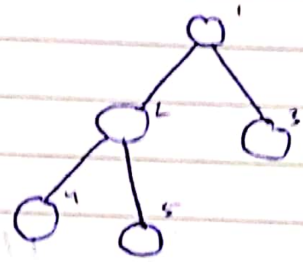
Heap

Date: 15-3-20 Page:

Complete Binary Tree

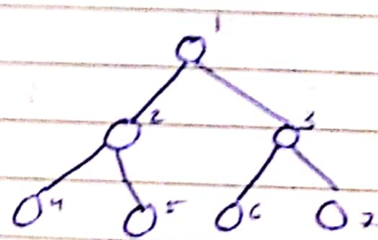
All leaf & Nodes are present in left-to-right manner.

No place where left-node is missing but right node is present.



Full Binary Tree

Each & Every node has full children.



max-nodes = $2^H - 1$ where H=height

Full Binary Tree is Complete B Tree.

Nodes are counted row wise

☆ Height of a complete Binary Tree / Heap

$\boxed{ceil(\log_2(N+1)) - 1}$ → if root is 0th position

$\boxed{ceil(\log_2(N+1))}$ → if root at 1th position

☆ Max nodes in Complete Bi Tree

$\boxed{2^{H+1} - 1}$ → if root at 0th pos.

$\boxed{2^H - 1}$ → if root at 1th pos.

Scanned with CamScanner

Date: ______ Page: ______

Tree position from Array →

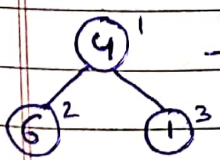
1	2	3	4	5	6
4	6	1	5	2	9

For any node at "i" index.

left child index → $2*i$
Right child index → $2*i+1$
its parent index → $\lfloor \frac{i}{2} \rfloor$

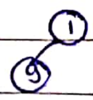
} Imp.

i	left child	Right child	Parent
1	2	3	0/null
2	4	5	1
3	6	–	1
4	Out-of-bound	–	2
5	–	–	2
6	–	–	3

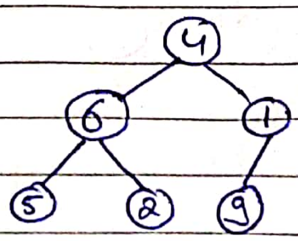


→ for index 1, its left child is element at index 2 & right child is element at index 3

→ for index 2, its left child is element at index 4

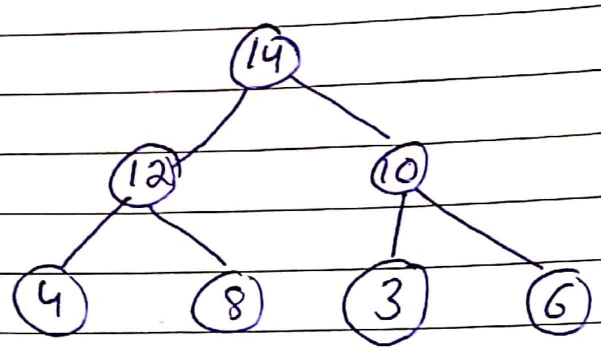


→ for index 3, its right child is element at index 7.

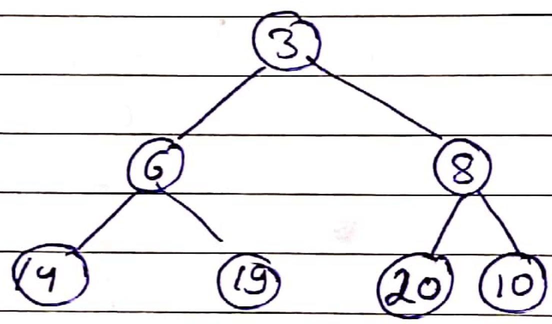


← Heap

- **Max-heap →** Heap in which root is bigger than its children.

