CS & IT ENGINEERING

Data Structures

Tree

Lecture No.- 07

By- Pankaj Sharma Sir

Topics to be Covered

Topic	Tree Part-07

Heap

Recap of Previous Lecture

Topic Tree Part-06

AVL tree questions

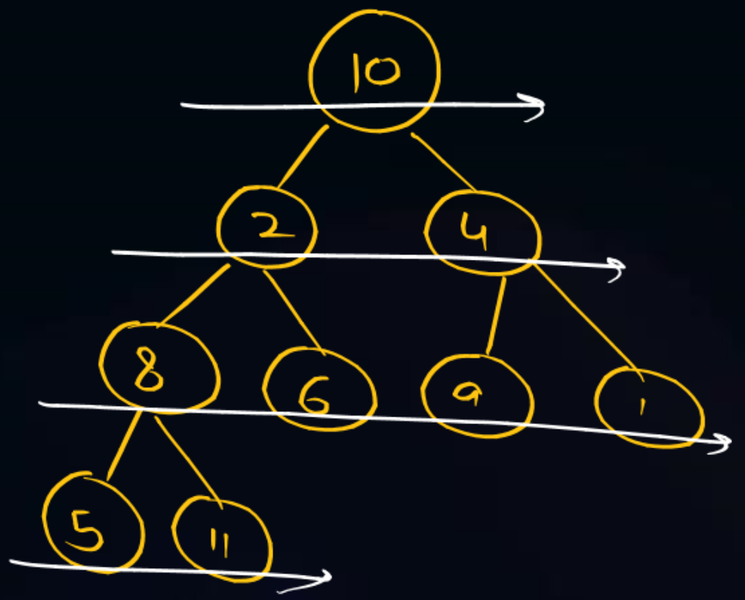
Heap → max heap
Min heap

Const. Heap by inserting keys one after another — $O(n \cdot \log n)$

Insert a key → $O(\log_2 n)$

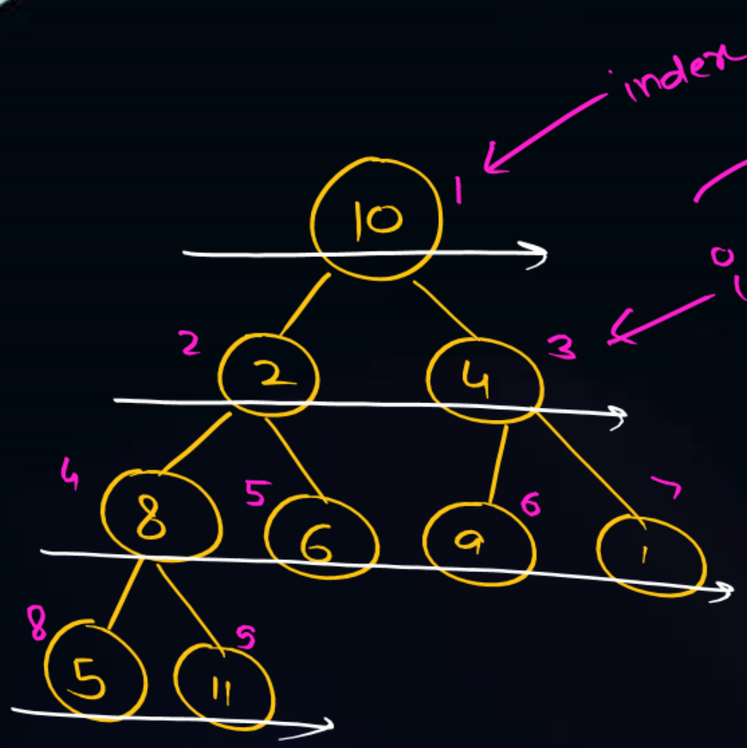
CBT → array representation

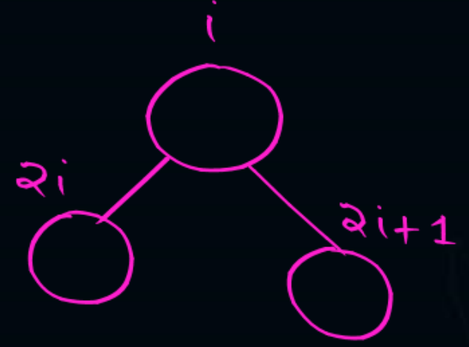
Topic : Tree



10	2	4	8	6	9	1	5	11	
1	2	3	4	5	6	7	8	9	

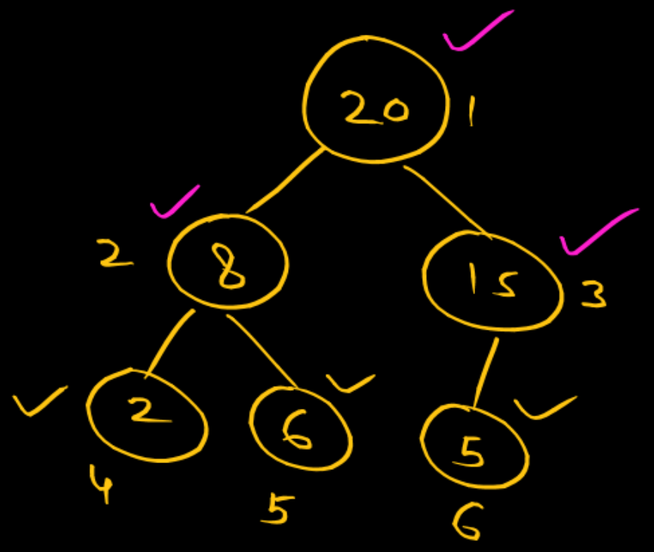
Topic : Tree

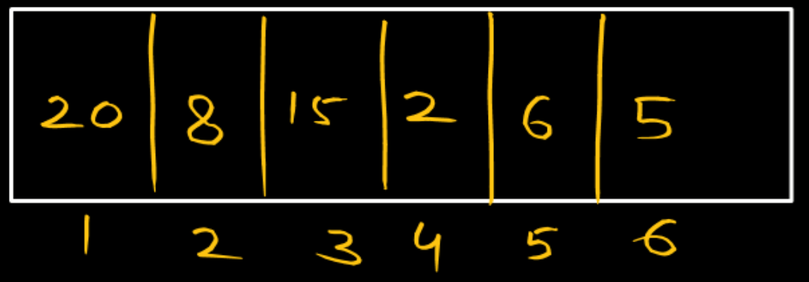




Given an array representing a CBT : 20, 8, 15, 2, 6, 5

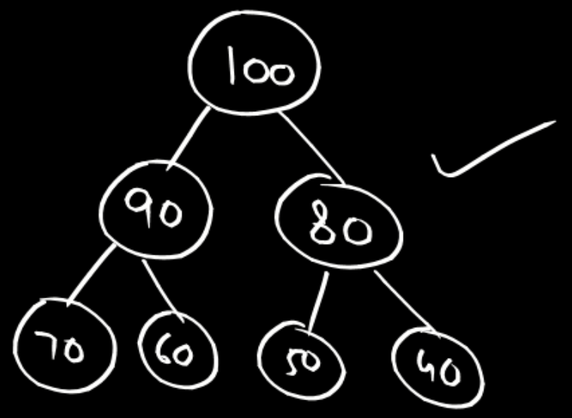
Is it rep. a max heap?





