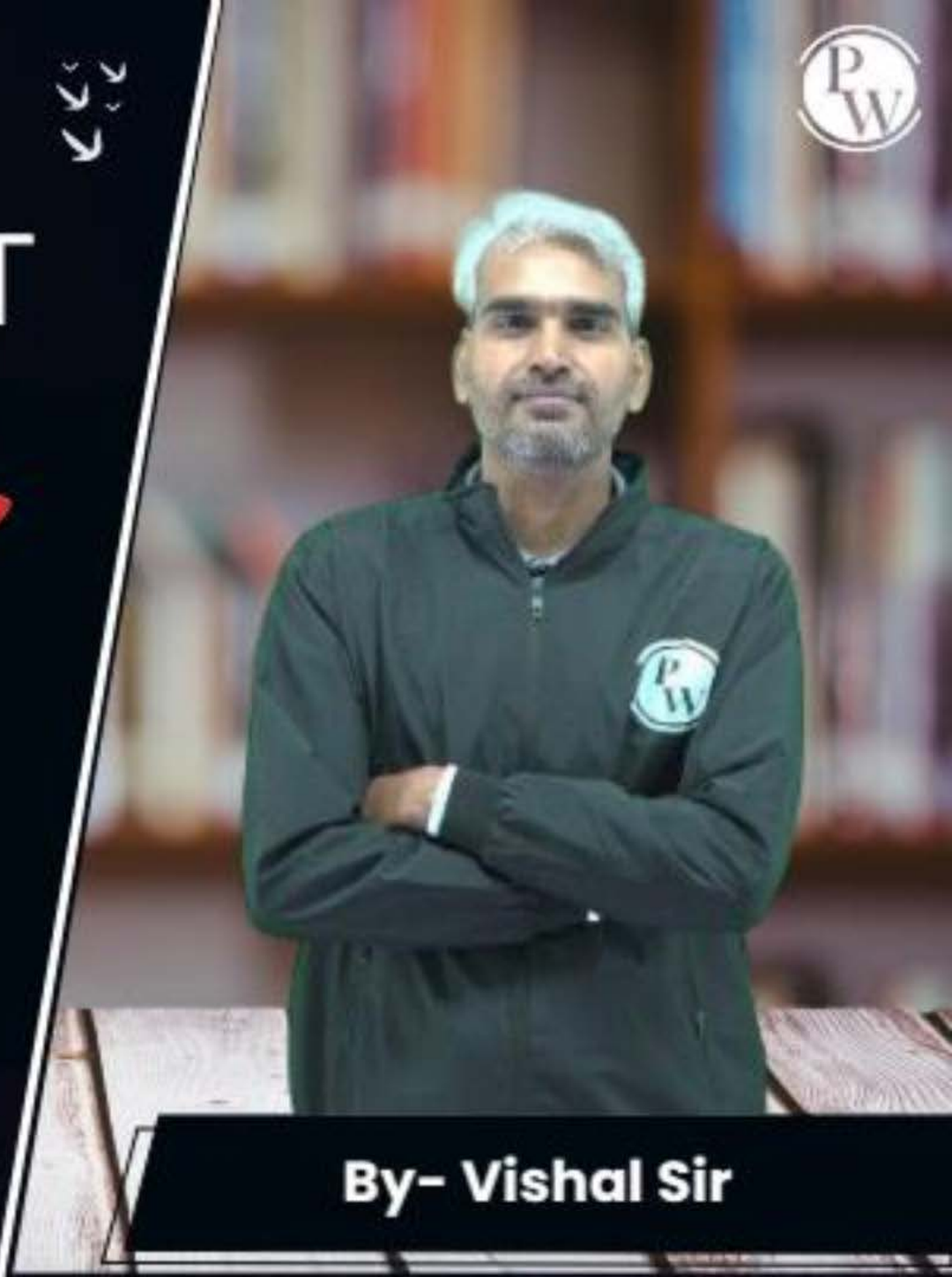
Computer Science & IT

Database Management System

File organization and indexing

Lecture No. 06



By- Vishal Sir

Recap of Previous Lecture

Topic

Practice questions

Slide

Topics to be Covered

- Topic: Insertion into B tree
- Topic: Deletion from B tree

Slide

* Insertion in B tree :→ - Insertion will always be performed at leaf node.

- ④ Let order of a node of B tree = P {∵ Max no. of keys = P-1}
- ① Search for the leaf node that can hold the key value that needs to be inserted
- ② If leaf node is not already full {i.e. number of keys < (P-1)} then simply insert the key in the ascending order
- ③ If leaf node is already full {i.e. no. of keys in leaf node = (P-1)} then because of insertion of this new key there will be overflow in that node. In that case we will insert the new key in the leaf node in ascending order, then we will split the node into two parts and we will promote the median position key to the parent node

③ (i) If parent node was not already full, then it will be able to accomodate the Promoted Key, and no further action will be required.

③ (ii)
- If parent node was already full, then split that node into two parts, and promote the median Position Key to the parent node of that node.
  - This may go upto root node, and if root node is also full, then we will split the root node as well into two parts, and the median position key will become the new root node {i.e.: Height of tree will increase by 1}

Q:- Let order of B tree = 4.
And keys to be inserted are.
90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25
in the same order.

