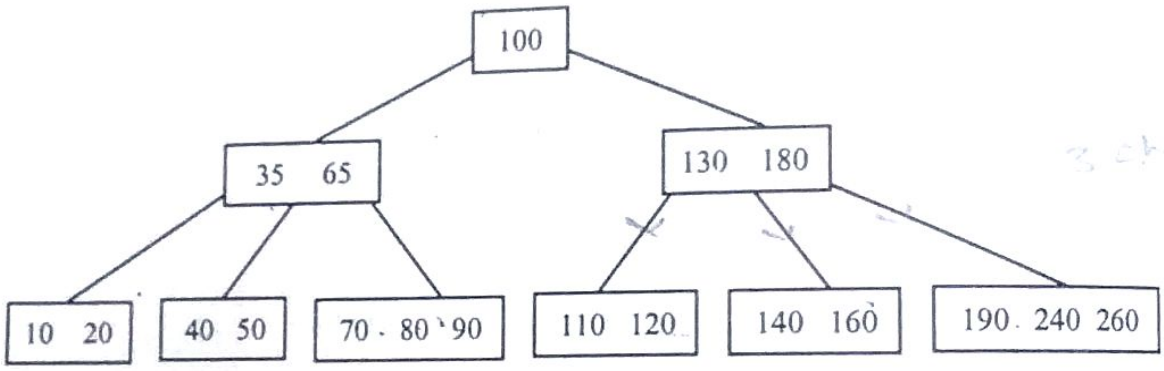
B Tree –

In multiway search tree, so many nodes have left subtree but no right subtree. Similarly they have right subtree but no left subtree. Insertion of key also increases the height of tree. As we know access time in tree is totally dependent on level of tree. So our aim is to minimize the access time which can be through balanced tree only. So we have a need to take all the leaf nodes at same level and non leaf nodes should not contain the empty

subtree. So for a order n tree, maximum number of children should be n and each node can contain k keys where k <= n-1. For balancing the tree each node should contain n/2 keys(except root). So the B-tree of order n can be defined as-

1. Each node has at least n/2 and maximum n non empty children.
2. All leaf nodes will be at same level.
3. All leaf nodes can contain maximum n-1 keys.
4. All non leaf nodes can contain m-1 keys where m is the number of children for that node.
5. Keys in non leaf node will divide the left and right subtree where value of left subtree keys will be less and value of right subtree keys will be more than that particular key.

Let us take a B-tree of order 5-



Here we can see all leaf nodes are at same level. All non leaf nodes have no empty subtree and they have keys 1 less than number of their children.

Insertion in B tree –

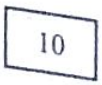
Insertion of key requires first traversal in B-tree. Through traversal it will find that key to be inserted is already existing or not. Suppose key does not exist in tree then through traversal it will reach leaf node. Now we have two cases for inserting the key-

1. Node is not full
2. Node is already full

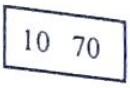
In the first case we can simply add the key in node. But in second case we will need to split the node into two nodes and median key will go to the parent of that node. If parent is also full then same thing will be repeated until it will get non full parent node. Suppose root is full then it will split into two nodes and median key will be root.
Let us take a list of keys and create a B-tree of order 5.

10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 190, 90, 180, 240, 30, 120, 140, 160

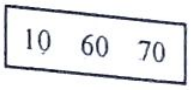
Step 1- Insert 10



Step 2- Insert 70

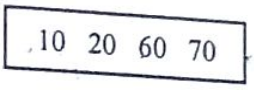


Step 3- Insert 60



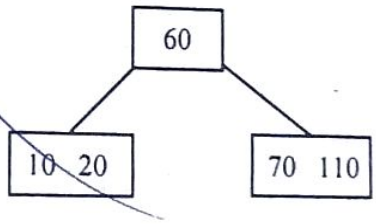
Here after insertion of 60, the keys in node will be sorted.

