Q1

171044098
Akif KARTAL

AVL Tree step by step

Build AVL Tree

"20, 30, 8, 47, 39, 18, 40, 98 "

- **Building** AVL Tree with given sequence of integers.
- While building tree if tree is not balanced apply **rotation.**
- Building is done step by step on paper and scanned.
- Each node in the AVL tree has **its balance value** beside the that node.

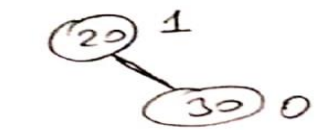
AVL TREE

Numbers = { 20, 30, 8, 47, 39, 18, 40, 98 }

Note = Numbers that is beside the nodes denotes the balance value of node.

Build AVL Tree

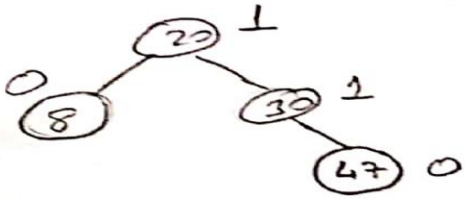
1



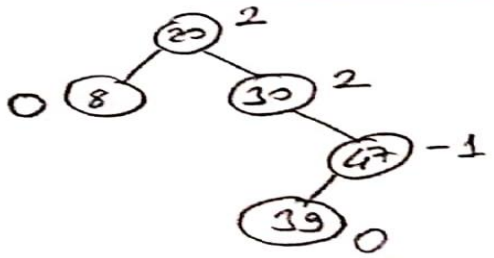
2

3

4



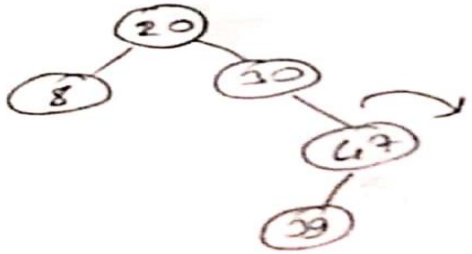
5

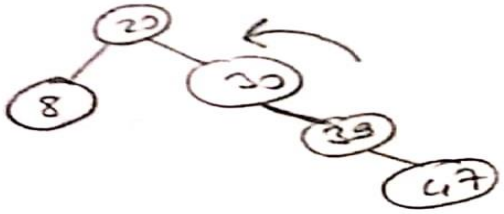


Now we have unbalanced tree

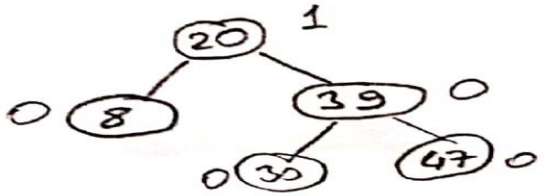
in 5th step 20 and 30 are unbalanced nodes so first we need to make right-left rotate on 30.

- Rotate right arround child

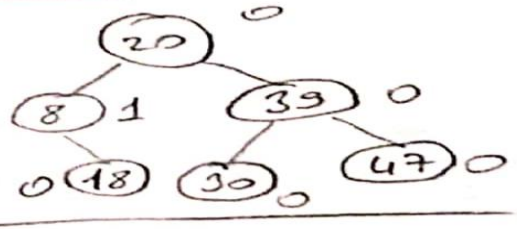




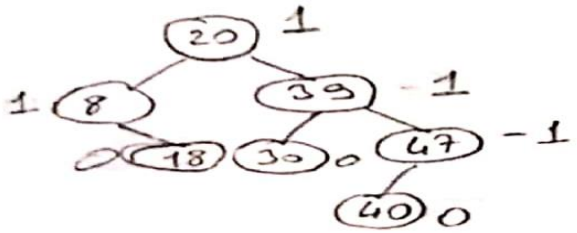
- Rotate left arround parent



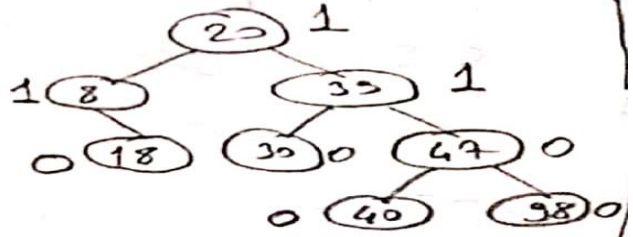
6



7



8



Building is done.

Tree is balanced.

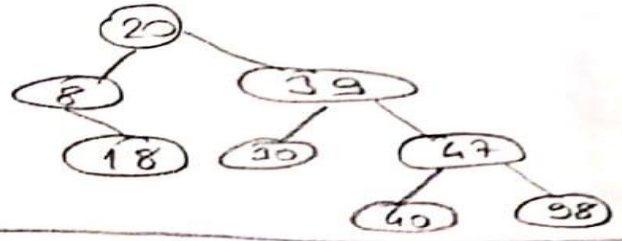
Removing from AVL Tree

"20, 30, 8, 47, 39, 18, 40, 98 "

- **Remove** all the items one by one in the same order (first in, first out).
- While removing an element apply **BST remove operations** after if tree is not balanced apply **rotation.**
- Each node in the AVL tree has its **balance value** beside the that node.

Removing from AVL Tree (20, 30, 8, 47, 39, 18, 40, 98)

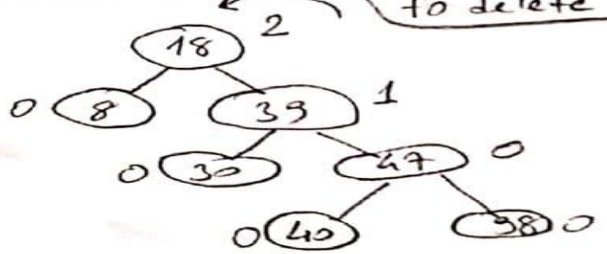
Tree



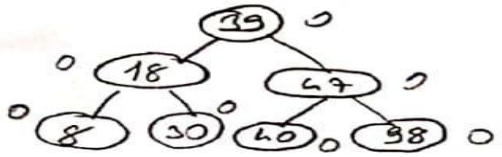
While deleting we apply BST remove operations.

To remove an element from an AVL tree first we need to replace it with the largest item in its left subtree. After that we make balance operations if tree is not balanced.

(1) Replace 20 with 18 then balance the tree. to delete 20.