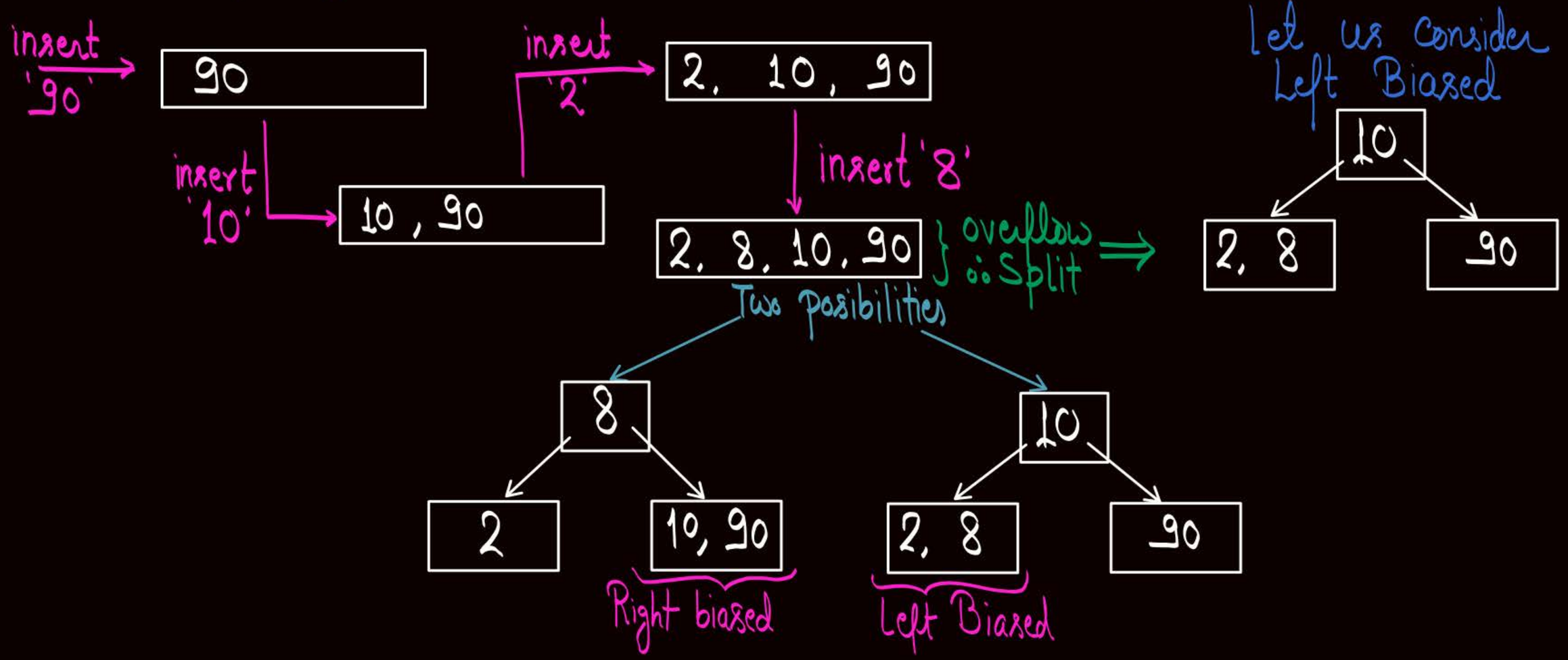
* Construct the B tree

Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

Min. no of keys w.r.t. root = 1

Min no. of child pointer w.r.t. root = 2

90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25

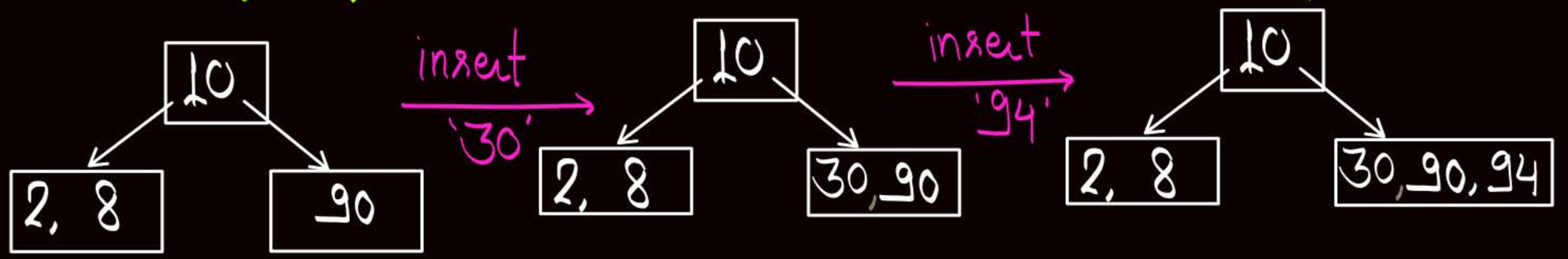


Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

Min. no of keys w.r.t. root = 1

Min no. of child pointer w.r.t. root = 2

90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25

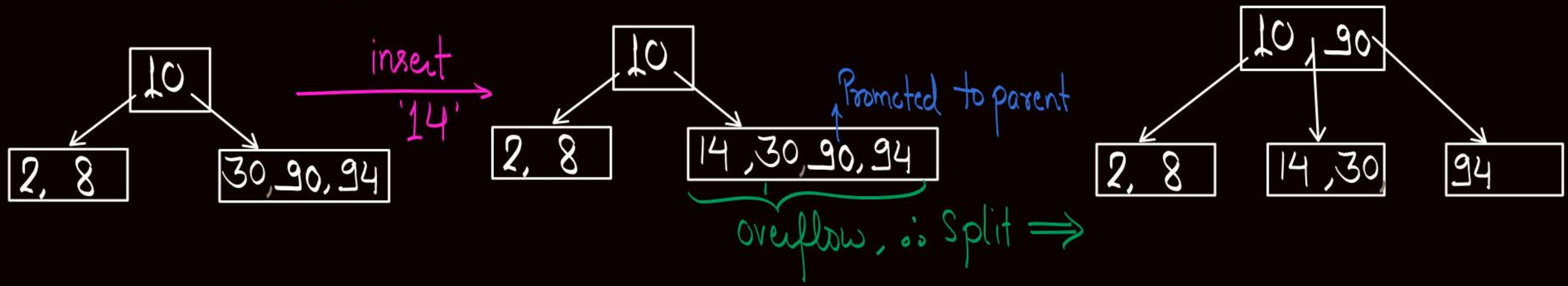


Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

Min. no of keys w.r.t. root = 1

Min no. of child pointers w.r.t. root = 2

