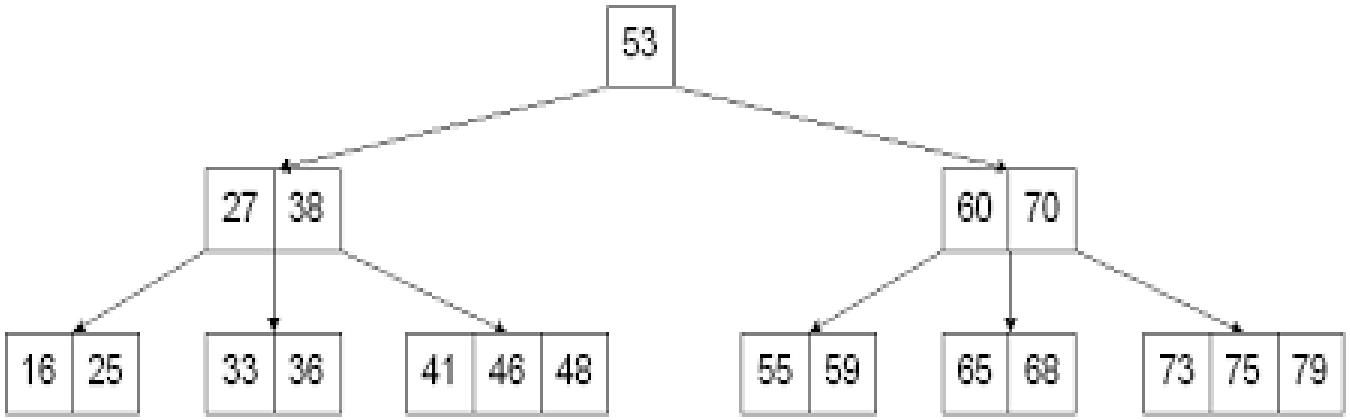
B Tree

- B-Tree is a M-way search tree that can be widely used for disk access.
- One of the main reason of using B tree is its capability to store large number of keys in a single node by keeping the height of the tree relatively small.

Properties of B-Tree:

1. B-Tree of order M will have atmost M children.
2. Every node in a B-tree except root and the leaf nodes has at least ⌈M/2⌉ children.
3. All leaves are at the same level.
4. All nodes may contain maximum M – 1 keys and minimum of ⌈M/2⌉ -1 keys.
5. All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in the range from k1 and k2.
6. The root has at least two children if it is not a leaf node.



B Tree...

Operations :

- **Searching:** Searching in B-tree is similar to BST

- **Insertion:**

  In a B-Tree, a new element must be added only at the leaf node. That means, the new key value is always attached to the leaf node only. The insertion operation is performed as follows...

  Step 1 - Check whether tree is Empty.

  Step 2 - If tree is **Empty**, then create a new node with new key value and insert it into the tree as a root node.

  Step 3 - If tree is **Not Empty**, then find the suitable leaf node to which the new key value is added using Binary Search Tree logic.

  Step 4 - If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.

  Step 5 - If that leaf node is already full, **split** that leaf node by sending middle value to its parent node. Repeat the same until the sending value is fixed into a node.

  Step 6 - If the splitting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

B Tree...

- Insert the following elements into a B-tree of order 3

1,2,3,4,5,6,7,8,9,10

Max no. of keys =M-1= 3-1=2

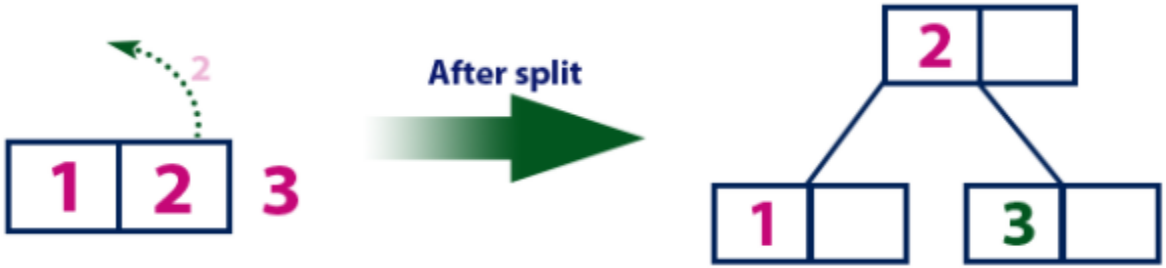
Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

Min no. of children = ⌈M/2⌉ =3/2=2

Insert 1

Insert 2

Insert 3



B Tree...

- Insert the following elements into a B-tree of order 3

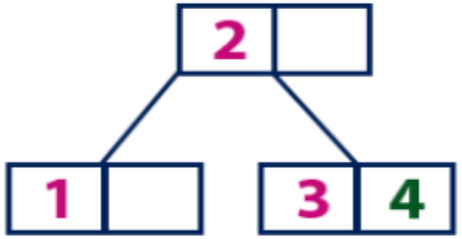
  1,2,3,4,5,6,7,8,9,10

Max no. of keys =M-1= 3-1=2

Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

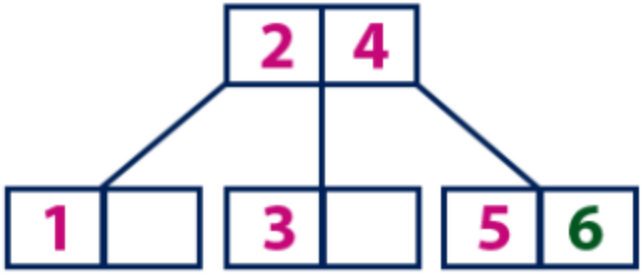
Min no. of children = ⌈M/2⌉ =3/2=2

Insert 4



Insert 5

Insert 6



B- Tree...

- Insert the following elements into a B-tree of order 3

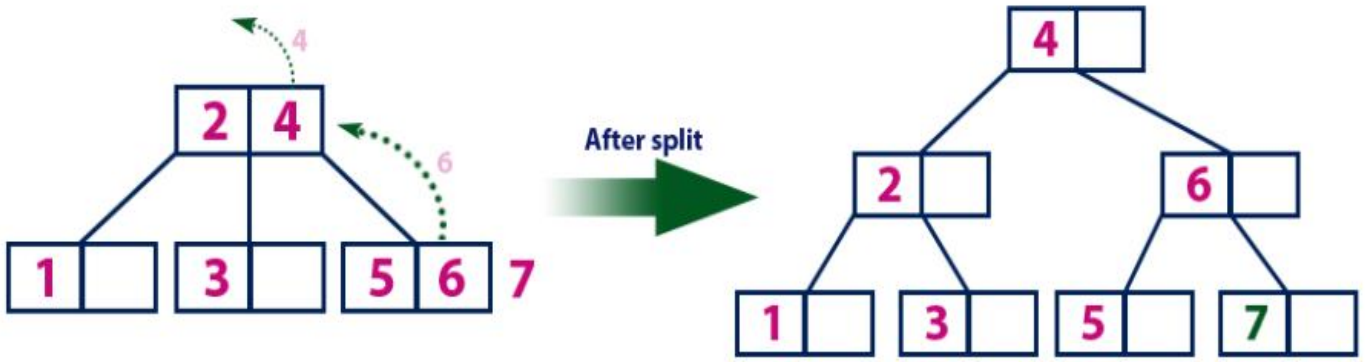
  1,2,3,4,5,6,7,8,9,10

Max no. of keys =M-1= 3-1=2

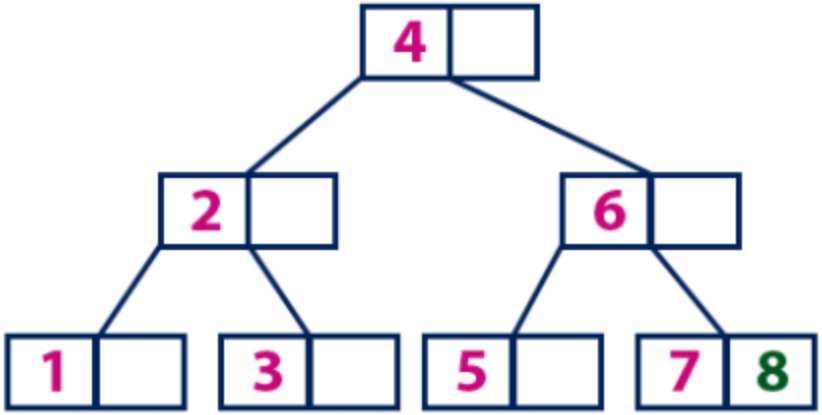
Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

Min no. of children = ⌈M/2⌉ =3/2=2

Insert 7



Insert 8



B Tree...

