Construct a B-Tree of order 5 with the following set of data.

4 8 26 11 2 16 17 5 1 19 and 23.

[Web sources].

order (m) = 5

key (k) = m-1 = 5-1 = 4.

max. keys = m-1 = 4

min. keys = m/2 = ⌈2.5⌉ = 2.

Insertion :-

4 8 26 11

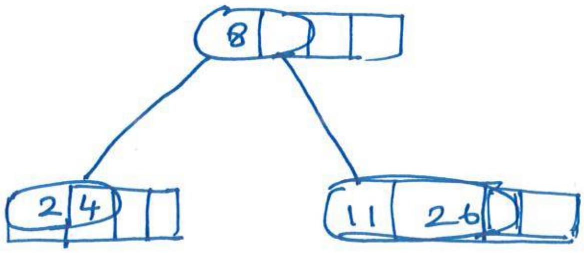
↑ insert 2

4 | 8 | 11 | 26

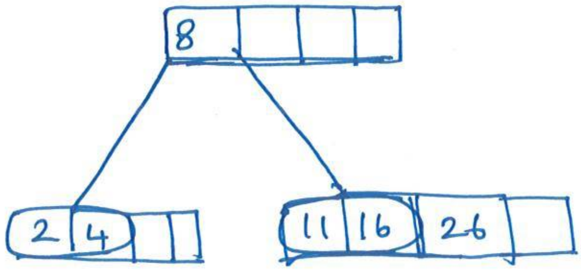
2 4 8 11 26.

↑

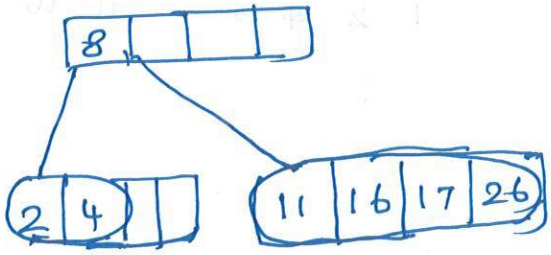
Insert 8:



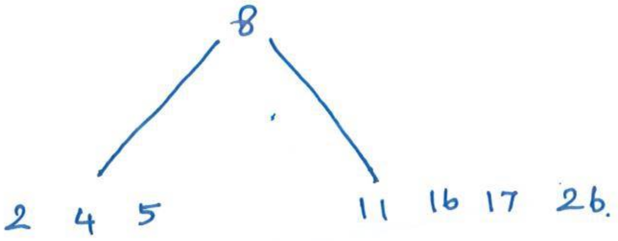
Insert 16:



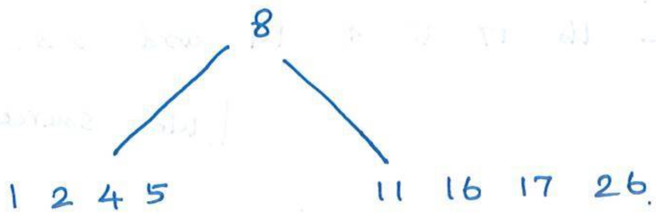
Insert 17:



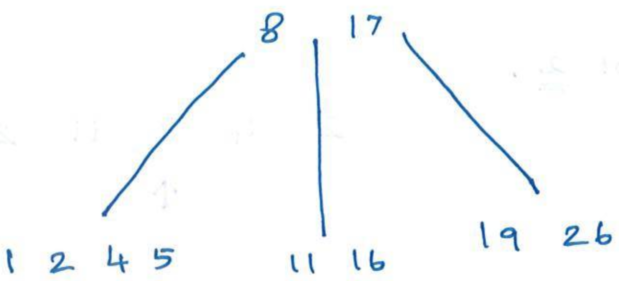
Insert 5:

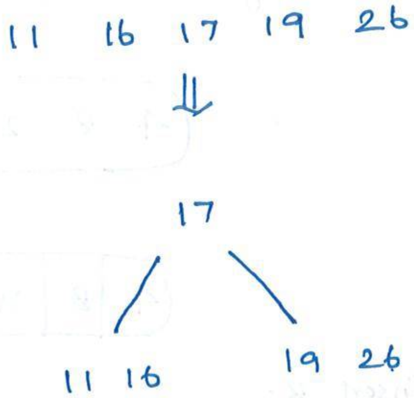


Insert 1:

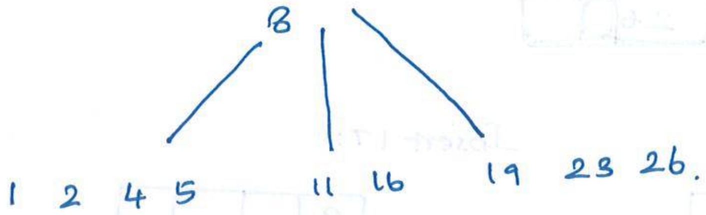


Insert 19:



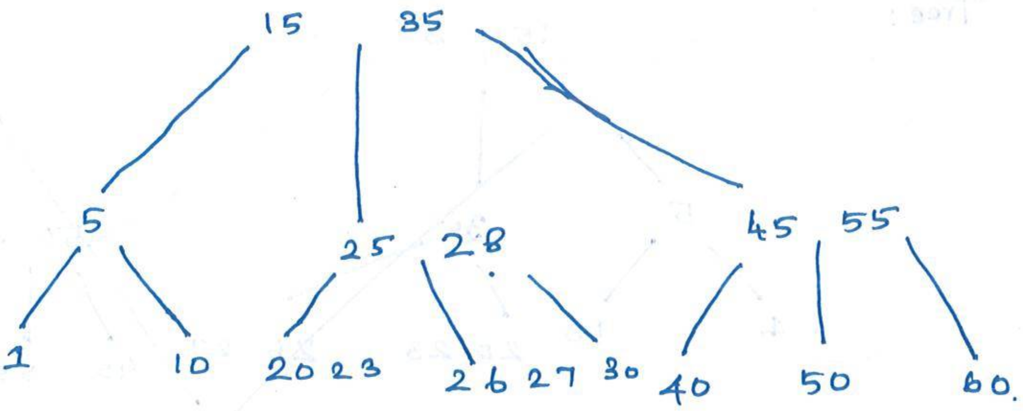


Insert 28:



B- Tree deletion:

Example :



Delete : 27

Rules for Deleting.

→ While deleting a tree, a condition called "Underflow".

→ underflow occurs when a node contains less than the minimum of keys it should hold.

Two ways to address this issue:

✓ Inorder Predecessor

✓ Inorder Sucessor.

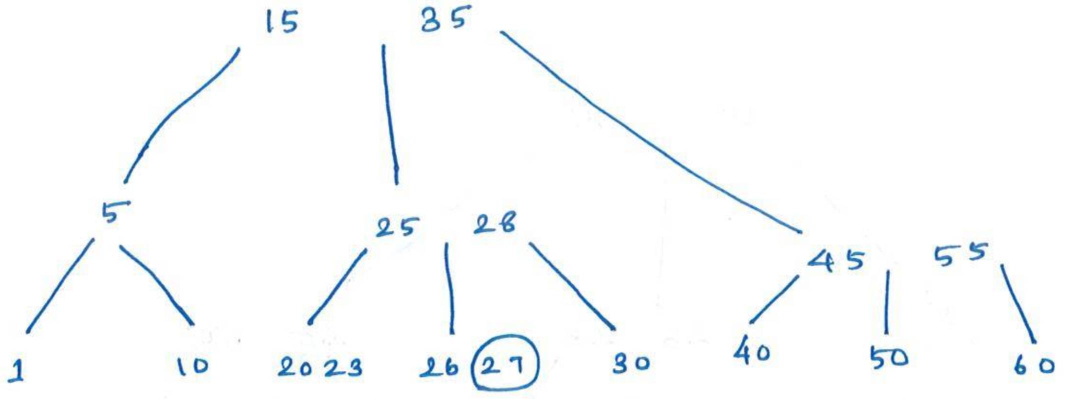
Inorder Predecessor.