90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25

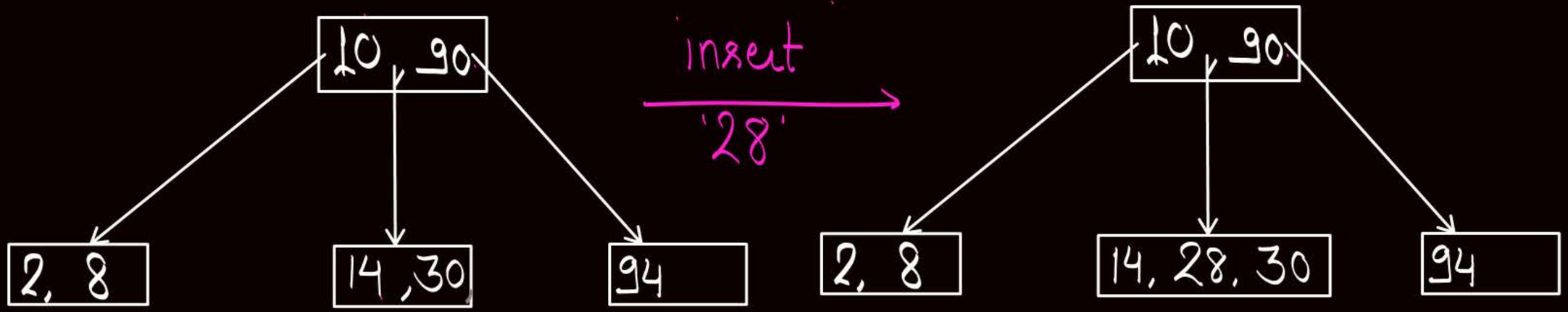


Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

Min. no of keys w.r.t. root = 1

Min no. of child pointer w.r.t. root = 2

90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25

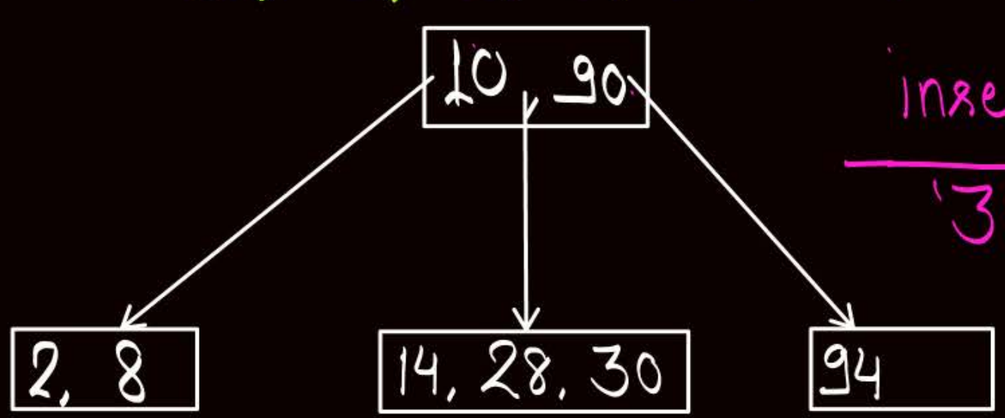


Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

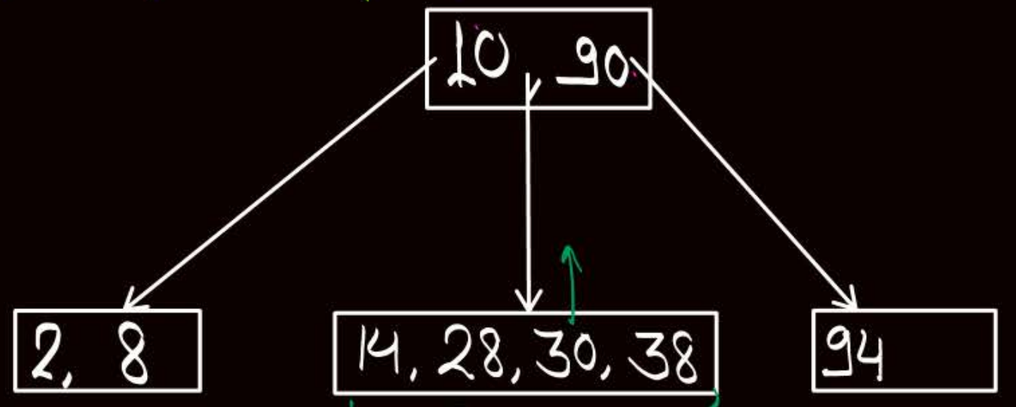
Min. no of keys w.r.t. root = 1

Min no. of child pointers w.r.t. root = 2

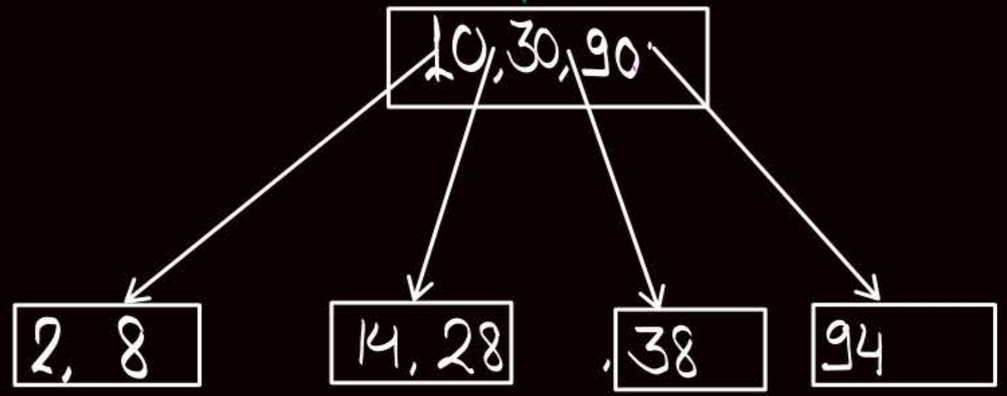
90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25



insert '38'