- Insert the following elements into a B-tree of order 3

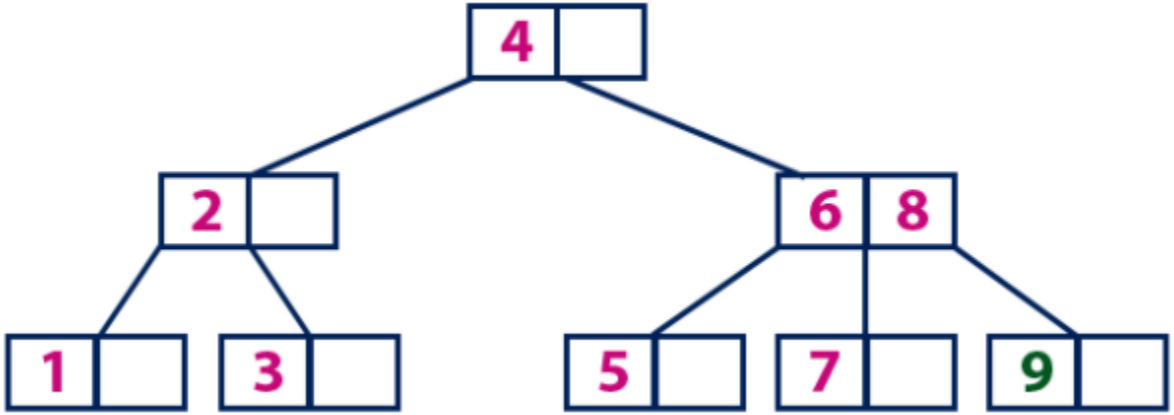
  1,2,3,4,5,6,7,8,9,10

Max no. of keys =M-1= 3-1=2

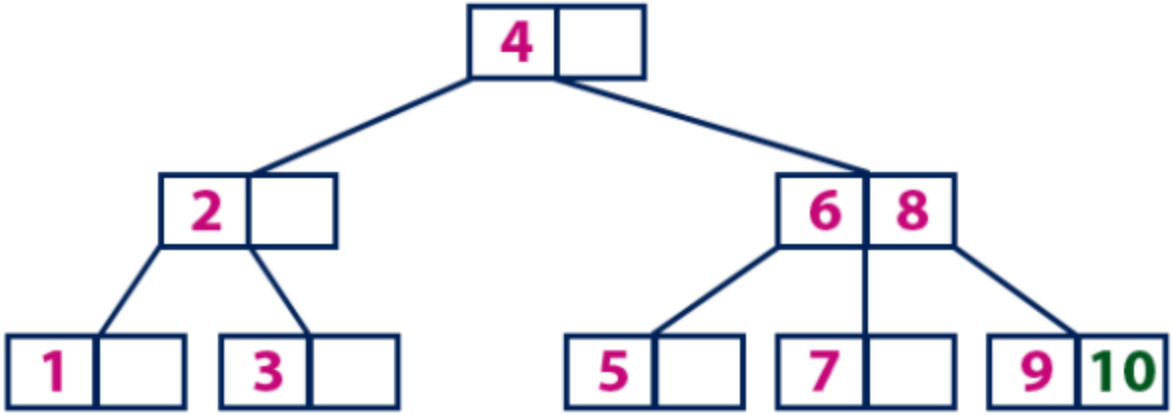
Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

Min no. of children = ⌈M/2⌉ =3/2=2

Insert 9



Insert 10



B Tree...

- Insert the following elements into a B-tree of order 3

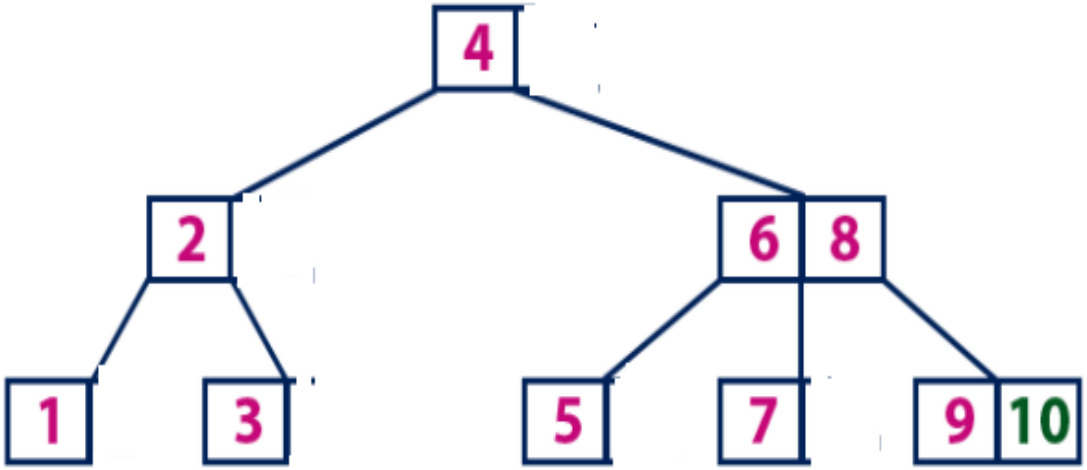
  1,2,3,4,5,6,7,8,9,10

Max no. of keys =M-1= 3-1=2

Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

Min no. of children = ⌈M/2⌉ =3/2=2

Finally, B-tree is

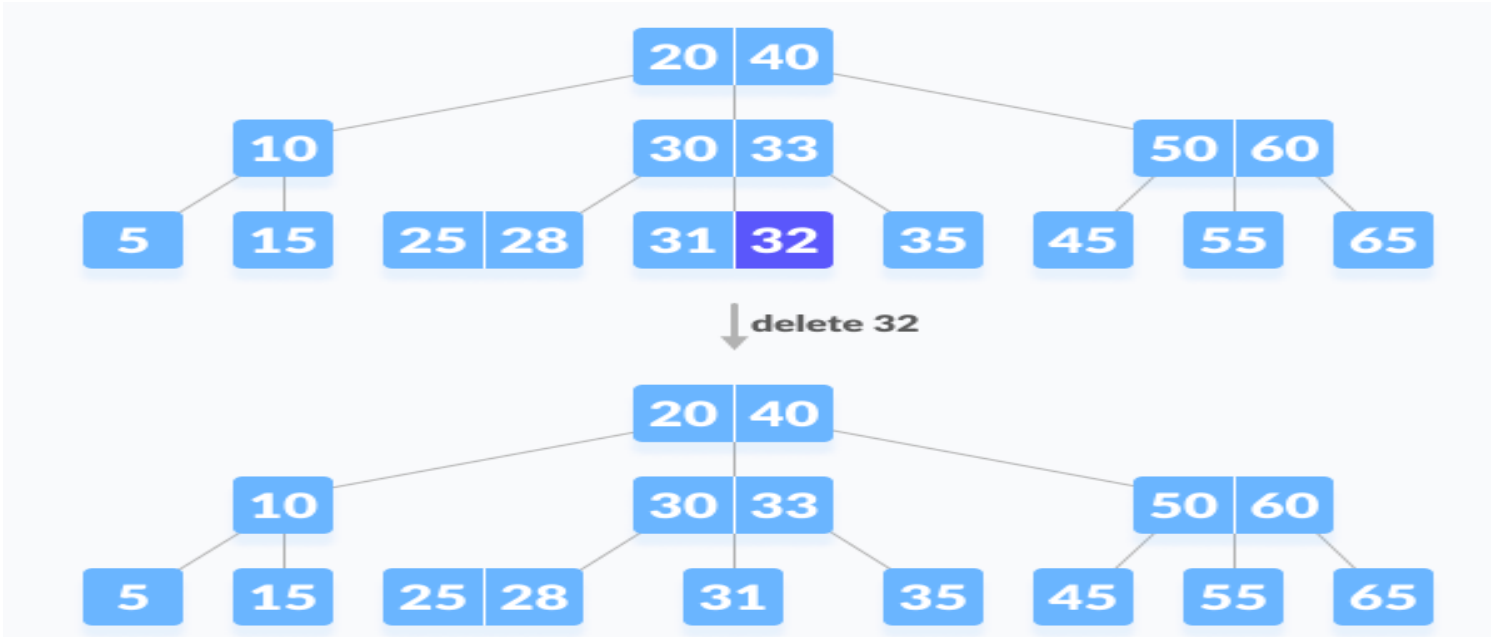


B Tree...

- **Deletion:**
  - Deleting an element on a B-tree consists of three main events: **searching** the node where the key to be deleted exists, **deleting** the key and **balancing** the tree if required.
  - While deleting a tree, a condition called **underflow** may occur. Underflow occurs when a node contains less than the minimum number of keys it should hold.
  - There are three main cases for deletion operation in a B tree.

Case I:The key to be deleted lies in the leaf. There are two cases for it.