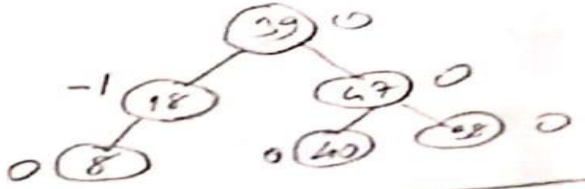


- Rotate left arround parent

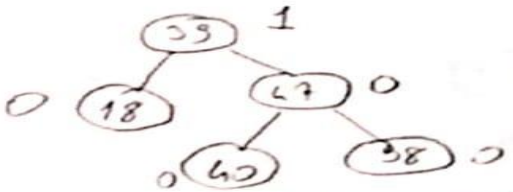


Now tree is balanced. Let's move on.

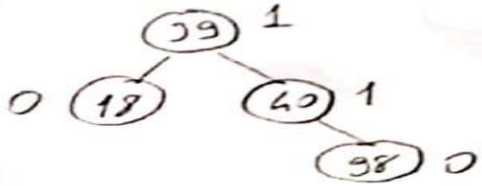
(2) Delete 30 directly



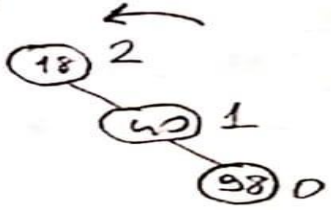
(3) Delete 8 directly



(4) To delete 47 replace it with 40



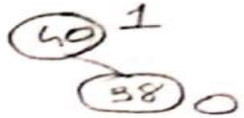
(5) To delete 39 replace it with 18



Now we need to rotate left arround parent to balance tree.

Rotate left then;

(6) Delete 18 directly



(7) Delete 40 directly

(8) Delete 98 directly

Deletion is done.

Red-Black Tree step by step

Build Red-Black Tree

"20, 30, 8, 47, 39, 18, 40, 98 "

- **Building** Red-Black tree with given sequence of integers.
- Building is done step by step on paper and scanned.
- While building we need to concern about followings;

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a null reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same.

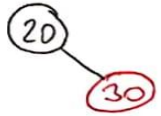
If any of them is violated we will be rebalancing by using recoloring, if it doesn't work: rotation and/or recoloring.

Build Red-Black Tree

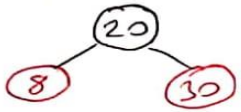
1)

Recolor the root

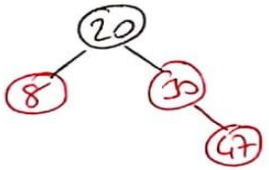
2)



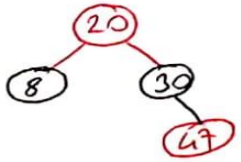
3)



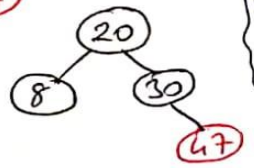
4)



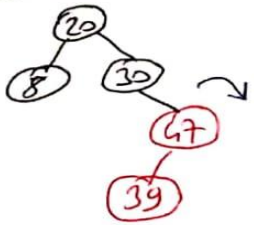
47 violates the rules.
47. uncle is red so recolor



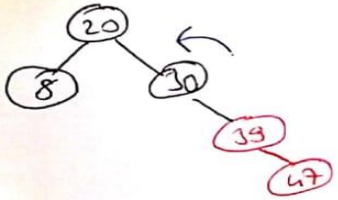
Recolor the root



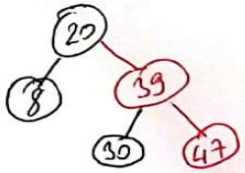
5)



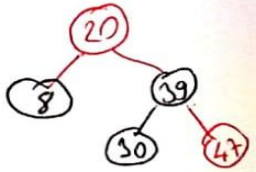
39 violates the rules
Rotate 47 right arround.



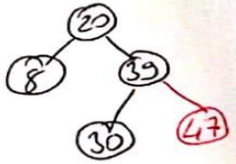
Rotate 30 left arround



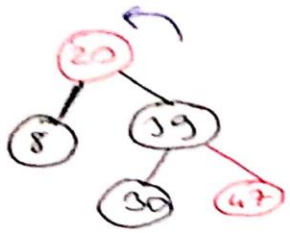
Recolor the parent and grandparent



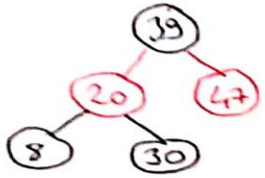
Recolor the root



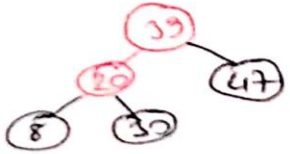
Now, this tree violates the rules
So we return back (don't change the root color)



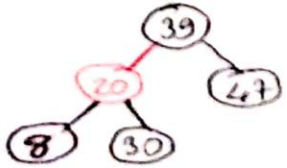
Rotate 20 left arround



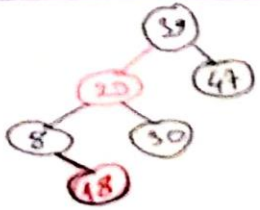
Change colors



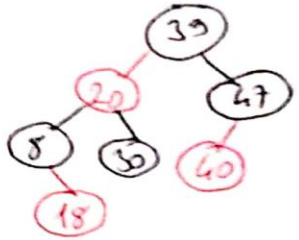
Change root color



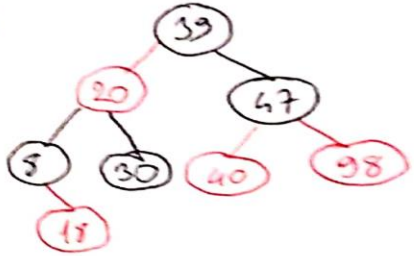
6)



7)



8)



Building is done.
Tree is balanced.

Removing from Red-Black Tree

"20, 30, 8, 47, 39, 18, 40, 98 "

- **Remove** all the items one by one in the same order (first in, first out).

