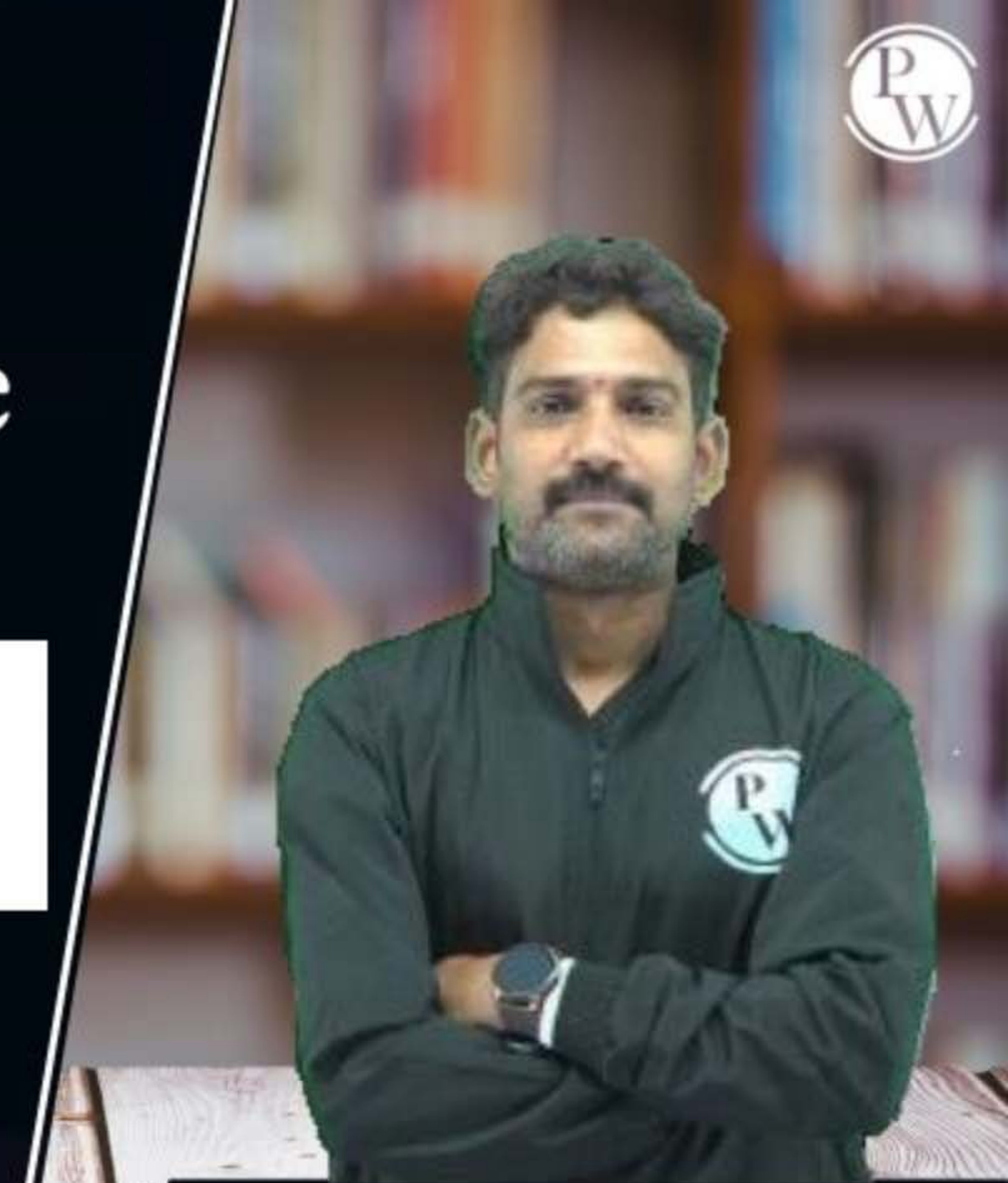
# Data Science & Artificial Intelligence

## Data Structures Through Python

### TREES

Lecture No.- 05



By- Satya sir

# Recap of Previous Lecture

- Binary Search Tree
  - Insertion
  - Search
  - Deletion
    - Leaf Node
    - A Node with 1 Child
    - A Node with 2 children
- Binary Heap : CBT, Parent > All children : Max-heap

  (OR)