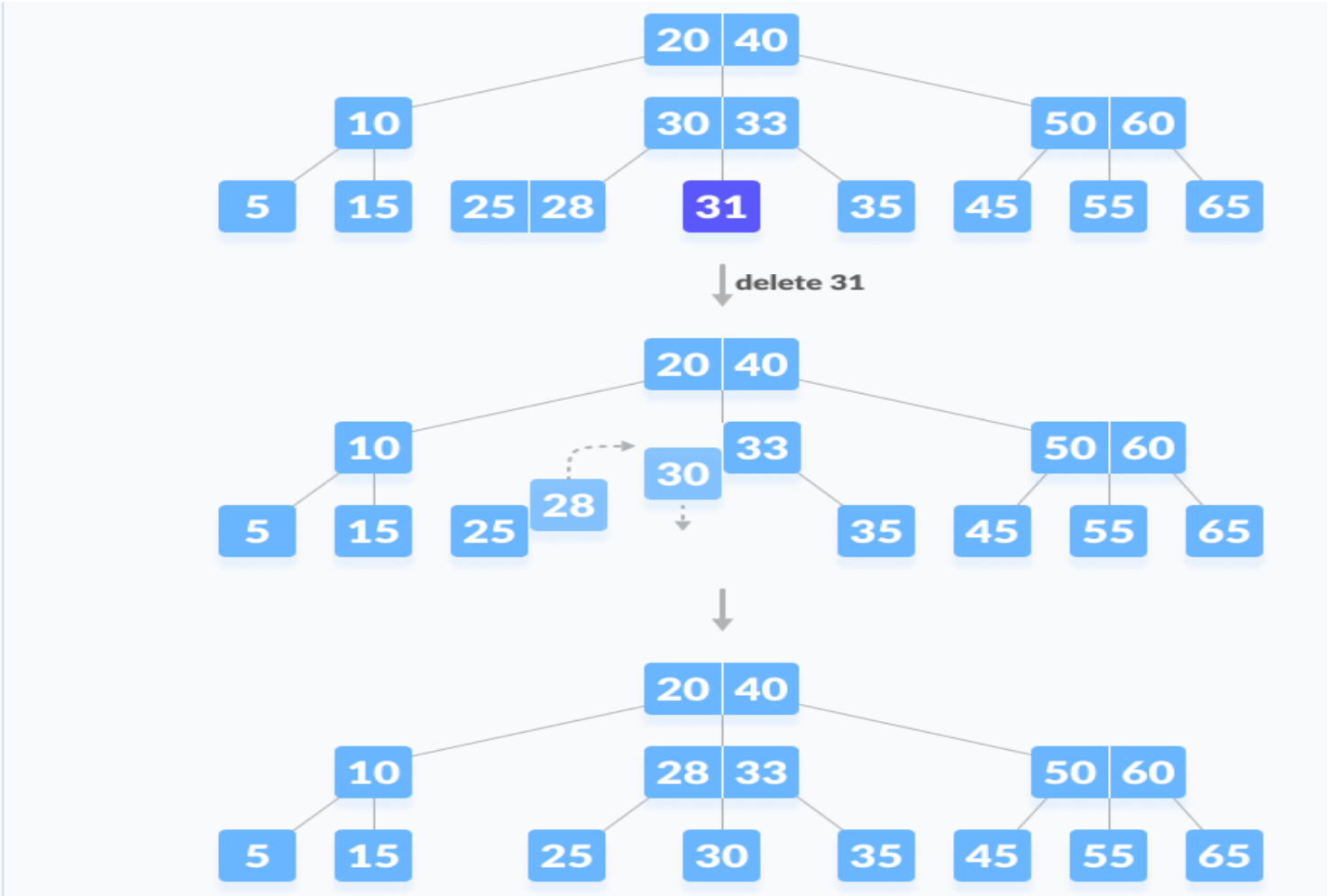
i) The deletion of the key does not violate the property of the minimum number of keys.



B Tree...

ii) The deletion of the key violates the property of the minimum number of keys.

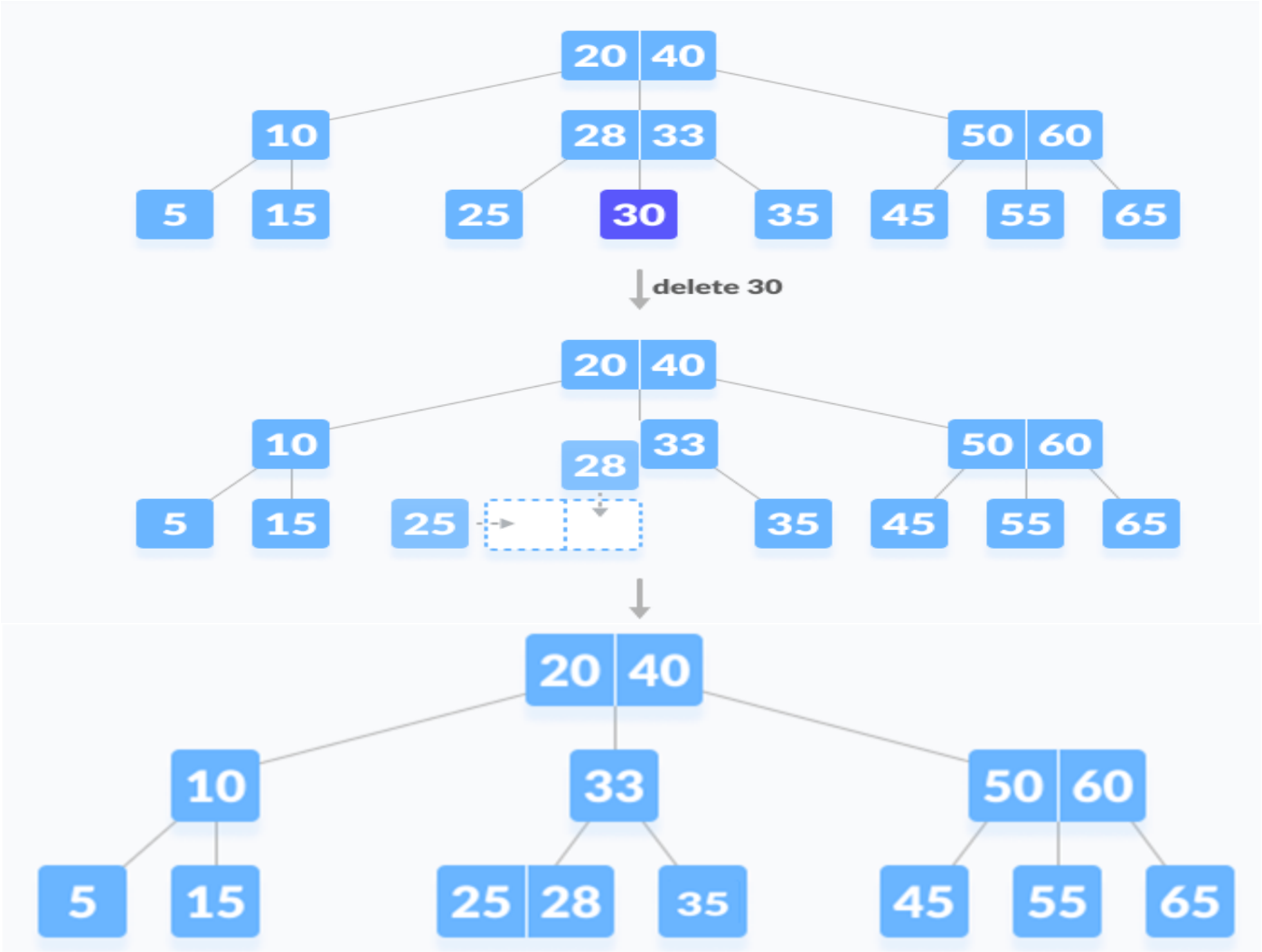
In this case, we borrow a key from its immediate neighbouring sibling node in the order of left to right. First, visit the immediate left sibling. If the left sibling node has more than a minimum number of keys, then borrow a key from this node. Else, check to borrow from the immediate right sibling node



B Tree...

iii) If both the immediate sibling nodes already have a minimum number of keys:

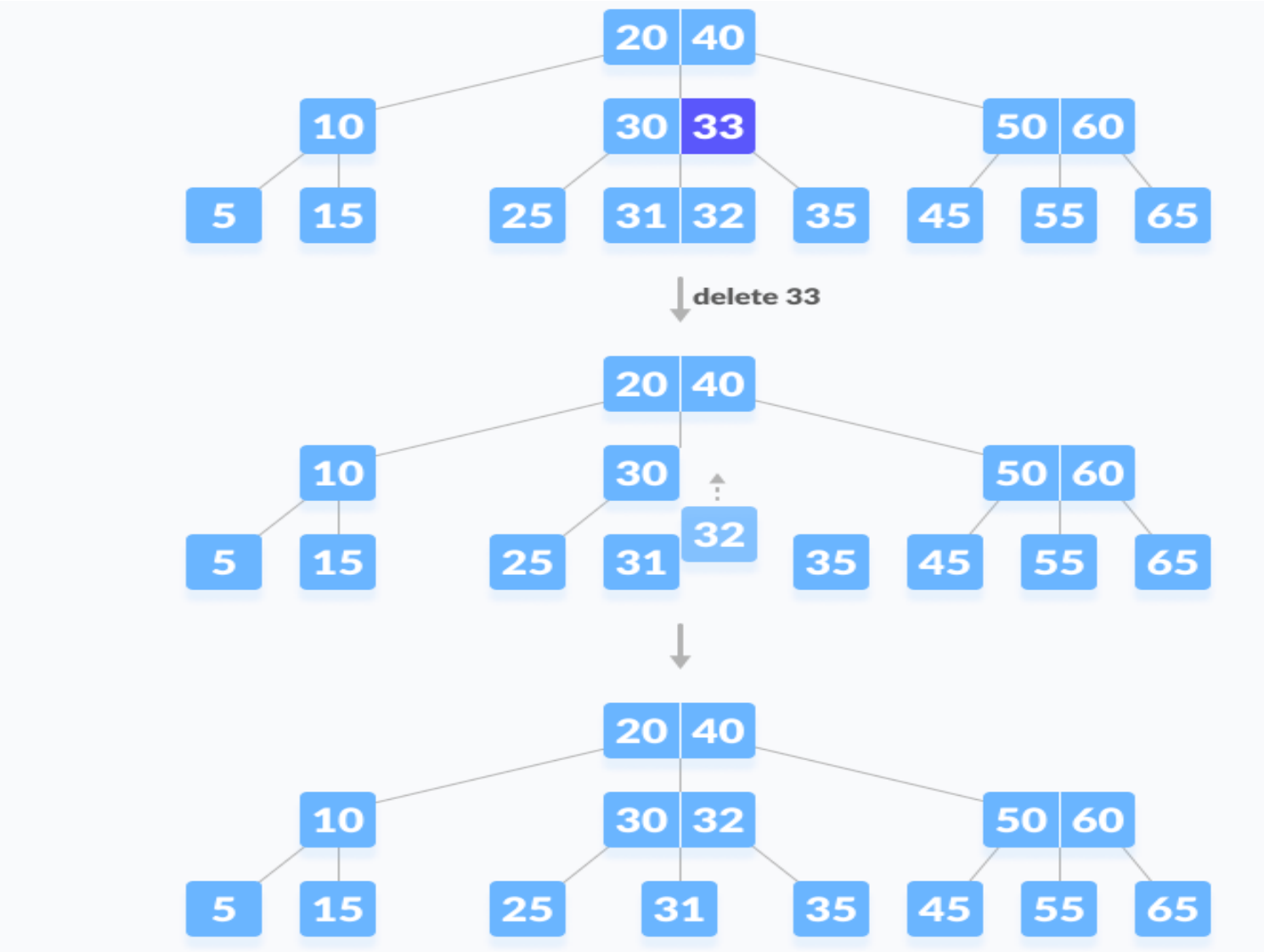
then merge the node with either the left sibling node or the right sibling node. This merging is done through the parent node.



B Tree...

Case II :If the key to be deleted lies in the internal node, the following cases occur.

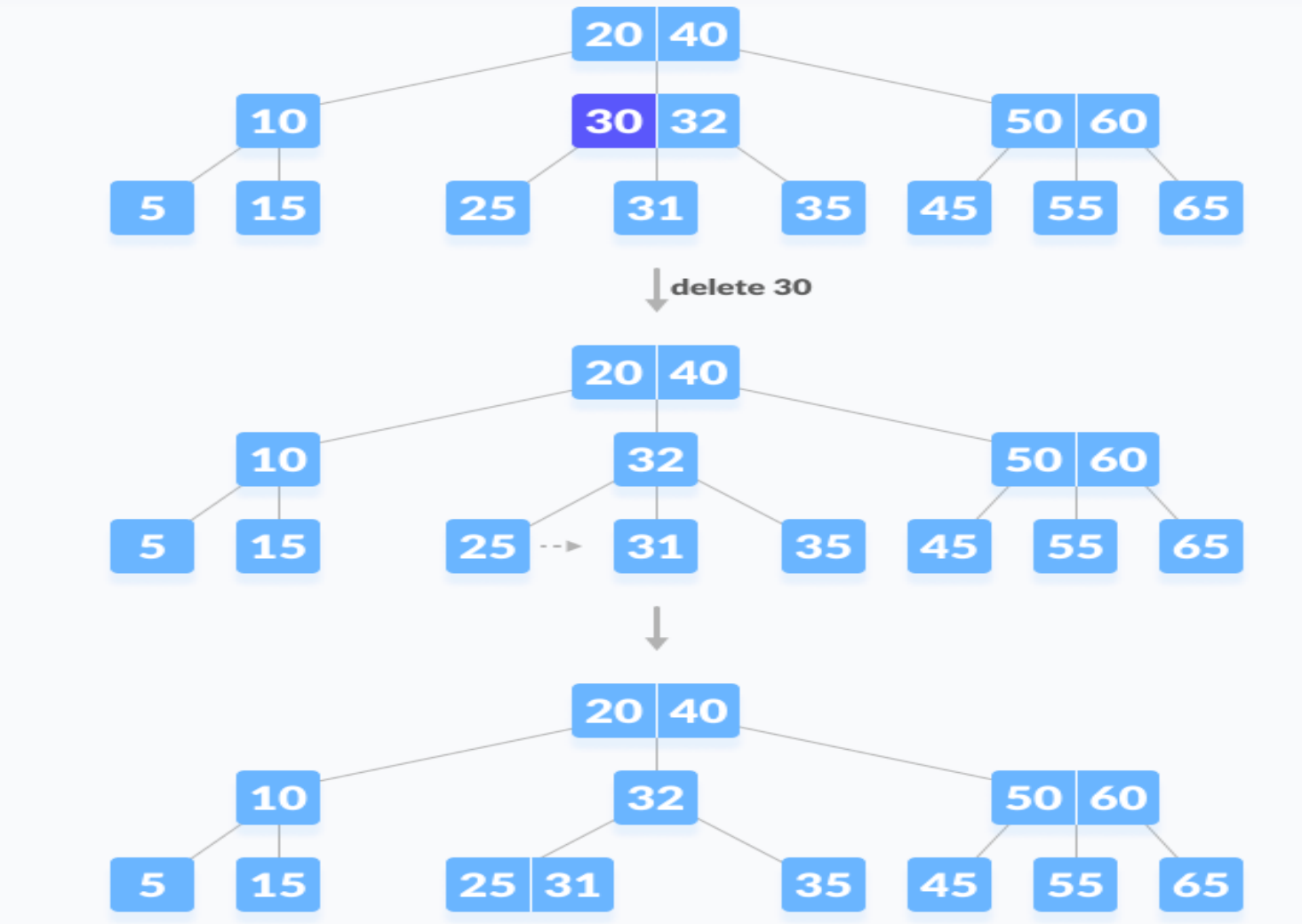
i) The internal node, which is deleted, is replaced by an Inorder predecessor if the left child has more than the minimum number of keys.



B Tree...

Case II :If the key to be deleted lies in the internal node, the following cases occur.

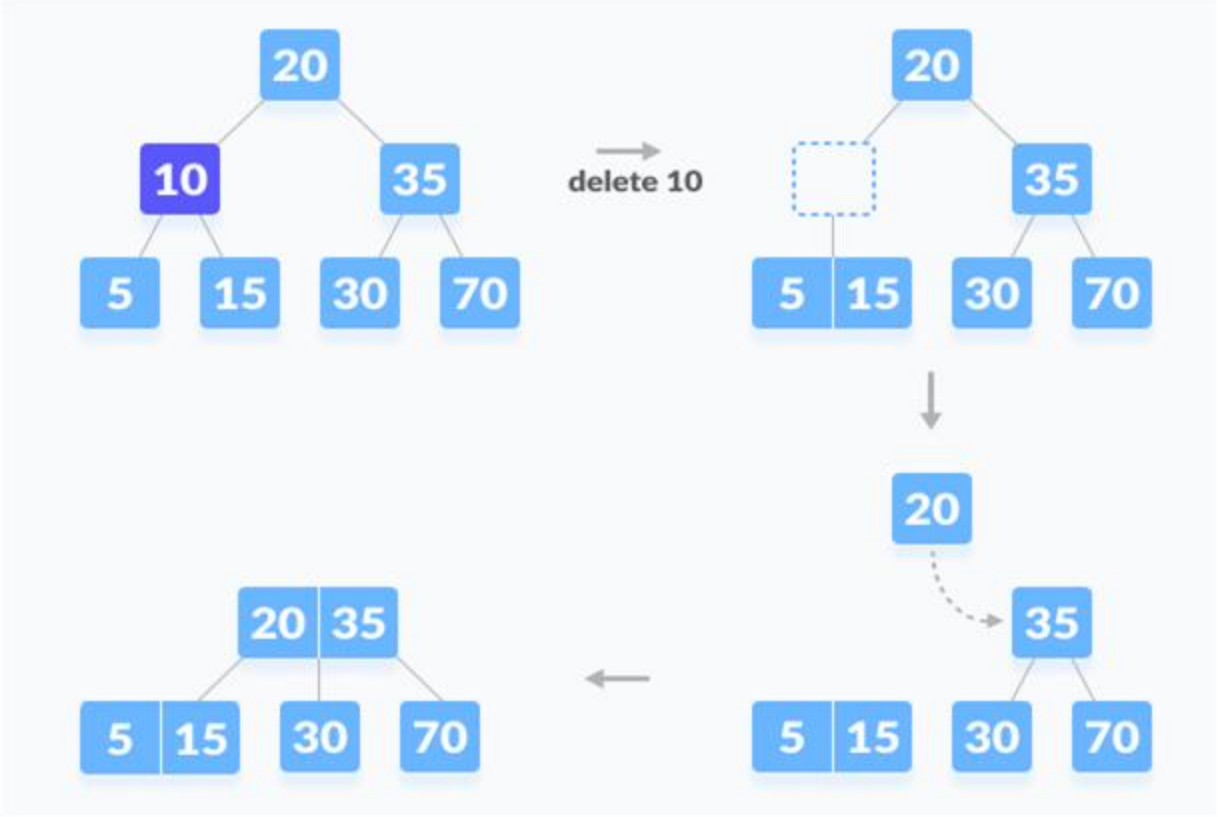
ii) If either child has exactly a minimum number of keys then, merge the left and the right children. After merging if the parent node has less than the minimum number of keys then, look for the siblings as in Case I.