The largest key on the left child of a node is called → inorder predecessor.

Inorder successor:

The smallest key on the right child of a node is called → Inorder successor.

Delete 27:



Deleting 27 <u>doesn't violate property 1</u>

Property 01 ⊗ The deletion of the keys doesn't violate property of the minimum number of keys a node should hold.

Deleting : 26.

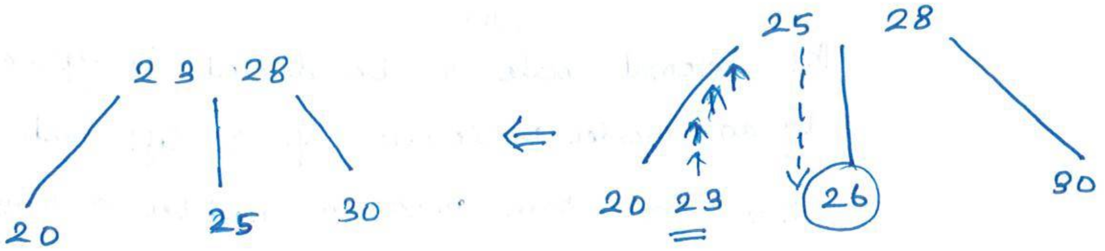
Deleting 26 results in above tree, It violates the property 01.

<u>Solution:</u>

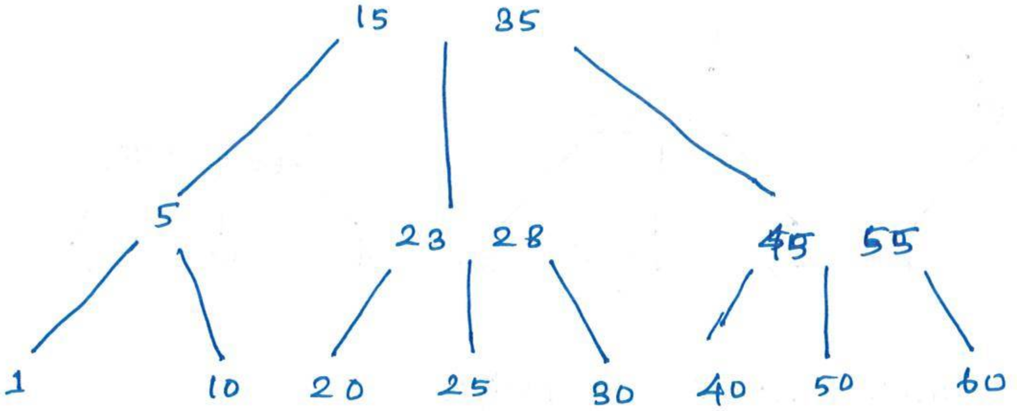
Borrow a key from its immediate neighbouring sibling node in the order of left to right.

<u>Condition:</u>

First check the immediate left sibling. If the sibling node has more than a minimum number of keys, then borrow a key from this node, else check to borrow from the immediate right sibling node.

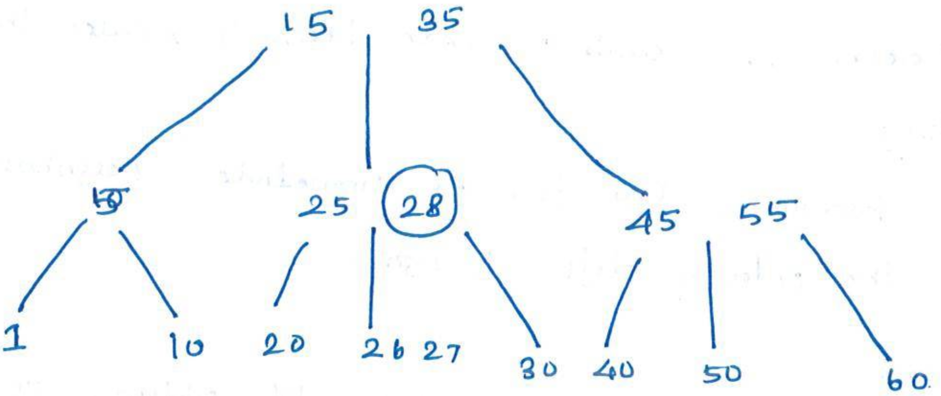


Delete 26



Internal node deletion:

Given data:



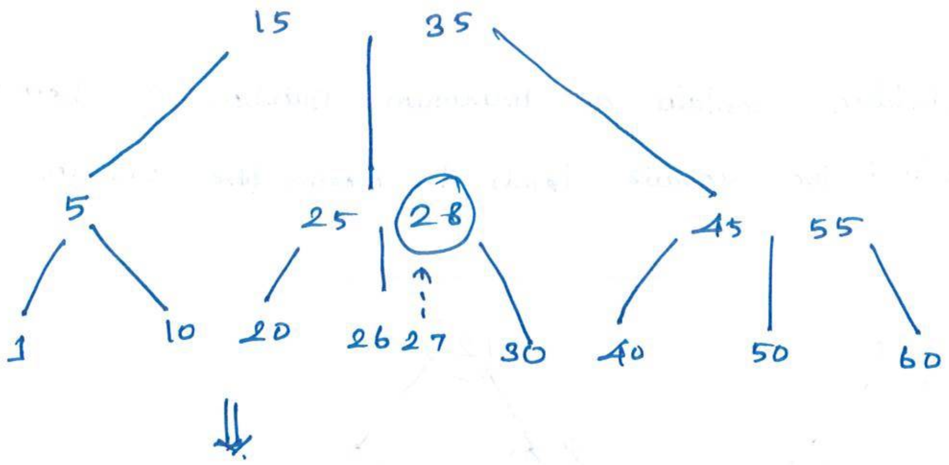
Delete 28: [Internal Node]

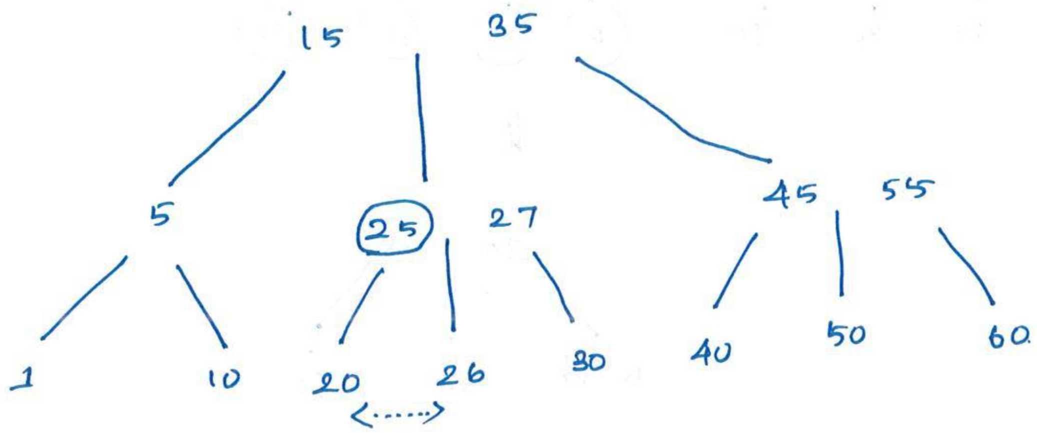
Condition: The internal node, which is delted, is replaced by an inorder predecessor - If the left child has more than minimum number of keys.

