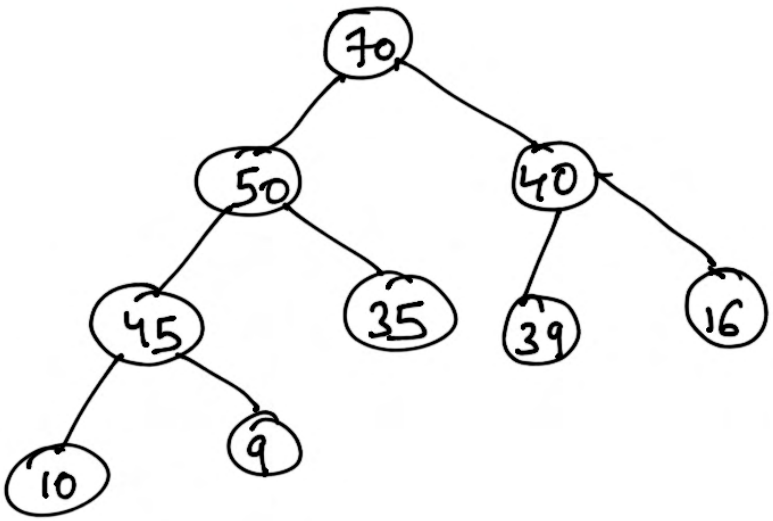
Heaps

→ Complete binary tree or almost complete (except last level, all levels are filled).

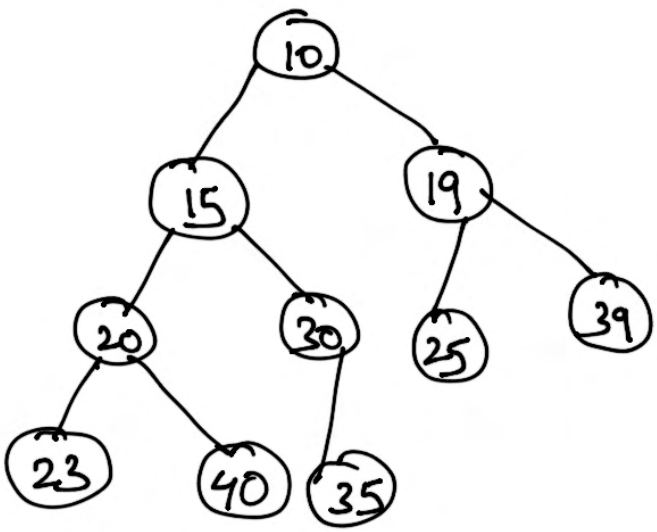
→ In Last level, all keys are aligned as left as possible.

Types

Max Heap

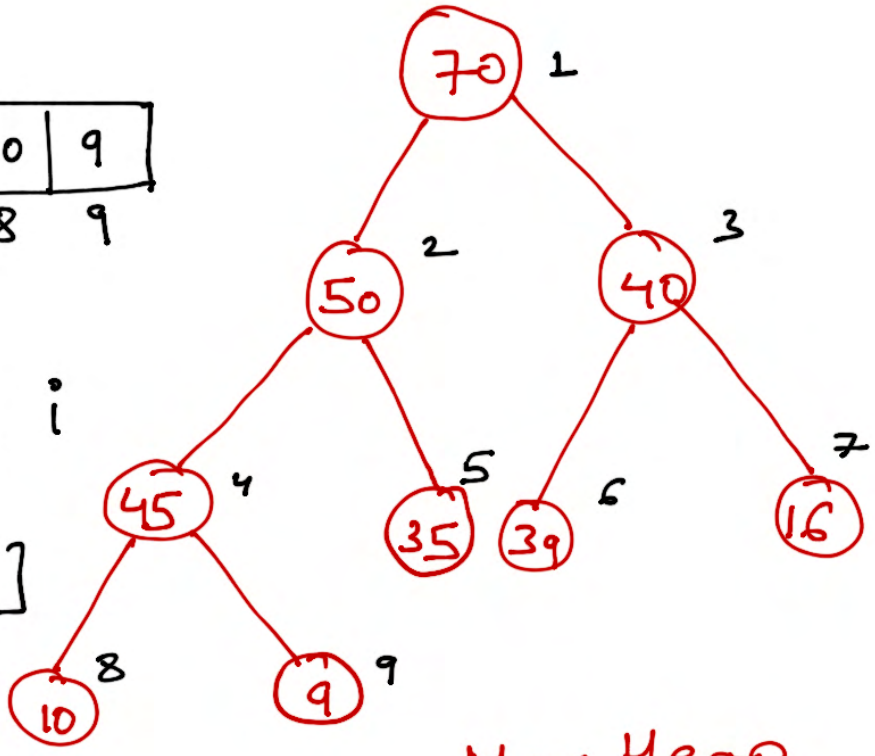


Min Heap



A

70	50	40	45	35	39	16	10	9
1	2	3	4	5	6	7	8	9



Max Heap

For every node i

$A[Parent(i)] \geq A[i]$

$i = 3$

$A[Parent(3)] \geq A[3]$

$A[1] \geq A[3]$

$70 \geq 40$

$i = 4$

$A[Parent(4)] \geq A[4]$

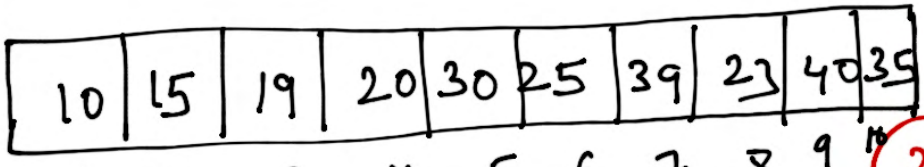
$A[2] \geq A[4]$

$50 \geq 45$

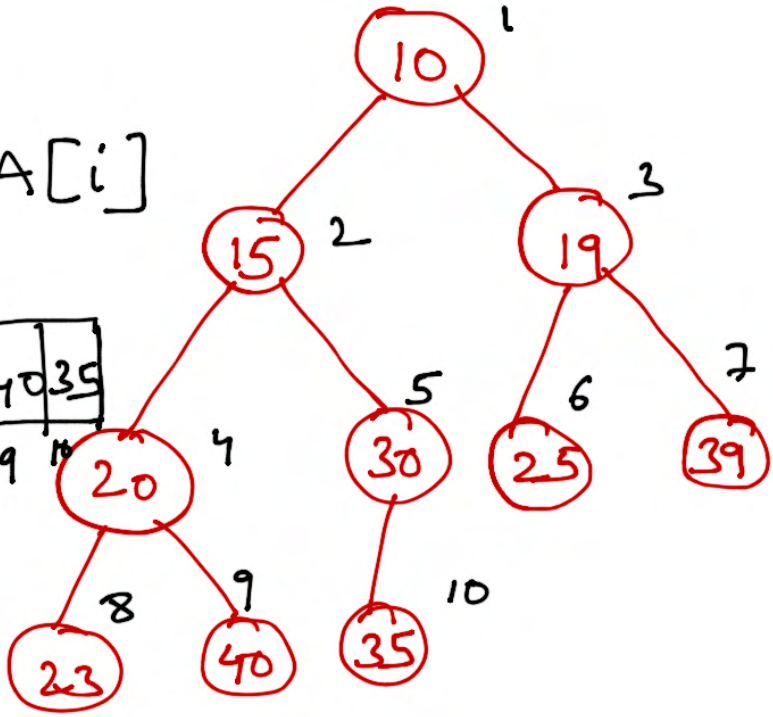
→ Root Node is the largest element.

