B-Tree (4 order):

*In computer science, B-tree is a self balancing tree data-structure that maintains sorted data and allows searches, sequential access, insertion and deletion in logarithmic time. The B tree generalizes the binary-search tree, allowing the nodes with more than two children. Unlike other self-balancing binary-search-tree, the B-tree is well suited for storage systems that read and write relatively large blocks of data, such discs. It commonly use in databases and file systems.

* Definition

1) Every node has at most m children
2) Every non-leaf node except root has at $m/2$ child nodes.
3) The root has at least two children if it is not a leaf node.
4) A non leaf node with key children contains k-1 keys.
5) All leaves appear in the same level and carry no information.

Note: when I research about B-tree, I encountered that "order" of tree has different definition. In this homework and also our book identify order as maximum number of children which is one more than the maximum number of keys.

*if order is 4 there must be 4 different child and 3 different key.

m = 4
key = 3

min children = $\frac{4}{2} = 2$ max children = 4
min keys = ceil$\left(\frac{m}{2}-1\right)$ = 1 → except root node
max keys = 3

Input = { 20, 30, 8, 42, 39, 15, 40, 24 }

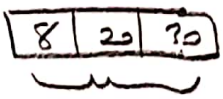
add(20)

(There is only one element)

add(30)

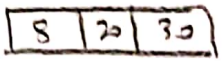
(There is a empty place which we can put item.)

add(8)



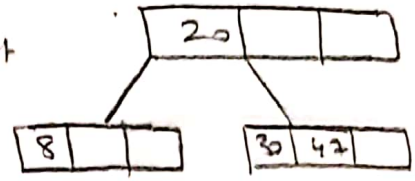
* Be careful that items in the same node must be in order in B tree.

add(47)



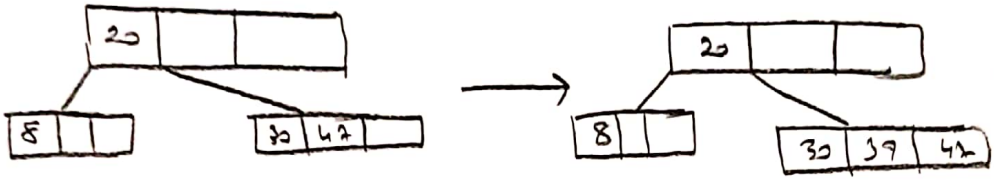
Root is in already full capacity

* we need to root into two different part of the array.



=

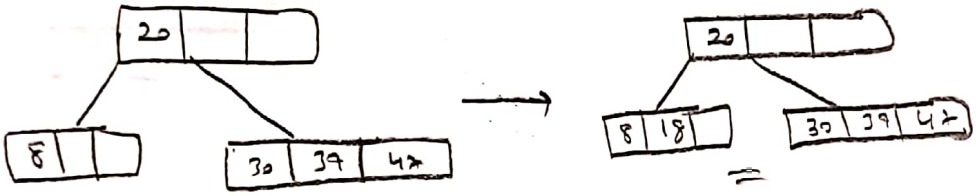
add (39)



* 39 has been added to right most side node because, there is an empty place, to put in order.

=

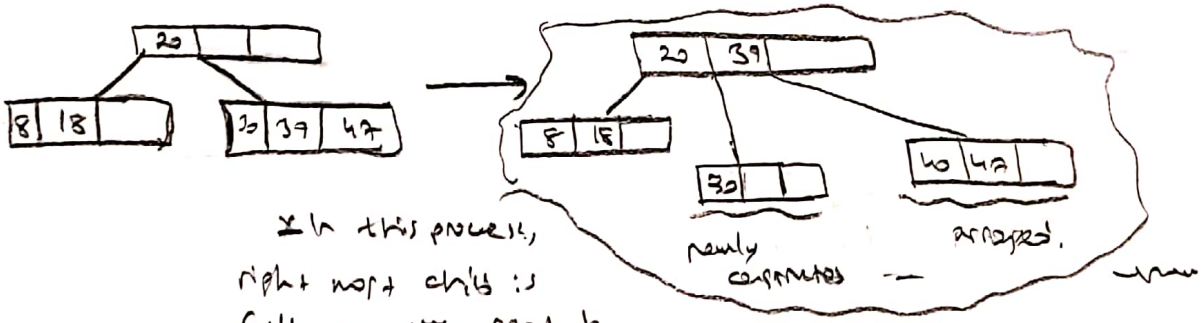
add (18)



* 18 has been added to left most side node because there is an empty place, to put in order.