Step 4- Insert 20



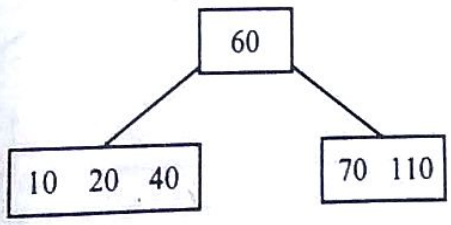
Here after insertion of 20, the keys in node will be sorted.

Step 5- Insert 110

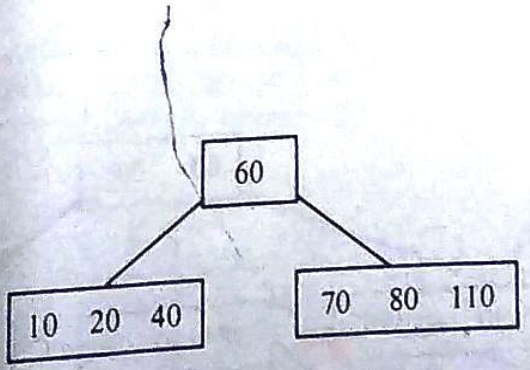


Here node was already full, so after insertion of 110, it splitted into two nodes, 60 is the median key so it will go in parent node but root node is splitted so it will become root.

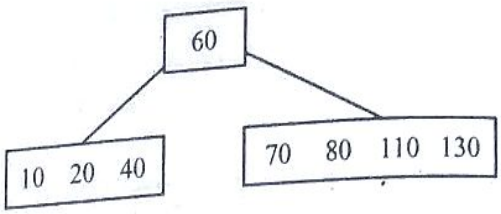
Step 6- Insert 40



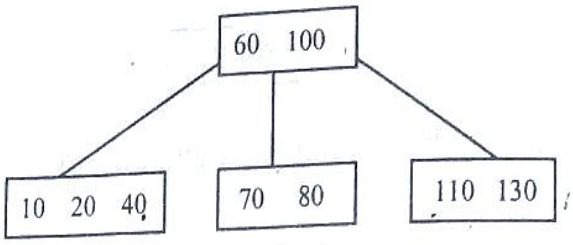
Step 7- Insert 80



Step 8- Insert 130

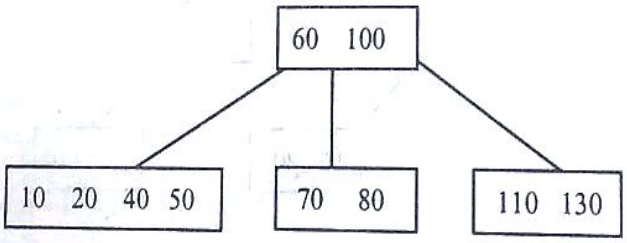


Step 9- Insert 100

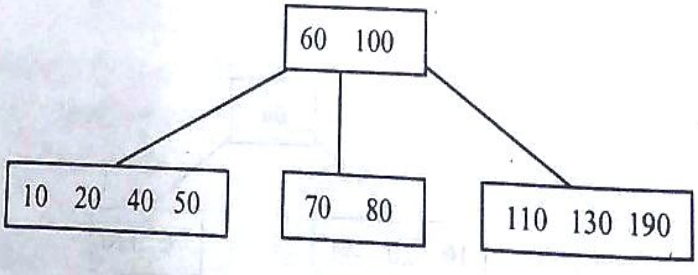


Here node was already full, so after insertion of 100 it splitted into two nodes, 100 is the median key, so it will go to the parent node.