for every Node i

$A[parent(i)] \leq A[i]$



A



Min Heap

$i = 4$

$A[Parent(4)] \leq A[4]$

$A[2] \leq A[4]$

$15 \leq 20$

$i = 5$

$A[Parent[5]] \leq A[5]$

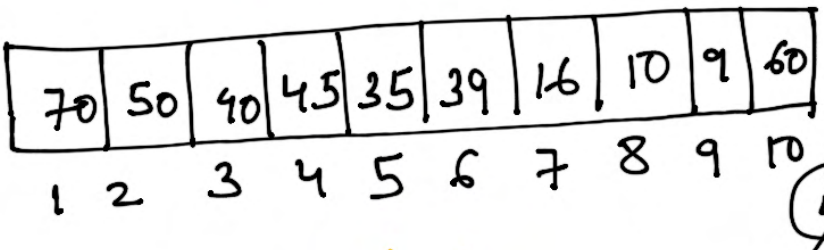
$A[2] \leq A[5]$

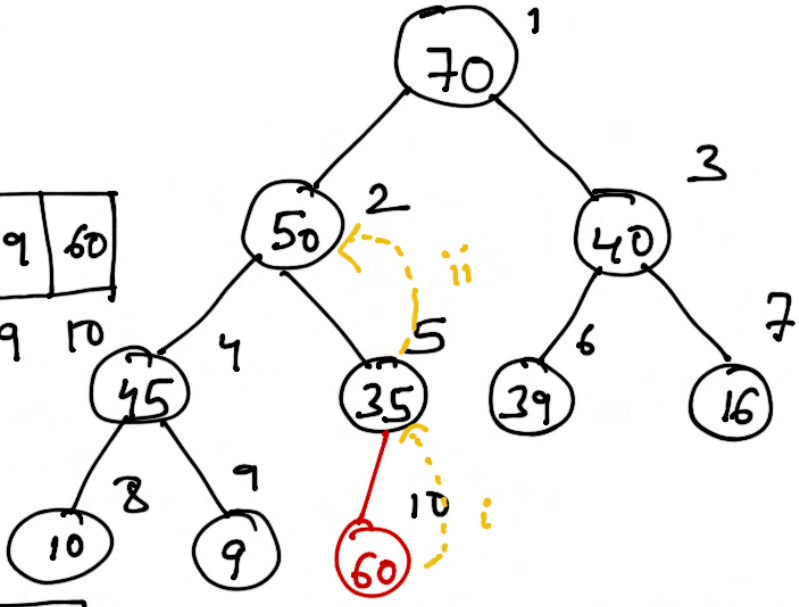
$15 \leq 30$

Insertion in Max Heap

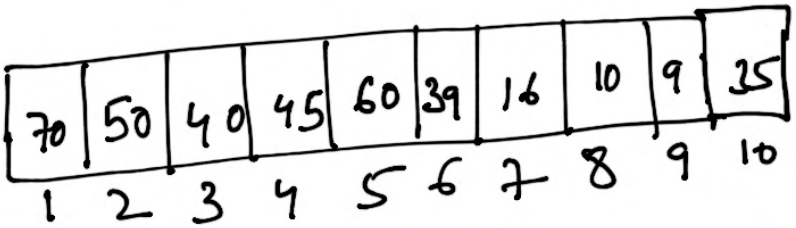
Insert '60'

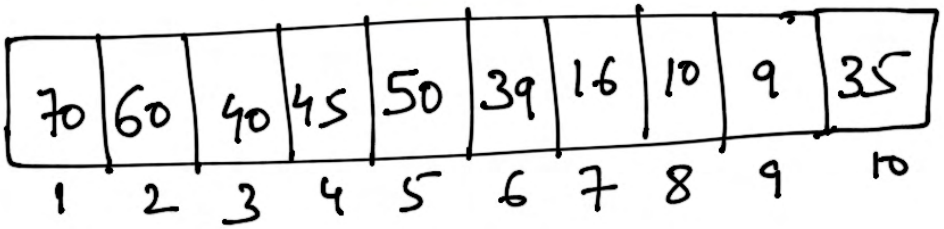
→ Data is always inserted from leaf node





Max Heap

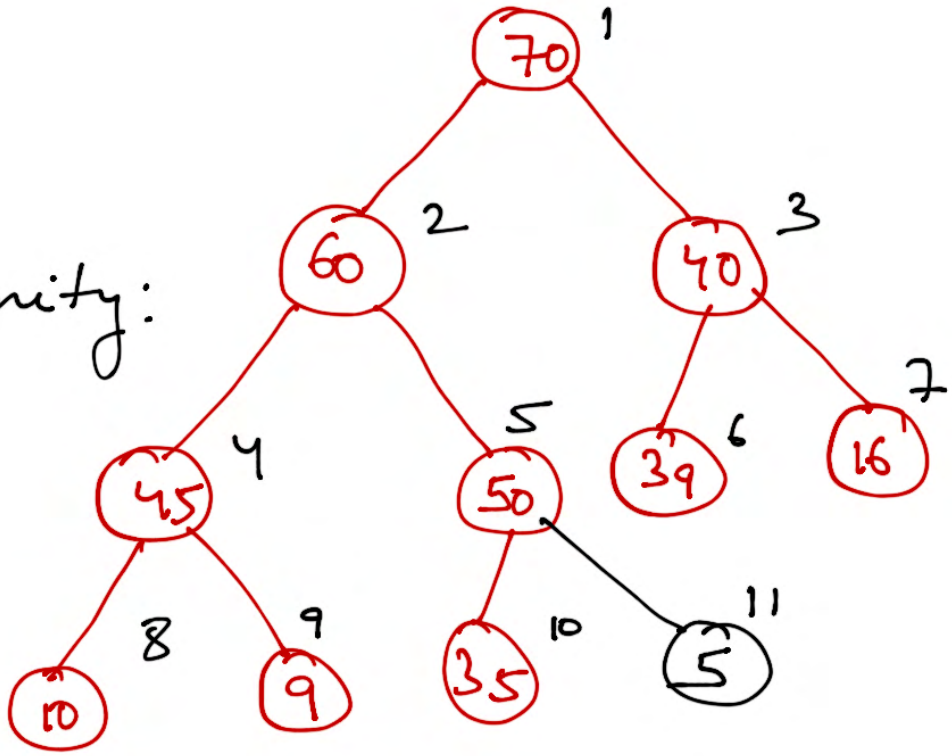




Insert '5'

Time complexity:

$O(\log n)$



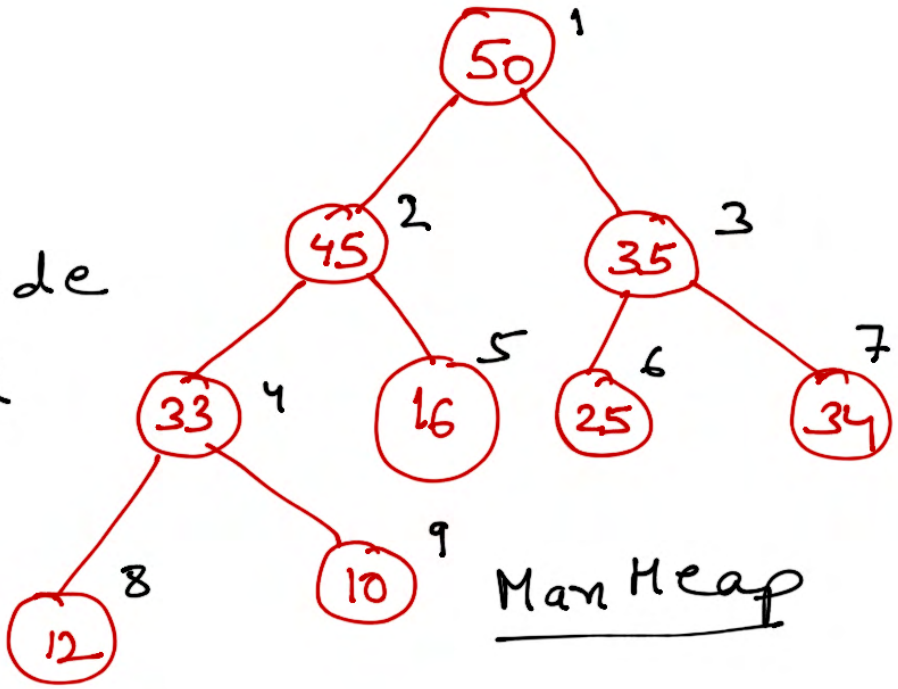
A

70	60	40	45	50	39	16	10	9	35	5
1	2	3	4	5	6	7	8	9	10	11

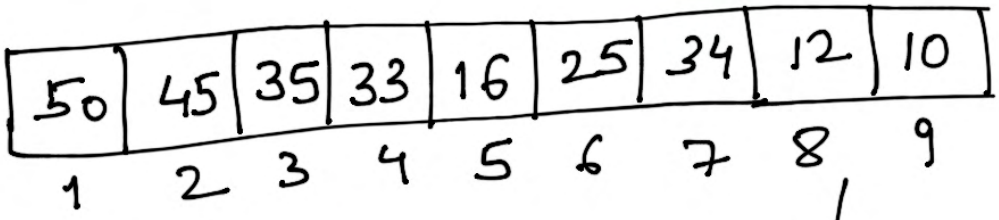
```
insertHeap(A, n, value)
{  n = n+1
   A[n] = value
   i = n
 while (i > 1)
  {  Parent = ⌊i/2⌋
       if (A[Parent] < A[i])
       { swap(A[parent], A[i])
              i = Parent
       }
       else { return }
} }
```

