~~B-TREES~~ HEAP TREES

⇒ **MIN HEAP**

- highest priority means smallest value → min^m heap tree.
- Store min^m value at root of min^m heap tree.
- The tree structure will help to provide $O(\log n)$ worst case for both inserting new element & delete min.

⇒ **MAX HEAP**

- highest priority means largest value → max^m heap tree
- Store max^m value at root of max^m heap tree

Binary heap has 2 properties:-

1) Structure Property
   - Has to be complete binary tree.

2) Ordering Property

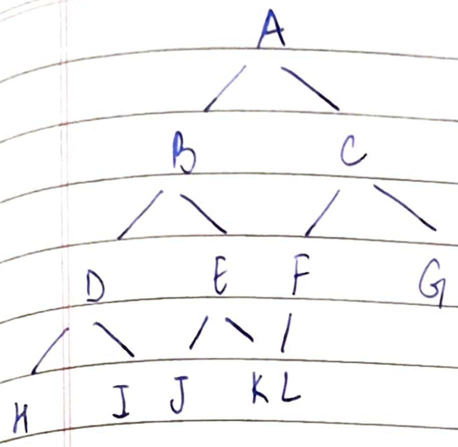
MIN HEAP → • each node is smaller than its children
• Smallest element will be located at root.

MAX HEAP → • each node is greater than its children
• Largest element will be located at root.

Implementing Binary Heap with an Array:-

There are NO holes in Heap Tree, so it can be stored compactly using an array structure.

The first element (root) can be stored in array position 1.

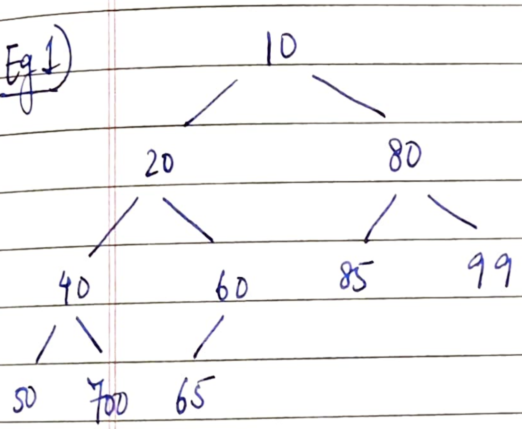


	A	B	C	D	E	F	G	H	I	J	K	L	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

For node: i
left child: $2i$
right child: $2i+1$
parent: $i/2$

① Insertion in Heap Tree

Eg 1)



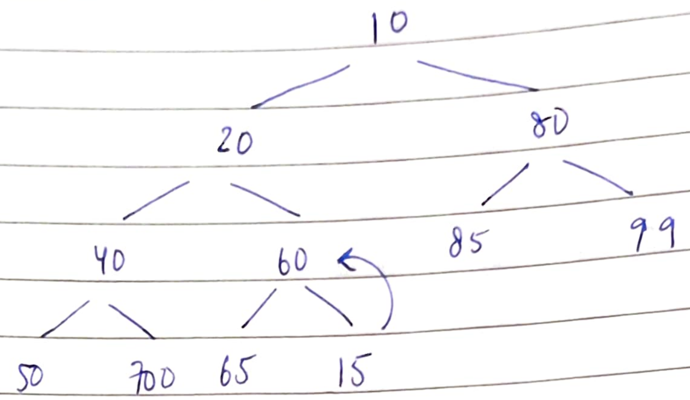
Insert 15

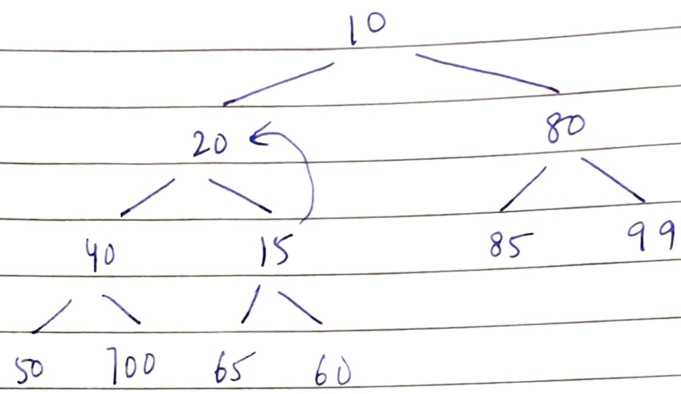
	10	20	80	40	60	85	99	50	700	65
0	1	2	3	4	5	6	7	8	9	10