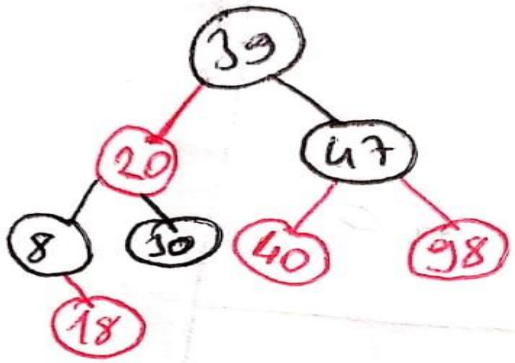
Removal follows the algorithm for a binary search tree . Recall that we remove a node only if it is a leaf or if it has only one child. Otherwise, the node that contains the inorder predecessor of the value being removed is the one that is removed.

If the node that is **removed is red,** nothing further must be done because red nodes do not affect a Red–Black tree's balance.

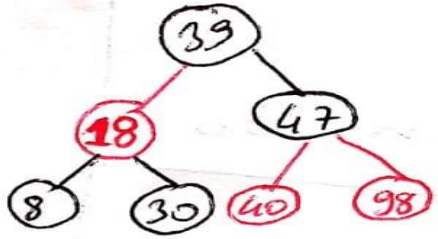
If the node to be **removed is black** and has a red child, then the red child takes its place, and we color it black.

However, if we remove a black leaf, then the black height is now out of balance. There are several cases that must be considered.

Tree

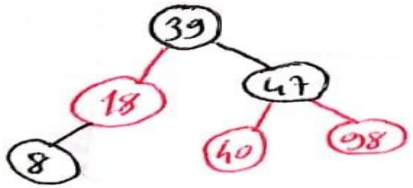


1) Remove 20



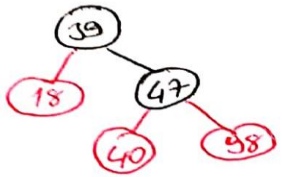
Replace it with 18

2) Remove 30

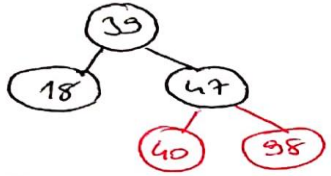


Remove directly since a leaf.

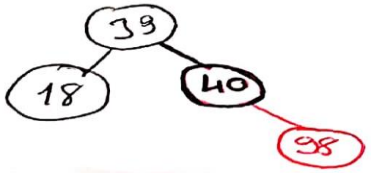
3) Remove 8



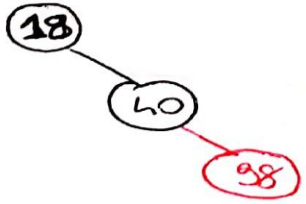
This tree violates the rules
We recolor 18



4) Remove 47 (Replace it with 40)

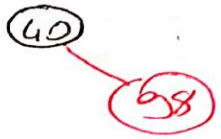


5) Remove 39



Replace it with 18

6)



Remove 18
Replace it with 40

7) Remove 40

Replace it with 98

8) Remove 98
- Removing is done.

2-3 Tree Tree step by step

Build 2-3 Tree

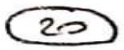
"20, 30, 8, 47, 39, 18, 40, 98 "

- **Building** 2-3 tree with given sequence of integers.
- Building is done step by step on paper and scanned.

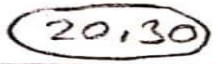
A 2-3 tree maintains balance by being built from the bottom up, not the top down.
Instead of hanging a new node onto a leaf, we insert the new node into a leaf.
If a node has 3 value because a node can’t store three values, the middle value propagates up to the 2-node parent and this leaf node splits into two new 2-nodes.

Numbers = {20, 30, 8, 47, 39, 18, 40, 98}

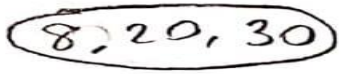
1) Add 20



2) Add 30



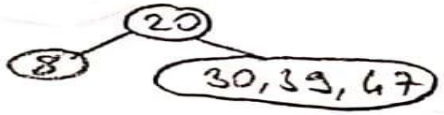
3) Add 8



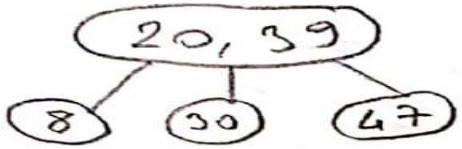
Now we have to move middle element to the upward and split others

4) Add 47

5) Add 39



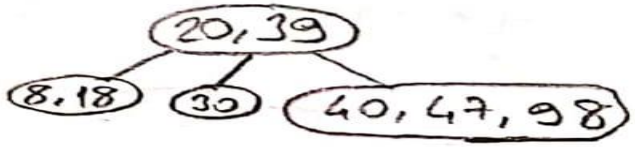
Now we have to do move and split.



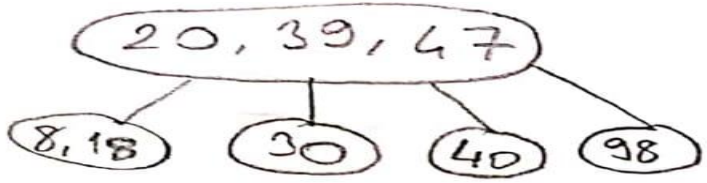
6) Add 18

7) Add 40

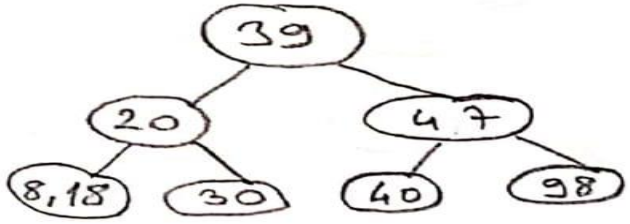
8) Add 98



Now we have to do move to the upward and split others



Repeat same process



Building is done.

Removing from 2-3 Treee

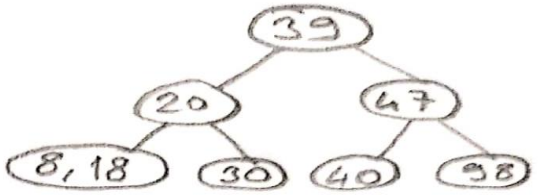
"20, 30, 8, 47, 39, 18, 40, 98 "

- **Remove** all the items one by one in the same order (first in, first out).