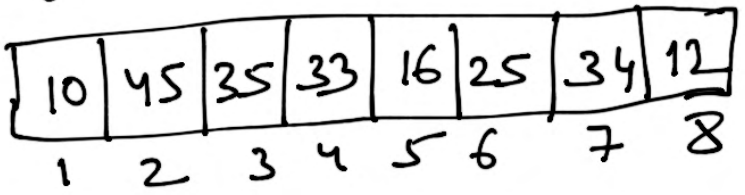
Deletion

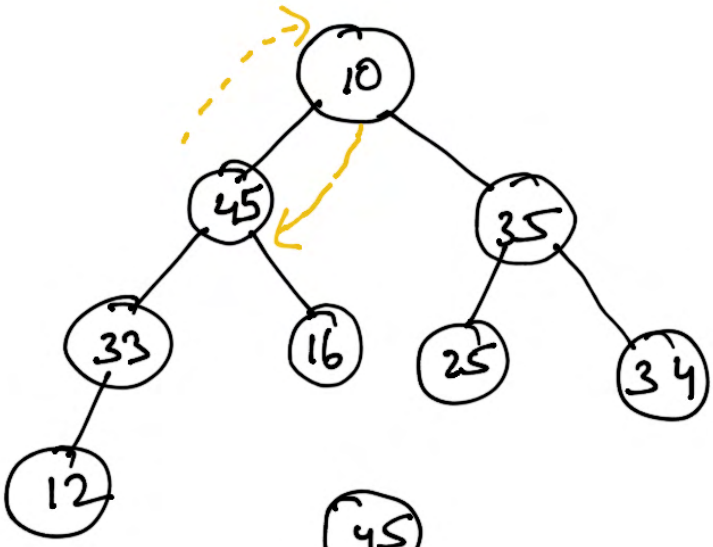
→ Only root node can be deleted

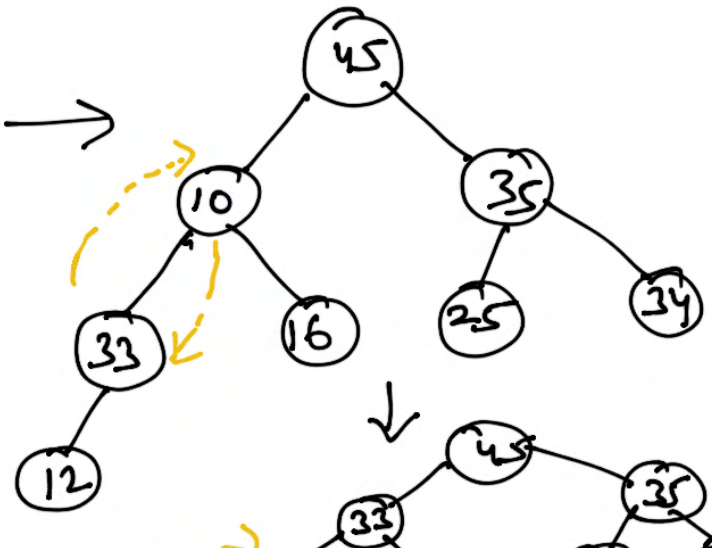


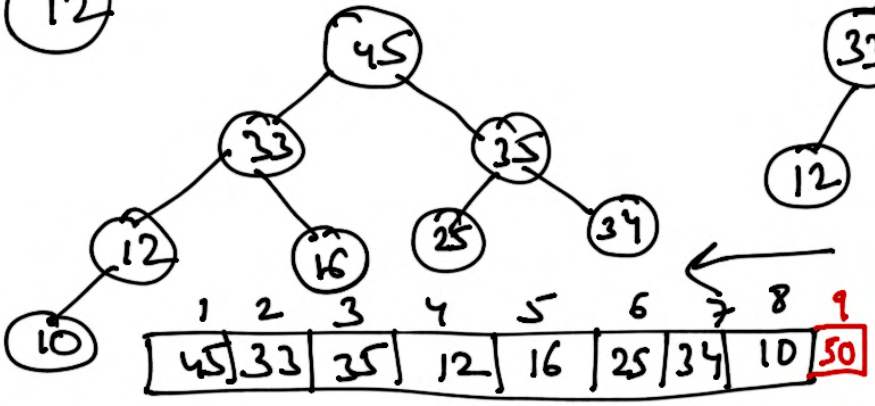
Max Heap

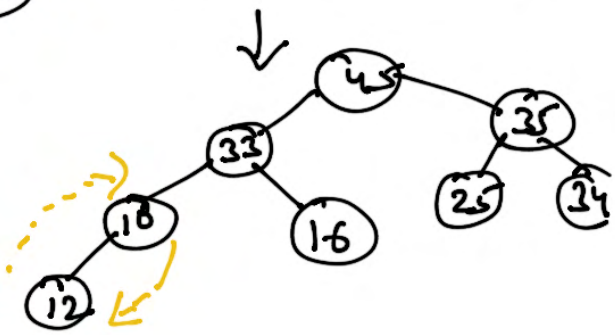








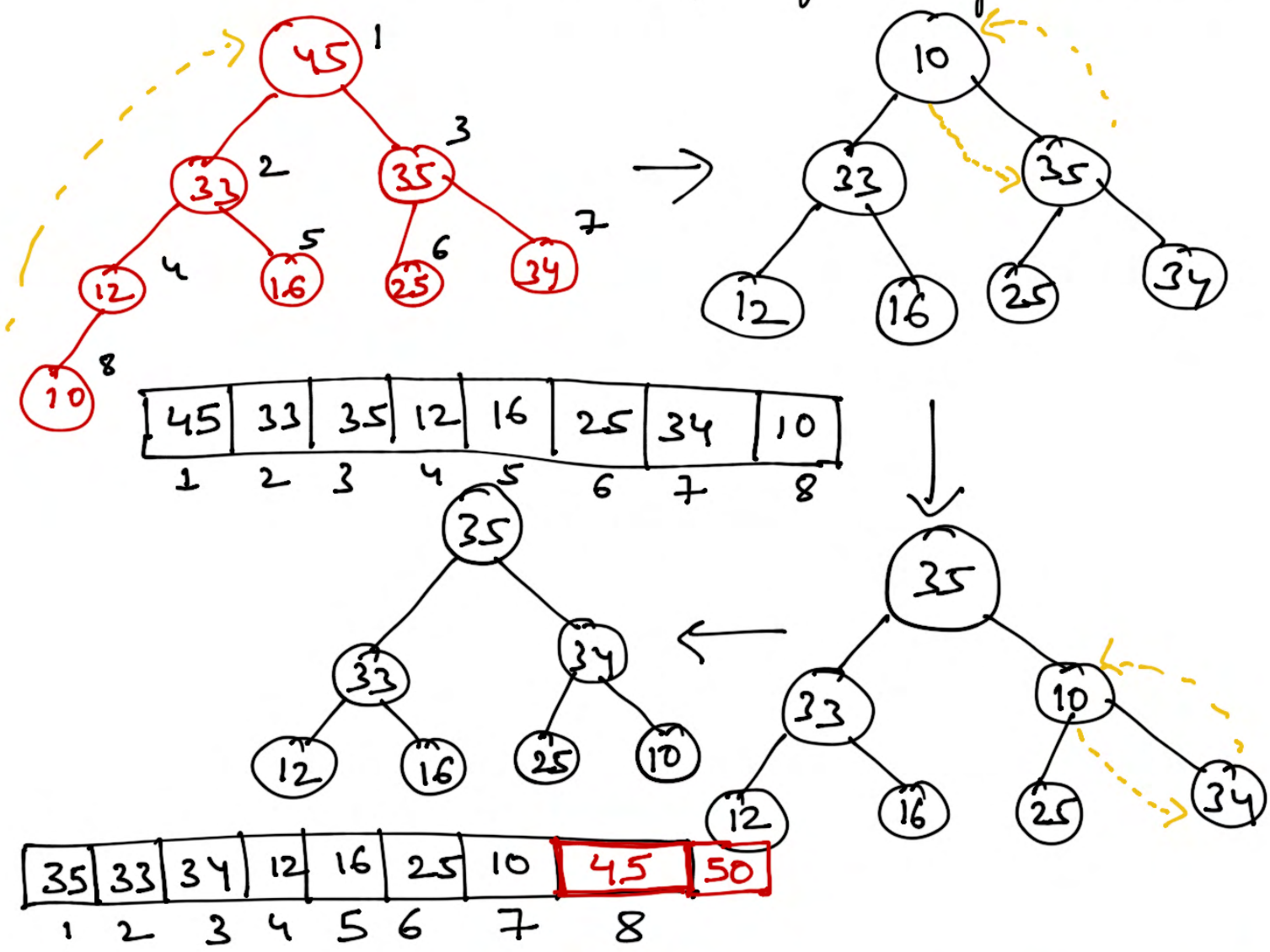




Left child = $2i$

Right child = $2i + 1$