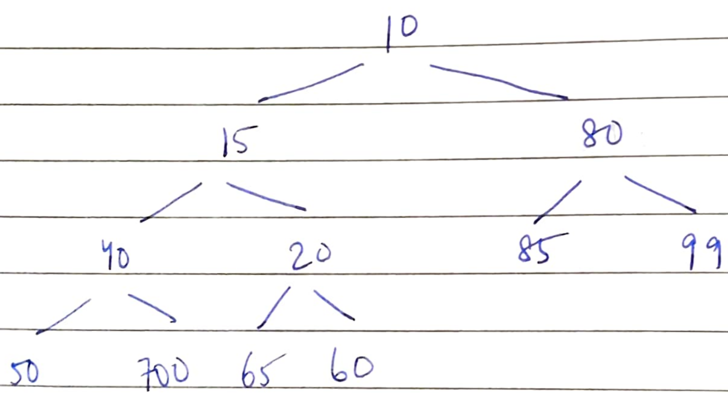
No holes ⇒ Insert it after the last element (pos^n 11)

10 20 80 40 60 85 99 50 700 65 15

But it violates heap property







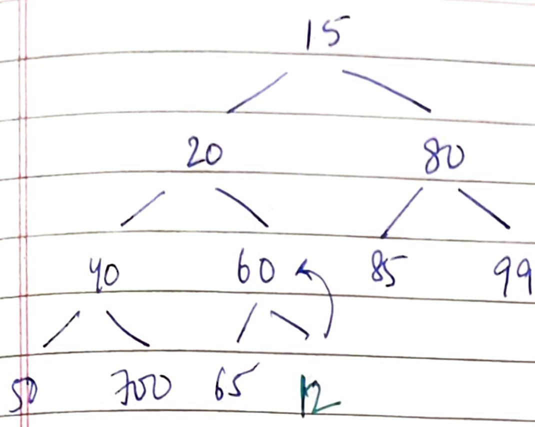
Now, satisfies Min heap property

10 15 80 40 20 85 99 50 700 65 60

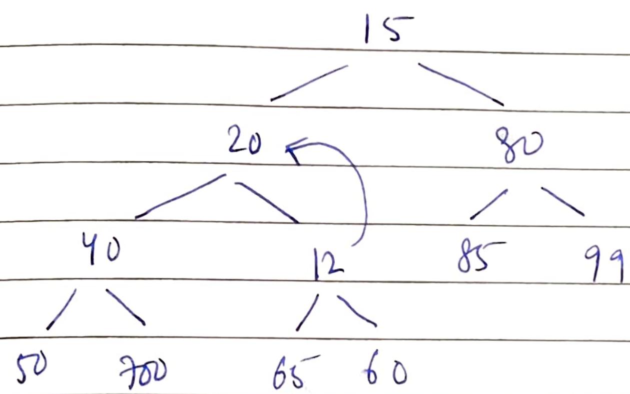
	15	20	80	40	60	85	99	50	700	65	12
0	1	2	3	4	5	6	7	8	9	10	11

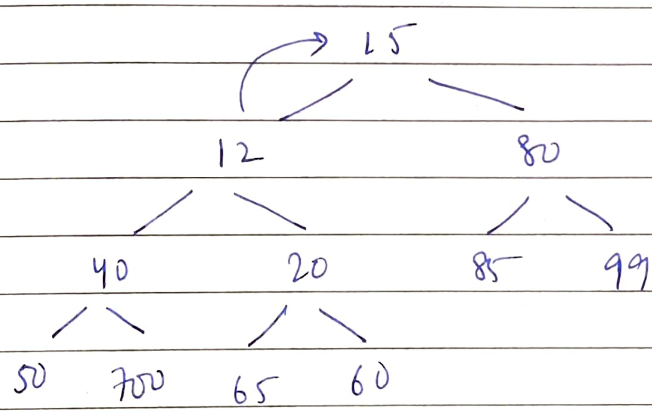
Eg 2)