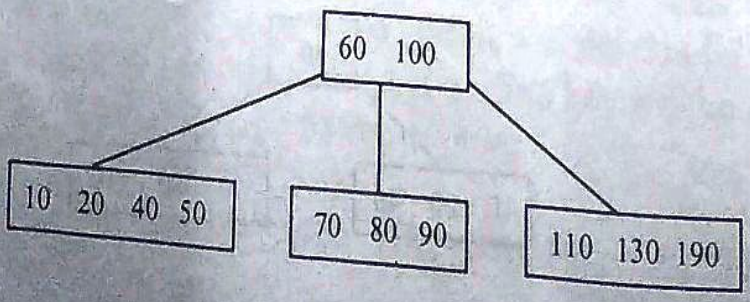
Step 10- Insert 50



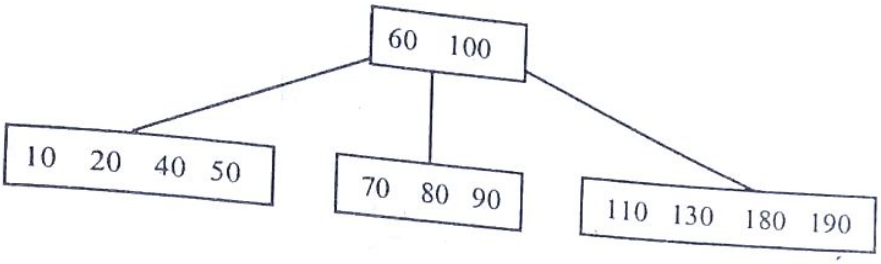
Step 11- Insert 190



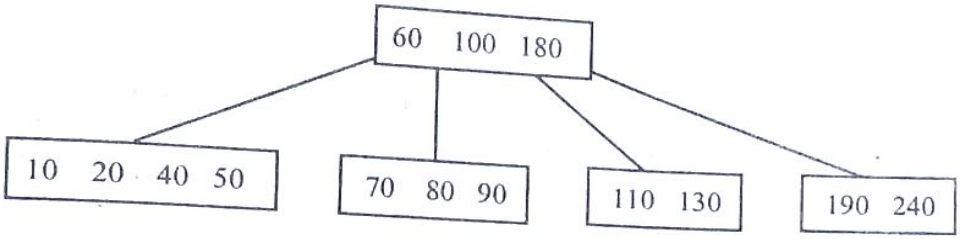
Step 12- Insert 90



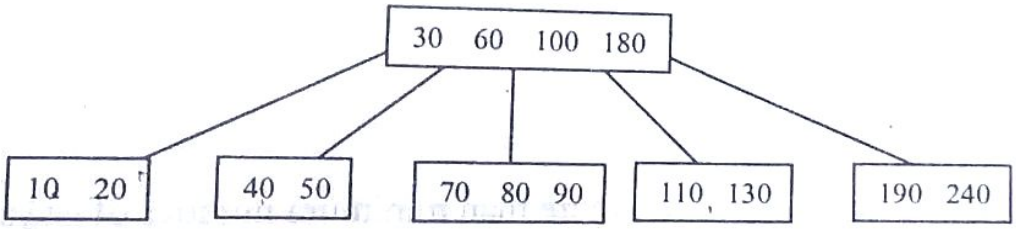
Step 13- Insert 180



Step 14- Insert 240

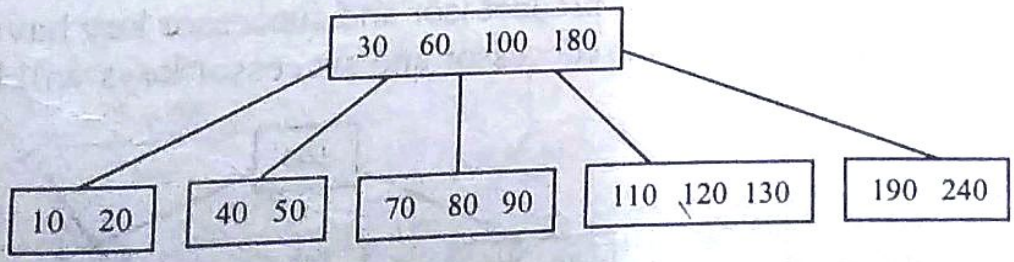


Step 15- Insert 30

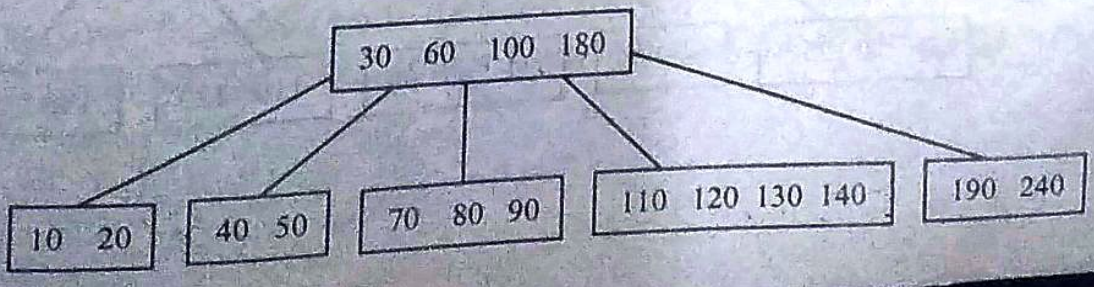


Here node was already full, so after insertion of 30 it splitted in two nodes, 30 is the median key so it will go into parent.