  Parent < All children : min-heap
- Insertion into Binary Heap

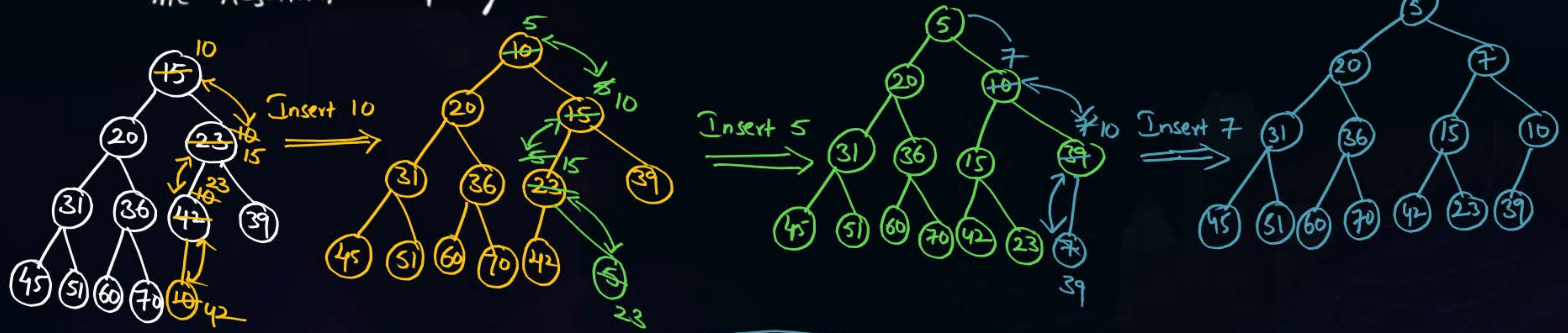
# Topics to be Covered

- Deletion from a Binary Heap
- Construction of a Binary Heap
- Heap orderings

## ~~2 mins Summary~~

H/w Question

#Q. Consider a Binary Heap, with Elements 15, 20, 23, 31, 36, 42, 39, 45, 51, 60, 70. The Resultant Heap after Insertion of Elements 10, 5, 7 in that Order is ______




Resultant heap: 5, 20, 7, 31, 36, 15, 10, 45, 51, 60, 70, 42, 23, 39

## Topic : Binary Heap - 2

# Deletion from Binary Heap

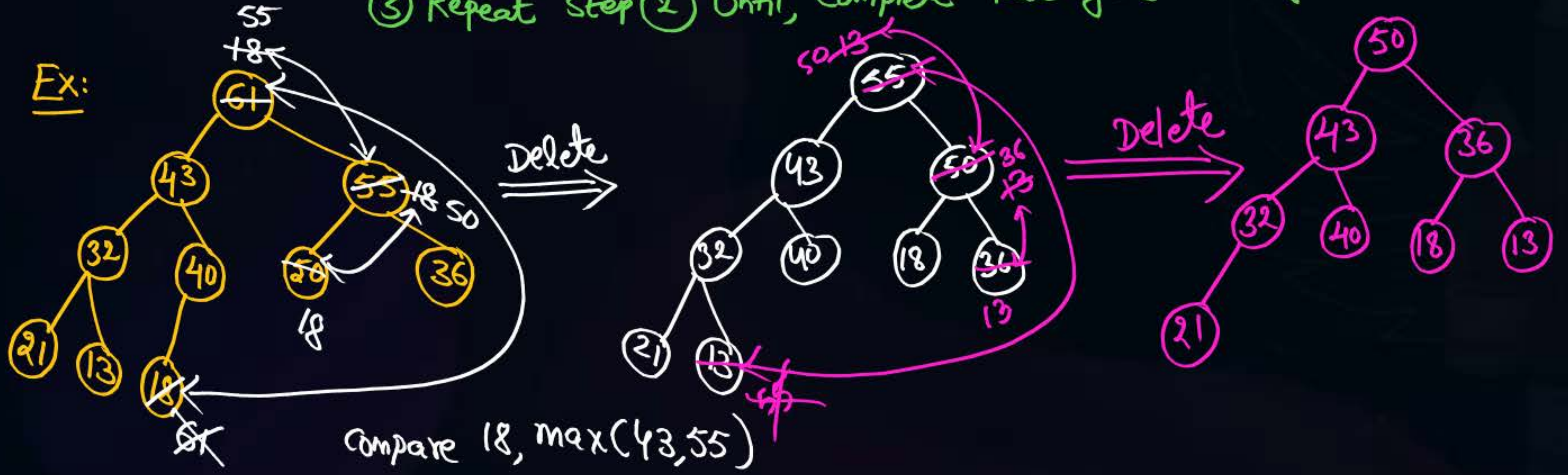
- In a Binary Heap, Deletion is done always at root Node only.