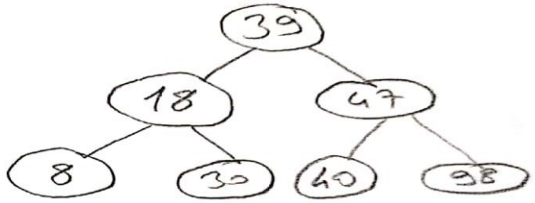
Removing an item from a 2-3 tree is generally the reverse of the insertion process

- If the item to be removed is in a leaf, simply delete it
- If it's not in a leaf, remove it by swapping it with its inorder predecessor in a leaf node and deleting it from the leaf node.
- If removing a node from a leaf causes the leaf to become empty,
  - ✓ items from the sibling and parent can be redistributed into that leaf
  - ✓ or the leaf can be merged with its parent and sibling nodes

Tree

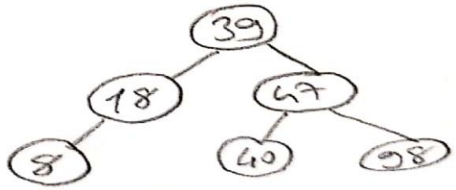


1) Remove 20

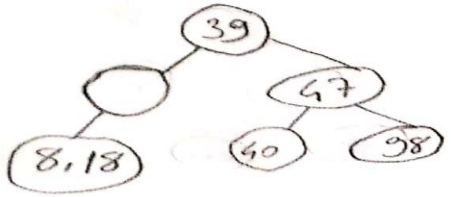


Replace 20 with 18
which is its leaf predecessor.

2) Remove 30



Remove directly since it is a leaf.
Now merge 8 and 18

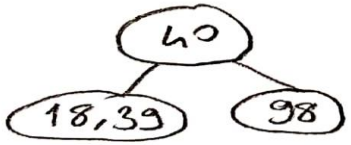


Repeat for other side (39 and 47)

3) Remove 8

4) Remove 47

Replace it with 40
Now merge 39 and 18



5) Remove 39

6) Remove 18

7) Remove 40

8) Remove 98
- removing is done.

Skip List step by step

Build Skip List

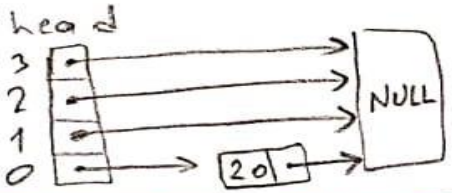
"20, 30, 8, 47, 39, 18, 40, 98 "

- **Building** Skip List tree with given sequence of integers.
- Building is done step by step on paper and scanned.
- Deafult skip list capacity is 16, Max Level is 4.
- Level of the new node is given by me according to rules.

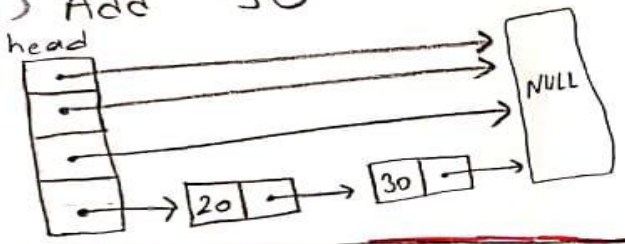
First we search the element in the list. A search always begins in the highest level list (the list with the fewest elements).
If the search algorithm fails to find the target, it will find its predecessor in the level-1 list, which is the target's insertion point.
While we know the insertion point, we need to determine the level of the new node.
The level is chosen at random based on the number of items currently in the skip-list.
The random number is chosen with a logarithmic distribution.

Building Skip-List (Capacity = 16, Max Level = 4)