Given an array representing a CBT : 100, 90, 80, 70, 60, 50, 40

Is it rep. a max heap?



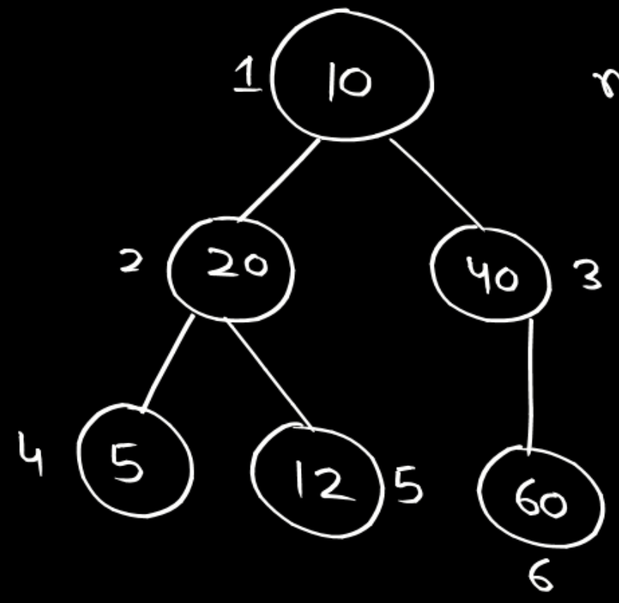
100	90	80	70	60	50	40
1	2	3	4	5	6	7

array decreasing sorted → always rep a max heap

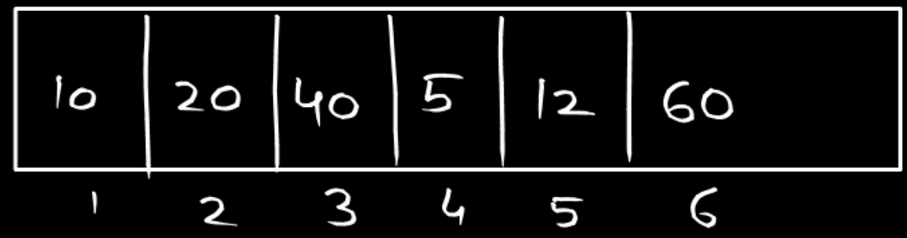
array inc. sorted → always rep. a min heap

Given an array representing a CBT : 10, 20, 40, 5, 12, 60

Is it rep. a max heap?

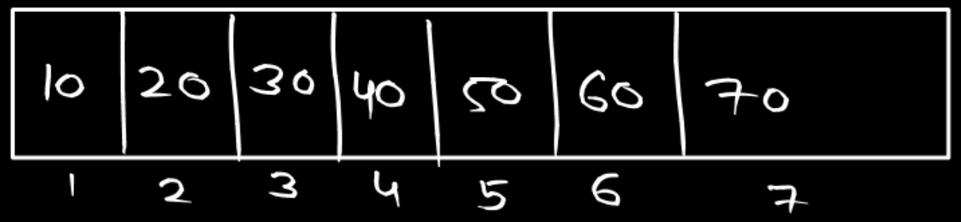


not a max heap



Given an array representing a CBT : 10, 20, 30, 40, 50, 60, 70

Convert it into max-heap.



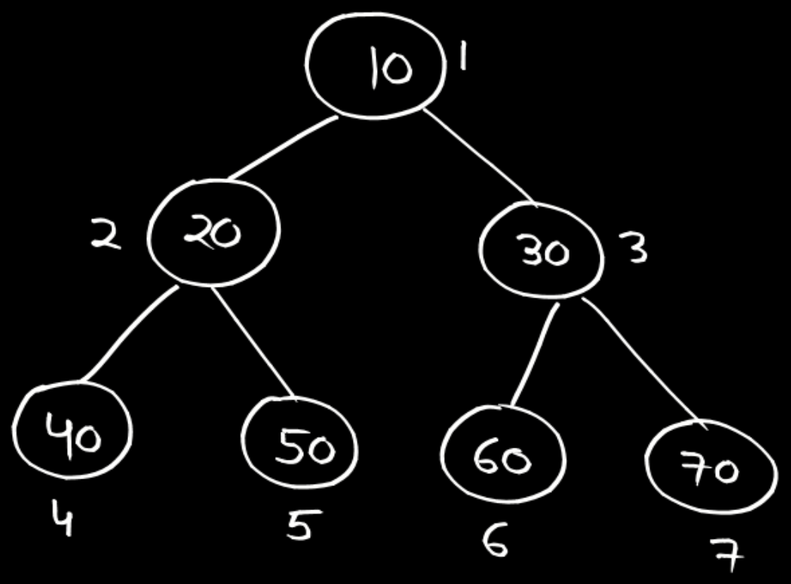
Sort → 70, 60, 50, 40, 30, 20, 10

$O(n \log n)$

better ?