**PROCEDURE :**
① Swap Root Node with Last leaf Node, Then Delete Node at last leaf.

② Heapify
- Compare Parent < Max(children), Swap : Max-Heap
- Compare Parent > min(children), Swap : min-heap

③ Repeat step ② Until, complete Tree gets Heapified.

Ex:




Compare 18, max(43,55)

18 < 55 True, Swap

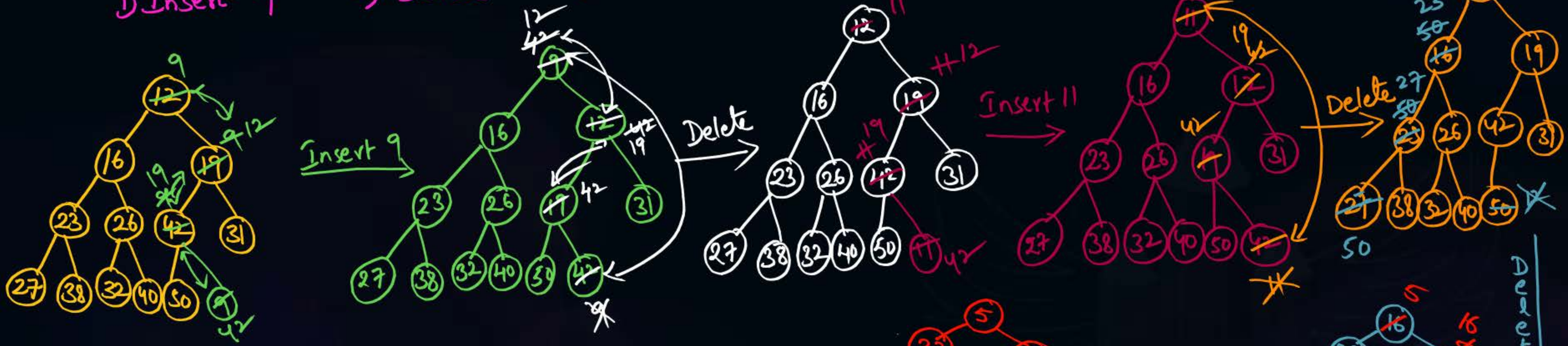
Resultant Max-heap

50, 43, 36, 32, 40, 18, 13, 21

#Q. Consider a min-heap with Elements 12, 16, 19, 23, 26, 42, 31, 27, 38, 32, 40, 50.

The Resultant min-heap after the following Operations in the same order is ______

1) Insert 9 2) Delete 3) Insert 11 4) Delete 5) Delete 6) Insert 5





Resultant min-heap: 5, 23, 16, 27, 26, 19, 31, 50, 38, 32, 40, 42

# Topic : Binary Heap - 2

***** Binary Heap is Used in Priority Queue and Heap sort Implementation.**

- In Max-heap, Biggest Value will always Present at Root Node.
- In min-heap, Smallest Value will always Present at Root Node.

- Let Priority of Nodes is directly Proportional to Value of Node. ⇒ Biggest value Node will have High Priority.
  So, Max-heap if used, PQ Can delete Element/Access Element at O(1) time.
- Let Priority of Nodes is Inversely Proportional to Value of Node ⇒ Smallest Value Node have high Priority
  So, min-heap is Used to Perform deletion/access Operations at O(1) time.
- In heap sort Implementation, For ascending order (small to big) ⇒ min heap is used : $\boxed{\text{Delete, heapify}}$ Repeat until all Sorted array is obtained.
  for descending order (Big to small) ⇒ Max heap is Used : Repeatedly Delete, Heapify.

# Topic : Binary Heap - 2

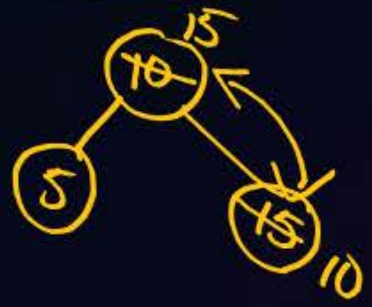
No. of Comp : $\lceil \log_2^n \rceil$ (or) $\lfloor \log_2^n \rfloor$

#. Construct a Max-heap, by Inserting Elements in the order : 5, 10, 15, 20, 25, 30, 35, 40