overflow ∴ Split

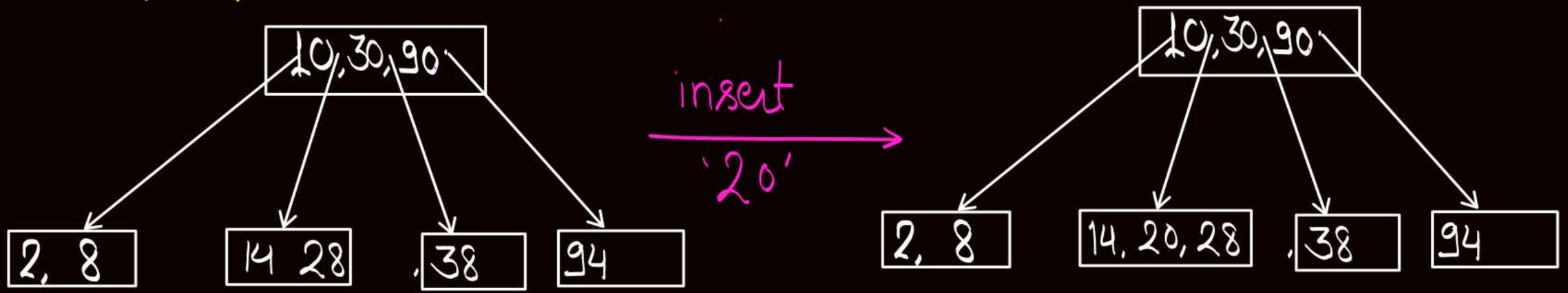


Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

Min. no of keys w.r.t. root = 1

Min no. of child pointers w.r.t. root = 2

90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25

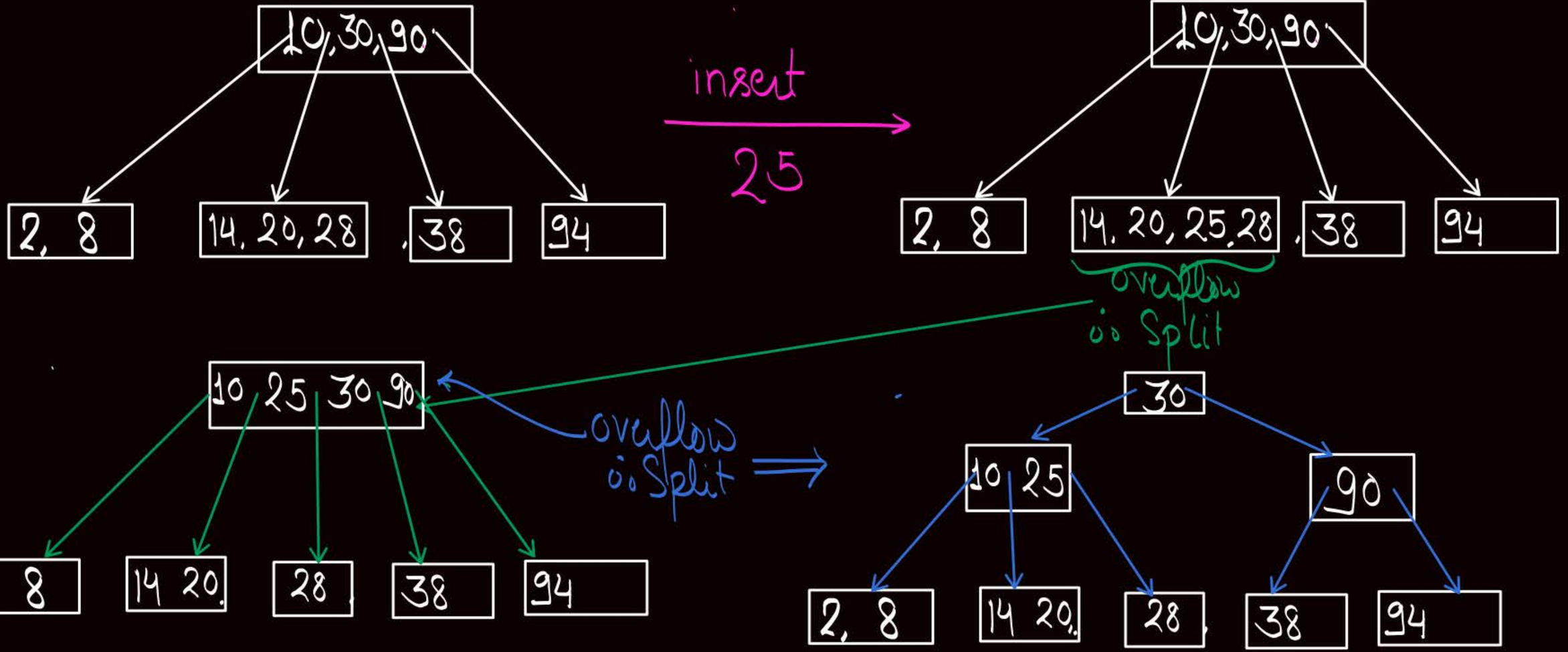


Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

Min. no of keys w.r.t. root = 1

Min no. of child pointers w.r.t. root = 2

90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25

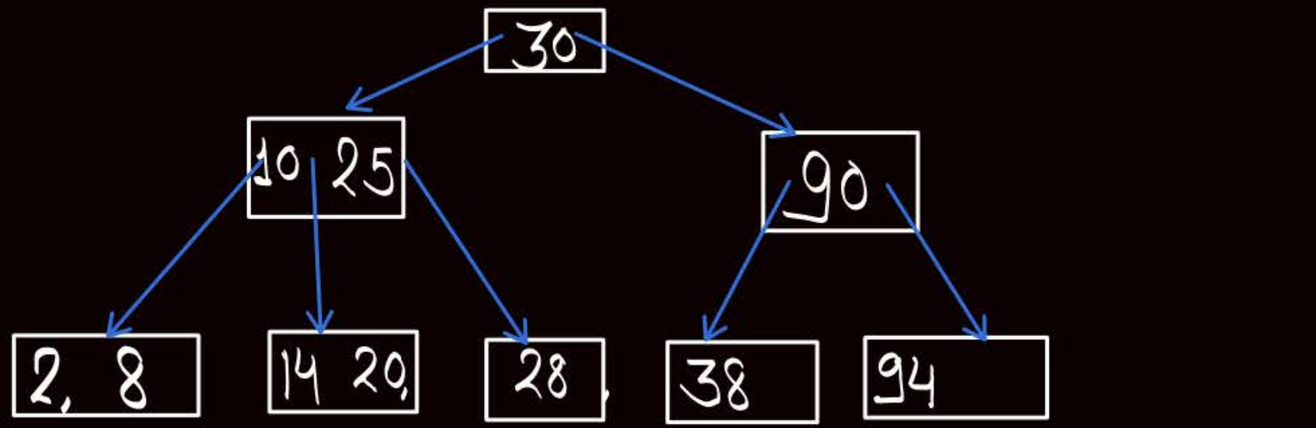


Q:- Let order of B tree = 4. ∴ Maximum no. of keys a node can hold = (4-1) = 3

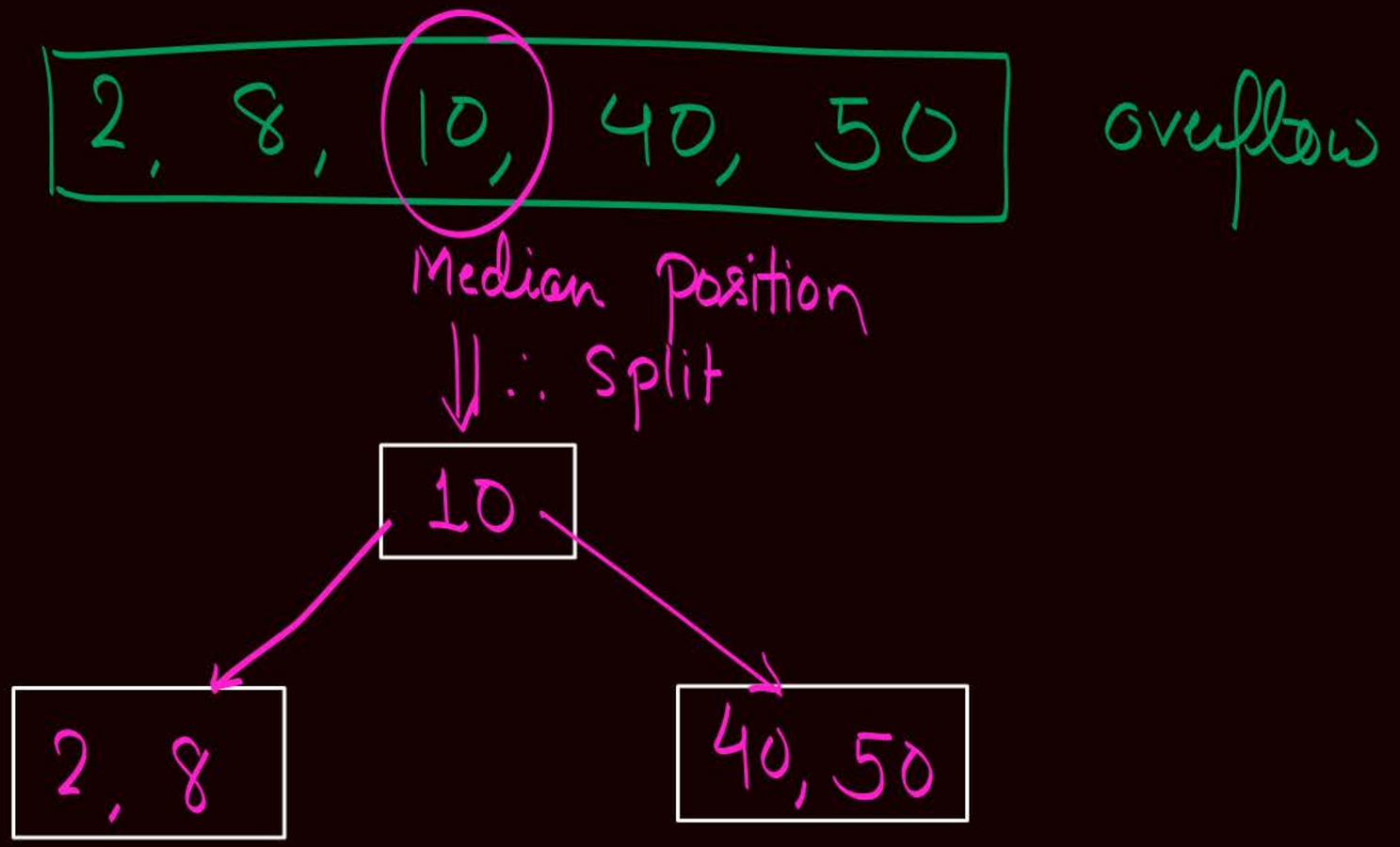
Min. no of keys w.r.t. root = 1

Min no. of child pointer w.r.t. root = 2

90, 10, 2, 8, 30, 94, 14, 28, 38, 20 & 25



Final tree after insertion of all keys


2, 8, 10, 40, 50
overflow
Median position
∴ split
10
2, 8
40, 50

Deletion from B tree

* Deletion from B Tree :- → { "key" that needs to be deleted may be in a leaf node or may be in an internal node }