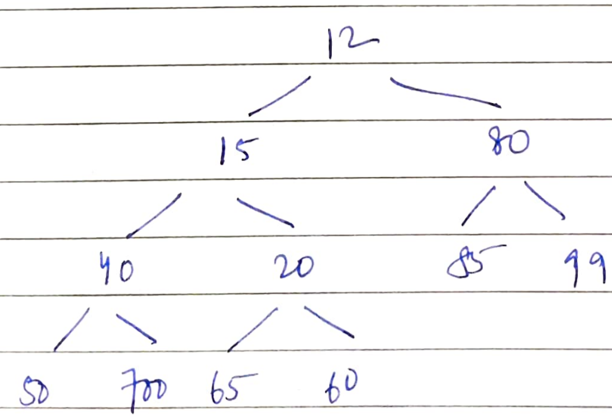
Insert 12



Violates heap property







	12	15	80	40	20	85	99	50	700	65	60
0	1	2	3	4	5	6	7	8	9	10	11

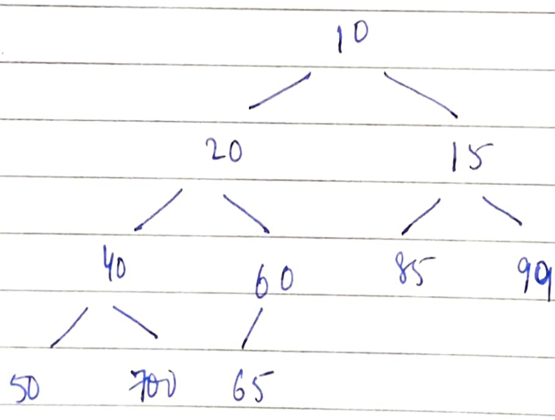
CODE:

```
hole = size + 1;
Heap[hole] = val;
while (hole >1 && val < Heap[hole/2]){
    Heap[hole] = Heap[hole/2];
    hole = hole/2;
}
Heap[hole] = val;
```

② Deletion in Heap Tree
Delete - Min

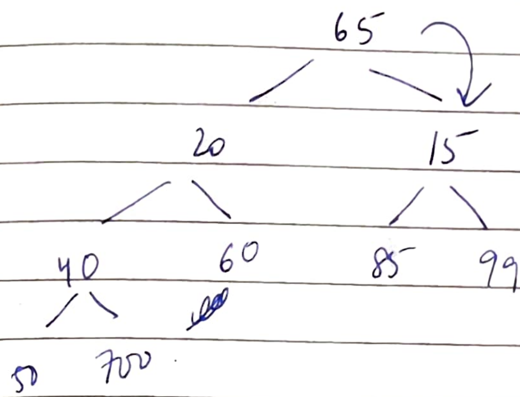
- Remove root (i.e. always the min!)
- Creates a hole
- Put 'last' leaf node at this hole
- Compare its value with 2 children
- If needed, swap node with smaller child
- Repeat these 2 steps until no swaps needed.

