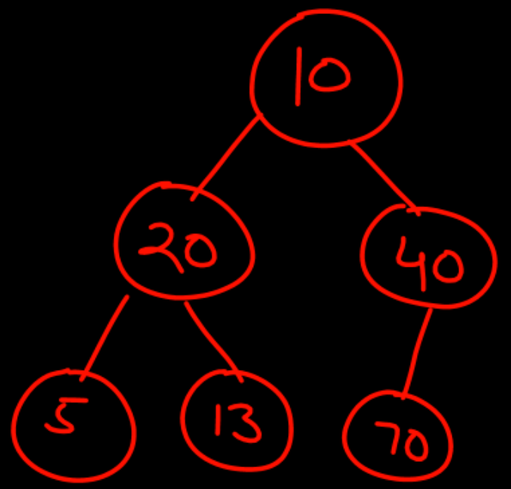
Consider the array rep. a CBT : 100, 90, 80, 70, 60, 50, 40

Is it a max-heap?



Given an array of a CBT : 10, 20, 40, 5, 13, 70

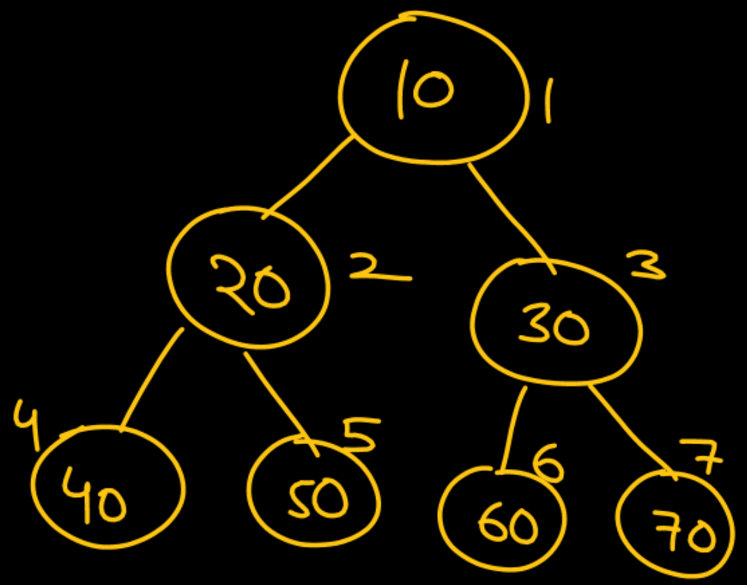
Is it a max-heap?



No

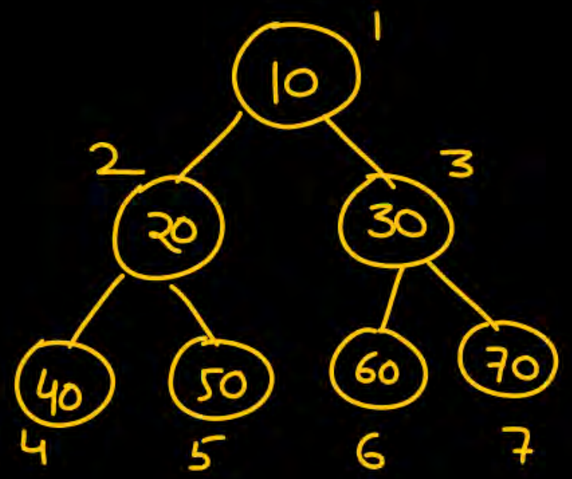
Q. An array rep. a CBT, : 10, 20, 30, 40, 50, 60, 70

Convert it into a max-heap.



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


10 | 20 | 30 | 40 | 50 | 60 | 70
Sort
70, 60, 50, 40, 30, 20, 10
Heap
O(n log n)