→ Compare the children with the root node & swap if required

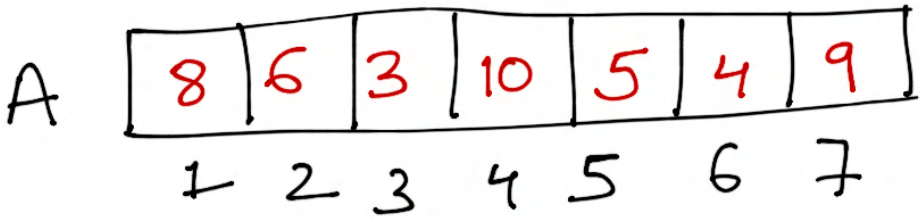


Max Heap —Deletion→ Ascending order

Min Heap —Deletion→ Descending order

Priority Queue

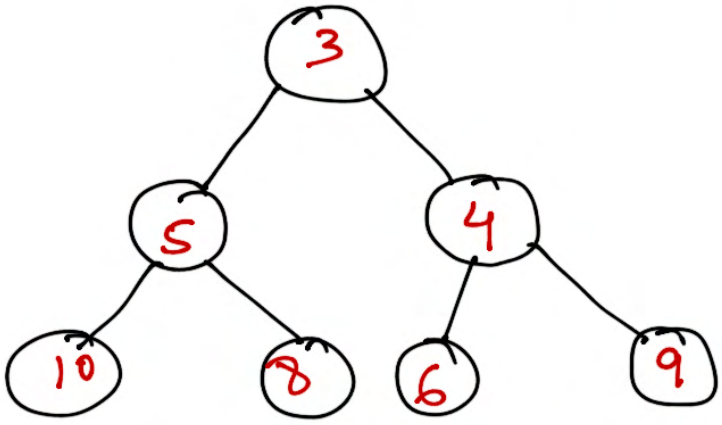
→ Based on priority



O(n)

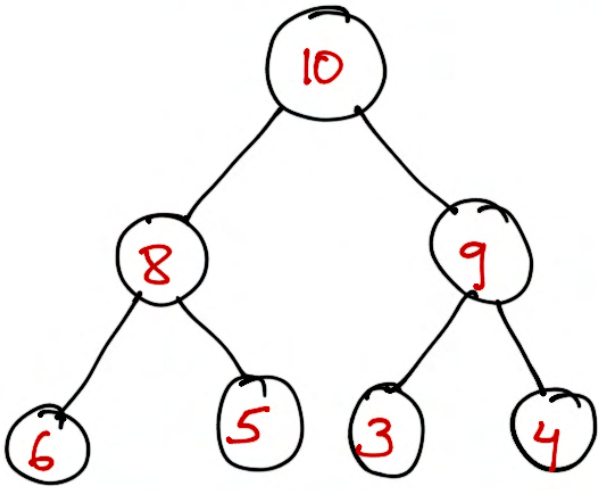
Smaller number → Higher priority

Larger number → Higher Priority



Min Heap

O(log n)