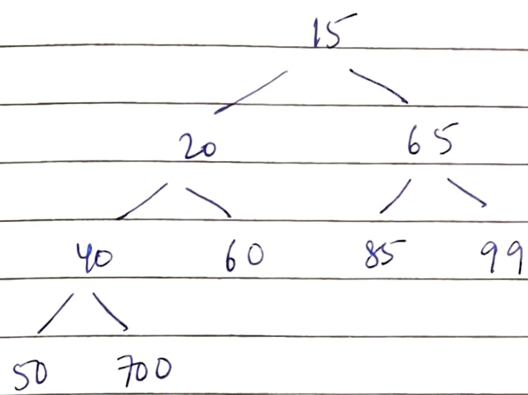
Eg 1) Delete Min. from this Tree



	10	20	15	40	60	85	99	50	700	65	
0	1	2	3	4	5	6	7	8	9	10	11

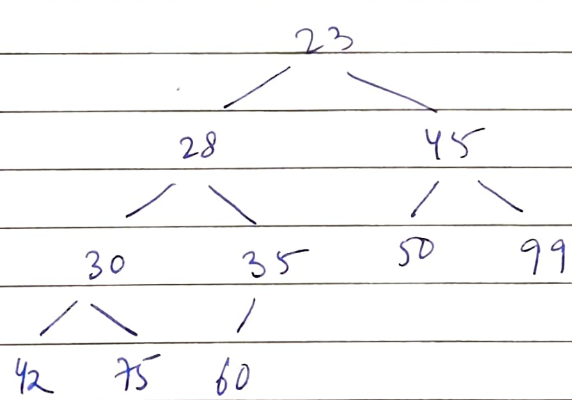
Deletion creates an hole in the tree

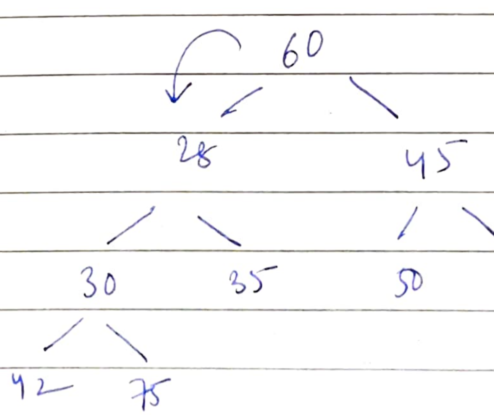


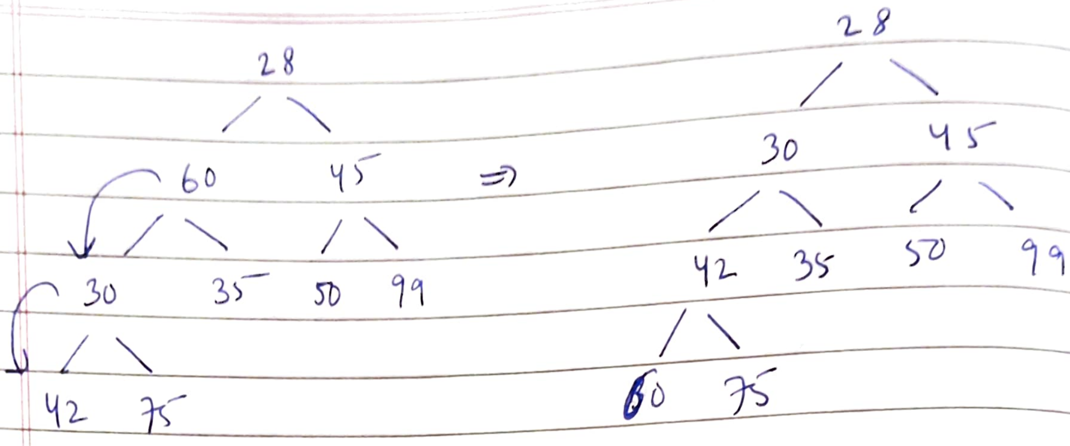


No more adjustments needed

Eg 2) Delete Min.







CODE:

```
int percolateDown(int hole, int val){

while (2* hole < size){
    left = 2* hole;
    right = left + 1;
    if (right < size && Heap[right] < Heap[left]){
        target = right;}
    else{
        target = left;}
    if (Heap[target] < val){
        Heap[hole] = Heap[target];
        hole = target;}
    else { break;}
}
return hole;
}
```

Page (Insert)

T.C. to insert item on heaptree : $O(\log n)$
insert one element at a time

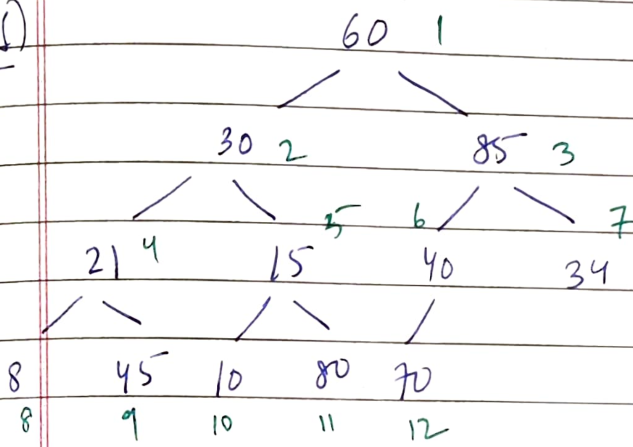
③ Build Heap (Heapify)

For n elements, T.C. : $O(n \log n)$ (Build Heap)

Can we do it in $O(n)$? (Heapify)

- Put all elements randomly on a heap tree.
- Start examining nodes from position $n/2$.