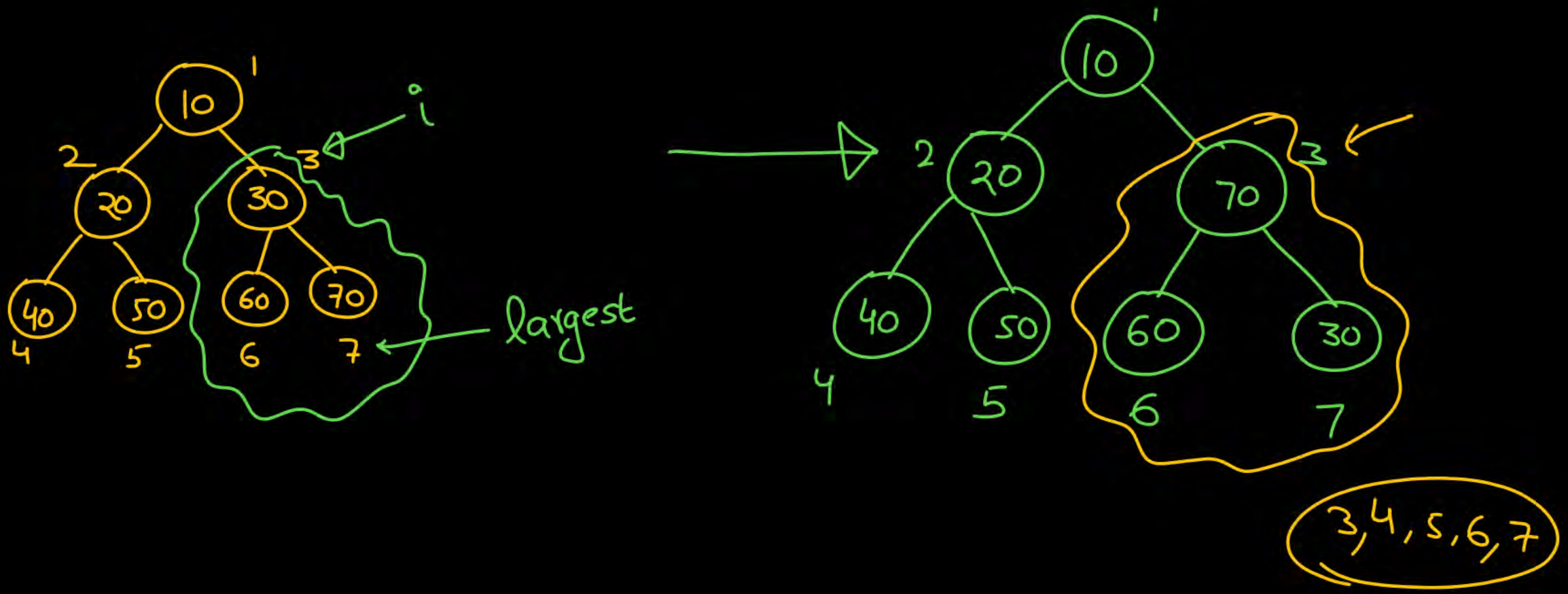


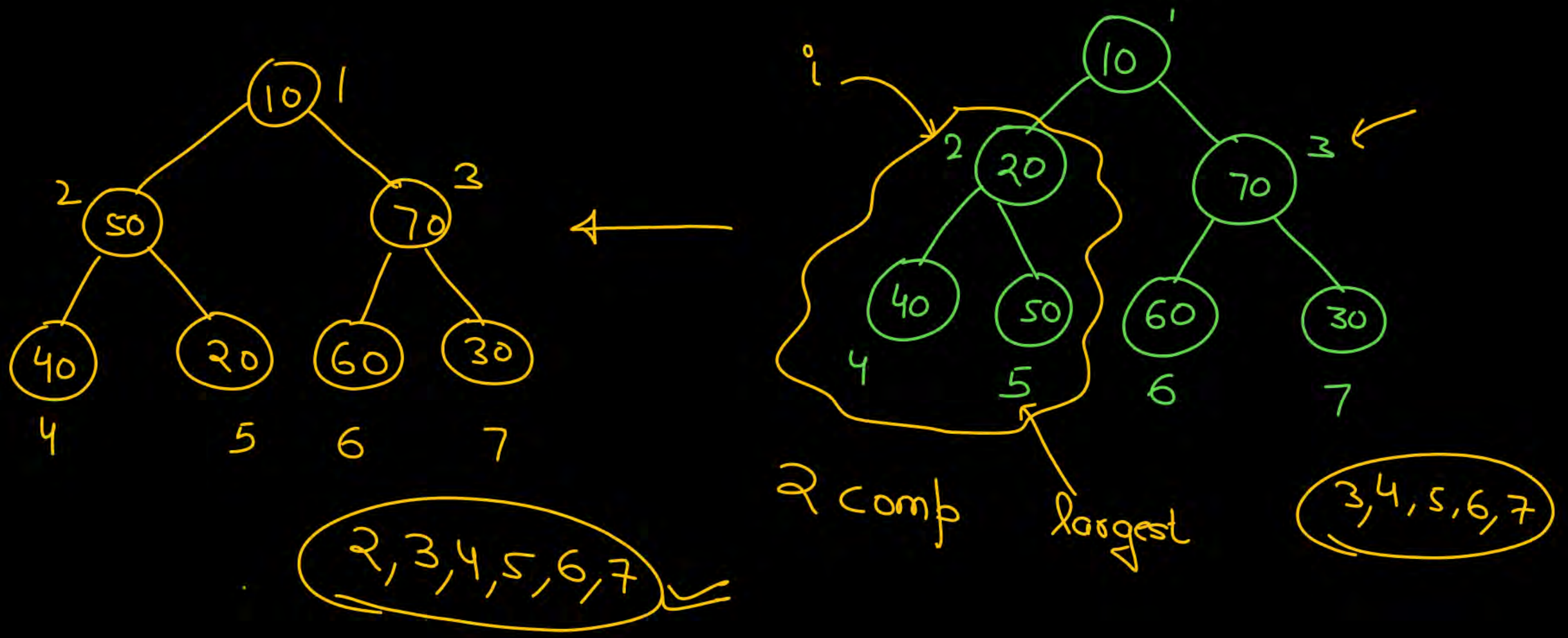
4, 5, 6, 7 ✓

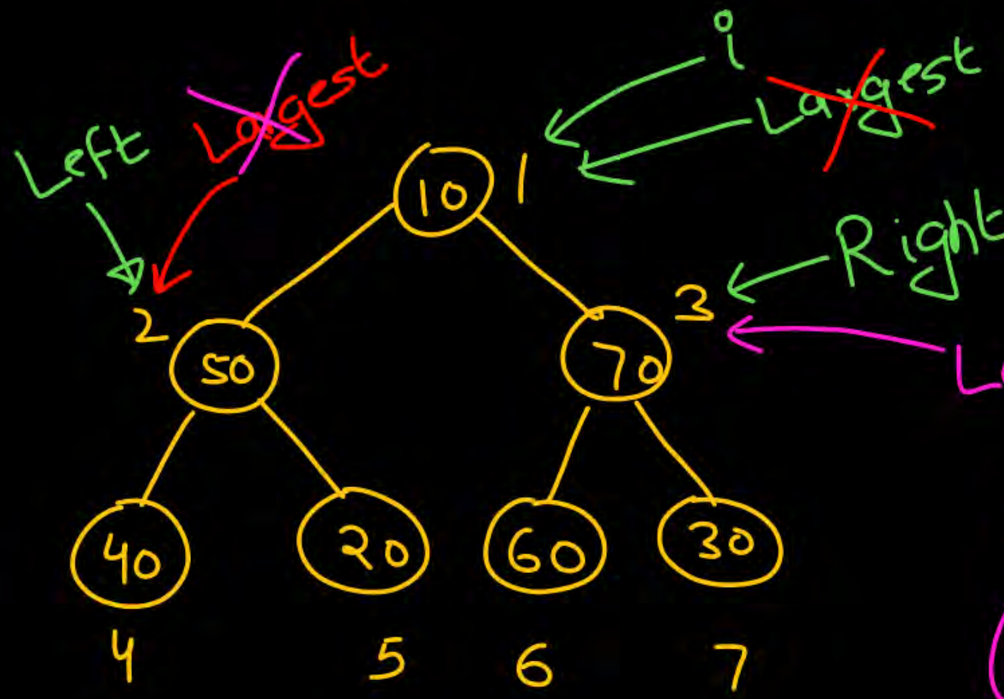
All leaf node satisfy heap property

index of inter nodes ⇒ (1, 2, 3)

index of internal node ⇒ 1 to $\left\lfloor \frac{n}{2} \right\rfloor$


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


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



Heapify (A, 7, 1) (n, i)