Build_heap method

↳ Heapify algorith

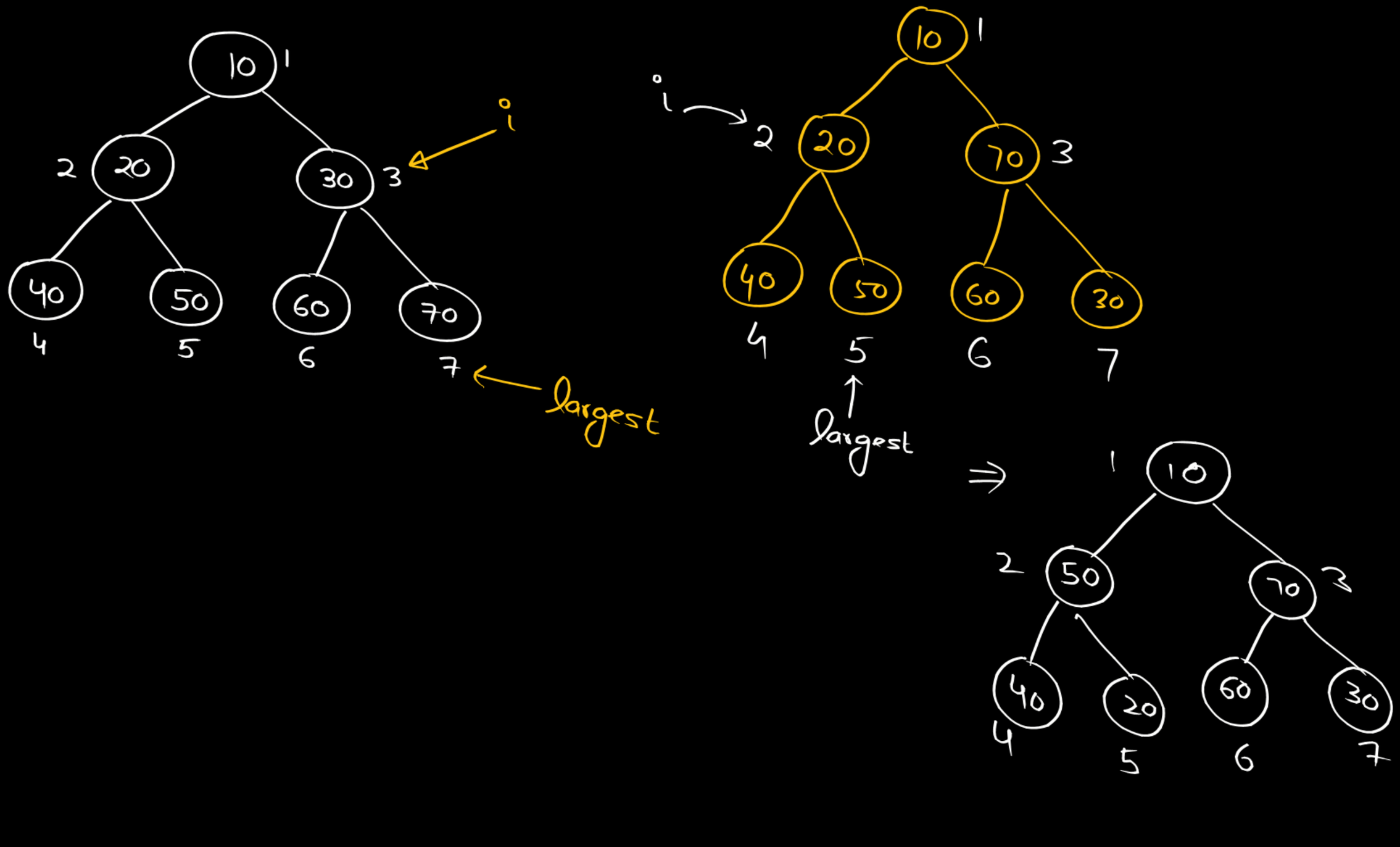


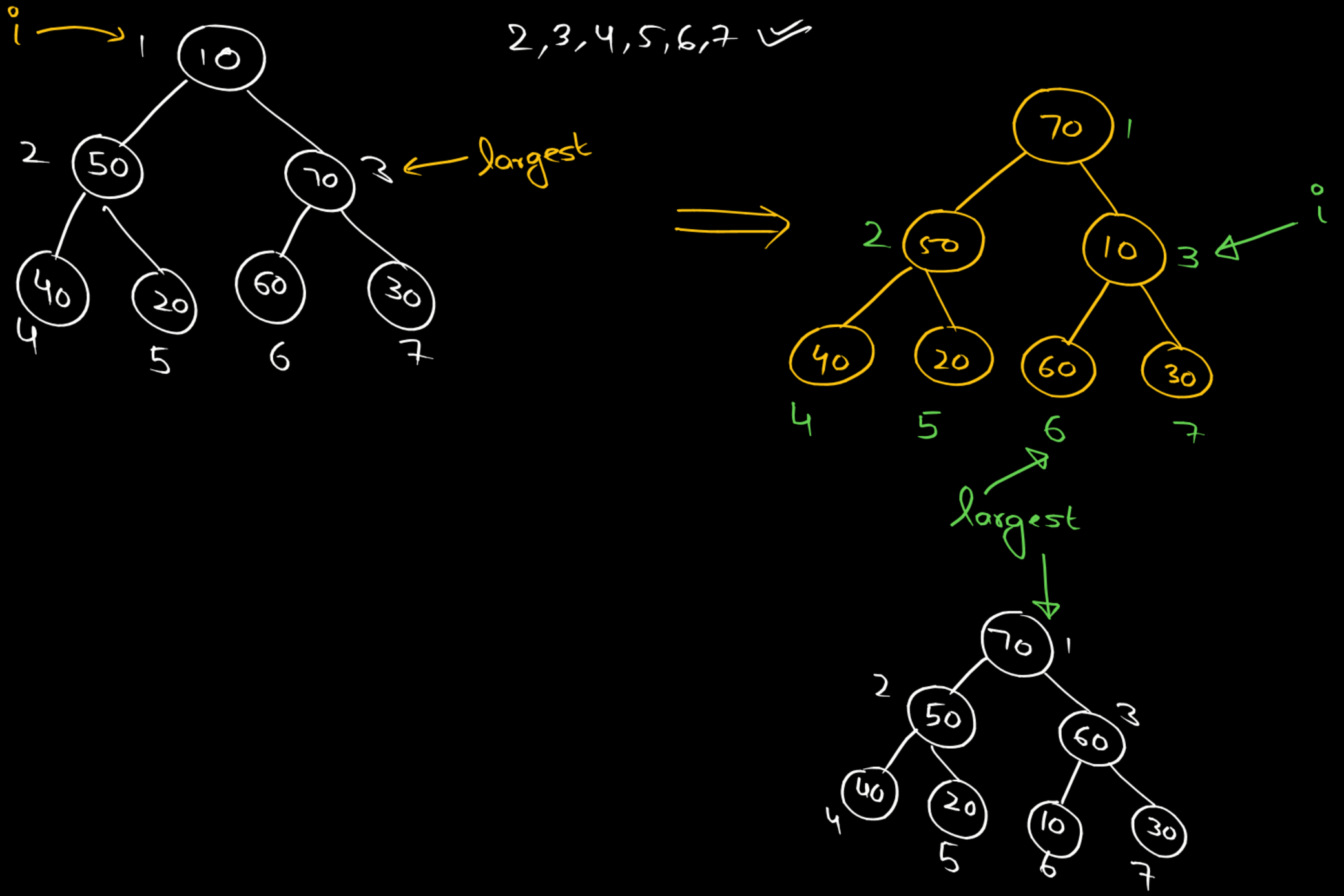
index of internal node = 1 to $\lfloor \frac{n}{2} \rfloor$

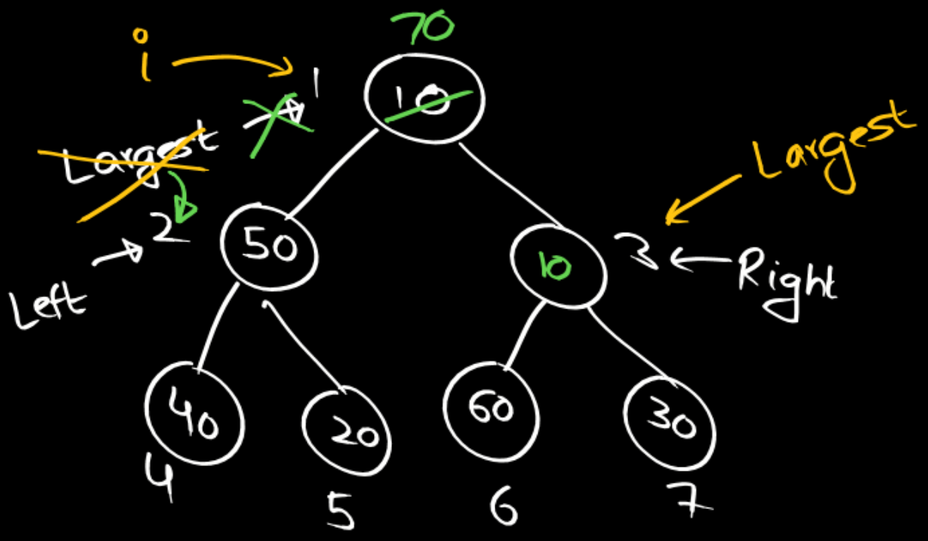
Every leaf node always satisfy max-heap property.