Insert 5

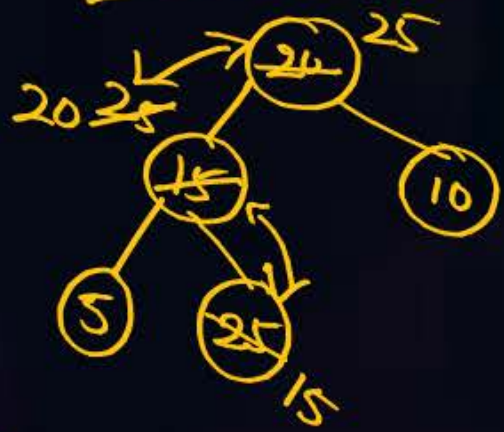
Insert 10

Insert 15



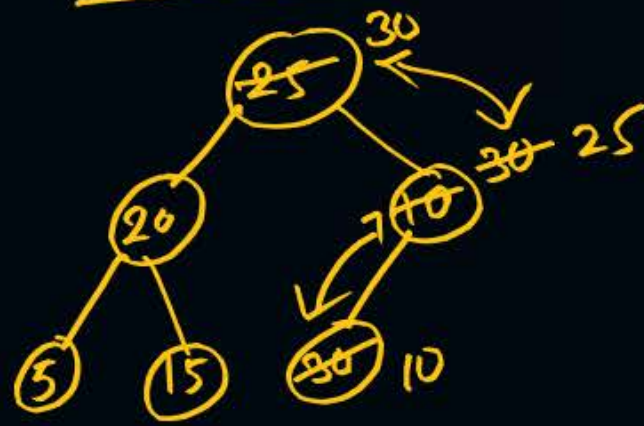
Insert 20

Insert 25

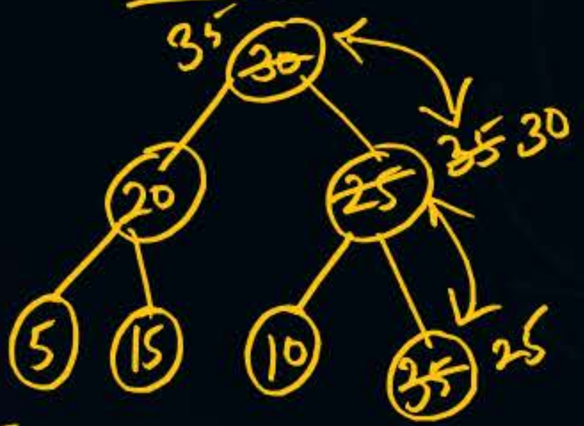


$\boxed{\text{Deletion T.C : } O(\log_2^n)}$

Insert 30

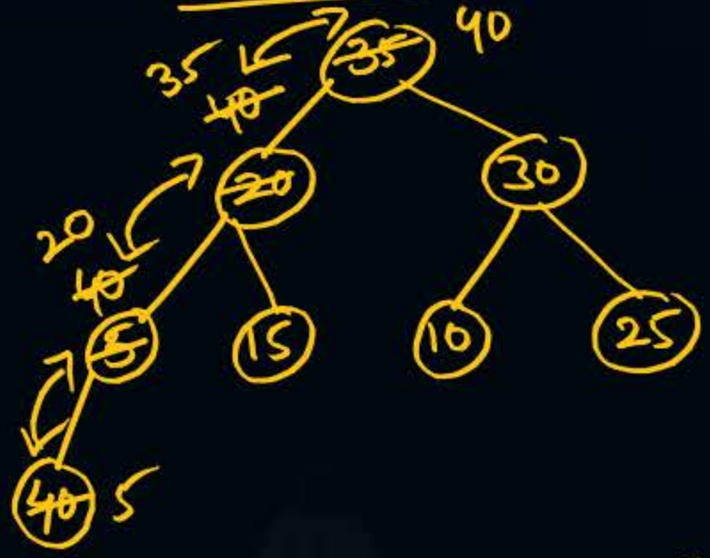


Insert 35



$\boxed{\text{Construction T.C} = O(n * \log_2^n)}$

Insert 40



Delete

Resultant Max-Heap

Insertion

$\boxed{\text{Time Complexity : } O(\log_2^n)}$

# Topic : Binary Heap - 2

$$^nC_r = \frac{n!}{(n-r)!\,r!}$$

## Heap orderings

The Total Number of Possible Binary Heap (Max/min) that can be formed with 'n' nodes is :

$$\boxed{T(n) = {}^{(n-1)}C_L * T(L) * T(R)}$$

$n <= 1 = 1$

Ex:

$n = 6$ Elements

$T(6) = {}^{(6-1)}C_3 * T(3) * T(2)$

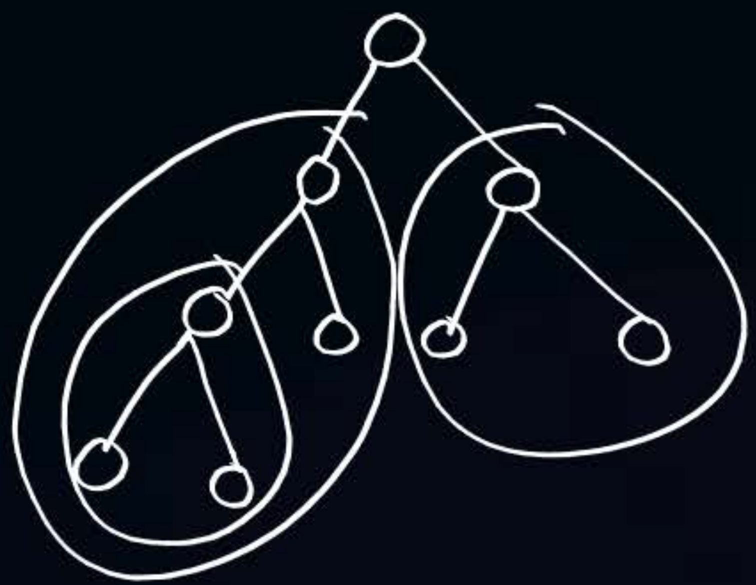
$T(L) = T(3) = {}^{(3-1)}C_1 * T(1) * T(1) = {}^2C_1 * 1 * 1 = 2$

$T(R) = T(2) = {}^{(2-1)}C_1 * T(1) * T(0) = {}^1C_1 * 1 * 1 = 1$

$= {}^5C_3 * 2 * 1 = \frac{5 \times 4 \times \cancel{3!}}{2! * \cancel{3!}} * \cancel{2} * 1 = \underline{\underline{20}}$

# Topic : Binary Heap - 2

Ex: $n=9$



$$T(9) = {}^{(9-1)}C_5 * T(5) * T(3)$$

$$= {}^{8}C_5 * \left[{}^{4}C_3 * T(3) * T(1)\right] * \left[{}^{2}C_1 * T(1) * T(1)\right]$$

$$= {}^{8}C_5 * \left[{}^{4}C_3 * \left[{}^{2}C_1 * T(1) * T(1)\right] * T(1)\right] * \left[2 * 1 * 1\right]$$

$$= {}^{8}C_5 * \left({}^{4}C_3 * 2 \times 1\right) * 2$$

$$= \frac{8 \times 7 \times \cancel{6} \times \cancel{5!}}{\cancel{3!} * \cancel{5!}} * \frac{4 \times \cancel{3!}}{1! * \cancel{3!}} * 2 \times 2 = 8 \times 7 \times 4 \times 2 \times 2 = \underline{\underline{896}}$$

# Topic : Implementation Of BST in Python

Go through it

```
#Node Creation
class Node:
  def __init__(self, data):
    self.left = None
    self.right = None
    self.data = data
```

```
#Node Insertion
  def insert(self, data):
# Compare the new value with the parent node
    if self.data:
      if data < self.data:
        if self.left is None:
          self.left = Node(data)
        else:
          self.left.insert(data)
      elif data > self.data:
          if self.right is None:
            self.right = Node(data)
          else:
            self.right.insert(data)
    else:
      self.data = data
```

# Topic : Implementation Of BST in Python

Traversal Implementation

```
def inorderTraversal(self, root):
    res = []
    if root:
      res = self.inorderTraversal(root.left)
      res.append(root.data)
      res = res + self.inorderTraversal(root.right)
    return res
```

```
def PreorderTraversal(self, root):
    res = []
    if root:
      res.append(root.data)
      res = res + self.PreorderTraversal(root.left)
      res = res + self.PreorderTraversal(root.right)
    return res
```

```
def PostorderTraversal(self, root):
  res = []
  if root:
    res = self.PostorderTraversal(root.left)
    res = res + self.PostorderTraversal(root.right)
    res.append(root.data)
  return res
```

# 2 mins Summary

- Deletion from Heap
- Construction of Heap
- Time Complexities
- Heap orderings.

THANK - YOU