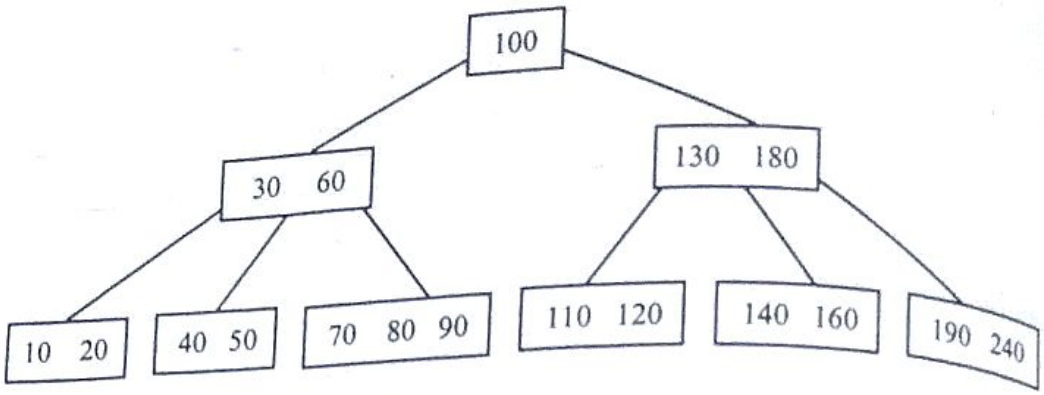
Step 16- Insert 120



Step 17- Insert 140



Step 18- Insert 160



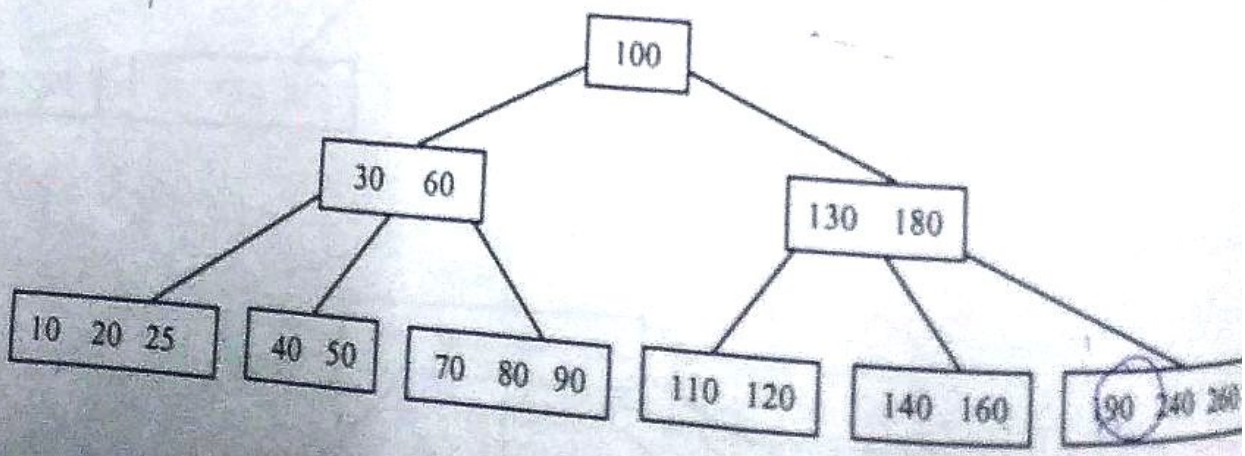
Here node was already full, so after insertion 160 it splitted in two nodes, 130 is the median key so it will go into parent. Here root is already full, so it splitted in two nodes, 100 is the median key so it will become the new root.