index – 4, 5, 6, 7

index of internal node = 1 to $\lfloor \frac{7}{2} \rfloor \Rightarrow 1, 2, 3$


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


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



Ye exact Algorithm नहीं है

Heapify (A, i, n)

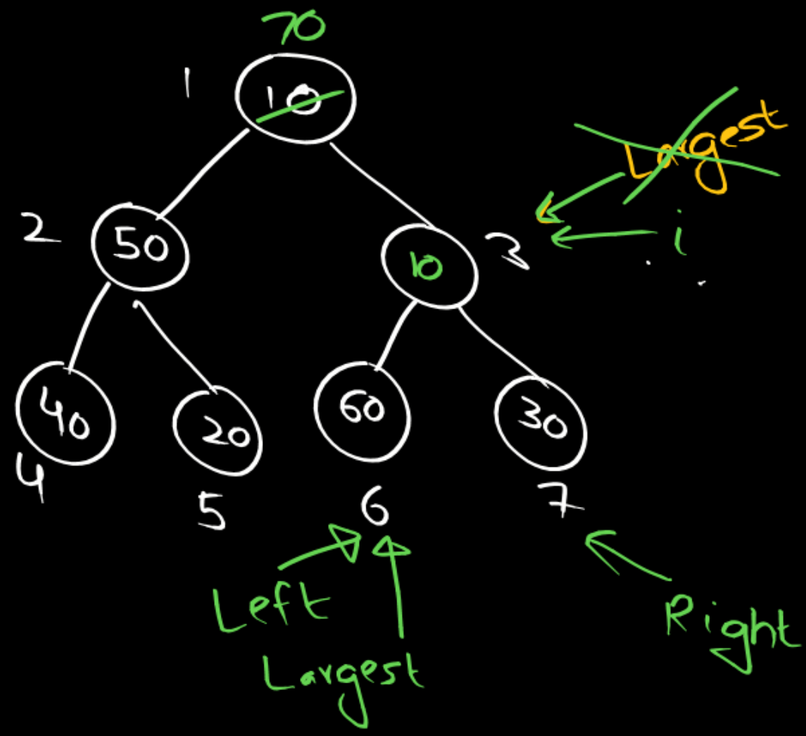
n ⇒ 7
i = 1

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

2.) if A[Left] > A[Largest]
Largest = Left ;

3.) if A[Right] > A[Largest]
Largest = Right ;

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



Heapify (A, i, n)

n ⇒ 7
i = 1

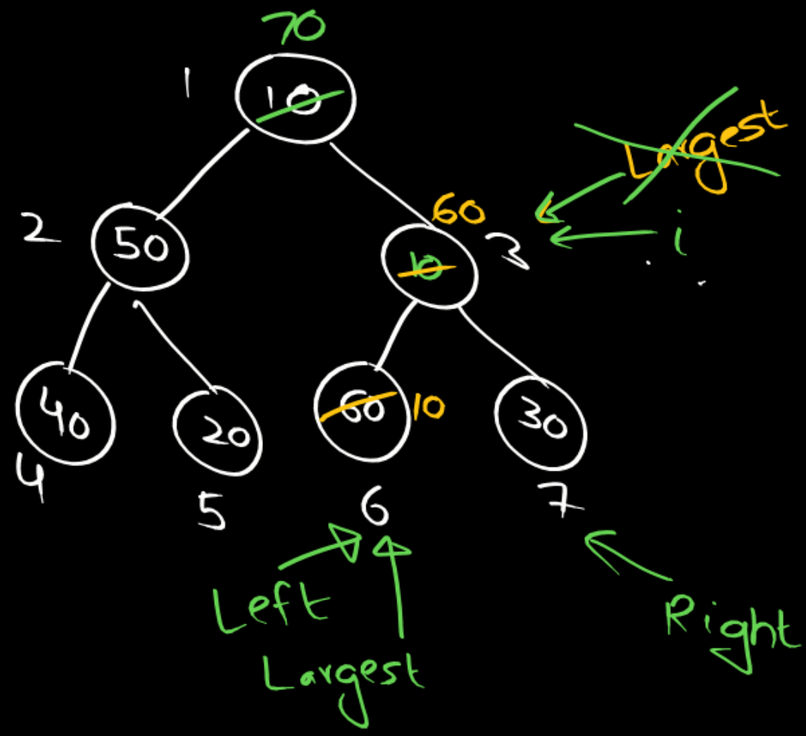
1.) Left = 2*i ;
Right = 2*i + 1 ;
Largest = i ;

2.) if A[Left] > A[Largest]
Largest = Left ;

3.) if A[Right] > A[Largest]
Largest = Right ;

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

Ye exact Algorithm नहीं है



Heapify (A, i, n)

n ⇒ 7
i = 1

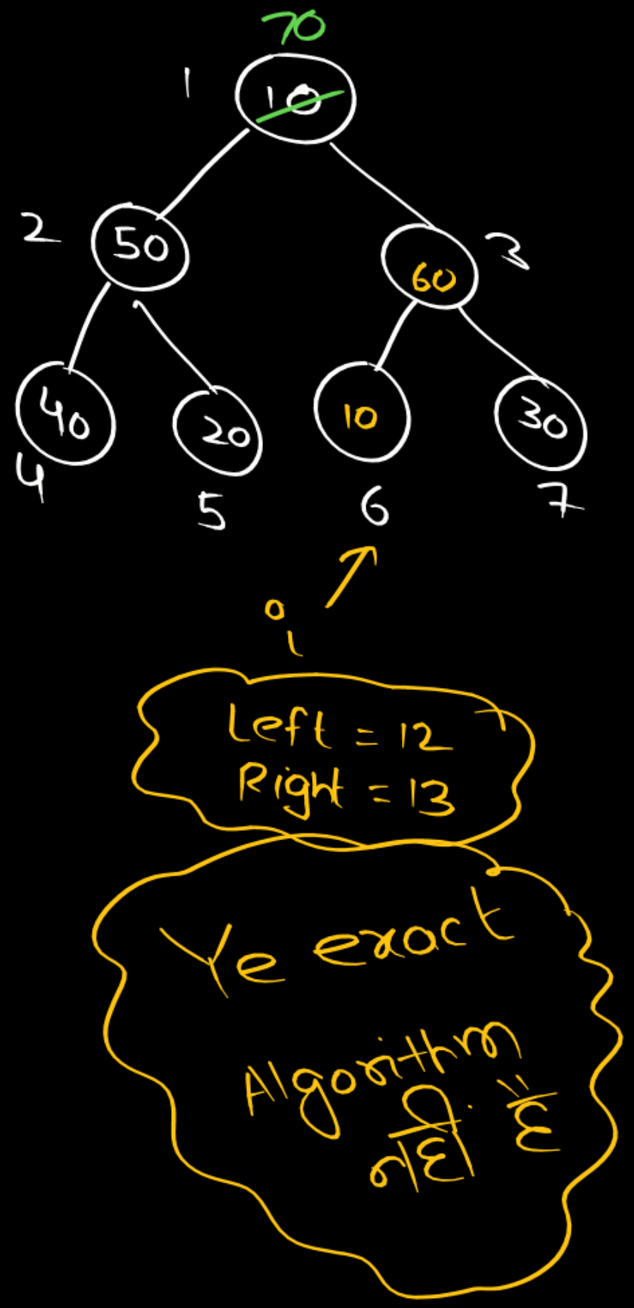
1.) Left = 2*i ;
Right = 2*i + 1 ;
Largest = i ;