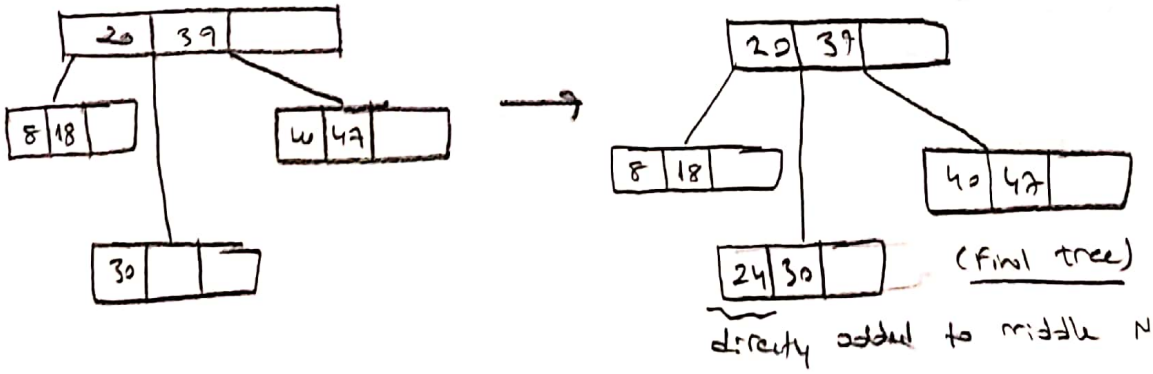
=

add (40)



* In this process, right most child is full, so we need to divide in to two part. and create a new node in middle.

add(24)



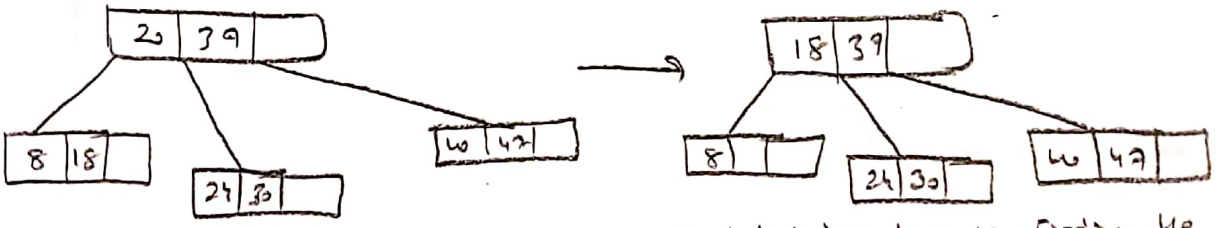
directly added to middle Node

(Final tree)

* removing from B tree operation type depends on what node that the element is belonging to. If node is leaf node, there will be different operation, if node is internal node there will be a different operation.

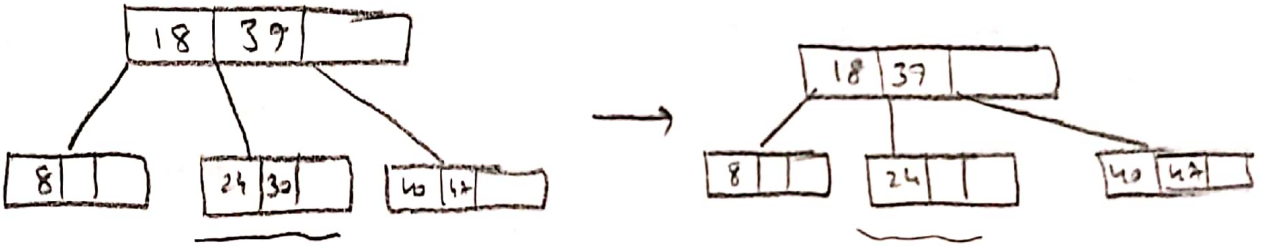
* removing the internal node based on exchanging with inorder predecessor or inorder successor.

remove (20)



* what has done is finding the inorder successor and replacing with removing element location
* while doing that we need to check our min-key value is "1".

remove (30)



* There are more than min-key element in it

* we can directly remove the (30) from here.

remove (8)

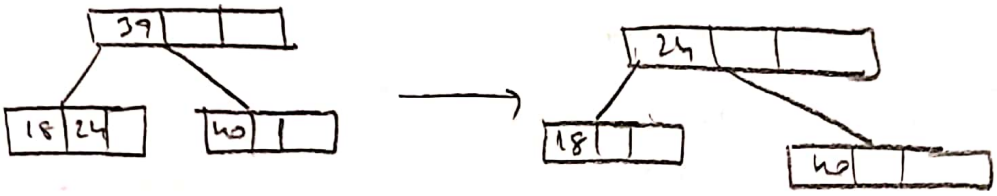
* when 8 is removed, left most node will be empty so, there must be merging operation

* Thus, middle and first node merged together. 18 taken from root

remove (47)

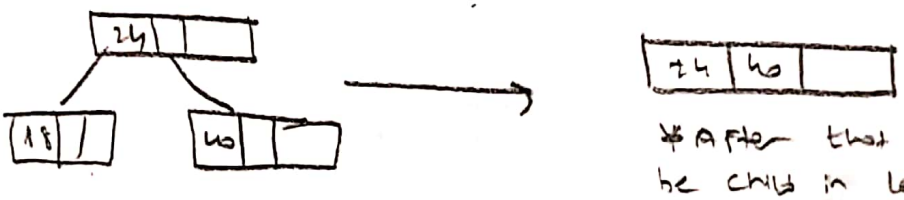
* Just removed from that node. there is restriction about min key.

remove (39)



* After deleting (39), what is searched is inorder predecessor & that elment replaces it with removed element.

remove (18)

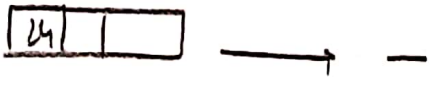


* After that point there will not be child in left most node so merging must be done. After merging there will be only root node.

remove (40)

* Just remove from root

remove (24)



* No longer, there is no elment in root.