Deletion in B-tree –

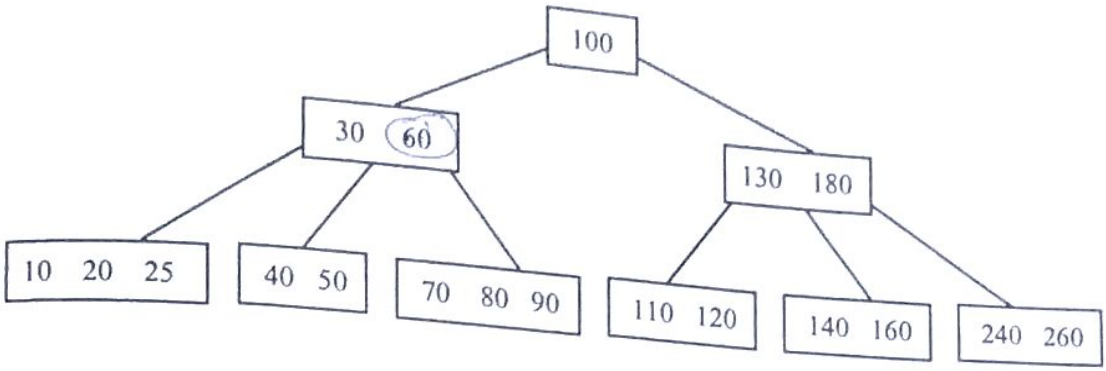
Deletion of key also requires first traversal in B-tree. After reaching on particular node, two cases may occur-

1. Node is leaf node
2. Node is non leaf node

For the first case suppose node has more than minimum number of keys then it can be easily deleted. But suppose it has only minimum number of keys then first we will see the number of keys in adjacent leaf node if it has more than minimum number of keys then first key of the adjacent node will go to the parent node and key in parent node which is partitioning will be combined together in one node. Suppose now parent has also less than the minimum number of keys then the same thing will be repeated until it will get the node which has more than the minimum number of keys.
For the second case key will be deleted and it's predecessor or successor key will come on its place. Suppose both nodes of predecessor and successor key have minimum number of keys then the nodes of predecessor and successor keys will be combined.