1. Left = 2*i , Right = 2*i+1 ,
   Largest = i

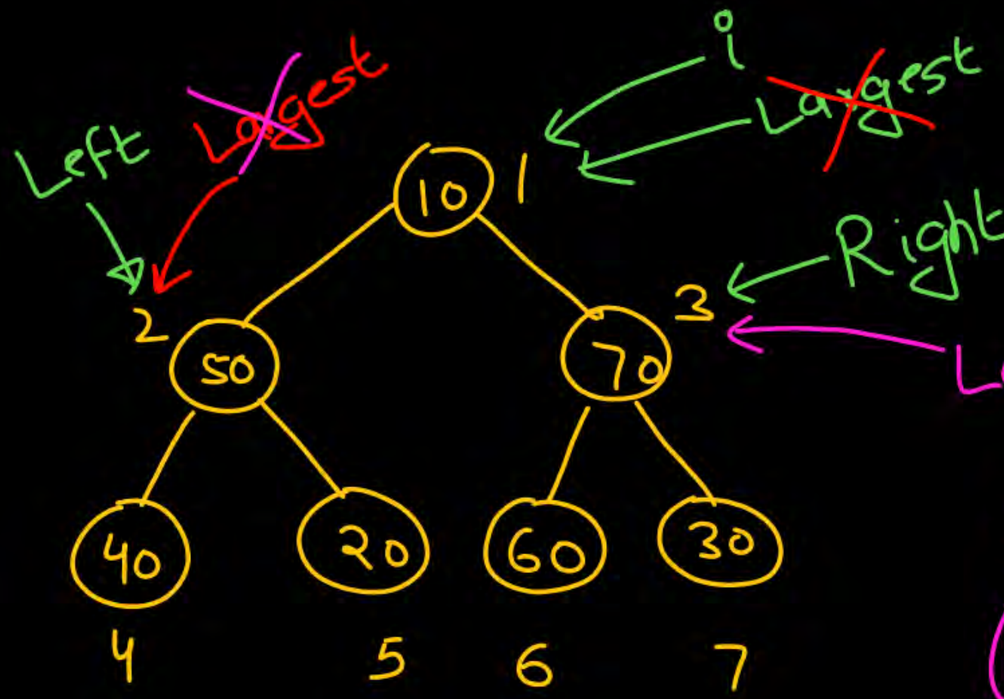
ये Exact Algo नहीं है

A[Largest] < A[Left]

Largest = Left

A[Largest] < A[Right]

Largest = Right



Heapify (A, 7, 1)
n i

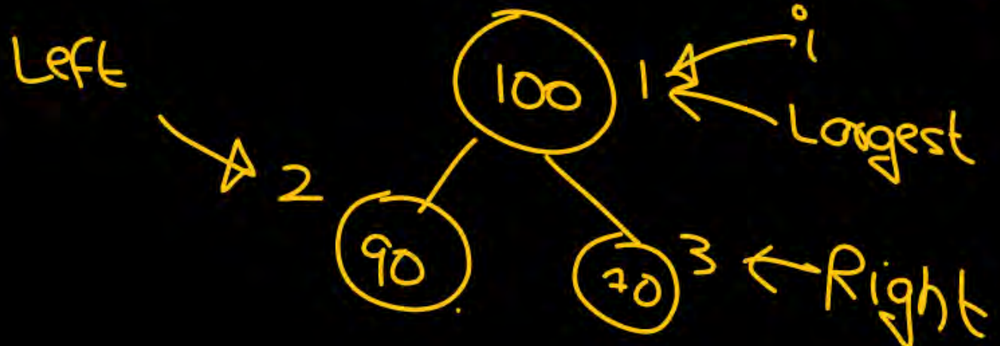
1. Left = 2*i , Right = 2*i+1 ,

Largest = i

ये Exact Algo नहीं है

A[Largest] < A[Left]