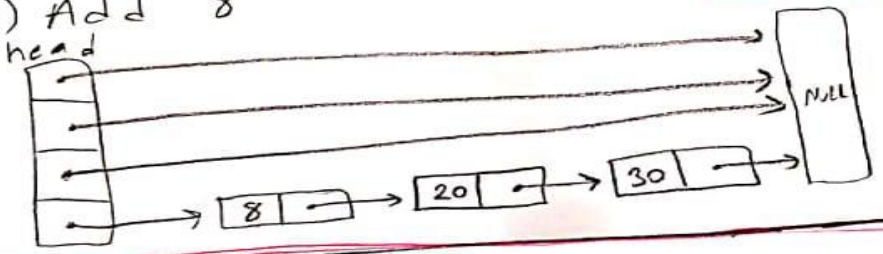
1) Add 20



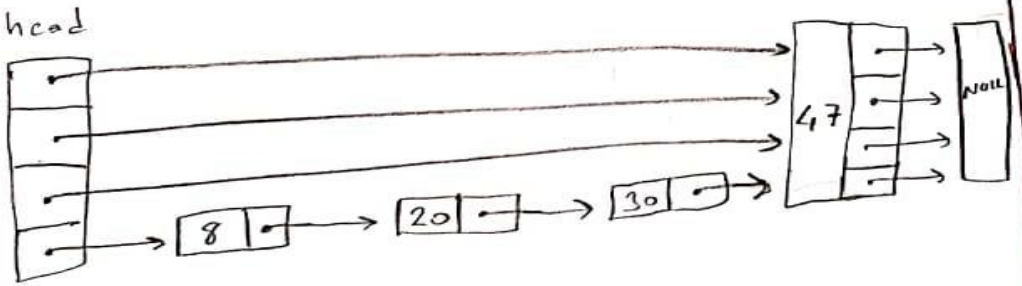
2) Add 30



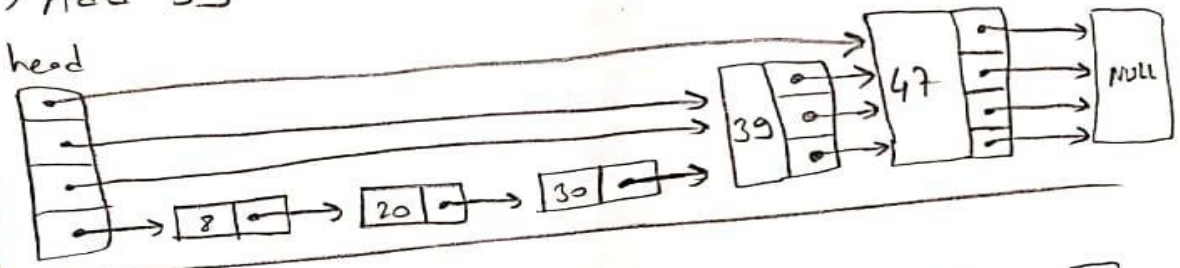
3) Add 8



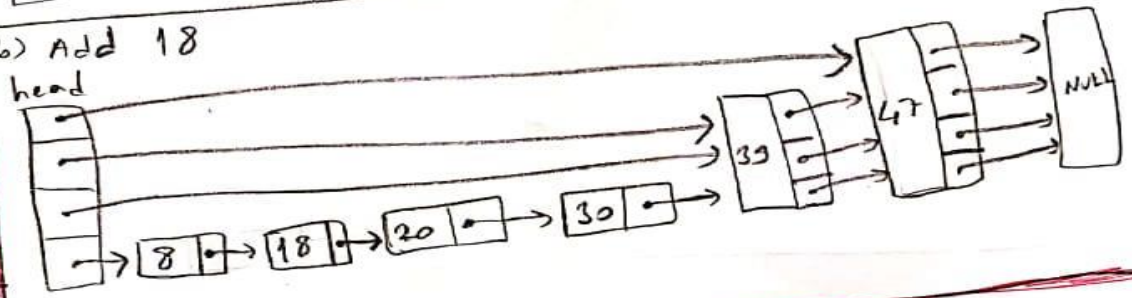
4) Add 47



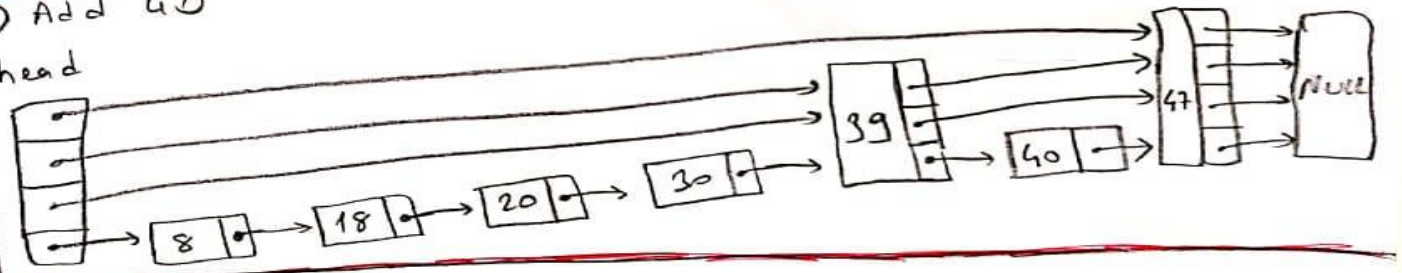
5) Add 39



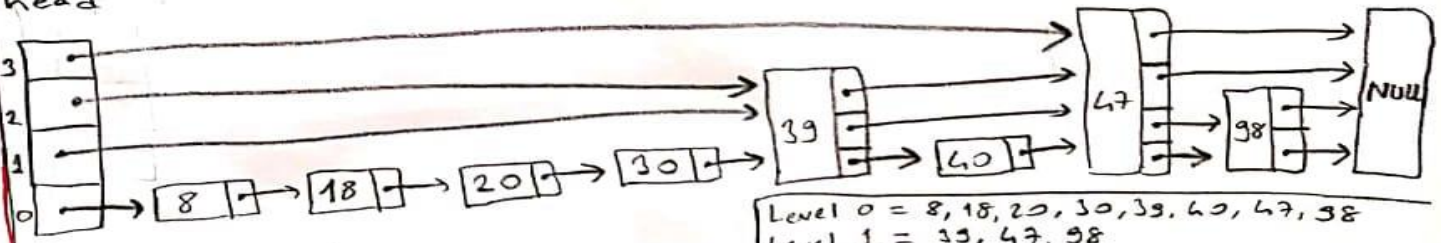
6) Add 18



7) Add 40



8) Add 98



Building is done.

Level 0 = 8, 18, 20, 30, 39, 40, 47, 98
Level 1 = 39, 47, 98
Level 2 = 39, 47
Level 3 = 47

Removing from Skip List

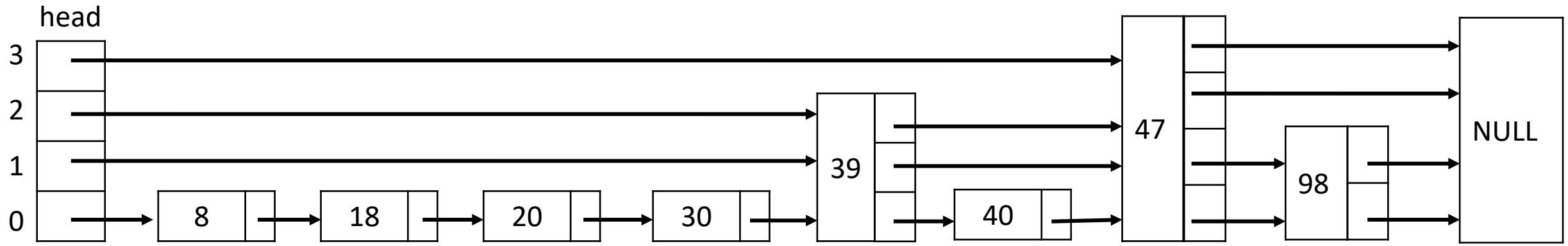
"20, 30, 8, 47, 39, 18, 40, 98 "

- **Remove** all the items one by one in the same order (first in, first out).
- Removing is done step by step on this document and showed.

First we search the element in the list. A search always begins in the highest level list (the list with the fewest elements).

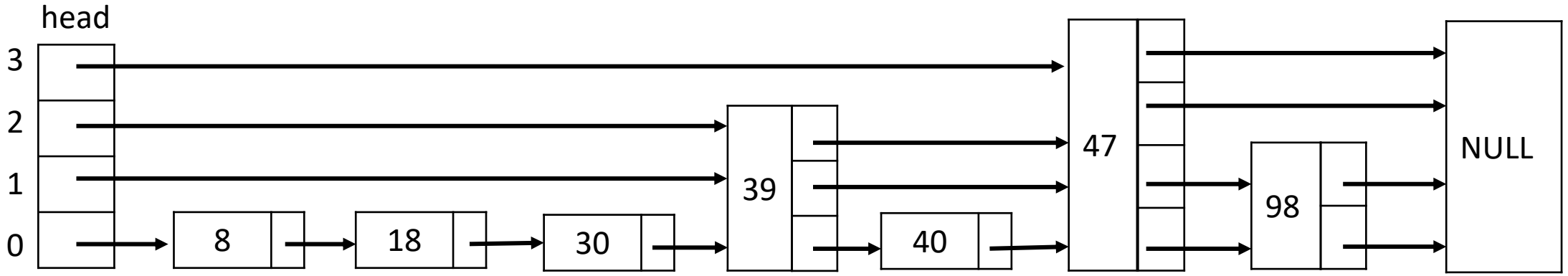
Along the way we recorded the last node visited at each level. We use these nodes to delete target node.