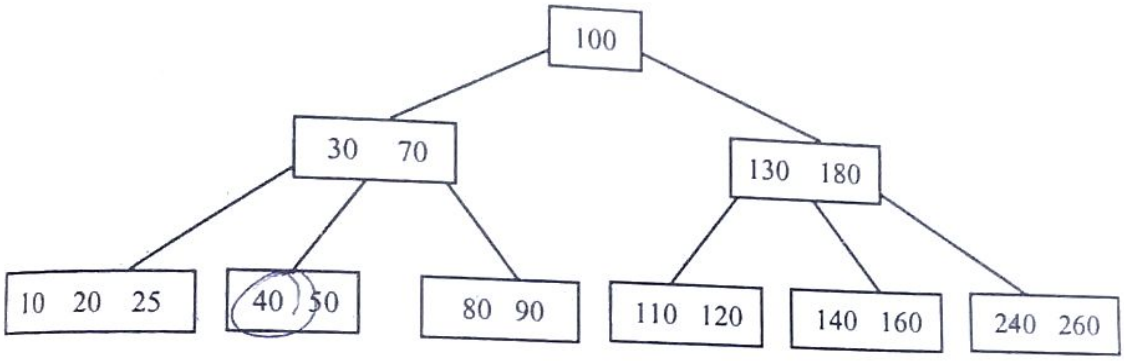


1. Delete 190



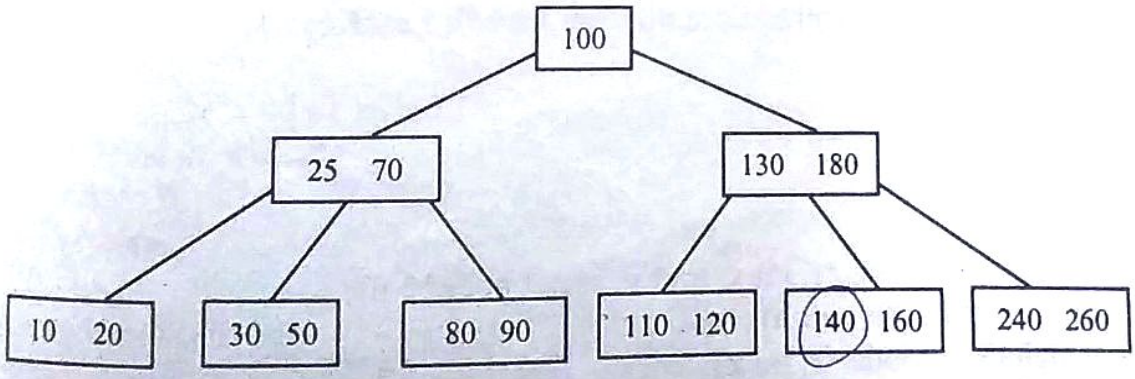
Here 190 is in leaf node, so we have a need to delete this from only leaf node.

2. Delete 60



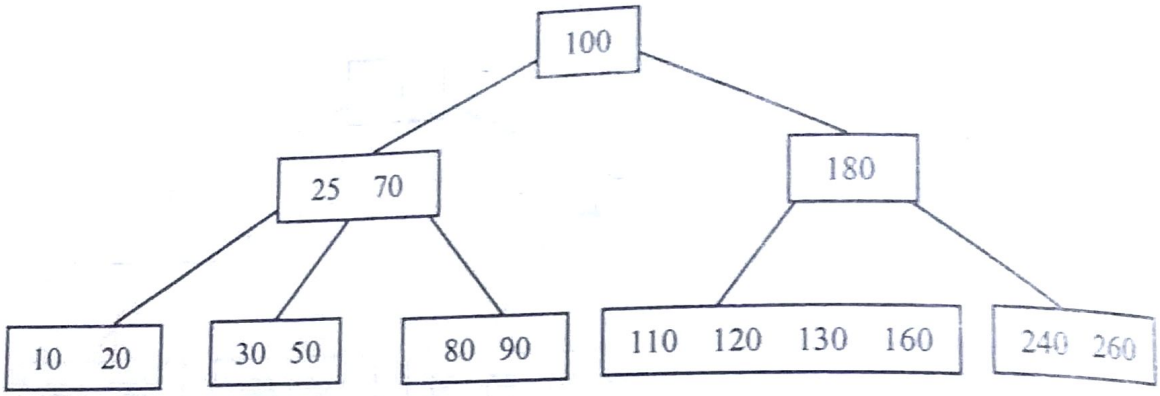
Here 60 is in non leaf node. So first it will be deleted from the node and then the element of right side child will come in the node.

3.Delete 40

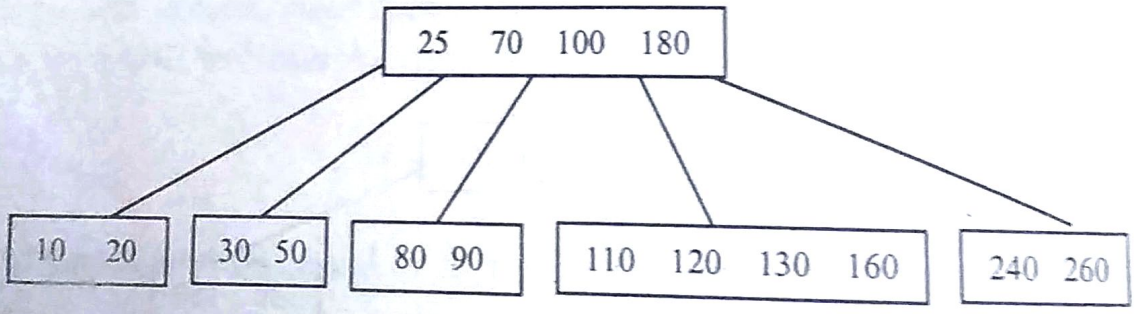


Here first 40 will be deleted from leaf node then left side element in the parent node will come in leaf node and then last element of the left side node of the parent node will come in parent node.