③ If key that needs to be deleted is present in internal node of B tree, then replace it by

* ✓ in-order predecessor {i.e. largest key in left sub-tree}
  or
* ✓ in-order successor {i.e smallest key from right sub-tree}

And then delete the key from leaf node

④ If key is already at the leaf node delete it from that leaf node

* ① After deletion of the key from leaf node if leaf node still contain the minimum number of keys required, then no further action is needed

② After deletion of the key from leaf node if leaf node becomes deficient {i.e. Contains less than the minimum number of keys required}

2(i) Check with your immediate sibling if they can help.
{first check with left sibling, then check with right sibling}

ⓐ If left sibling can help then borrow (re-distribute) from left sibling
Largest key from the left sibling will be promoted to Parent node, and corresponding key from parent {i.e. Key b/w this two nodes} will be transferred to deficient node

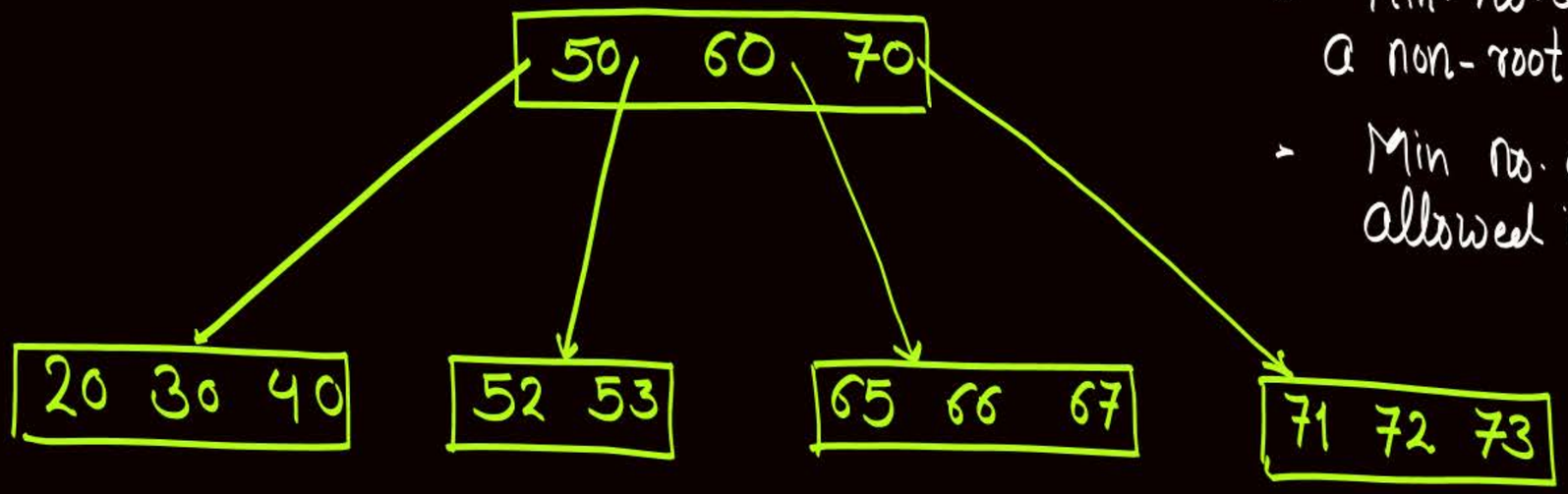
ⓑ If right sibling can help then borrow (re-distribute) from right sibling
Smallest key from right sibling will be promoted to Parent node, and corresponding key from Parent node {i.e; Key between those two nodes} will be transferred to deficient node

2 (ii) If none of your immediate sibling can help, then

- merge deficient node with one of its sibling along with one key (i.e. the key b/w deficient node & its sibling) from its parent node into a single node., as a result of this process the parent node may become deficient

- If parent node does not become deficient then stop.
- If parent node become deficient, then the same process will be repeated (i) Ask from their sibling if they can help.
  (ii) If not, then merge with one sibling along with one key from parent node
  - This may go upto root node, if root node becomes deficient then height of tree will reduce by "1".

Q:- Consider the following B tree of order = 5.

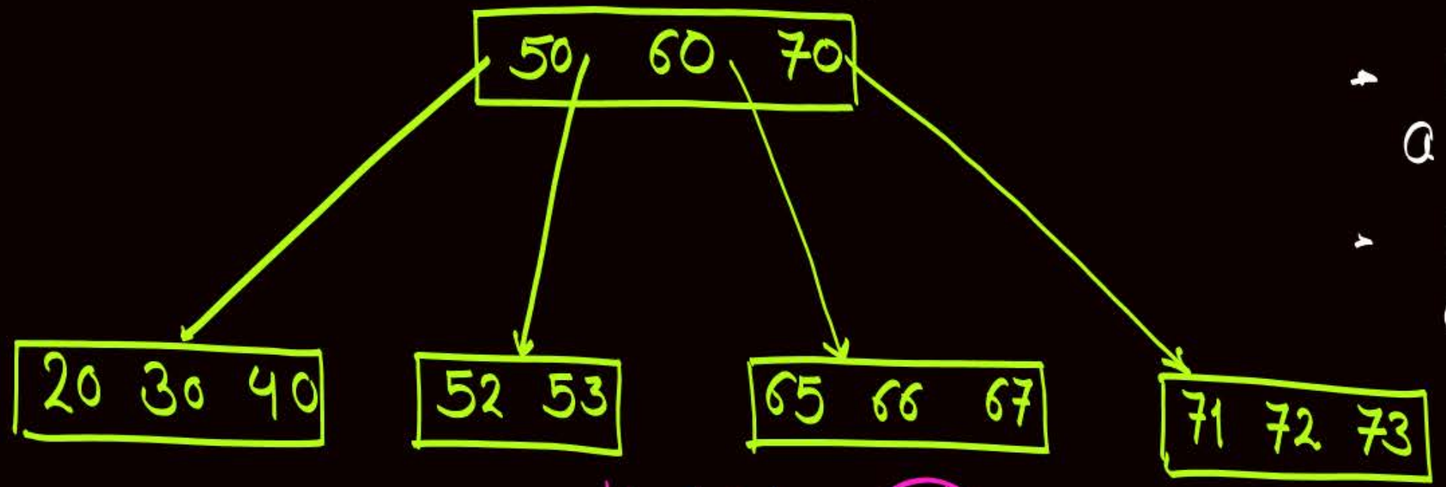


- Min. no. of keys in a non-root node = $\lceil \frac{5}{2} \rceil - 1 = 2$
- Min no. of keys allowed in a root node = 1