Skip-List(which was created.)

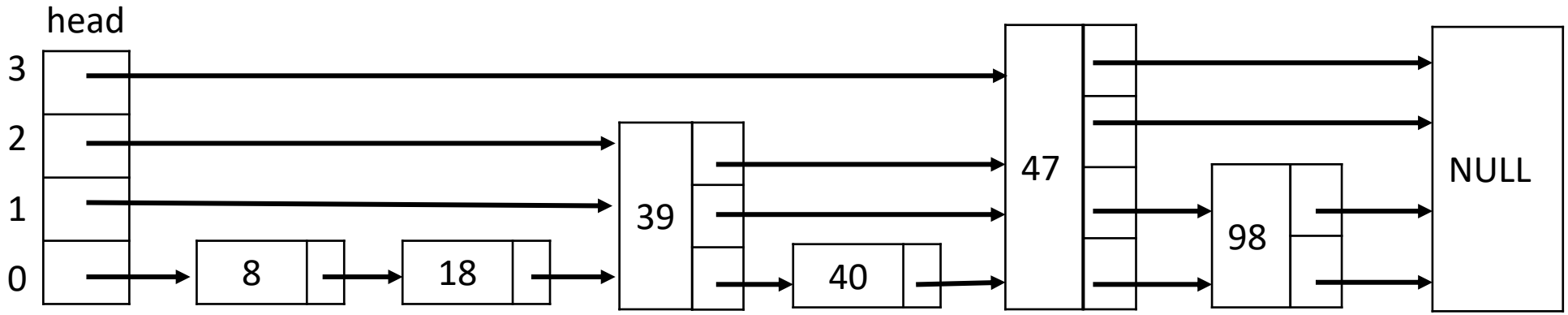


Level 0 = 20, 30, 8, 47, 39, 18, 40, 98
Level 1 = 39, 47, 98
Level 2 = 39, 47
Level 3 = 47