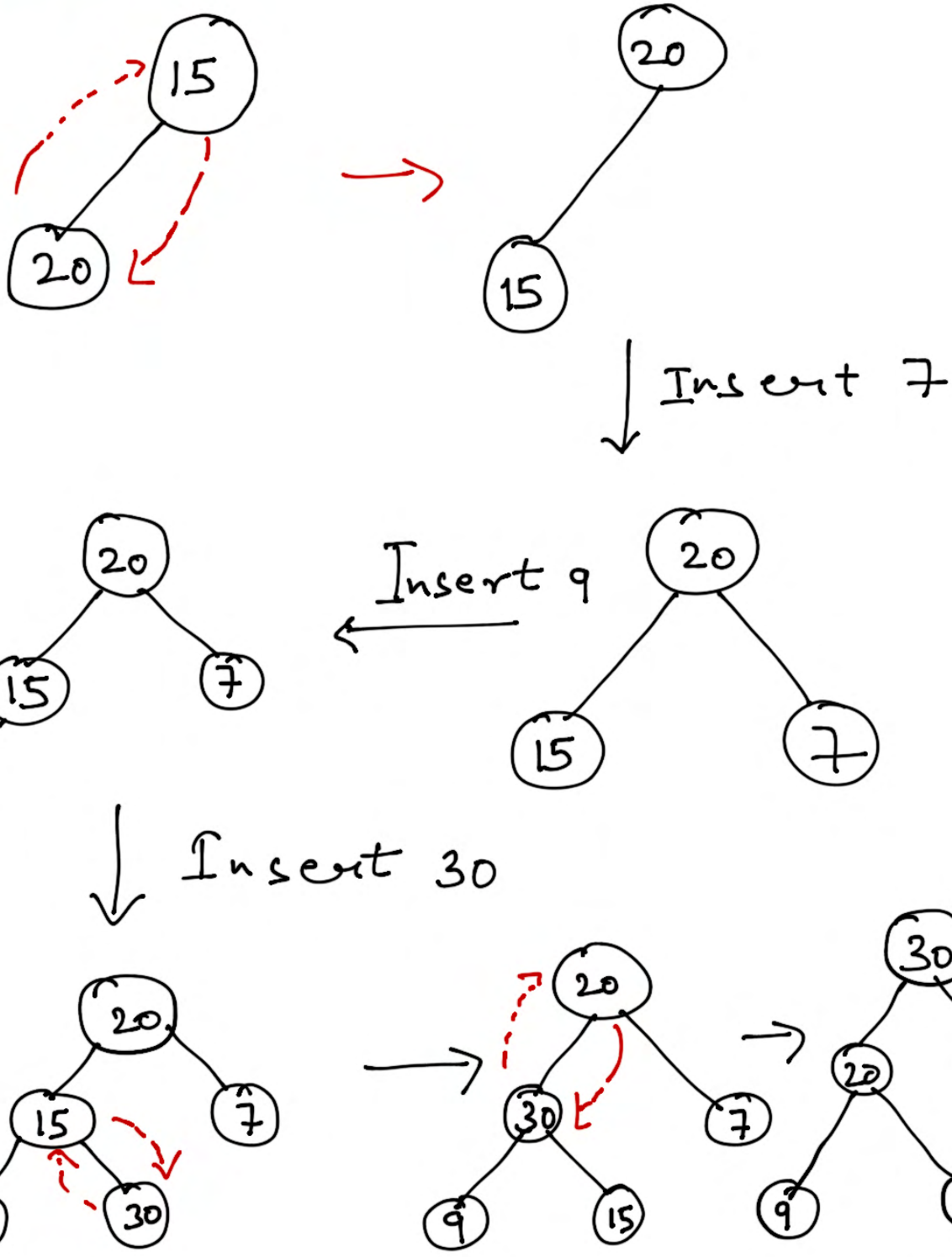


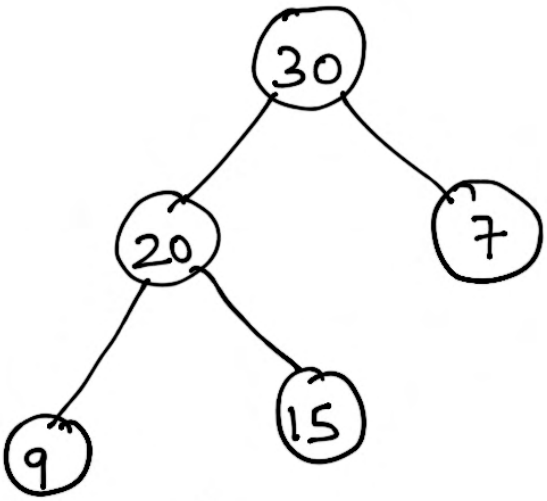
Max Heap

O(log n)

Heap Sort

	1	2	3	4	5
A	15	20	7	9	30





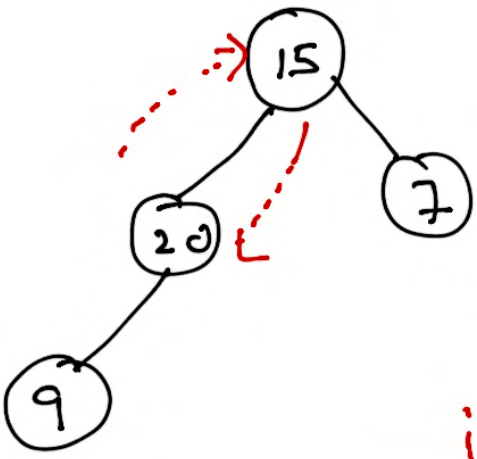
A

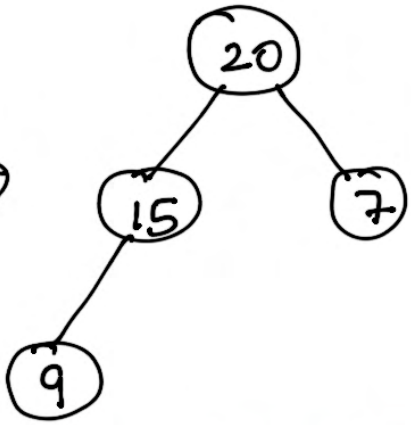
30	20	7	9	15
0	1	2	3	4

-

Max Heap

Delete '30'



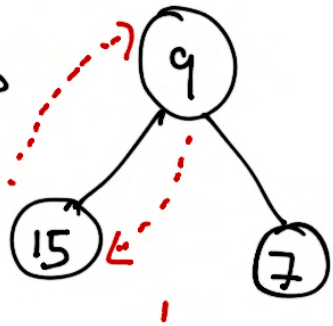


Delete '20'

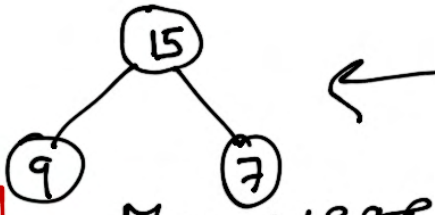
15	20	7	9	30
0	1	2	3	4

Max Heap

20	15	7	9	30
0	1	2	3	4

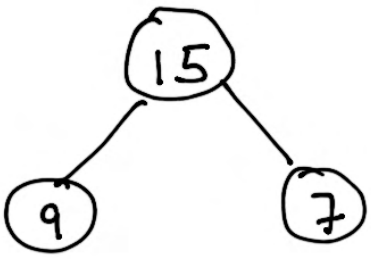


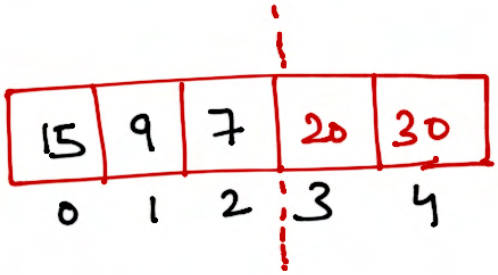
9	15	7	20	30
0	1	2	3	4



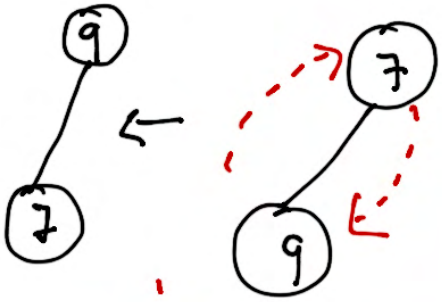
Max Heap

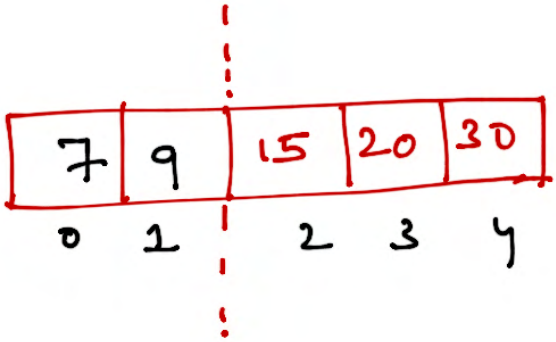
15	9	7	20	30
0	1	2	3	4

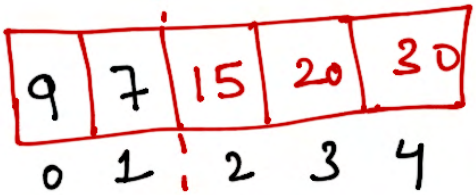




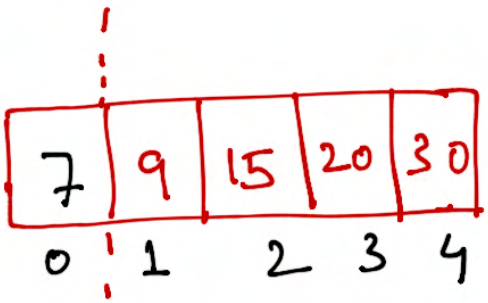
Delete '15'



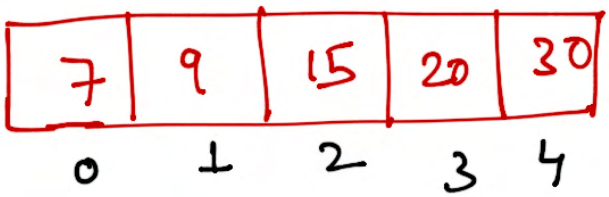




Delete '9'



Delete '7'



Sorted
(Ascending Order)