Delete the following keys from the above B tree in the same order

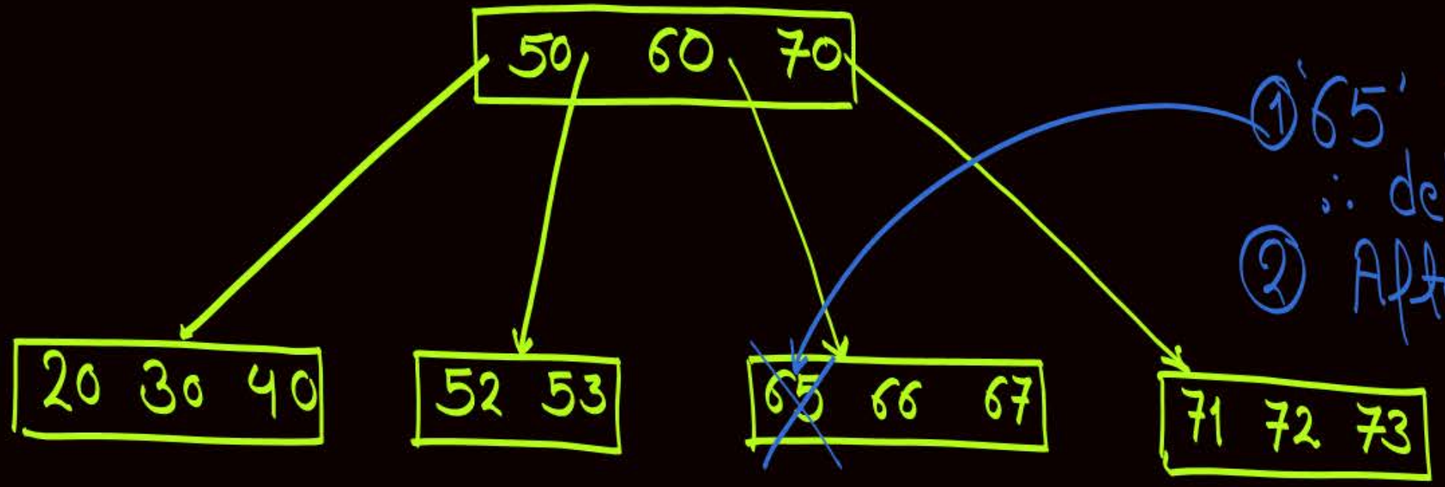
65, 53, 66, 52, 60

delete 65, 53, 66, 52, 60



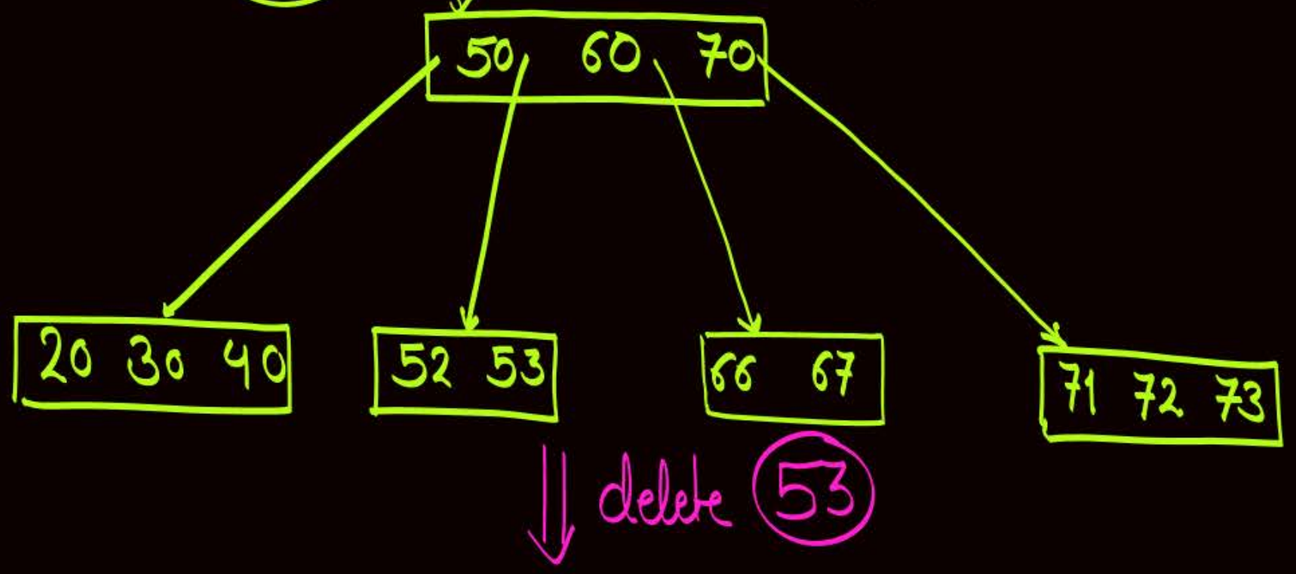
- Min. no. of keys in a non-root node = $\lceil \frac{15}{2} \rceil - 1 = 2$
- Min no. of keys allowed in a root node = 1

↓ delete (65)



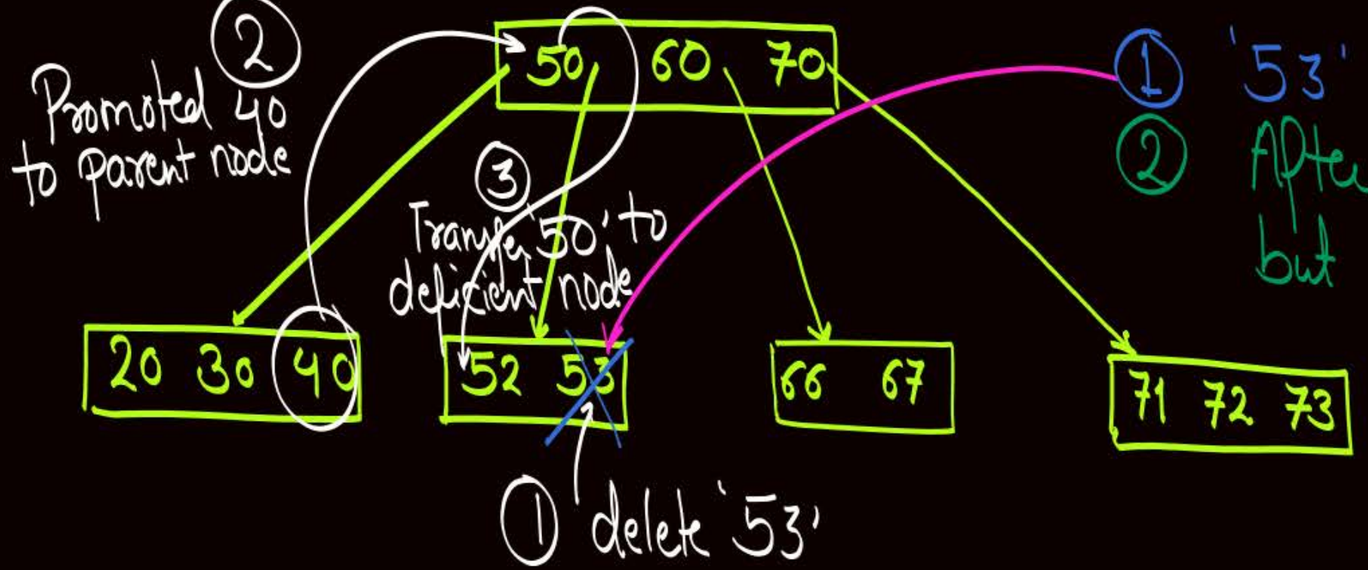
① '65' is already at the leaf node
∴ delete from leaf node
② After deletion, no underflow
∴ Stop