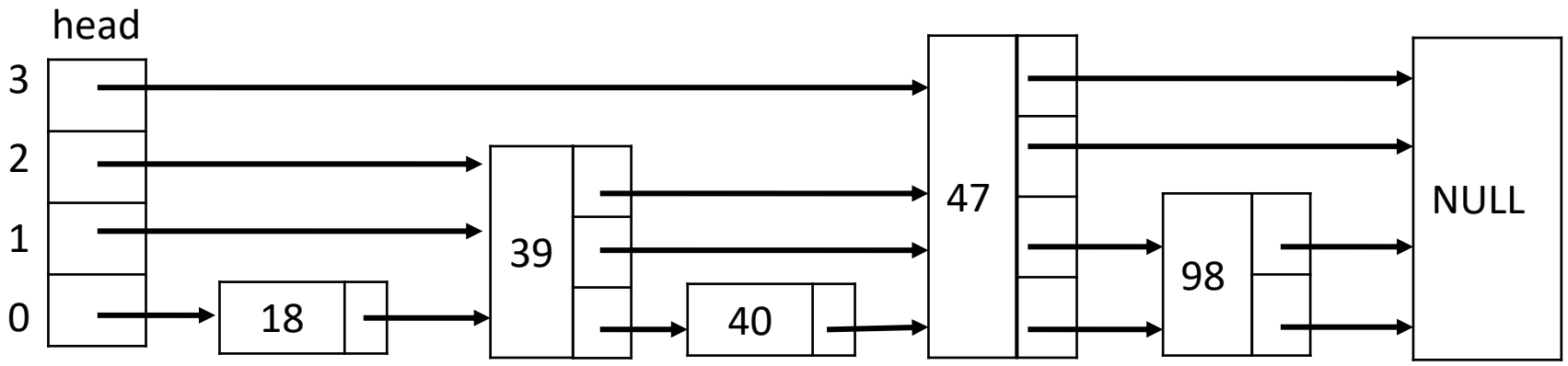
Remove 20



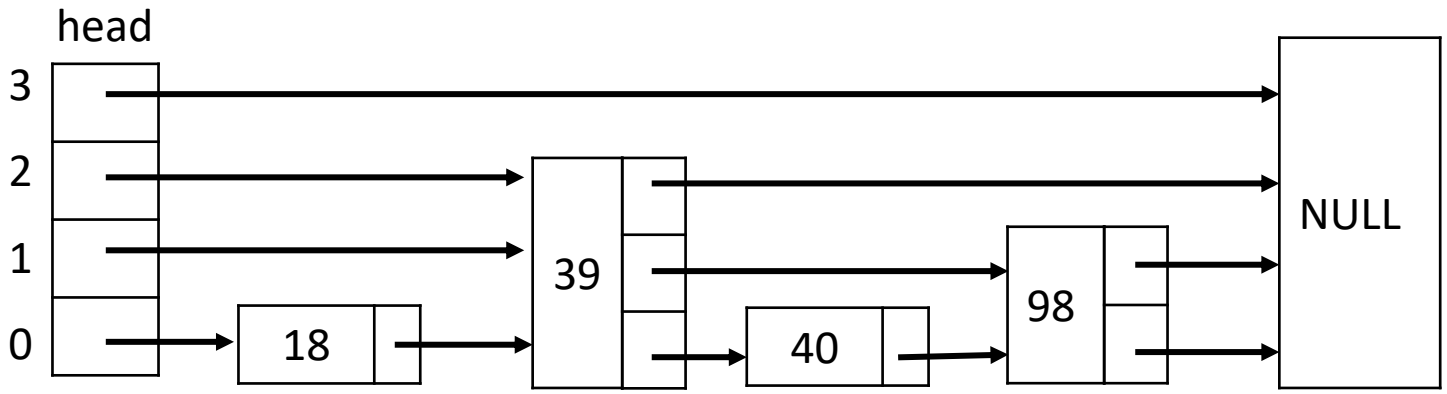
Remove 30



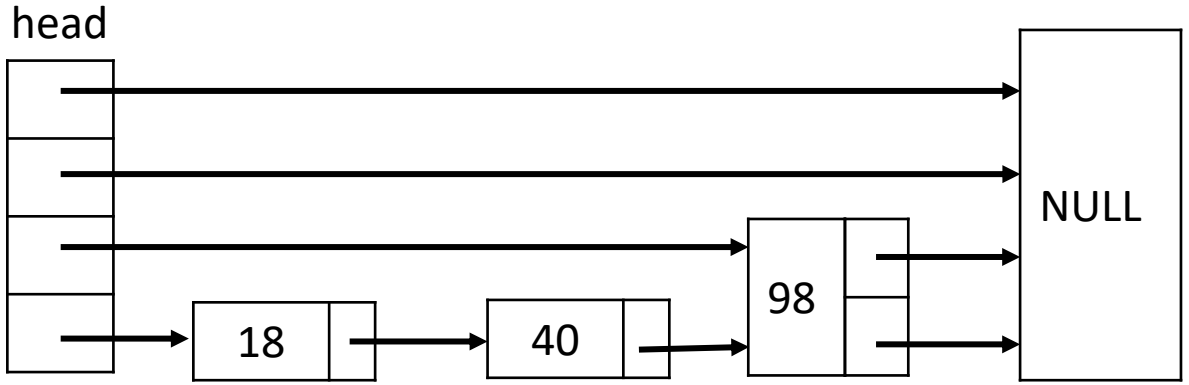
Remove 8



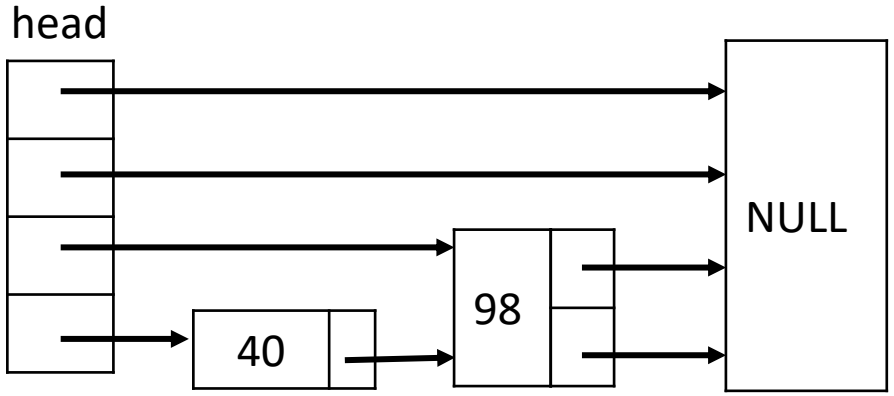
Remove 47



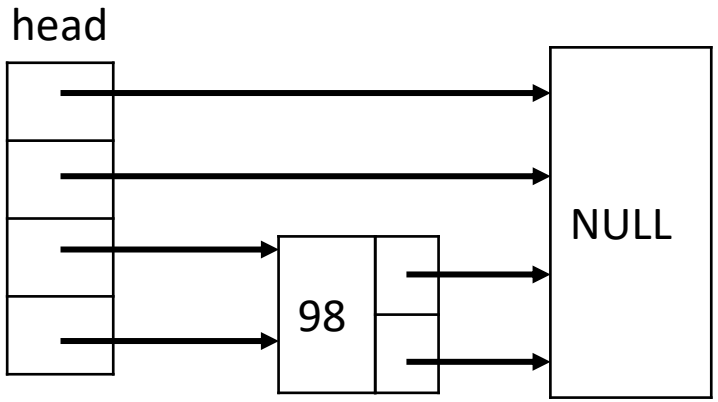
Remove 39



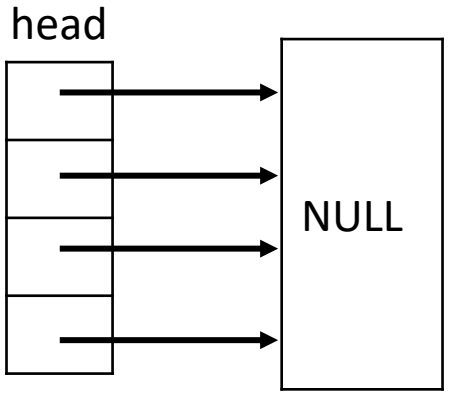
Remove 18



Remove 40



Remove 98



Removing is done.

Build B-Tree with order 4

"20, 30, 8, 47, 39, 18, 40, 98 "

- **Building** B-Tree with given sequence of integers.
- Building is done step by step on paper and scanned.

Insertion is Similar to 2-3 trees, insertions take place in leaves.
If a leaf to receive the insertion is full, it is split into two nodes, each containing approximately half the items, and the middle item is passed up to the split node's parents.
If the parent is full, it is split and its middle item is passed up to its parent, and so on.

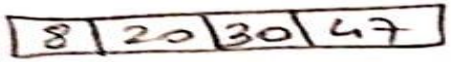
Adding B-Tree with order 4

1) Add 20

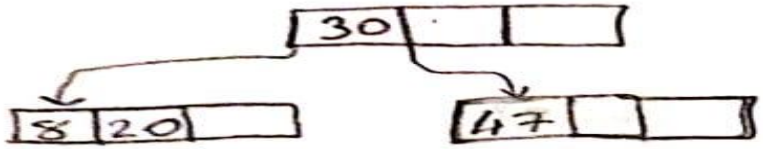
2) Add 30

3) Add 8

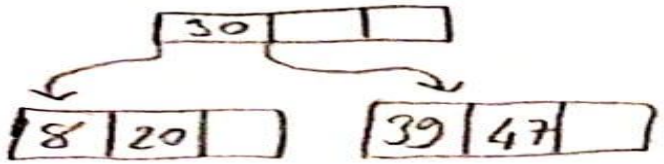
4) Add 47



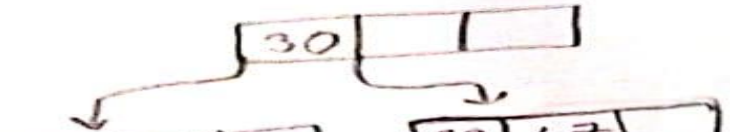
Now we have to move middle element to the upward and Split the others (order is 4)