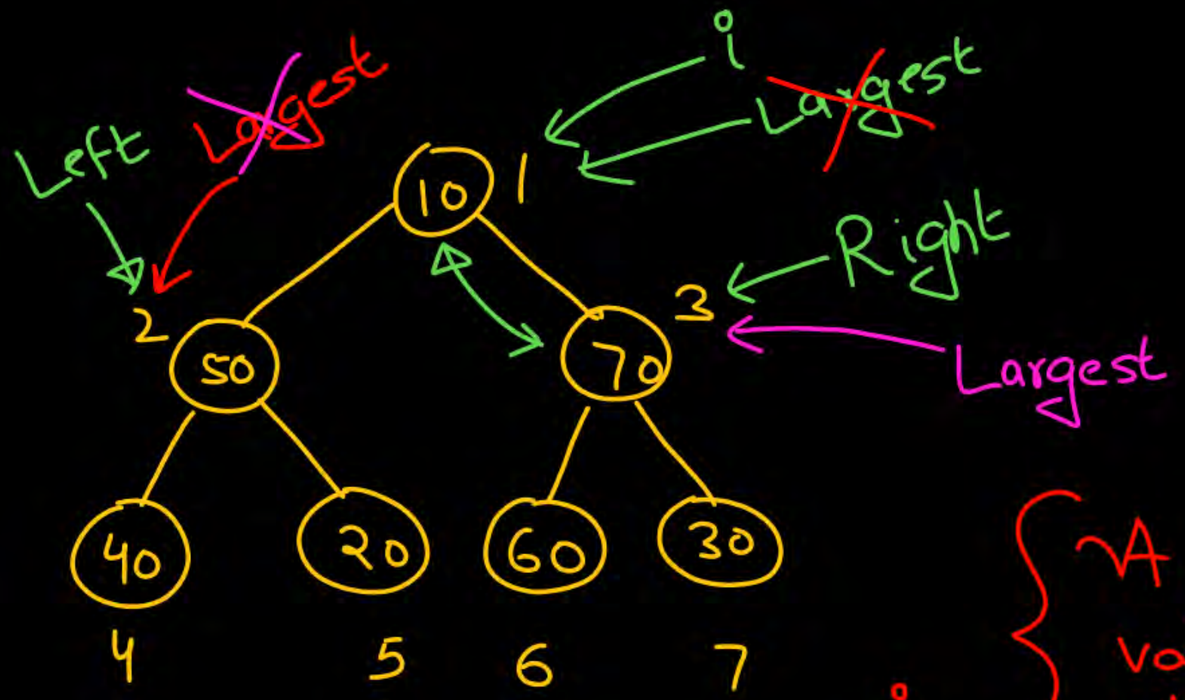
Largest = Left



A[Largest] < A[Right]

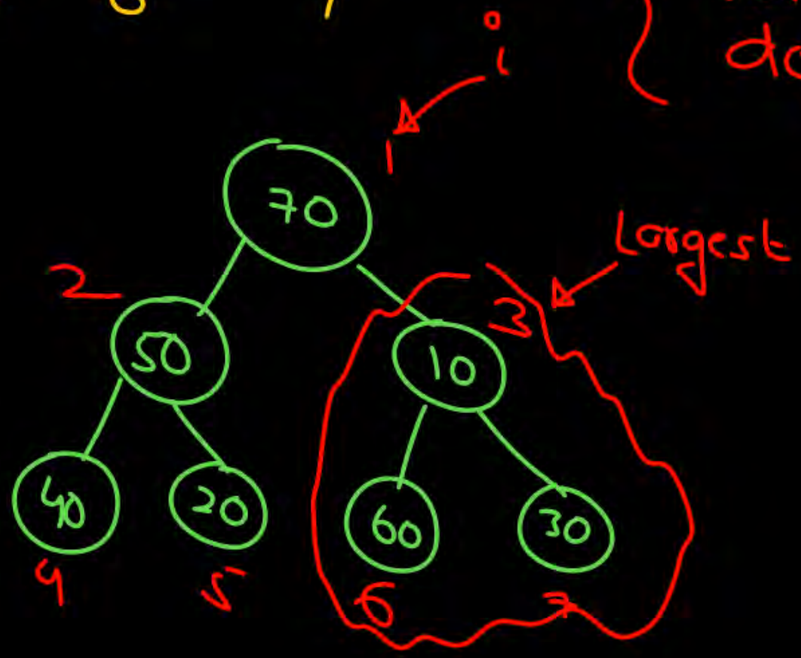
Largest = Right

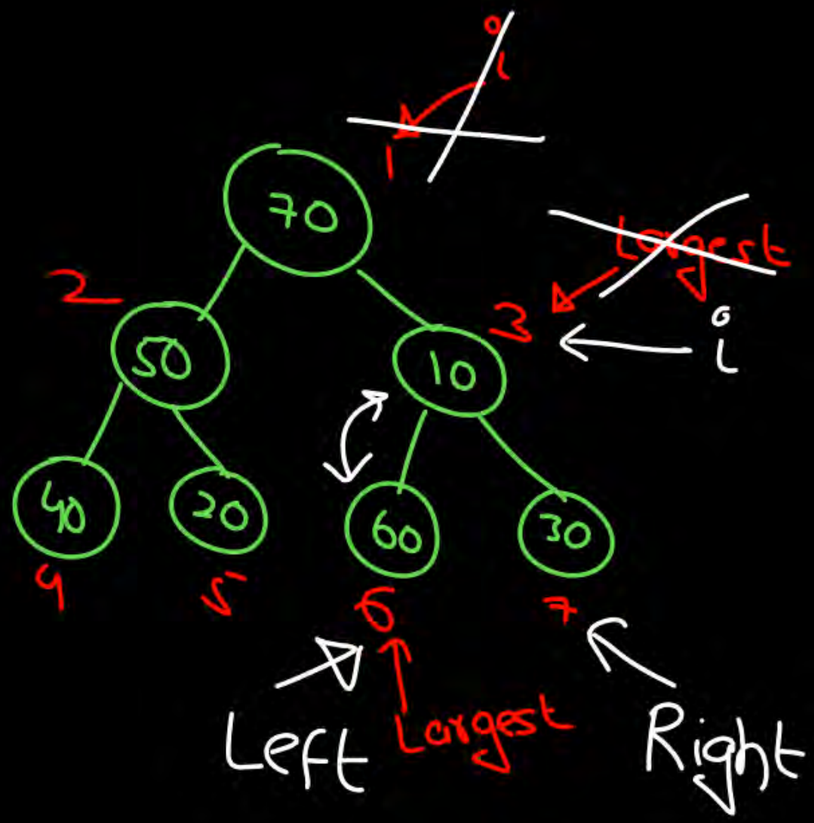
if (i != Largest)