delete 65, 53, 66, 52, 60



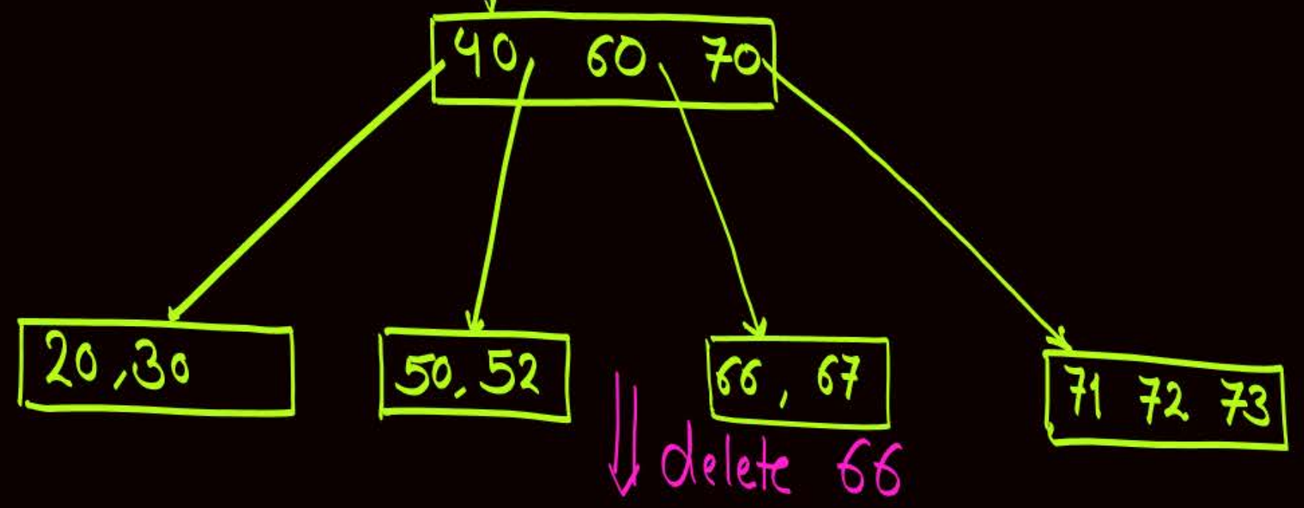
- Min. no. of keys in a non-root node = $\lceil \frac{15}{2} \rceil - 1 = 2$
- Min no. of keys allowed in a root node = 1

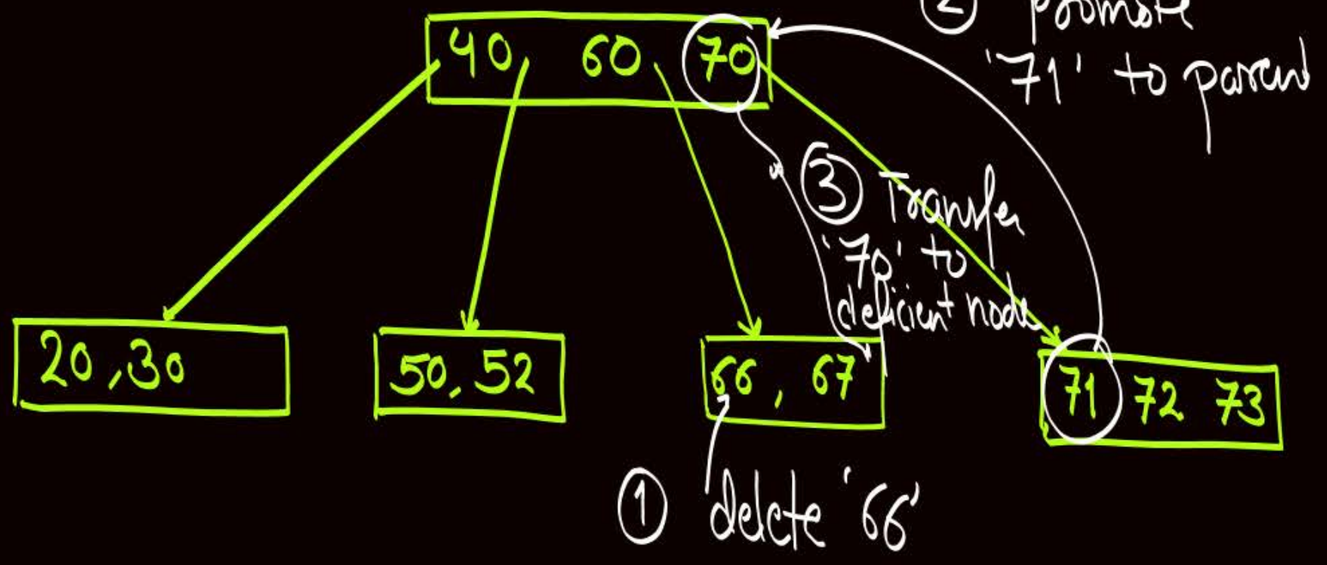
delete 53



1. '53' is at leaf node, ∴ delete from leaf node
2. After deletion leaf node becomes deficient but left sibling can help
   ∴ Redistribute from left sibling via parent

delete 65, 53, 66, 52, 60



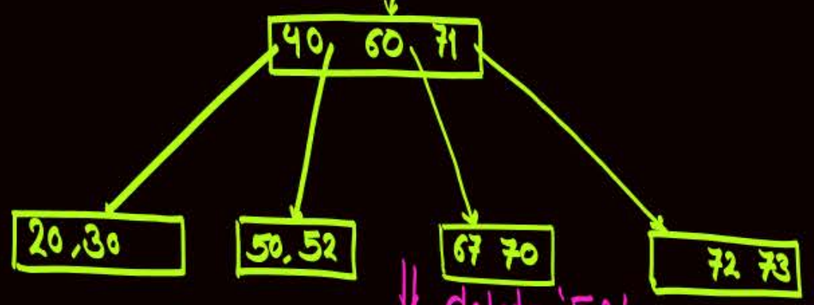


- Min. no. of keys in a non-root node = $\lceil \frac{5}{2} \rceil - 1 = 2$
- Min no. of keys allowed in a root node = 1