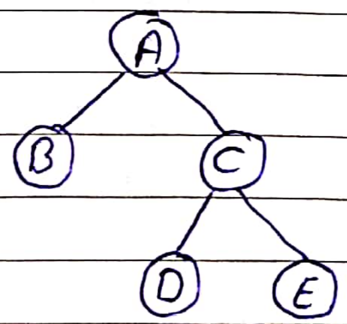


- **Min-heap →** Root is smaller than its children.



→ Missing Elements →

A	B	C	–	–	D	E



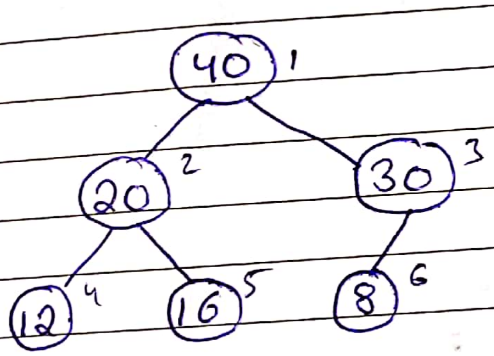
Date: ________ Page: ________

★ Insertion in Max-heap →

Complexity → $O(\log N)$

◦ Element move from leaf to Root (upward)

Insert → 58.



$7/2 \to 3$

1	2	3	4	5	6	7
40	20	30	12	16	8	58

(3) $58 > 30$ ✓

$3/2 \to 1$

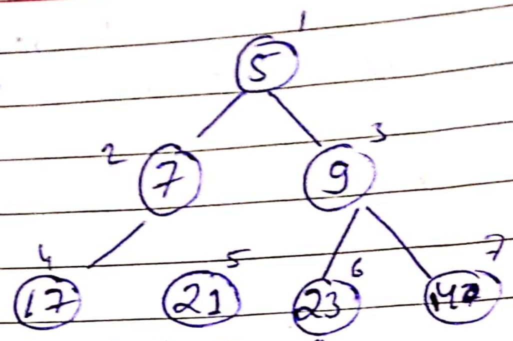
	3	7
	58	30

(1) $58 > 40$ ✓

	3	7
58	40	30

★ Creating Max-heap = $O(n \log n)$ = Creating Min-heap

Min-heap →



1	2	3	4	5	6	7	8
5	7	9	17	21	23	14	4

Insert - 4 →

8/2 ⇒ 4

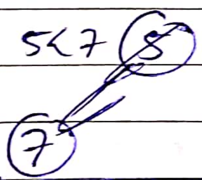
4   4 < 17   ✗ ✓

4	8
4	17

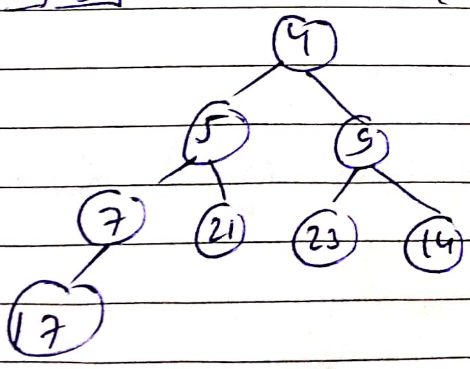
4/2   2   4 < 7   ✓

1	2	3	4	5	6	7	8
5	~~7~~ 4	9	7	21	23	14	17

2/2   1   5<7



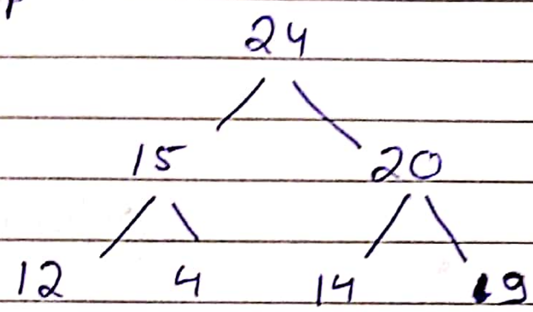
4	5	9	7	21	23	14	17



Deletion in Heap → O(log N) [Down-to-up]