2.) if A[Left] > A[Largest]
Largest = Left ;

3.) if A[Right] > A[Largest]
Largest = Right ;

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

Ye exact Algorithm नहीं है



Heapify (A, i, n)

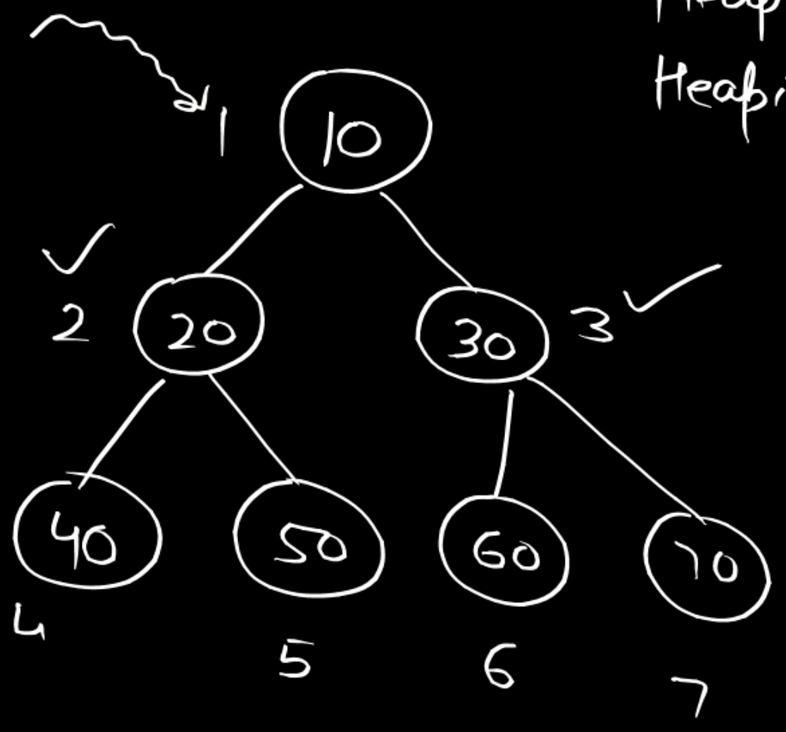
n ⇒ 7
i = 1

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

2.) if A[Left] > A[Largest]
Largest = Left ;

3.) if A[Right] > A[Largest]
Largest = Right ;

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



Heapify ⇒ 3
Heapify ⇒ 2
Heapify ⇒ 1

Heapify (A, i, n)

n ⇒ 7
i = 1

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

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

Largest = Left ;

if(Right<=n && A[Right] > A[Largest])

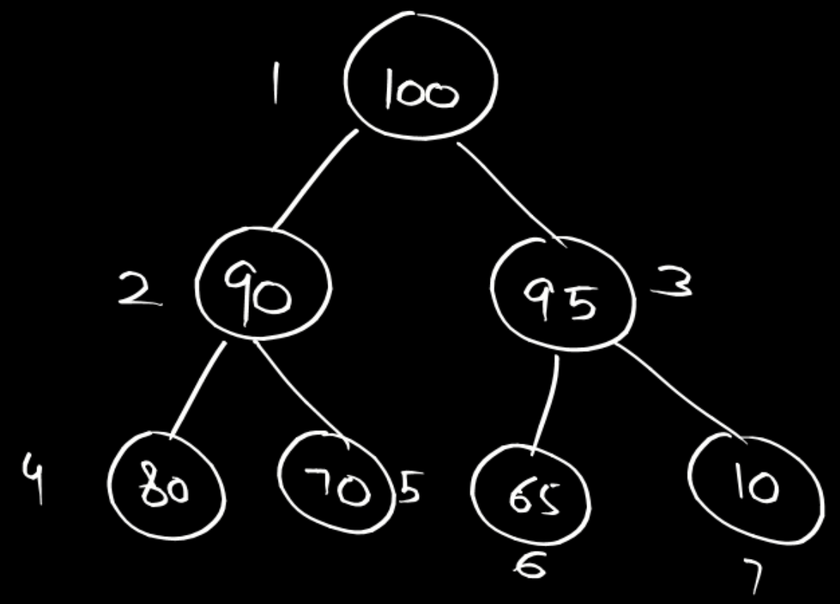
Largest = Right ;

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

Max_heap

Build - Heap $\Rightarrow O(n)$

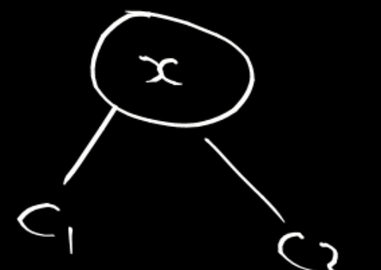
100	90	95	80	70	65	10
1	2	3	4	5	6	7



① Find_max $\Rightarrow$ return A[1]