4. Delete 140



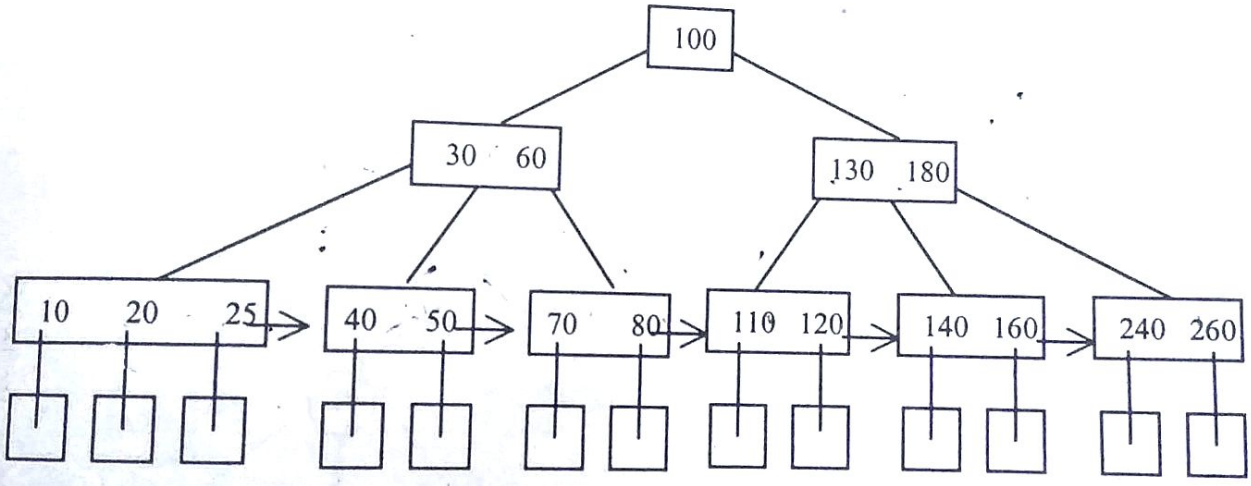
But at least two elements should be in the child of root so it will go in root node.



B+ tree-

In B- tree we can access record only randomly. We cannot traverse the records sequentially. After finding particular record we cannot get the previous or next record because it does not provide sequential traversal. B+ tree has the both properties, random access as well as sequential traversal.

In B+ tree all the keys which are in non leaf node will be in leaf node also and each leaf node will point to the next leaf node, so we can traverse sequentially also. Here keys in leaf node will point to the full record attached with that particular key. Let us take a B+ tree of order 5-



Here we can see all the keys in non leaf node are also in leaf node and each leaf node is pointing to the next leaf node.

Now suppose we want to search the key 30 then first it will be compared with 100 then will go to the left subtree, then it will compare with 30 and it will go to the left subtree. Now it's in leaf node and it will find 30 and will get the record pointed by the key 30. Now it can traverse sequentially in another leaf node also. So it is getting property to traverse sequentially all the keys.

Insertion in B+ tree-

Insertion in B+ tree is same as B-tree. But in one case where key is inserted in leaf node and it is already full then the median key will be in left splitted node and as well as in parent node. One pointer is also needed for pointing left node to right node.

Deletion in B+ tree-

Deletion in B+ tree is also same as B-tree. But if key that is going to be deleted belongs to non leaf node then we should delete that key from leaf node only because it's a partitioning key and key in non leaf node does not point to the record.

Digital Search Tree-