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

{A smaller value came down }





{A smaller value came down }

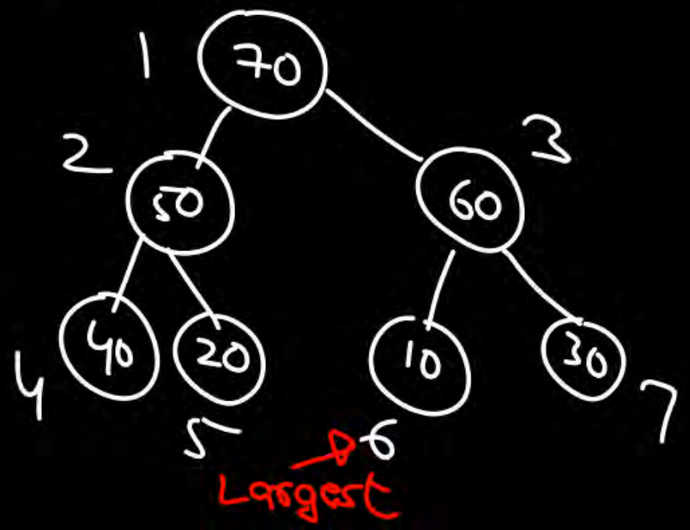
```
if ( i != Largest)
  {
  swap (A[i], A[Largest]);
  Heapify (A, n, Largest);
```

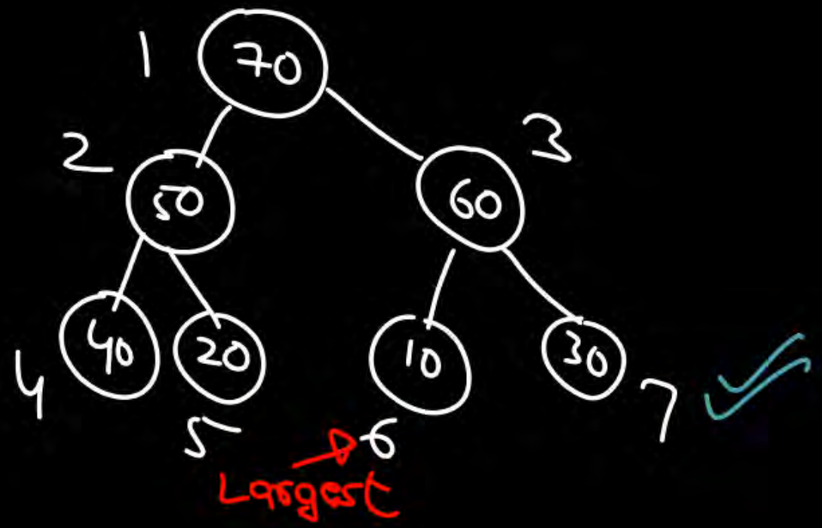
Heapify (A, 7, 6)

i

Left = 12

Right = 13






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

i = 6

Largest = 6

Left = 12, Right = 13

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

```
                    i
Heapify(A, 7, 6)
```

Left = 12

Right = 13

Telegram

4 am

```
Heapify (A, i, n)
1. Left = 2i , Right = 2i+1, Largest = i ;
2. if (Left <= n && A[Largest] < A[left])
                Largest = Left ;
3. if (Right <= n && A[Largest] < A[Right])
                Largest = Right;

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


THANK YOU GW
SOLDIERS !