B Tree...

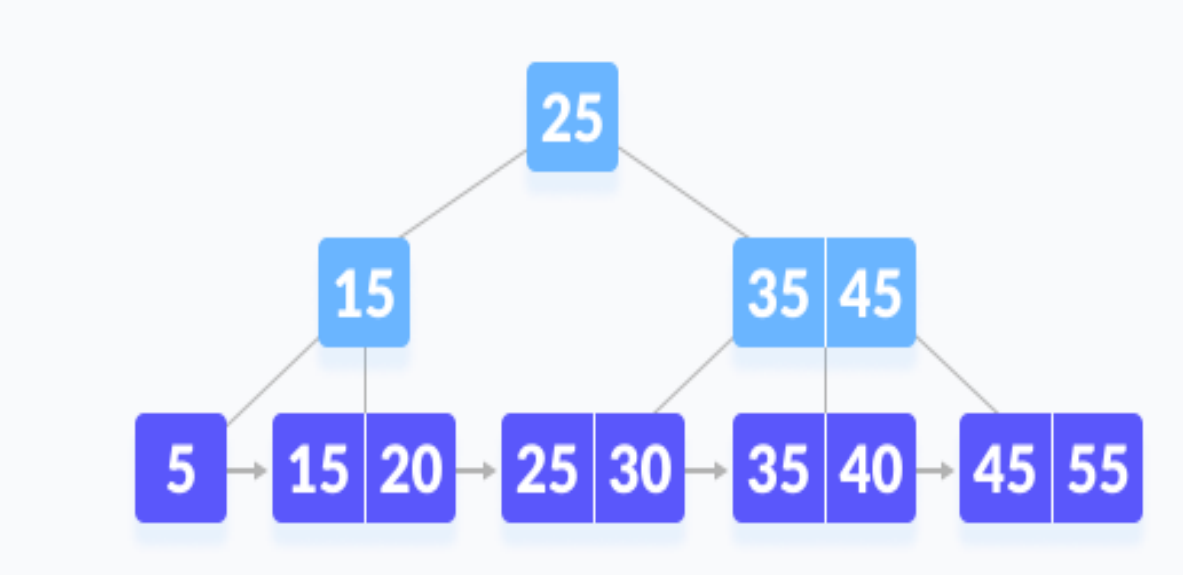
Case III : In this case, the height of the tree shrinks. If the target key lies in an internal node, and the deletion of the key leads to a fewer number of keys in the node (i.e. less than the minimum required), then look for the inorder predecessor and the inorder successor. If both the children contain a minimum number of keys then, borrowing cannot take place. This leads to Case II(ii) i.e. merging the children.

Again, look for the sibling to borrow a key. But, if the sibling also has only a minimum number of keys then, merge the node with the sibling along with the parent. Arrange the children accordingly (increasing order).



B+ Tree...

- B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
- In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values. The internal nodes of B+ tree are often called index nodes.
- The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.



B+ Tree...

Operations:

- **Insertion:**

  Case I

  If the leaf is not full, insert the key into the leaf node in increasing order.

  Case II

  1. If the leaf is full, insert the key into the leaf node in increasing order and balance the tree in the following way.
  2. Break the node at m/2th position.
  3. Add m/2th key to the parent node as well.
  4. If the parent node is already full, follow steps 2 to 3.

Example:

Construct B+ Tree of order 3 for the elements 5,15, 25, 35, 45.

Max no. of keys =M-1= 3-1=2

Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

Min no. of children = ⌈M/2⌉ =3/2=2

B+ Tree...

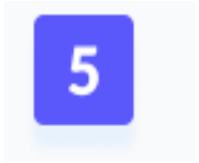
Construct B+ Tree of order 3 for the elements 5,15, 25, 35, 45.

Max no. of keys =M-1= 3-1=2

Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

Min no. of children = ⌈M/2⌉ =3/2=2

Insert 5



Insert 15

Insert 25

B+ Tree...

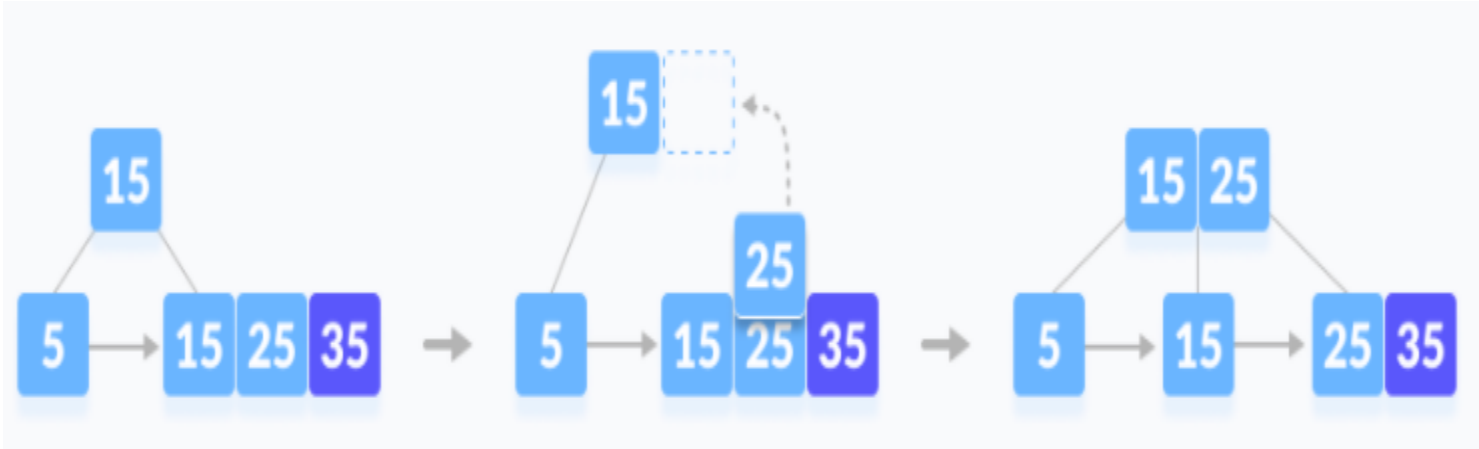
Construct B+ Tree of order 3 for the elements 5,15, 25, 35, 45.