(Or)

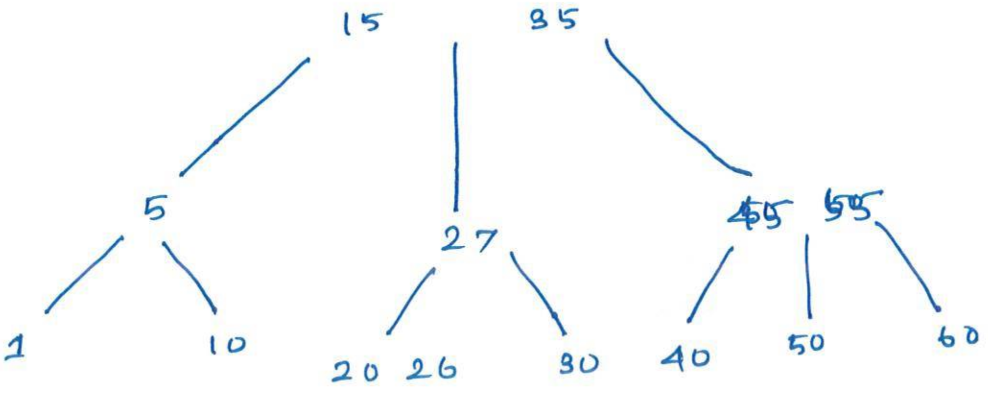
The internal node to be deleted is replaced by an inorder predecessor if the left child has more than maximum number of keys.

Delete 28





Delete 25: The internal node to be deleted is replaced by an inorder successor if the right child has more than maximum number of keys



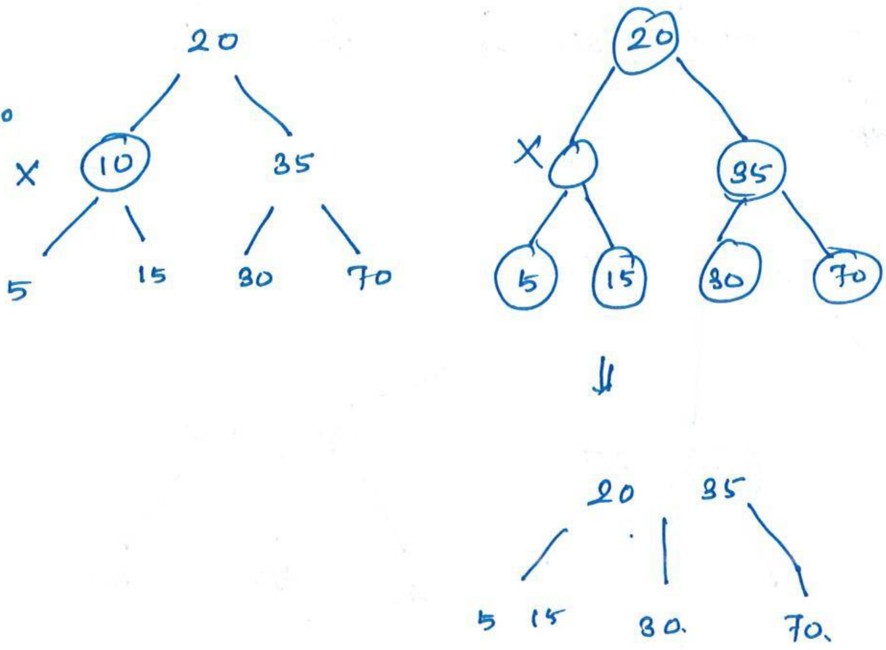
Condition : 03 :

Both children contain a minimum number of keys :-

Sol. : The scenario leads to merging the children.

Example

Delete : 10



major draw backs of B-Tree :-

→ Difficulty of traversing the keys sequentially.

Class work pbms.

Why we need B+ Tree :

→ To store the large amount of data which can't be stored in main memory.

→ limited storage space.

All the internal nodes of B+ Tree are stored in the main memory, where as leaf nodes are stored in the secondary m/y.

Basic difference between B-Tree and B+ Tree :

1. Search key:

B.-Tree : Search key can't be repeatedly stored.

B+ Tree : Redundant search keys can be present

2. Deletion:

B. Tree : Deletion of internal nodes are so complicated and time consuming.

B+ Tree: Deletion will never be a complexed process since the element will always deleted from its Leaf node.

3. Leaf node :-

B. Tree : Leaf node can't be linked together.

B+ Tree : Leaf nodes are linked together to make search options more efficient.