$O(1)$ ✓

② Find_Min $\Rightarrow$ $O(n)$ ✓

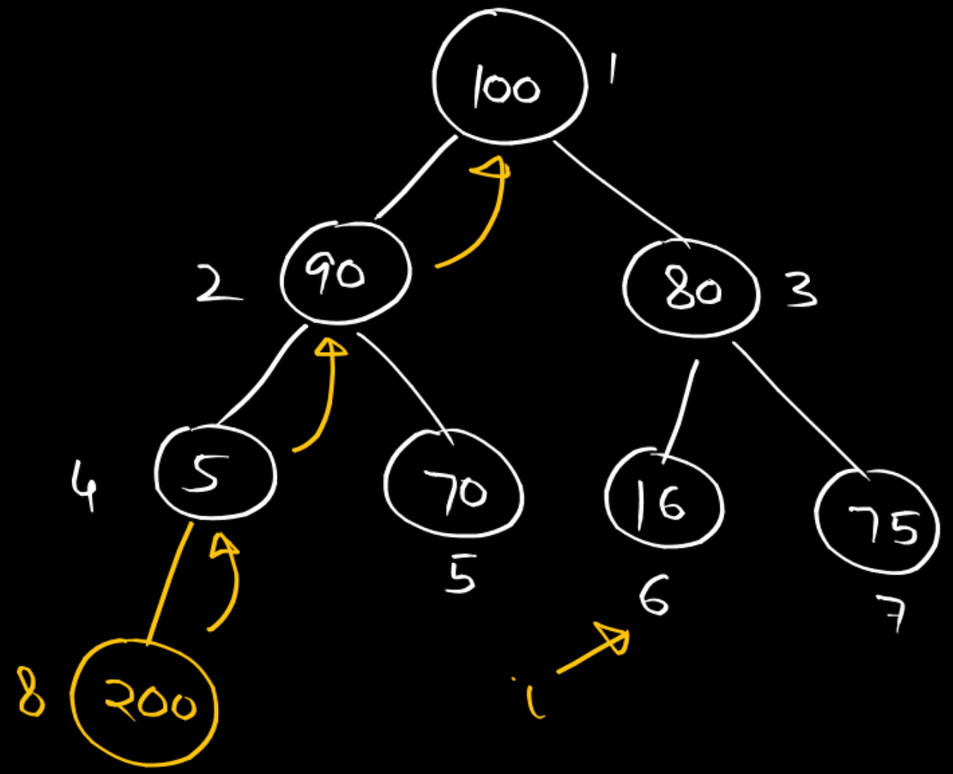


$x > c_1, c_2$

Min $\Rightarrow c_1 / c_2$

leaf nodes $= \lceil \frac{n}{2} \rceil$

$= \lceil \frac{7}{2} \rceil = 4$



1) Search $\Rightarrow O(n)$

2) Insert a key $\longrightarrow O(\log_2 n)$

$$Par(i) = \left\lfloor \frac{i}{2} \right\rfloor$$

Max_heap

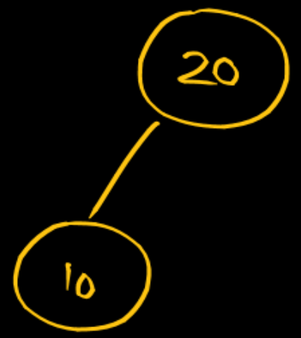
1) Find_Max : $O(1)$
2) Find_Min : $O(n)$
3) Insert a key : $O(\log_2 n)$
4) Search : $O(n)$
5) Extract_Max : $O(\log_2 n)$

Min_heap

1) Find_Min : $O(1)$
2) Find_Max : $O(n)$
3) Insert a key : $O(\log_2 n)$
4) Search : $O(n)$
5) Extract_Min : $O(\log_2 n)$

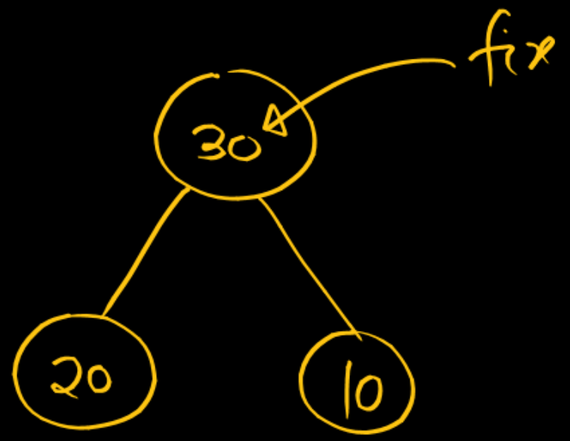
Given keys 10, 20.

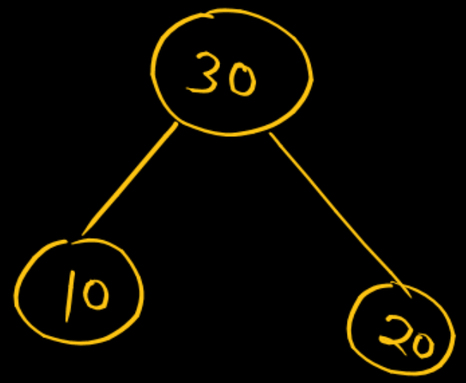
How many max heaps can be possible.

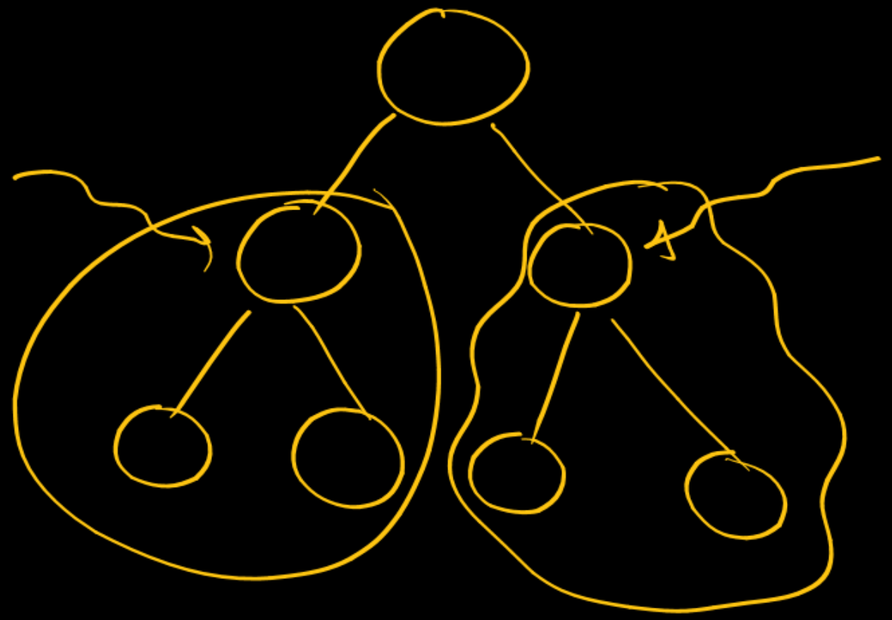


Keys : 10, 20, 30

max-heaps

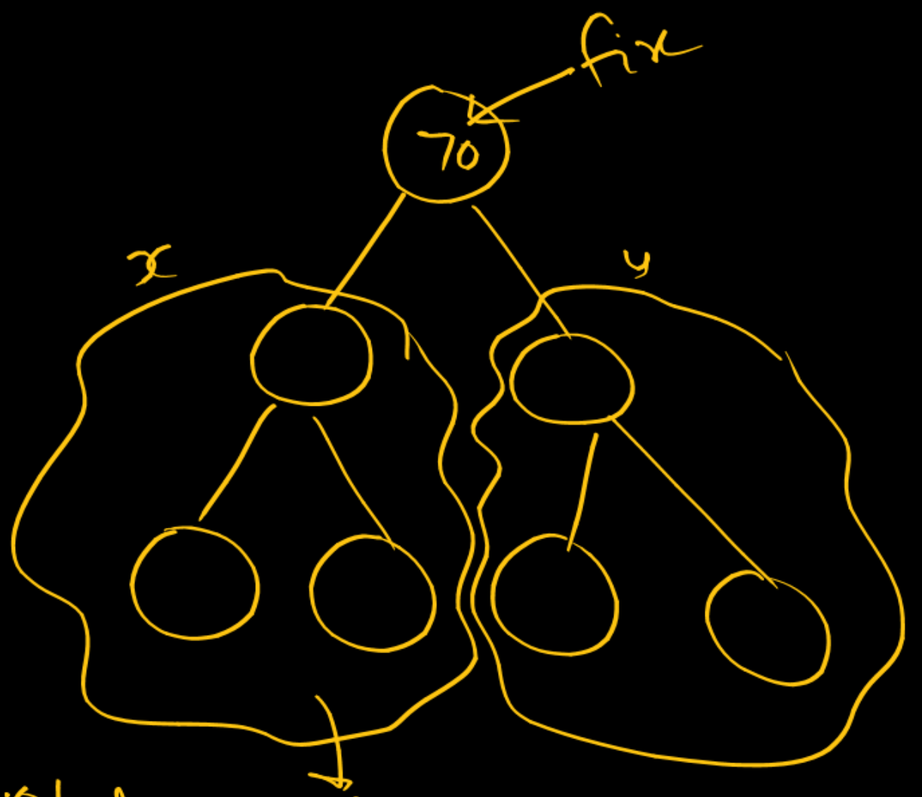






Keys : 10, 20, 30, 40, 50, 60, 70

7 - 1 ⇒ 6 keys are rem.