Time Complexity

Insert an element in a Heap

$= O(\log n)$

for 'n' elements, $= O(n \log n)$

Delete an element in a Heap

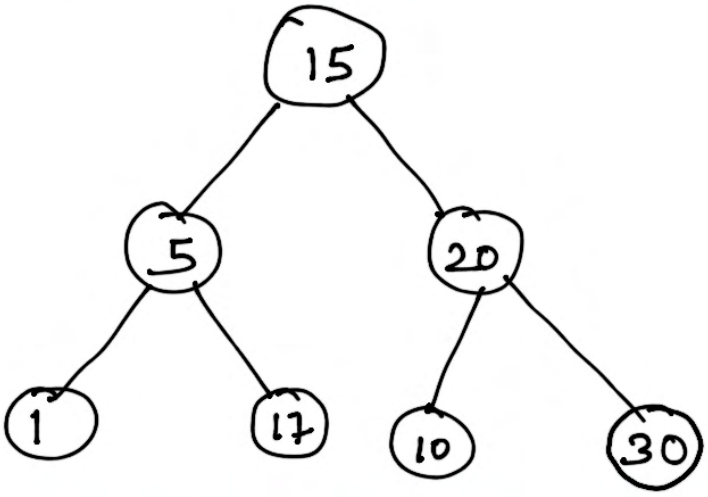
$= O(\log n)$

for 'n' elements $= O(n \log n)$

Total = $2 n \log n$

$= O(n \log n)$

Heapify



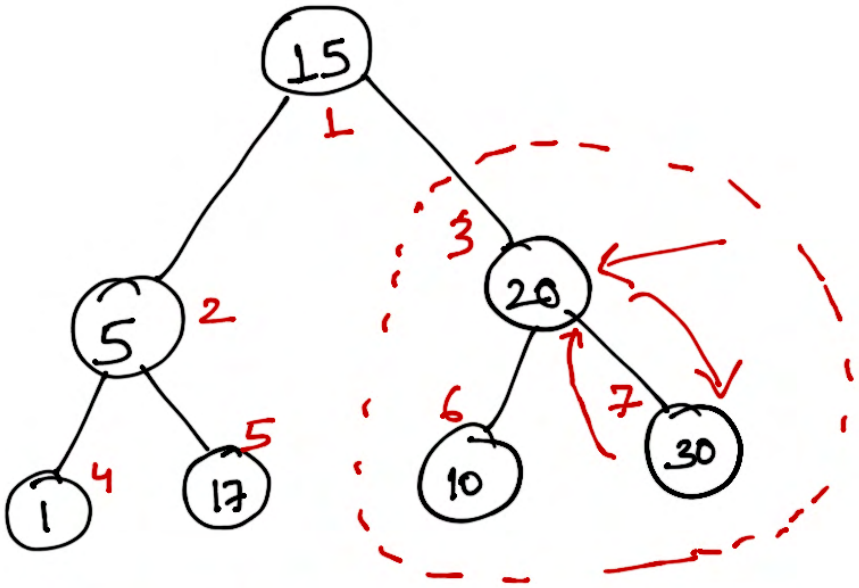
A	15	5	20	1	17	10	30
	1	2	3	4	5	6	7

↑

→ Apply Heapify only on non-leaf nodes.

→ In complete binary tree, leaf nodes are from $\lfloor \frac{n}{2} \rfloor + 1$ to n

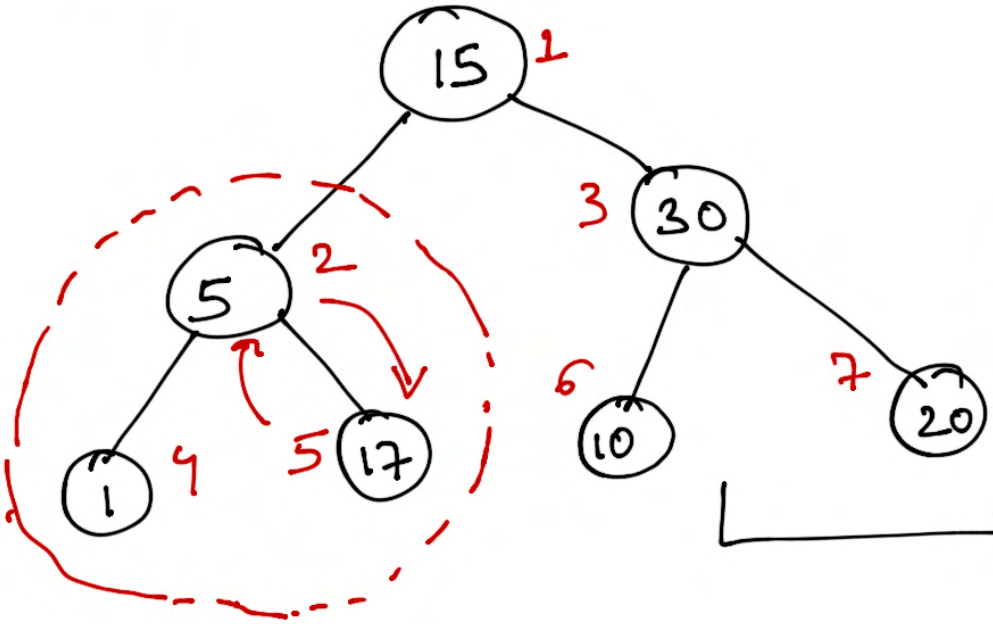
→ Heapify starts from non-leaf node with largest index $\lfloor \frac{n}{2} \rfloor$



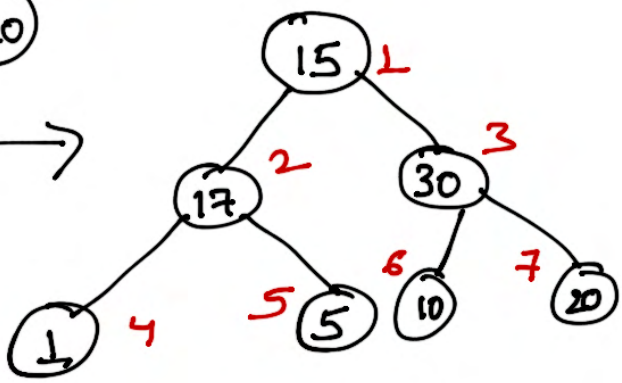
15	5	20	1	17	10	30
1	2	3	4	5	6	7

Heapify starts here

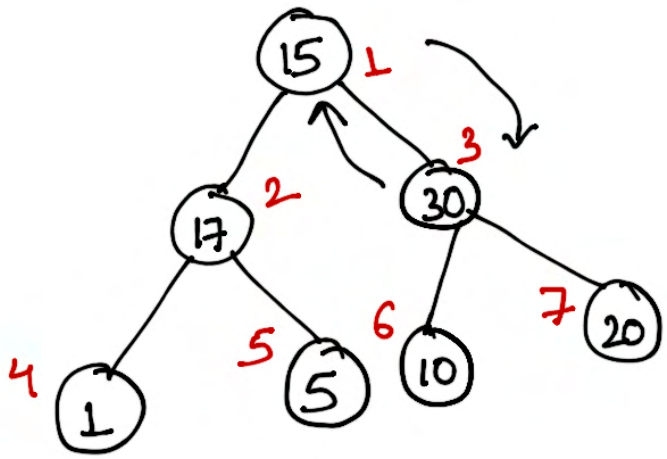
$$\left\lfloor \frac{n}{2} \right\rfloor = \left\lfloor \frac{7}{2} \right\rfloor = \lfloor 3.5 \rfloor = 3$$