Eg 1)



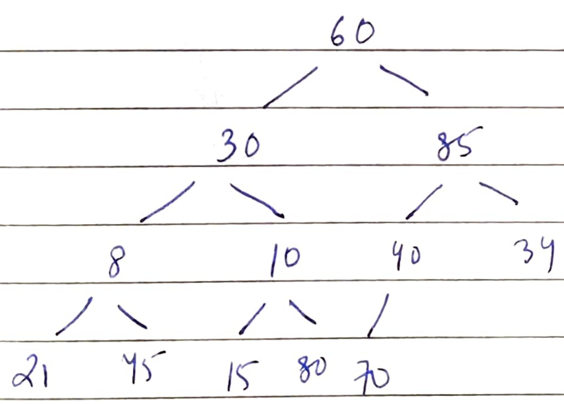
12 elements

$n/2 = 6$

Elem 6 : 40 in right place

Elem 5 : $15 \longleftrightarrow 10$

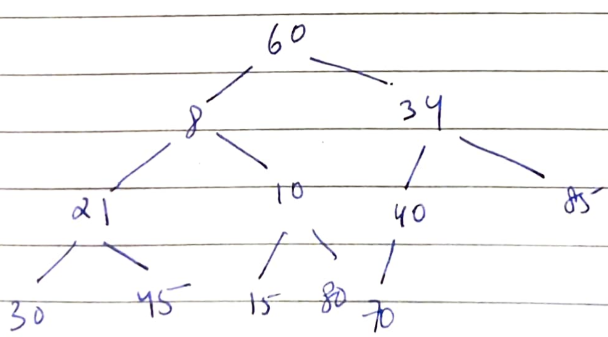
Elem 4 : $21 \longleftrightarrow 8$



Elem 3 : $85 \longleftrightarrow 34$

Elem 2 : $8 \longleftrightarrow 30$

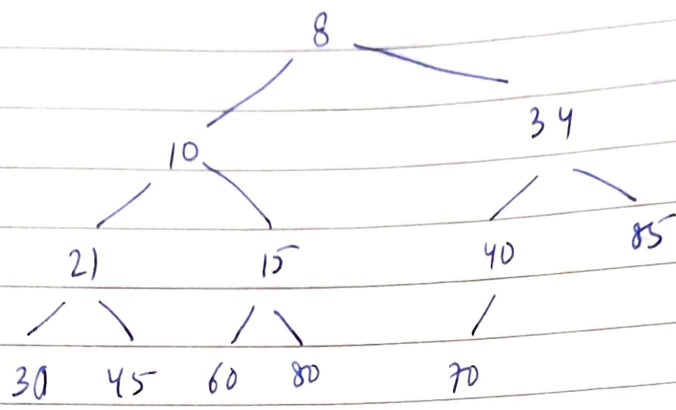
$21 \longleftrightarrow 30$



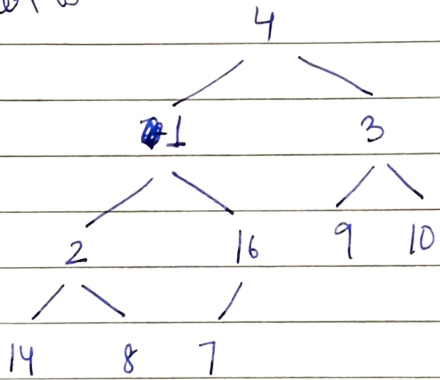
Elem 1 : $60 \longleftrightarrow 8$

$60 \longleftrightarrow 10$

$60 \longleftrightarrow 15$



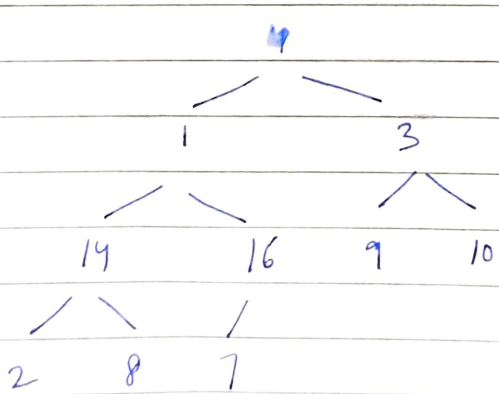
Eg 2) Heapify as MAX heap



10/2 = 5

Elem 5 : right place

Elem 4 : 2 ⟷ 14



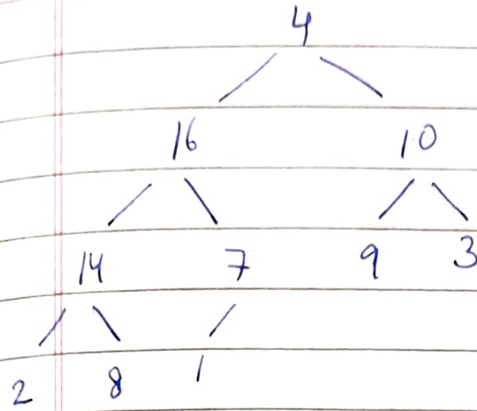
Elem 3 : 3 ⟷ 10

Elem 2 : 16 ⟷ 1, 1 ⟷ 7

~~Elem 3 : right place~~

~~Elem 2 : right place~~

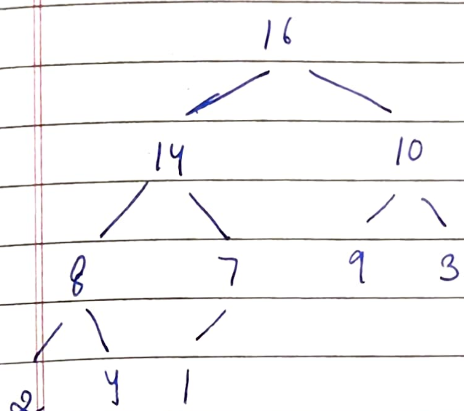
~~Elem 1 :~~



Elmnt: $4 \leftrightarrow 16$

$4 \leftrightarrow 14$

$4 \leftrightarrow 8$



⇒ Running Time of Heapify: $O(n)$

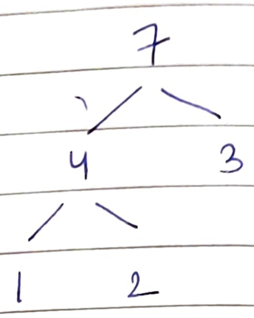