Select any 3 keys out of rem. 6 keys

⇒ $^6C_3 \times (\ 2\ )$

max-heaps with 7 distinct keys

$= 1 \times (^6C_3 \times 2) \times 2$

LT

No. of max-heaps with 3 distinct keys

$= \frac{6!}{3!3!} \times 2 \times 2$

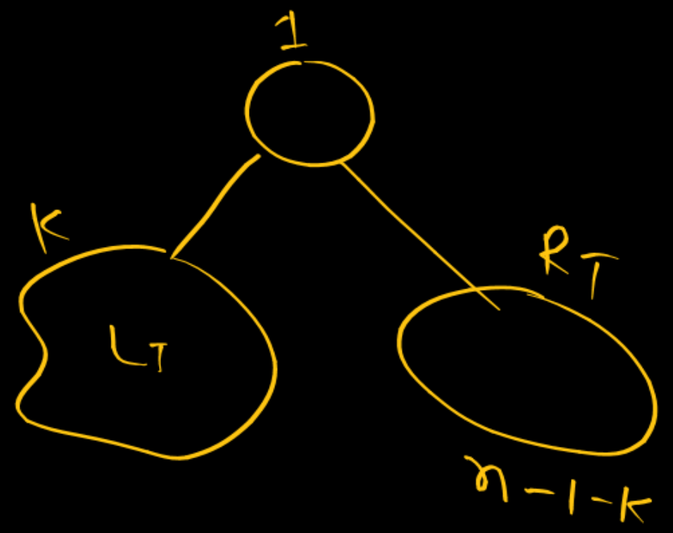
$= \frac{6 \times 5 \times 4 \times 3!}{3! \times 3!} \times 2 \times 2$

⇒ 80

n : distinct keys

$$F(n) = 1 \times \underbrace{{}^{n-1}C_k \times F(k)}_{L_T} \times \underbrace{F(n-1-k)}_{R_T}$$