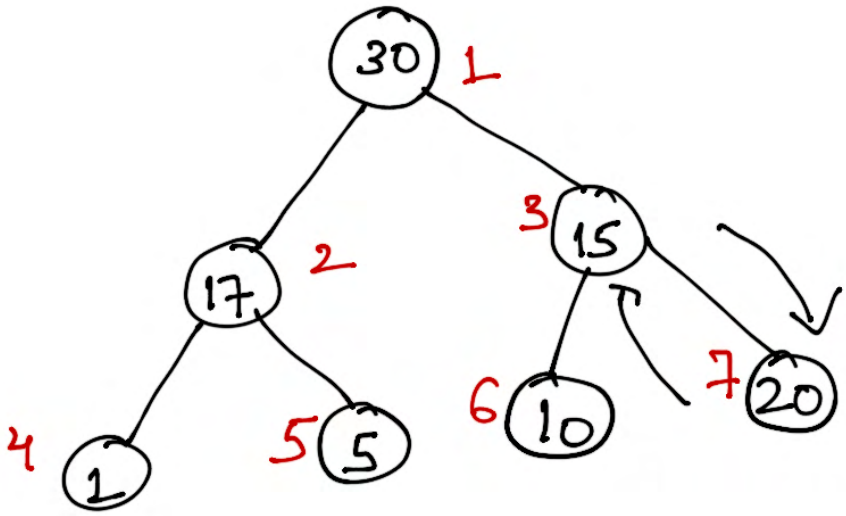
15	5	30	1	17	10	20
1	2	3	4	5	6	7



15	17	30	1	5	10	20
1	2	3	4	5	6	7



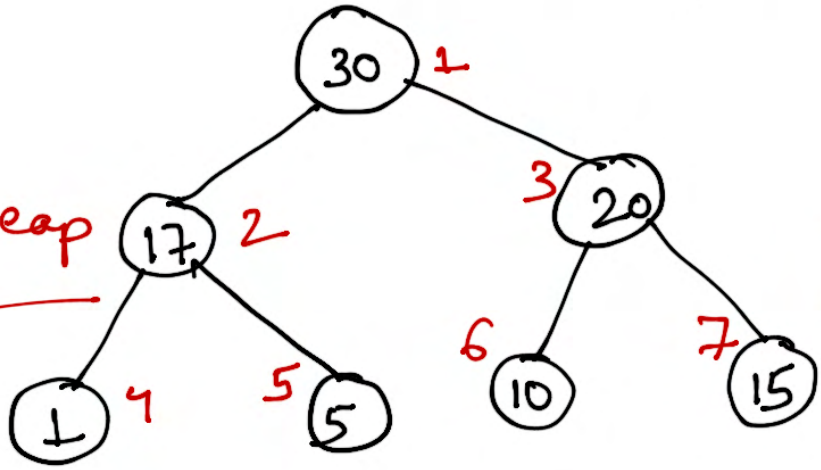
15	17	30	1	5	10	20
1	2	3	4	5	6	7



30	17	15	1	5	10	20
1	2	3	4	5	6	7

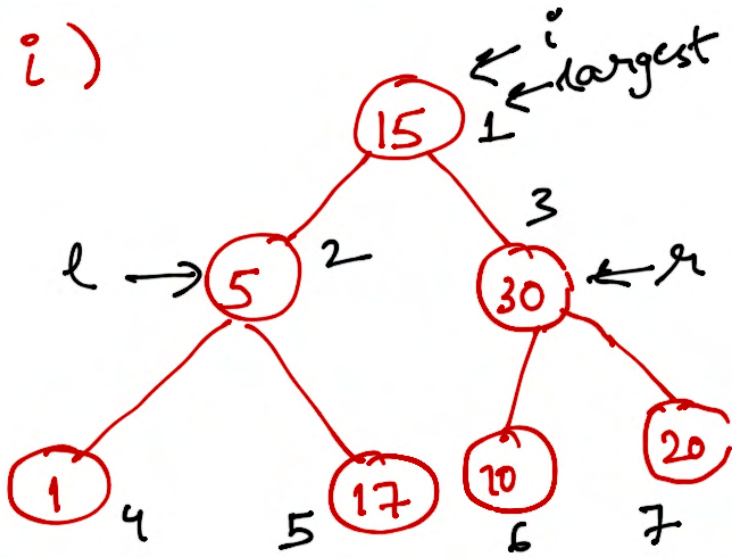
Do this until we reach end of array (i.e 7 in this case)

Max Heap

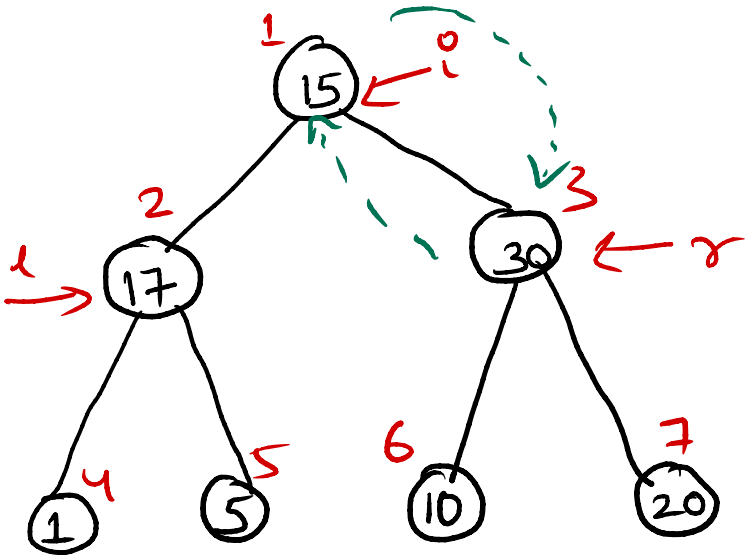


30	17	20	1	5	10	15
1	2	3	4	5	6	7

```
MaxHeapify (A, n, i)
{
 int largest = i ;
 int l = 2*i ;
 int r = 2*i + 1 ;
while(l ≤ n && A[l] > A[largest])
      { largest = l ;
      }
while(r ≤ n && A[r] > A[largest])
      { largest = r;
      }
if(largest != i)
  { swap( A[largest] ,A[i] );
     MaxHeapify(A, n, largest);
  }
}
```



Time Complexity : O(n)



largest = i = 1

a[l] > a[largest]

a[2] > a[1]

17 > 15

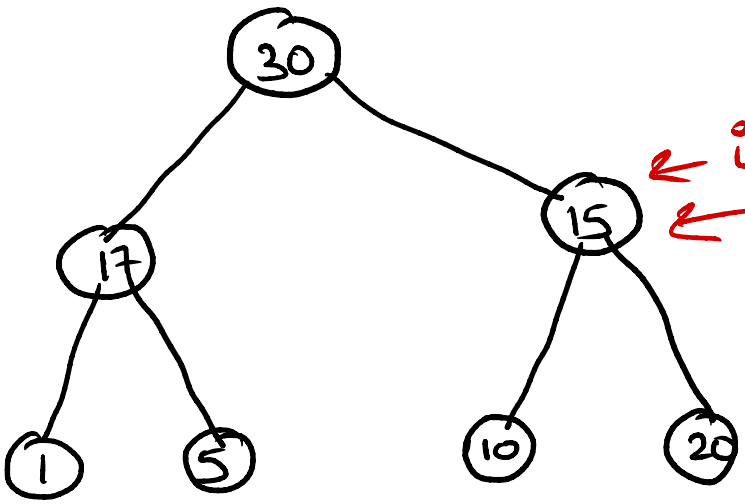
largest = l = 2

a[r] > a[largest]