Running Time of BUILD-MAX-HEAP: $T(n) = O(n)$

Building heap from random array:	$O(n)$
Deletion of one item:	$O(\log n)$
Deletion of K items:	$O(K\log n)$
Total time:	$O(n + K\log n)$

heapify
↳ We build a heap & then turn it into a sorted list by calling deleteMin/deleteMax.

HEAPSORT

↳ To sort an array using heap representations



- Build a max heap [7, 4, 3, 2, 1]
- Largest element will be at the root of the tree.
- Delete the root and swap with last element of the array. [1, 4, 3, 2] 7
- Call Max-Heapify on the new root

[4, 3, 2, 1]
~~[4, 1, 3, 2]~~ [1, 3, 2] 4, 7

[3, 2, 1]
[1, 2] 3, 4, 7

[2, 1]
[1] 2, 3, 4, 7

Final: [1, 2, 3, 4, 7]

Lec 20 - Binary Heaps - slide 16

Q) Why is Binary Heap preferred over priority Queue?

Priority Queue

	Insert	delMin	findMin
ord. array	O(n)	O(n)	O(1)
ord. list	O(n)	O(n)	O(1)
unord. array	O(1)	O(1)	O(n)
unord. list	O(1)	O(1)	O(n)

	insert	delMin	findMin
binary heap	O(logN)	O(logN)	O(1)