k : no. of nodes in L_T

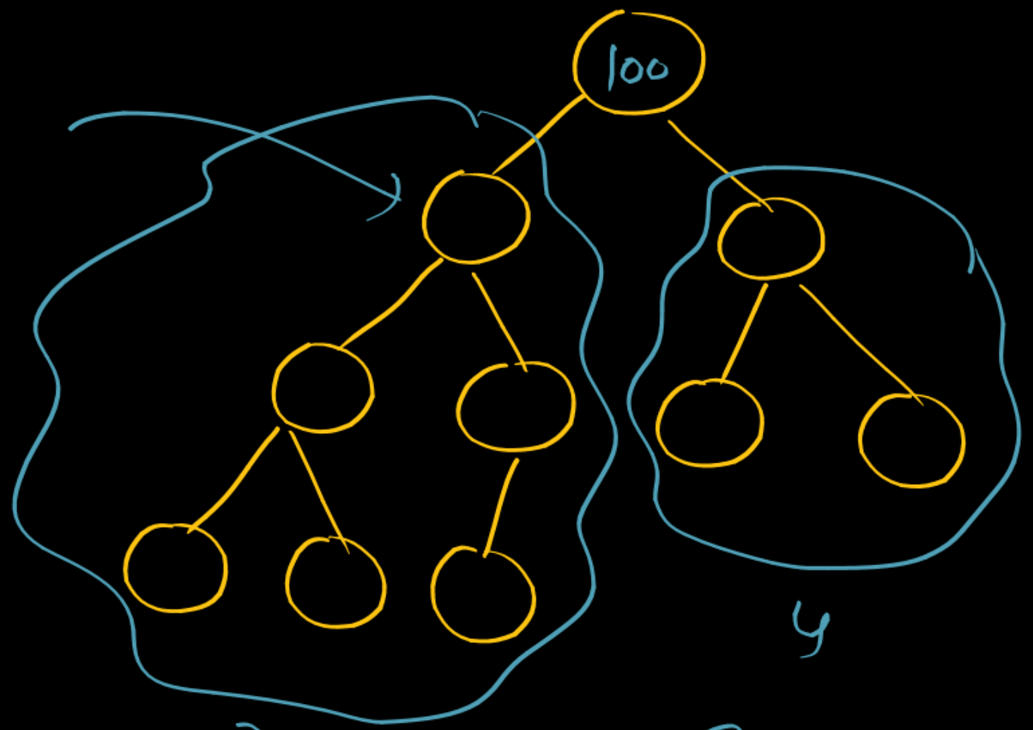


10 nodes

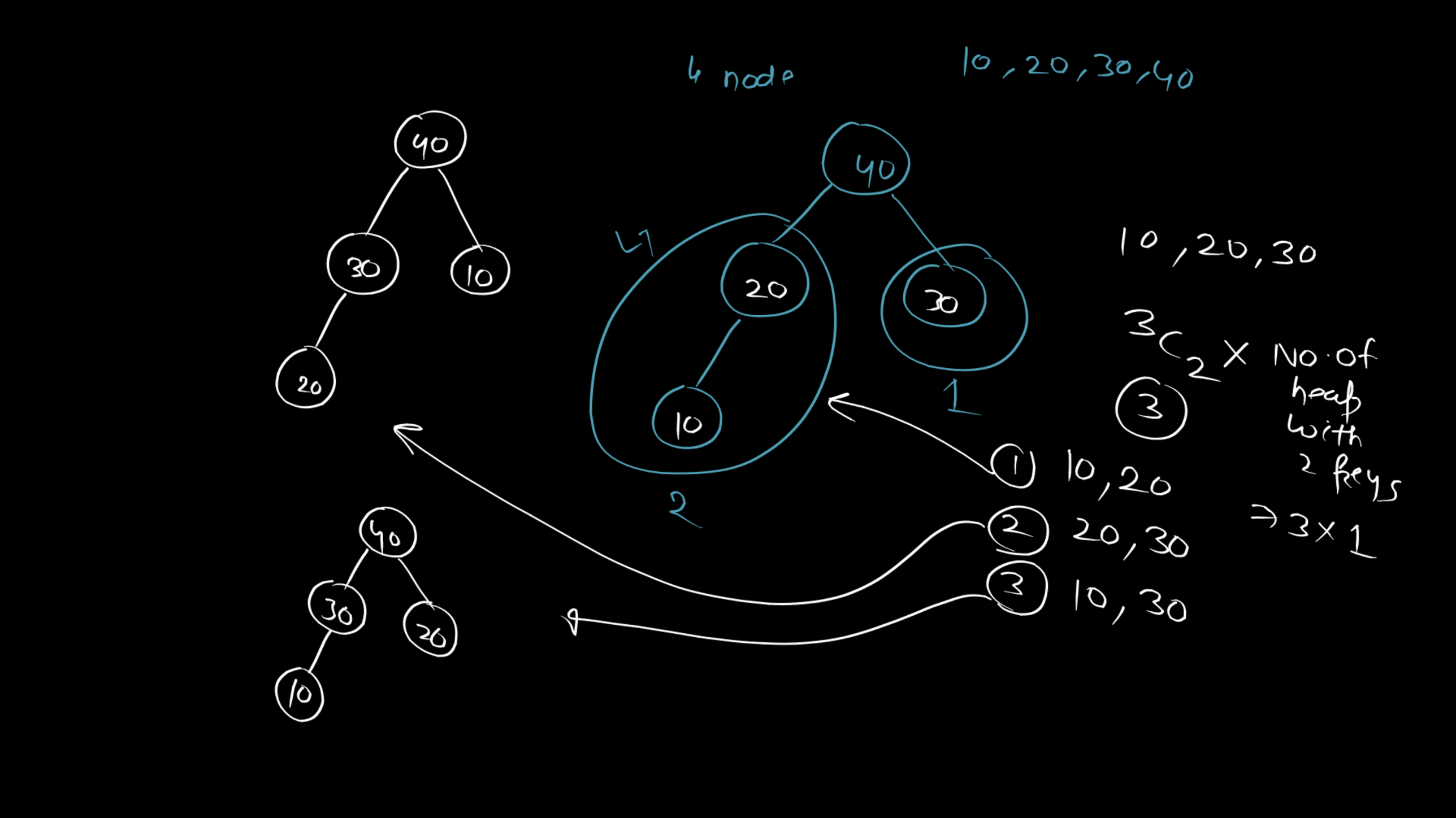
10, 20, — — 100

F(10)

⇒



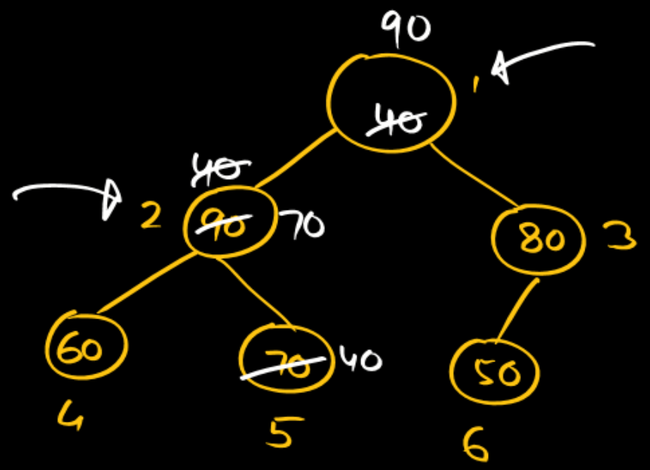
9 keys

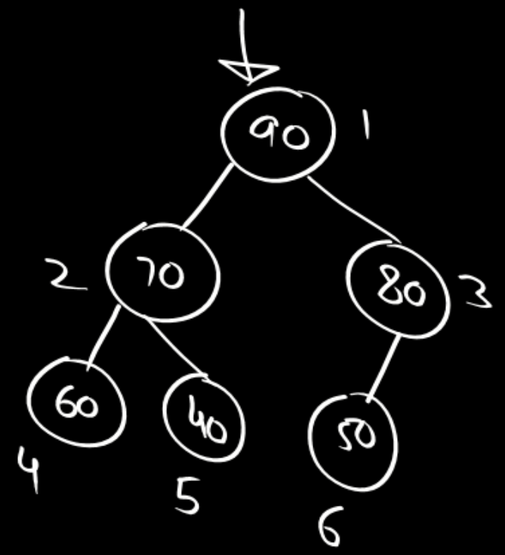

4 node
10, 20, 30, 40
40
30
10
20
40
20
30
10
4
2
1
10, 20, 30
$^3C_2$ × No. of heap with 2 keys
3
⇒ 3 × 1
1 10, 20
2 20, 30
3 10, 30
40
30
20
10

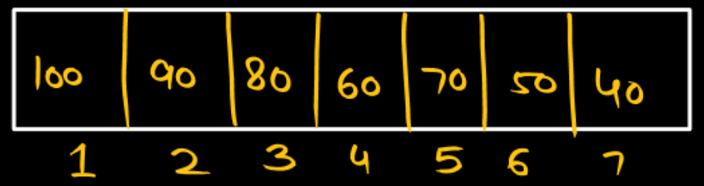
Graph

Hashing - 1

Hashing - 2





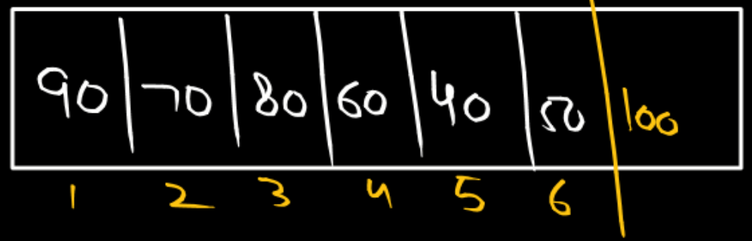


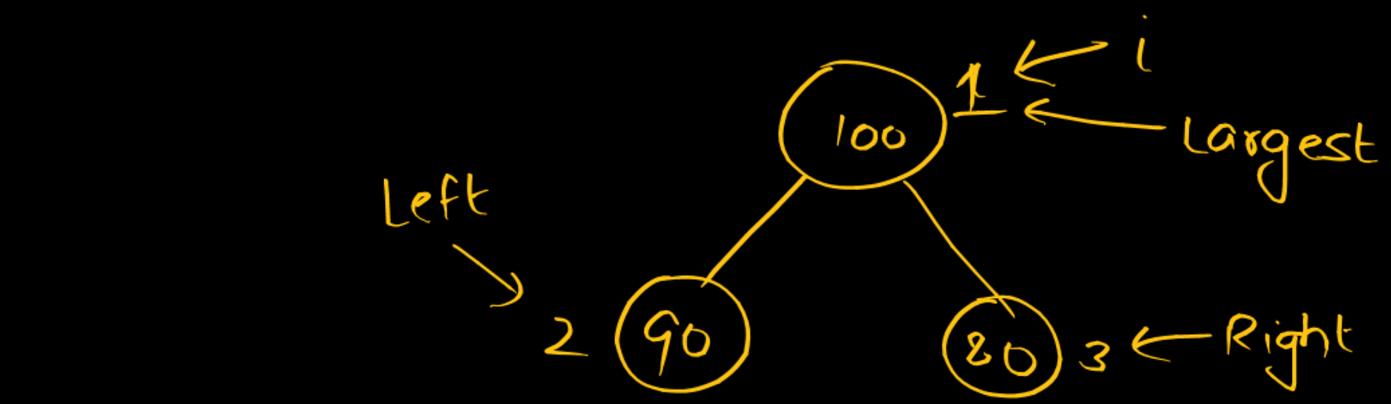
Extract_Max :

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

$n = n-1$;

Heapify (A, 1, n) ;






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

THANK - YOU