Max no. of keys =M-1= 3-1=2

Min no. of keys = ⌈M/2⌉ -1 =3/2-1=2-1=1

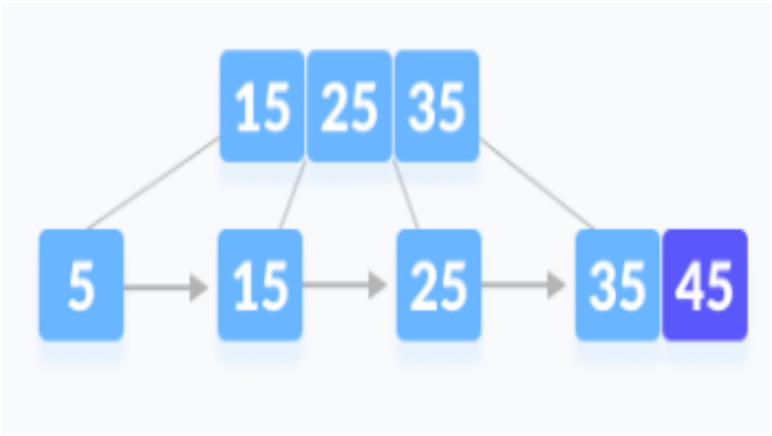
Min no. of children = ⌈M/2⌉ =3/2=2

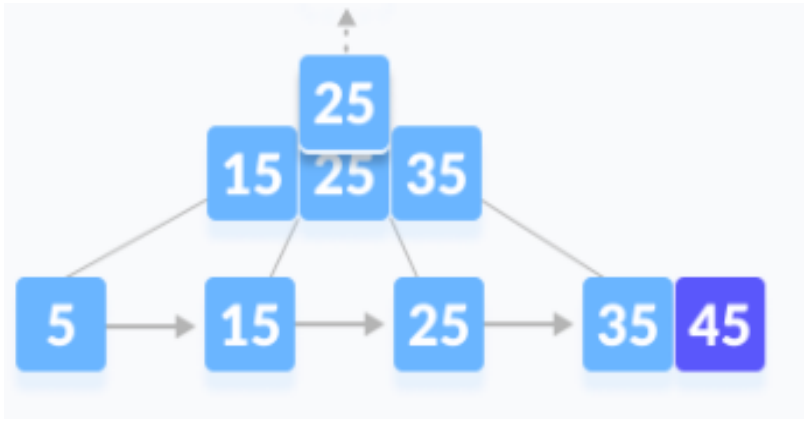
Insert 35

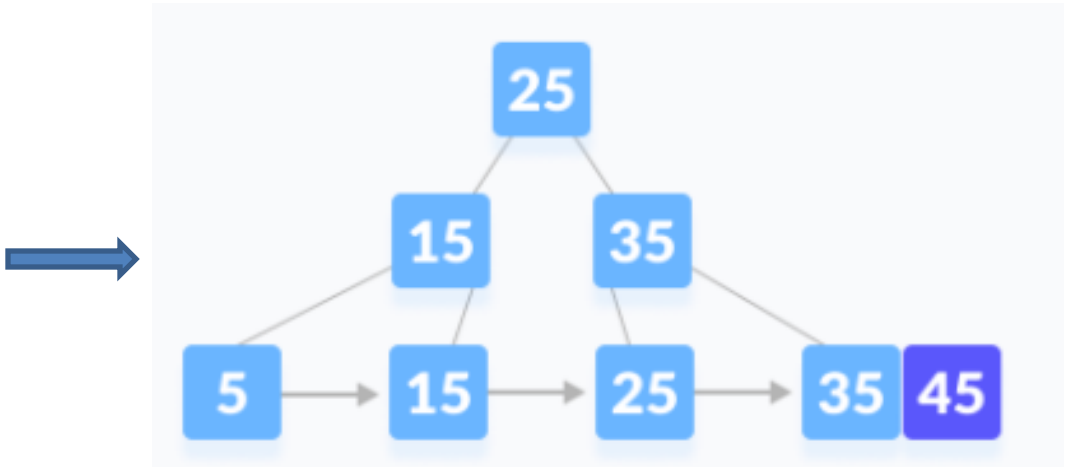


Insert 45

B+ Tree...







B+ Tree...

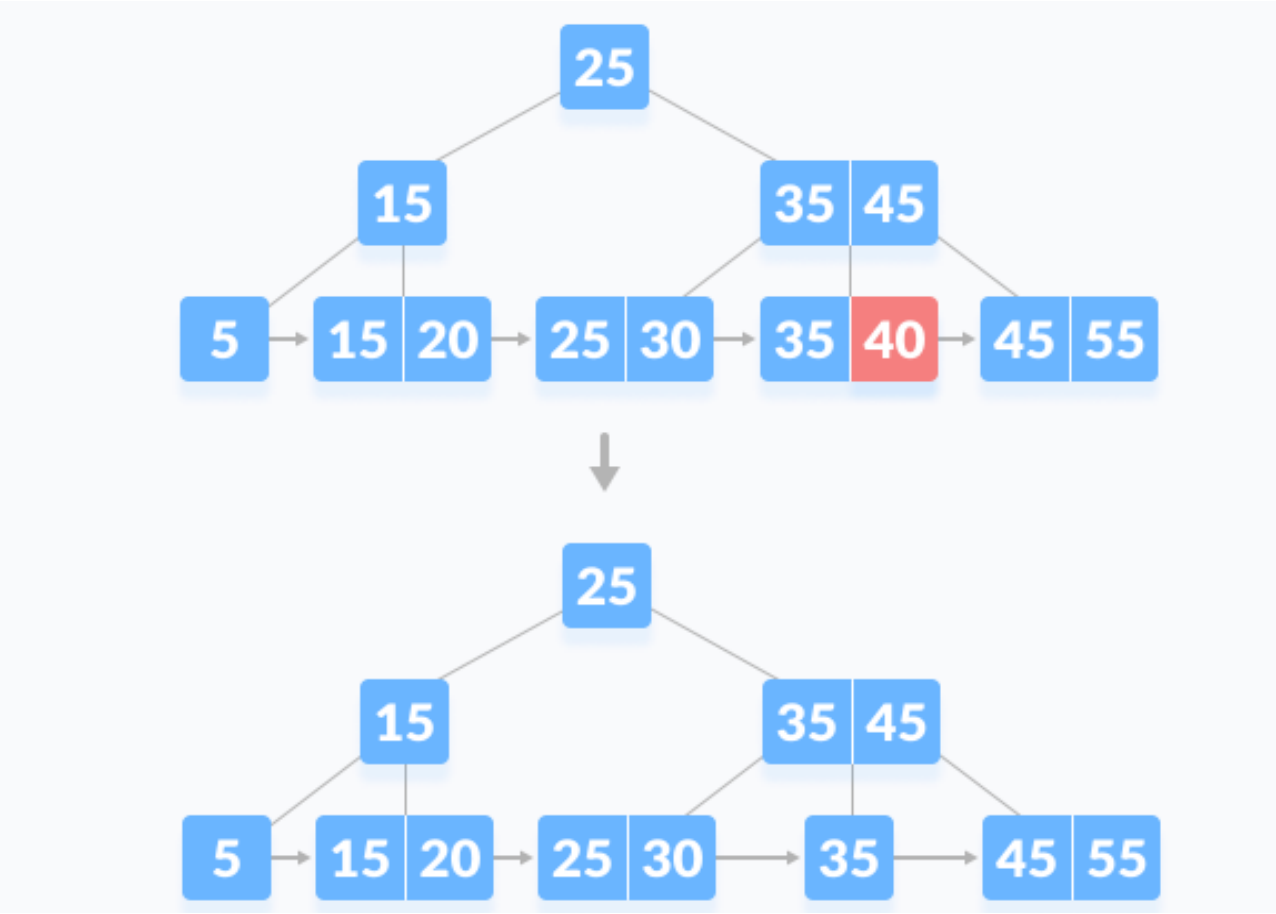
➢**Deletion: (B+ Tree of order 3)**

Case I:The key to be deleted is present only at the leaf node not in the indexes (or internal nodes).

There are two cases for it:

i) There is more than the minimum number of keys in the node. Simply delete the key.

Delete 40:



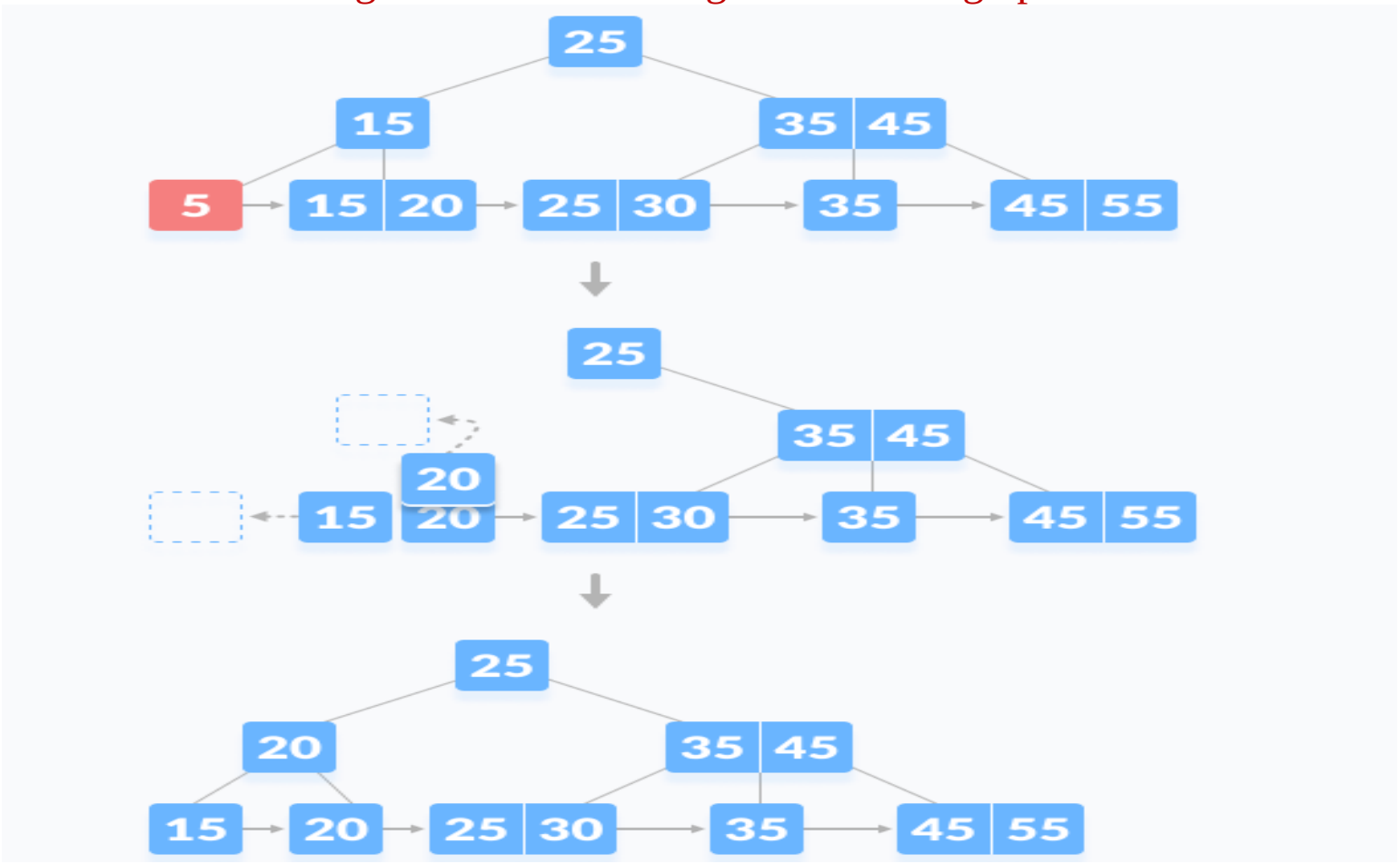
B+ Tree...

➢**Deletion: (B+ Tree of order 3)**

Case I:The key to be deleted is present only at the leaf node not in the indexes (or internal nodes).

ii) There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Minimum from right child through parent.

Delete 5:



B+ Tree...

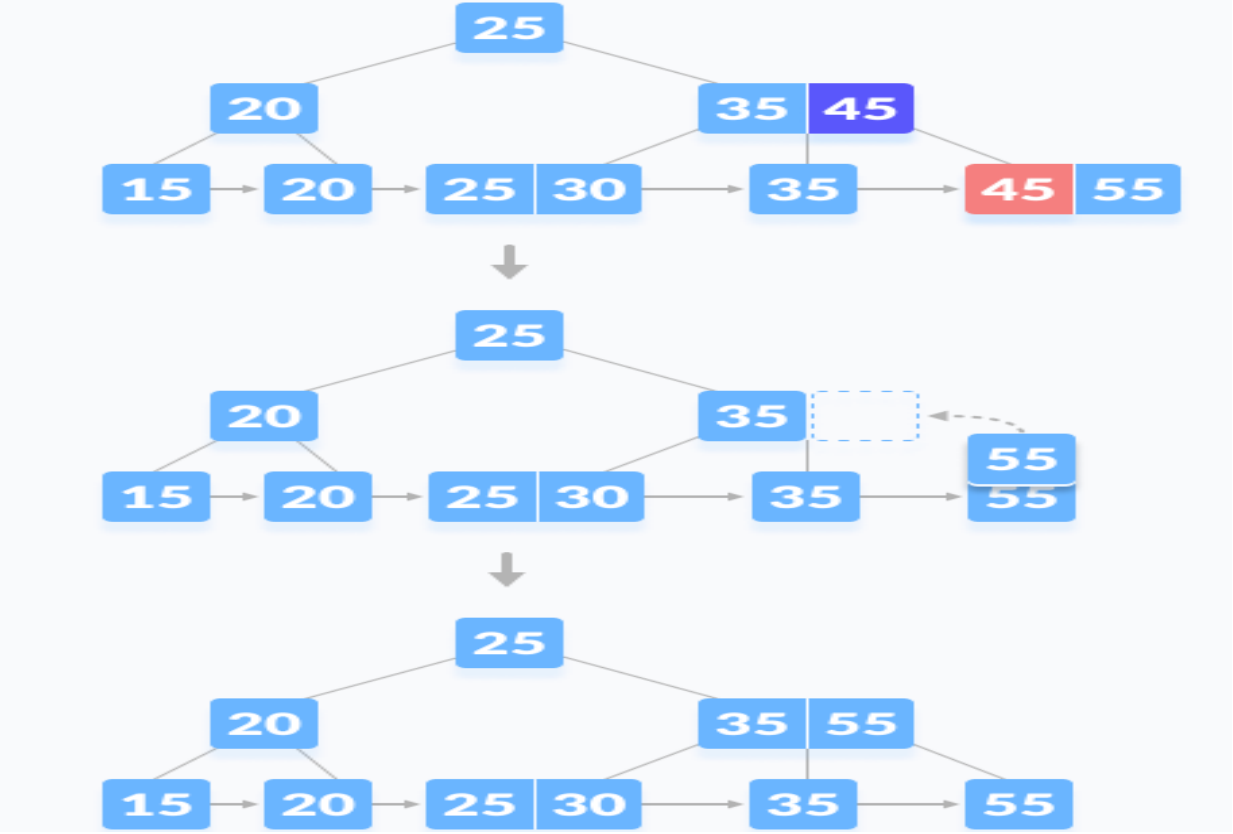
➢**Deletion: (B+ Tree of order 3)**

Case 2:The key to be deleted is present in the internal nodes and leaf node. Then we have to remove them from the internal nodes and leaf node.

There are the following cases for this situation:

i) If there is more than the minimum number of keys in the node, simply delete the key from the leaf node and delete the key from the internal node as well.Fill the empty space in the internal node with the Inorder successor.

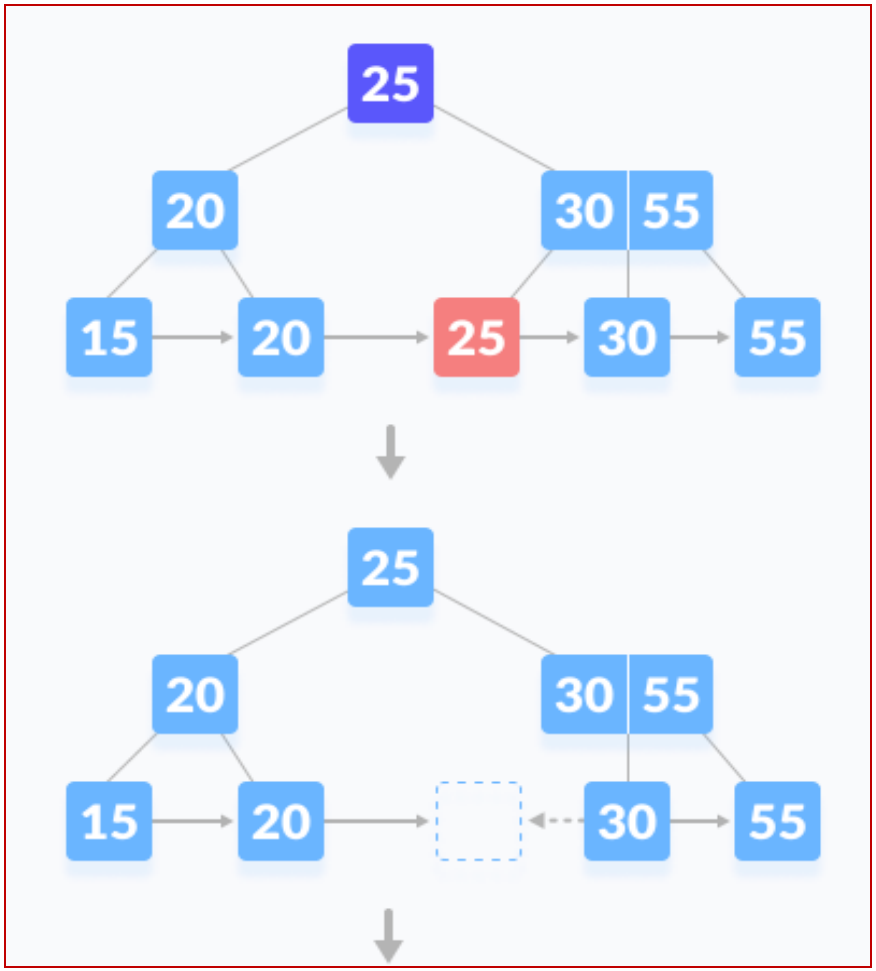
Delete 45:

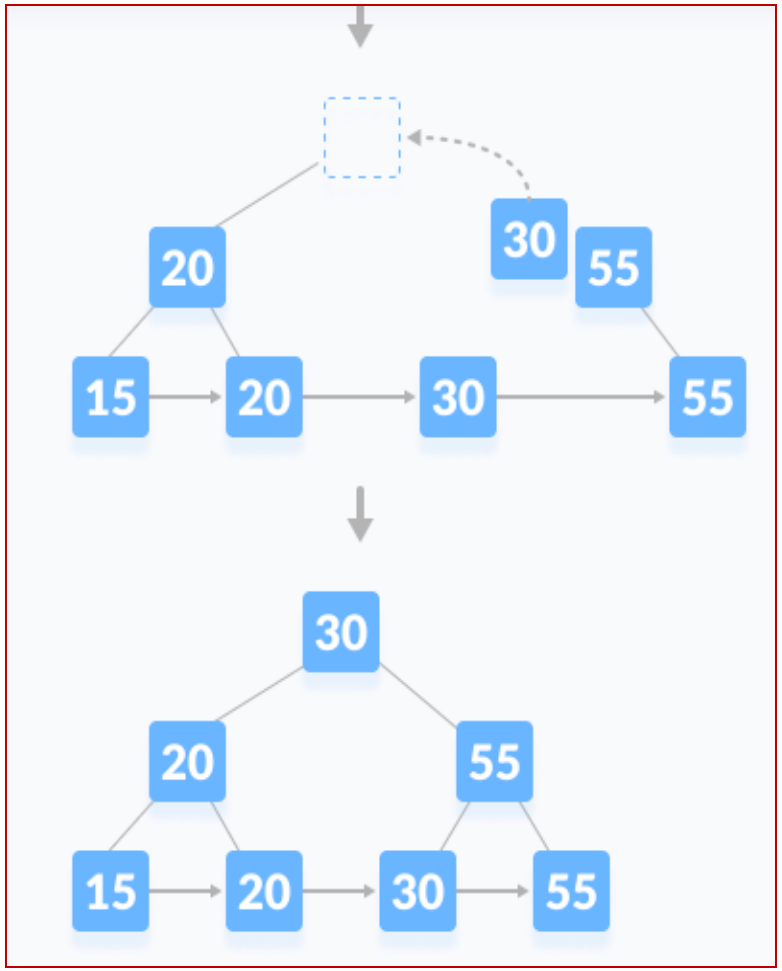


B+ Tree...

iii) This case is similar to Case 2(i) but here, empty space is generated above the immediate parent node.After deleting the key, merge the empty space with its sibling.
Fill the empty space in the grandparent node with the inorder successor.

Delete 25:

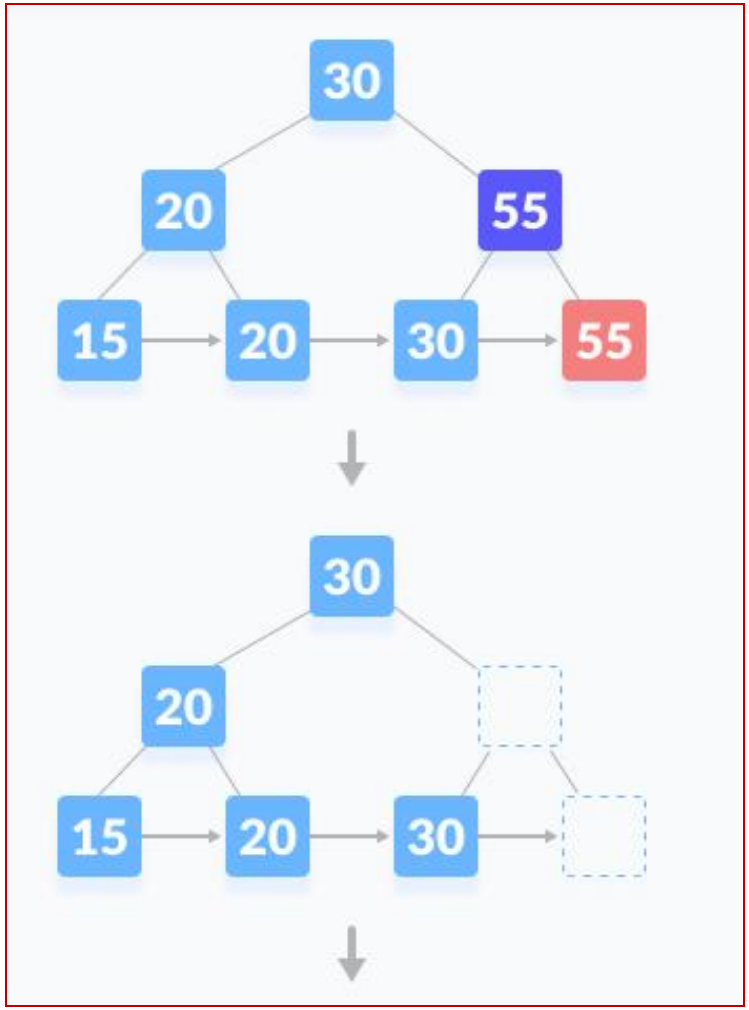


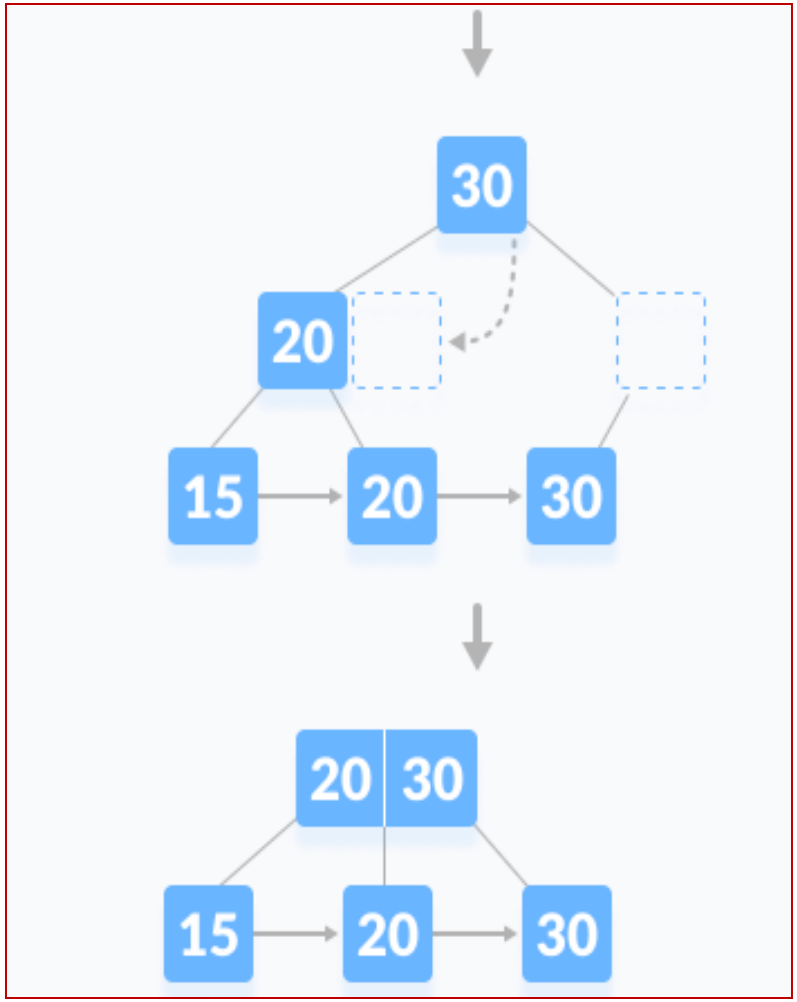


B+ Tree...

Case 3: In this case, the height of the tree gets shrinked. It is a little complicated.Deleting 55 from the tree below leads to this condition. It can be understood in the illustrations below.

Delete 55:





Comparison between B-Tree and B+ Tree

S.NO	B tree	B+ tree
1.	All internal and leaf nodes have data pointers	Only leaf nodes have data pointers
2.	Since all keys are not available at leaf, search often takes more time.	All keys are at leaf nodes, hence search is faster and accurate..
3.	No duplicate of keys is maintained in the tree.	Duplicate of keys are maintained and all nodes are present at leaf.
4.	Insertion takes more time and it is not predictable sometimes.	Insertion is easier and the results are always the same.
5.	Deletion of internal node is very complex and tree has to undergo lot of transformations.	Deletion of any node is easy because all node are found at leaf.
6.	Leaf nodes are not stored as structural linked list.	Leaf nodes are stored as structural linked list.
7.	No redundant search keys are present..	Redundant search keys may be present..