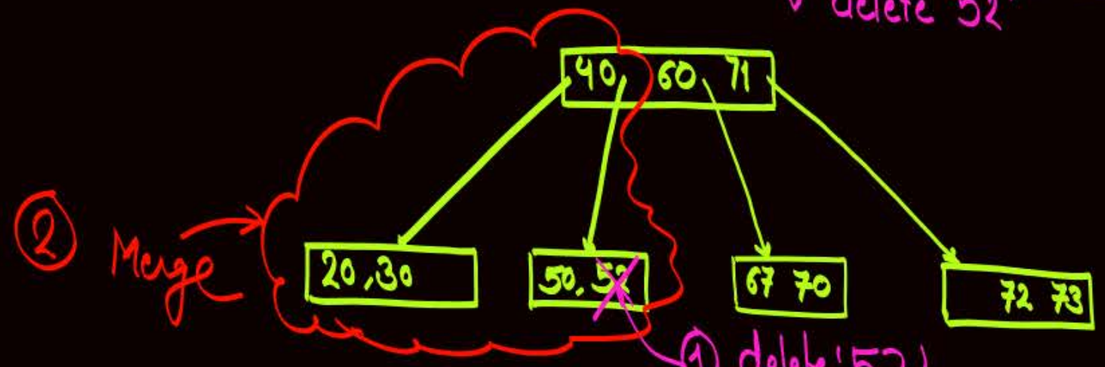
① Key is already at leaf node
∴ delete from leaf node

② After deletion, leaf node becomes deficient and left sibling can not help, but right sibling can help
∴ Redistribute from right sibling

delete 65, 53, 66, 52, 60

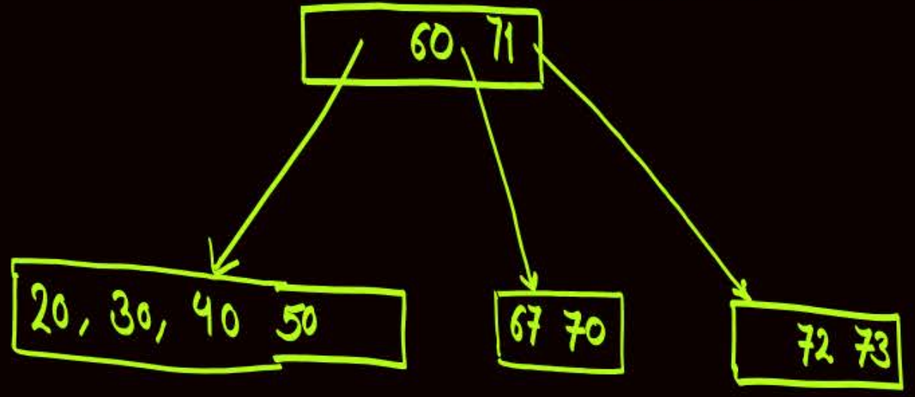


↓ delete '52'



② Merge

① delete '52'



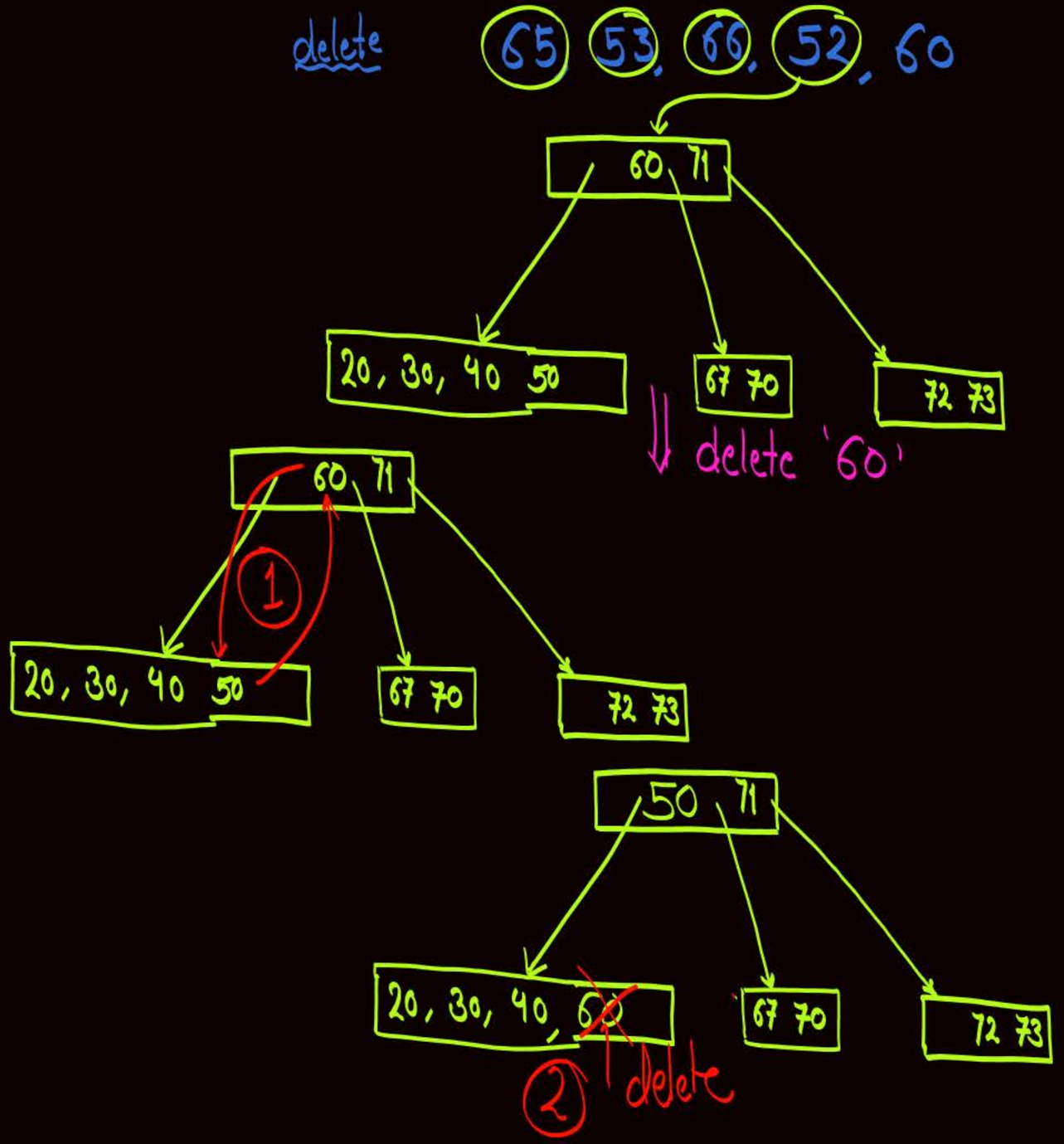
- Min. no. of keys in a non-root node = $\lceil \frac{5}{2} \rceil - 1 = 2$
- Min no. of keys allowed in a root node = 1

① '52' is at leaf node, ∴ delete it

② After deletion, leaf node becomes deficient, and none of its immediate siblings can help

Let us merge with left sibling ∴ Merge with one of its sibling along with '1' key from Parent node

delete (65) (53) (66) (52) 60



- Min. no. of keys in a non-root node = $\lceil \frac{5}{2} \rceil - 1 = 2$
- Min no. of keys allowed in a root node = 1

① '60' is at internal node
∴ Replace by in-order predecessor
or
in-order successor
{ let us replace by in-order predecessor i.e, '50'.

② After replacement, delete '60' from leaf node

③ After deletion, no underflow ∴ Stop

2 mins Summary

Topic Insertion into B tree

Topic Deletion from B tree

Slide

THANK - YOU

Slide