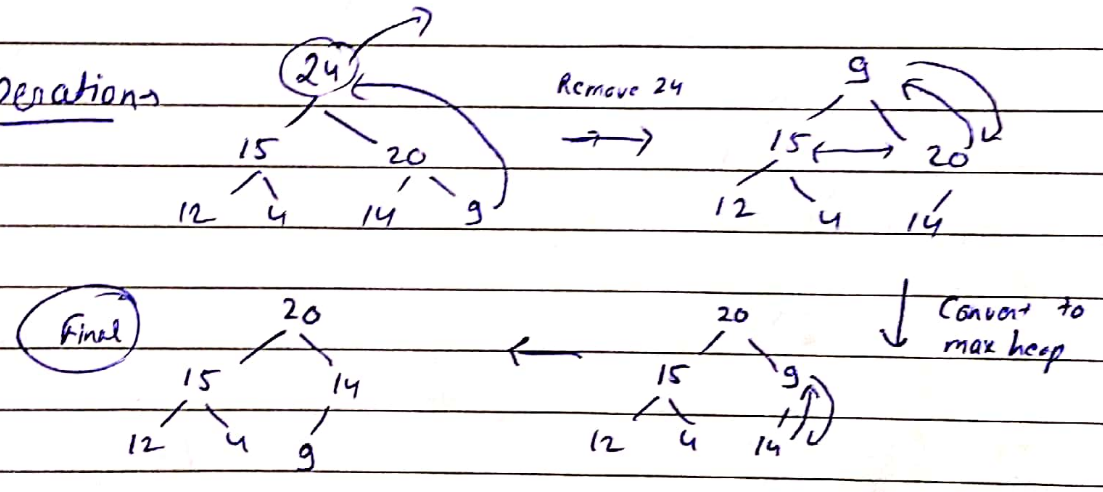
→ Only the root element is deleted.
→ Last element of heap takes its place.
→ After replacing, property of heap is restored (Max or Min)