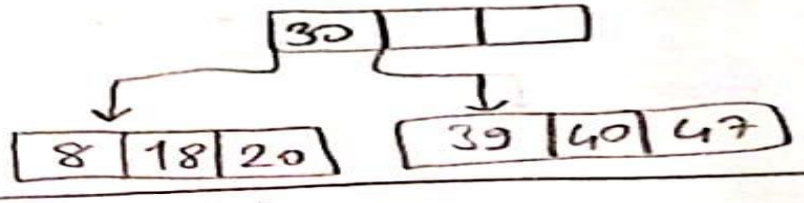
5) Add 39



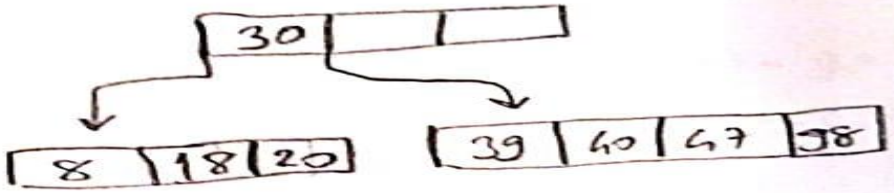
6) Add 18



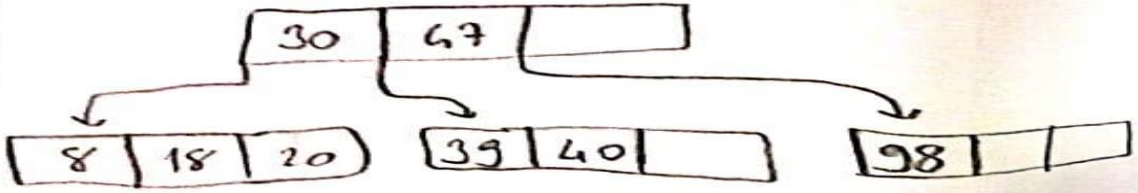
7) Add 40



8) Add 98



Now we have to move middle element to the upward and split the others (order is 4)



Building is done.

Removing from B-Treee

"20, 30, 8, 47, 39, 18, 40, 98 "

- **Remove** all the items one by one in the same order (first in, first out).

Removing an item is a generalization of removing an item from a 2-3 tree

The simplest removal is deletion from a leaf.

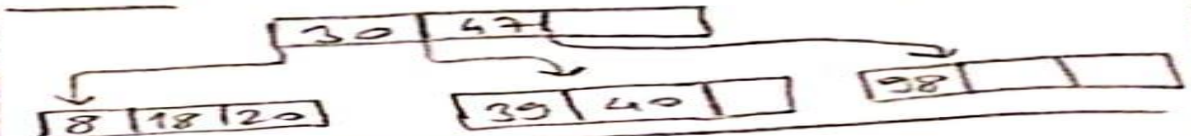
When an item is removed from an interior node, it must be replaced by its inorder predecessor (or successor) in a leaf.

If removing an item from a leaf results in the leaf being less than half full, **redistribution** needs to occur.

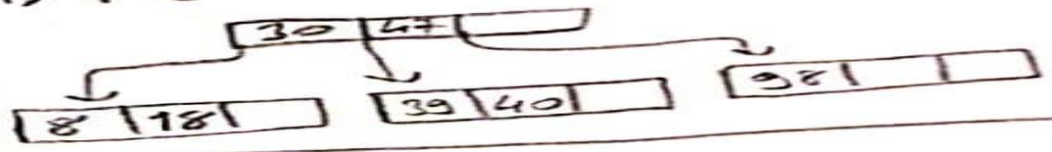
Redistribution can be merging it and its parent with its sibling.

Removing from B-Tree

Tree

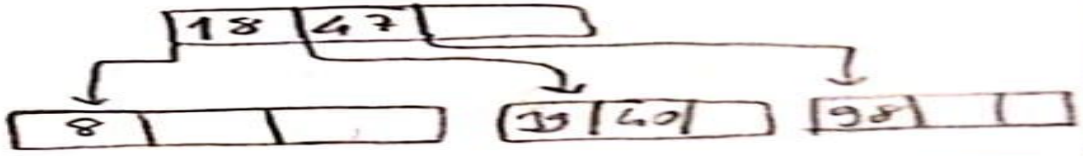


1) Remove 20



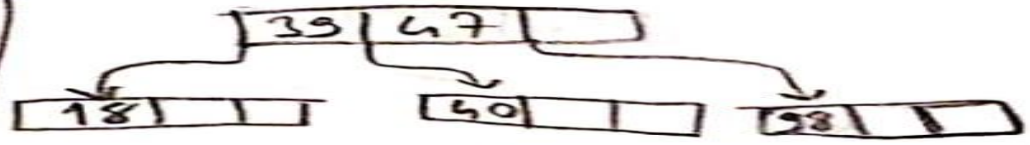
2) Remove 30

Replace it with inorder predecessor (18)



3) Remove 8

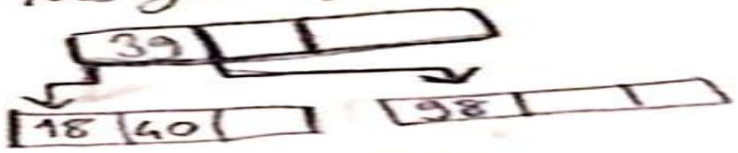
Now since the node contain 8 has only 1 element to delete 8 we need to barrow 1 element from right sibling To do this we use parent node.



We removed 8 by barrowing one element from right sibling

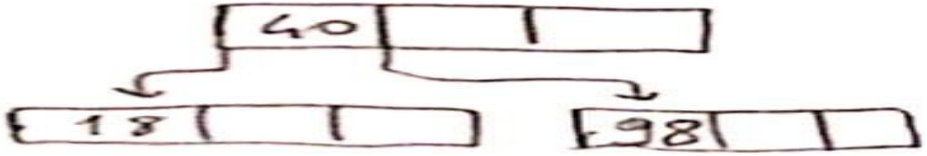
4) Remove 47

To remove 47 we can replace it inorder predecessor or successor but they contain minimum number of key so we have to do merge operation.

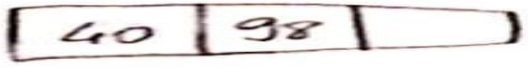


5) Remove 39

Replace it with 40



6) Remove 18



7) Remove 40

8) Remove 98

Removing is done.