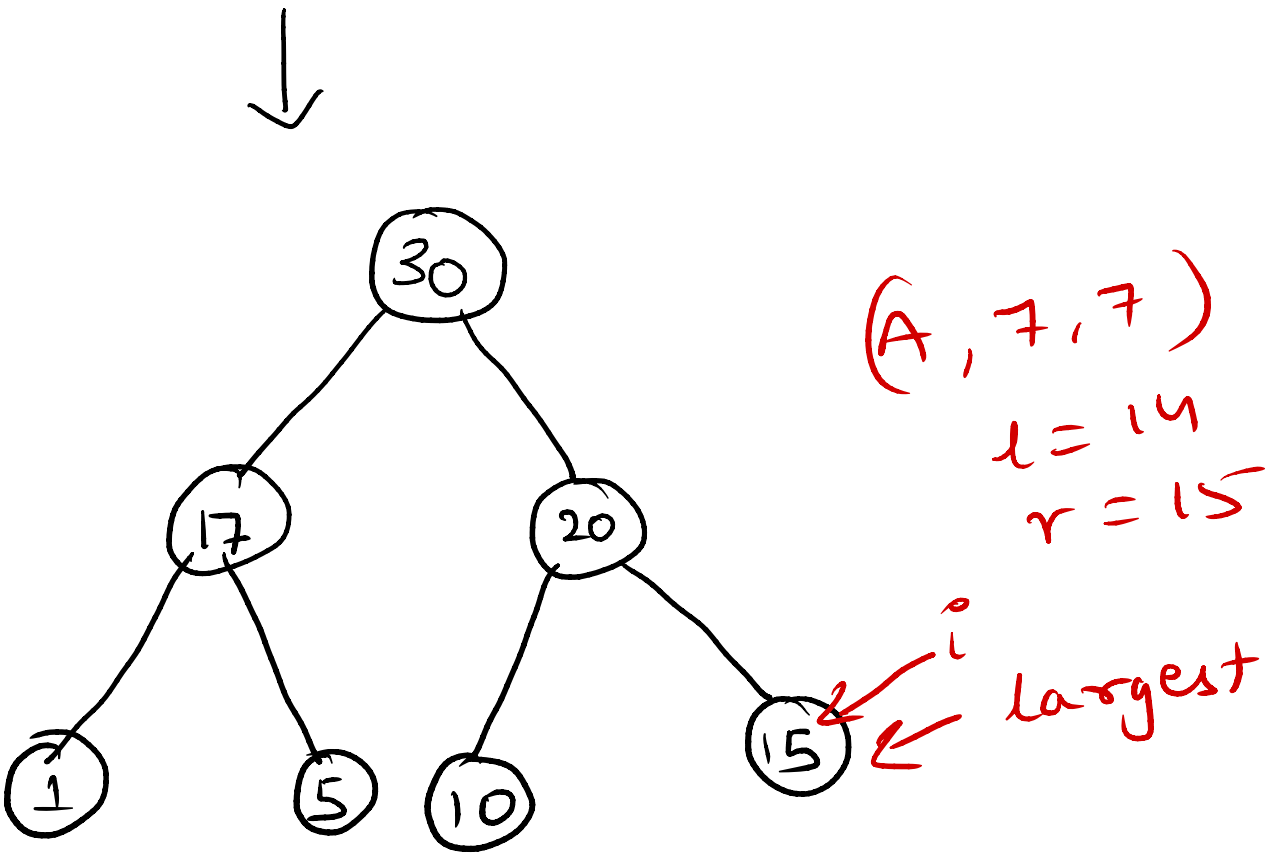
a[3] > a[2]

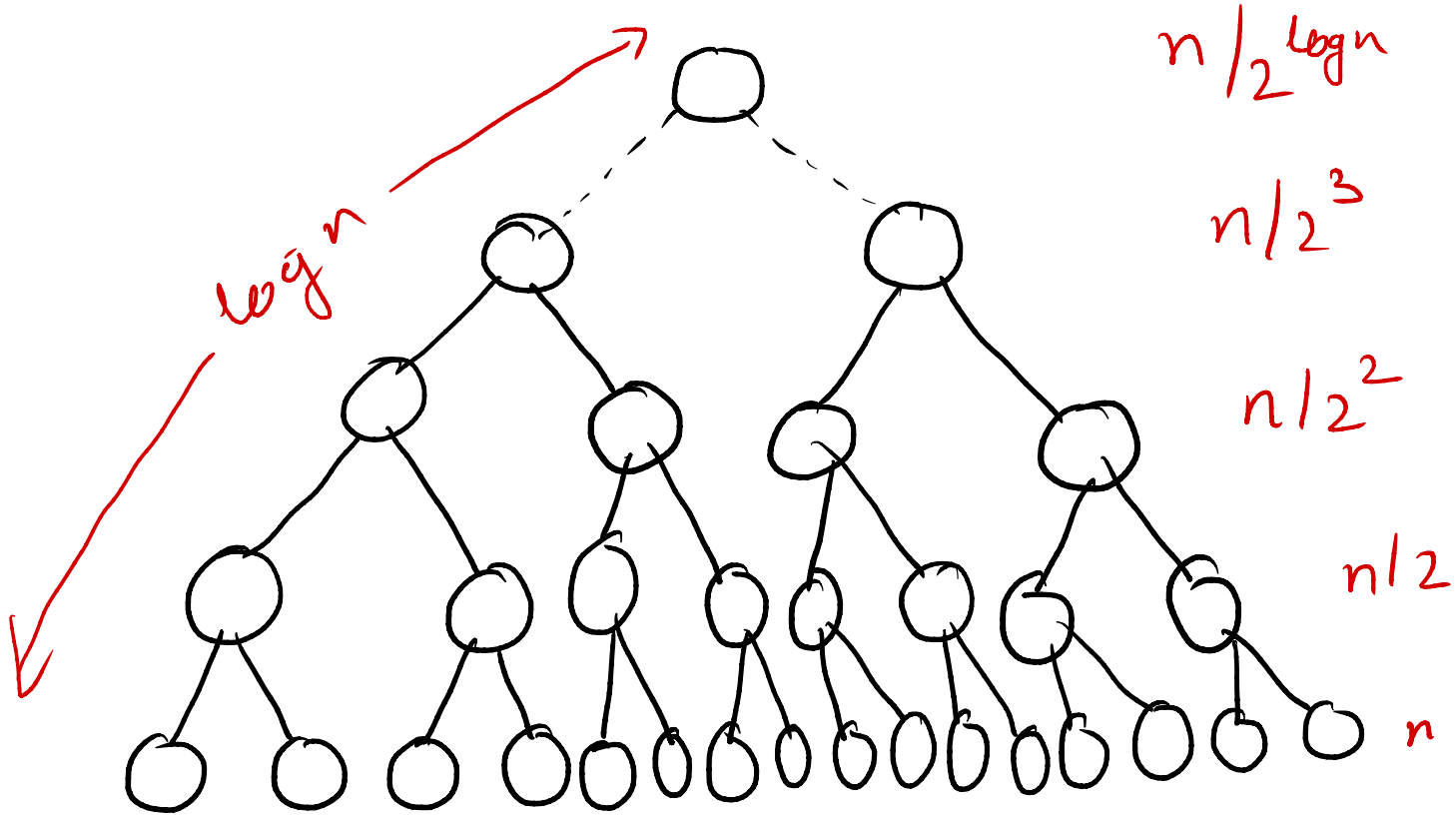
30 > 17

largest = 3



(A, 7, 3)


30
17
20
1
5
10
15
(A, 7, 7)
l = 14
r = 15
i
largest



Total swaps = S

$$S = \frac{n}{2^0} \times 0 + \frac{n}{2^1} \times 1 + \frac{n}{2^2} \times 2 + \frac{n}{2^3} \times 3 + \cdots + \frac{n}{2^{\log n}} \times \log n$$

$$= n\left[0 + \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \cdots + \frac{\log n}{2^{\log n}}\right] \quad \text{---(1)}$$

Multiply eqn(1) by $\frac{1}{2}$

$$\frac{S}{2} = n\left[\frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \frac{4}{2^5} + \cdots + \frac{\log n}{2^{\log n+1}}\right]$$

---(2)

Substract (2) from (1)

$$\frac{S}{2} = n\left[\left[\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots \frac{1}{2^{\log n}}\right] - \frac{\log n}{2^{\log n + 1}}\right]$$

Use GP formula $r < 1$

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

$$\frac{S}{2} = n\left[\left(\frac{\frac{1}{2}\left(1 - \frac{1}{2^{\log n}}\right)}{\frac{1}{2}}\right) - \frac{\log n}{2^{\log n + 1}}\right]$$

$$= n\left[\left(\frac{2^{\log n} - 1}{2^{\log n}}\right) - \frac{\log n}{2^{\log n + 1}}\right]$$

$$= n\left[\frac{n-1}{n} - \frac{\log n}{n \cdot 2}\right]$$

$$\frac{S}{2} = n\left[\frac{2n - 2 - \log n}{2n}\right]$$

$$S = 2n - 2 - \log n \quad \therefore O(n)$$

```
HeapSort (A, n)
{
    for (i = n/2 ; i >= 1 ; i--)        // Build Max Heap
    {
        MaxHeapify (A, n, i);
    }
    for (i = n ; i >= 1 ; i--)          // Delete
    {
        Swap ( A[1], A[i] );
        MaxHeapify (A, n, 1);
    }
```

Time Complexity : $O(n \log n)$