Max-heap →

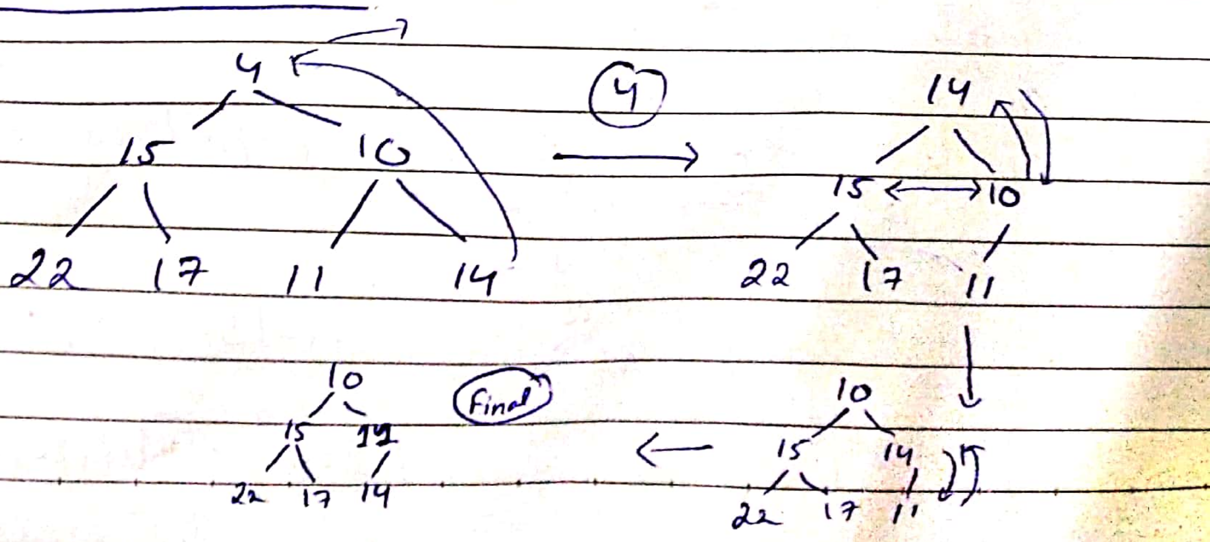


24	15	20	12	4	14	9

Remove operation →



Min-heap Removal →



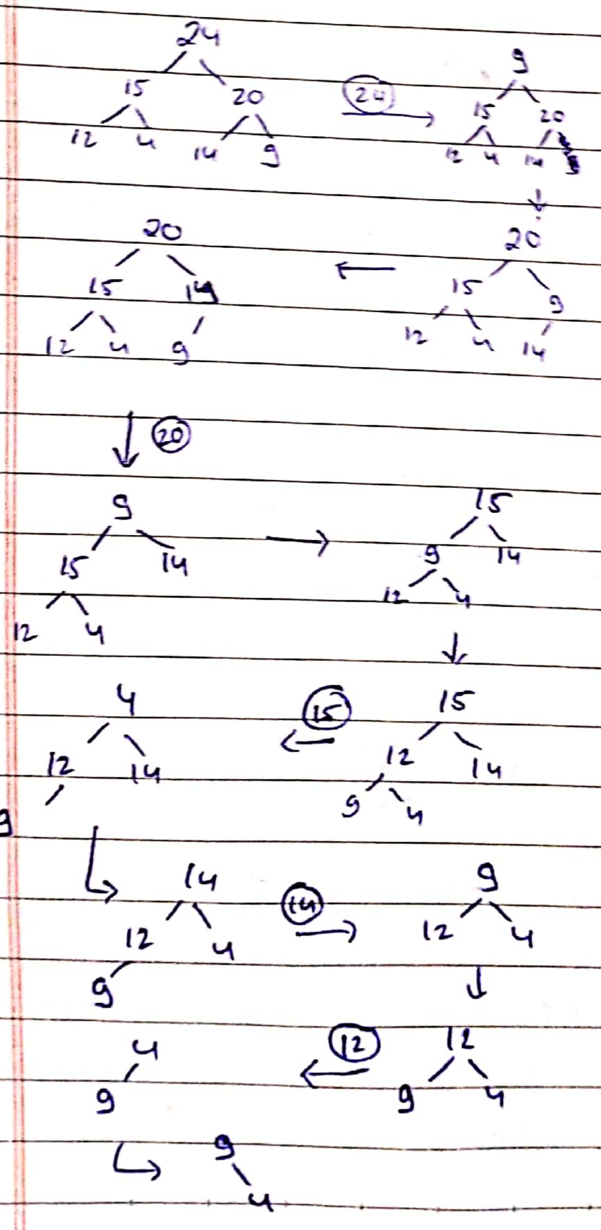
Sorting using Heap = Heap Sort

Time → $O(n \log n)$
Space → $O(1)$

Deleting element from Min/Max heap leads to sorted list.

Every removed element will be added to free space in list (same list)

From max-heap →



| 24 | 15 | 20 | 12 | 4 | 14 | 9 |

| 20 | 15 | 14 | 12 | 4 | 9 | 24 |

| 15 | 12 | 14 | 9 | 4 | 20 | 24 |

| 14 | 12 | 4 | 9 | 15 | 20 | 24 |

| 12 | 9 | 4 | 14 | 15 | 20 | 24 |

| 9 | 4 | 12 | 14 | 15 | 20 | 24 |

| 4 | 9 | 12 | 14 | 15 | 20 | 24 |

Sorted

Heap Sort Steps →

Step 1 → Convert Array into Max/Min heap

Step 2 → Remove root node & store removed element at last index.

Step 3 → Adjust the heap to attain its property & repeat 2-3.

Max-heap ⟶ Increasing Order Sort

Min-heap ⟶ Decreasing Order Sort.

☆ **Heapify** → Approach to create Max/Min heap with complexity $O(n)$ time.

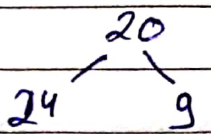
→ follows bottom-to-up approach.

1	2	3	4	5	6	7
24	15	20	22	4	24	9

Step 1 → $7/2 \to 3$ → until we reach index 3, all are leaf node. So we start from $i/2$

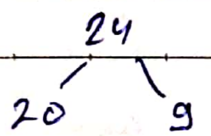
~~12~~ ~~4~~ ~~14~~ 9

Step 1 → $i = 3$
$2i = 6$
$2i+1 = 7$



yes

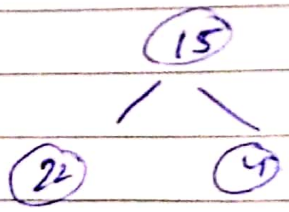
it is not max-heap yet so at lowest heigh, swap operations (1)



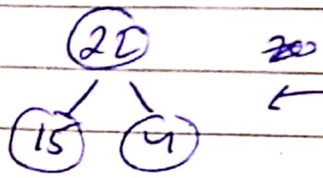
→ Max-heap.

Scanned with CamScanner

Step 2 → $i = 2$
$2i = 4$
$2i+1 = 5$

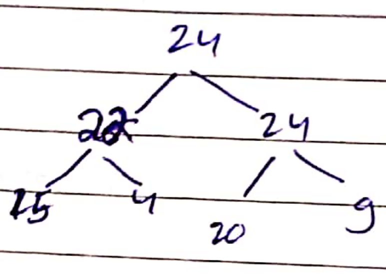


Operation require for rearragin 1.



← max heap.

Step 3 → $i = 1$
$2i = 2$
$2i+1 = 3$

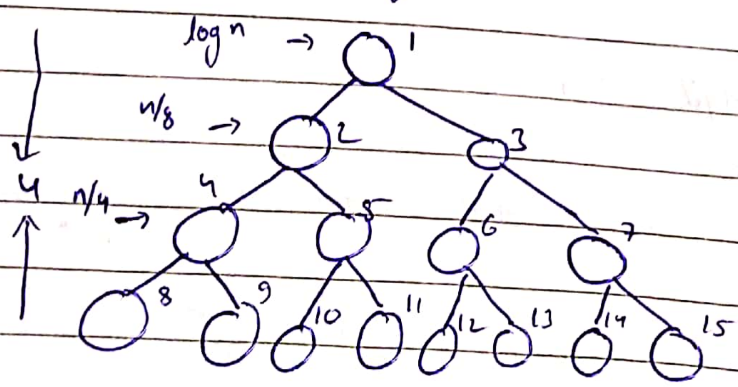


← max heap

So, max-heap was formed.

Total no. of Steps taken →

Depends on height of Tree ~~[illegible]~~. of length n.



← height of Tree. $\boxed{\log n}$

So, $n = 15$,

in general there will ~~be~~ always be
$n/4$ nodes with level 1,
$n/8$ nodes with level 2,
⋮
~~to~~ 1 node with level $\log n$.

[means lesser & lesser nodes with high level
level = height]

Date: ________ Page: ________

So, operations performed

at $n/4$ nodes $\rightarrow$ 1 constant operation

at $n/8$ nodes $\rightarrow$ $n/4$, $n/4$, 2 operation

at ~~to~~ $n/16$ nodes $\rightarrow$ $n/4, n/4, n/4,$ 3 operation

at 1 nodes. $\rightarrow$ $\log n$ operation

So

$$n/4(1) + n/8(2) + n/16(3) + \cdots 1(\log n)$$

let $n/4$ = constant

so $$\frac{1}{2^0} + \frac{2}{2^1} + \frac{3}{2^2} + \cdots \frac{k}{2^{k-1}} + \frac{1}{2^k}$$

$$= \frac{1}{2^0} + \frac{2}{2^1} + \frac{3}{2^2} + \cdots \frac{(k+1)}{2^k}$$

$$\Rightarrow 1 + 2 + 3 + \cdots (k+1) = k$$

$$O(n)$$

$\boxed{\text{So, Creating max-heap using Heapify} \rightarrow O(n)}$