B tree

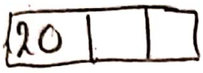
20, 30, 8, 47, 39, 18, 40, 95

order 4 ⇒ 4-1=3 keys max

4 childtannen

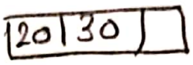
⇒ 1) 20

⇒

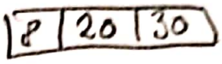


2) 30

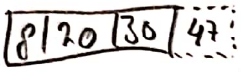
⇒



3) 8 (change places to preserve order

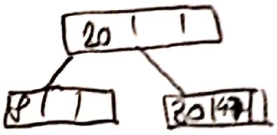


4) 47



⇒ split since 4 in one node

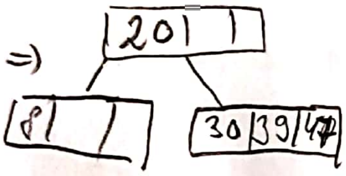
⇒ 20 (mid) goes up ⇒



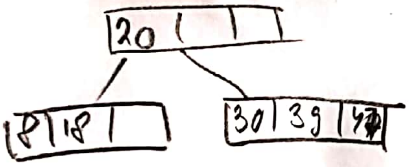
5) 39 (bigger than 20 ⇒ its right child

+ update positions to perserve order

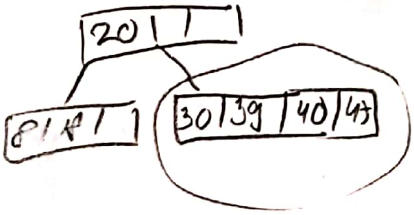
⇒



6) 18 (smalle than 20 ⇒ left)

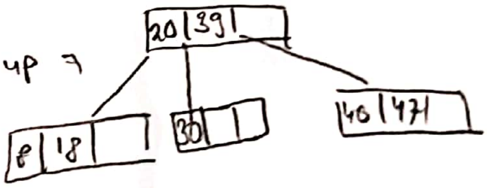


7) 40



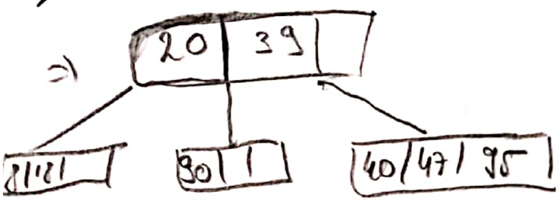
split

⇒ 39 goes up ⇒



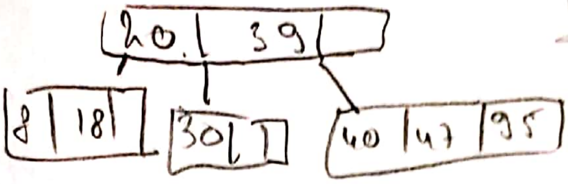
8) 95

⇒



Scanned with CamScanner

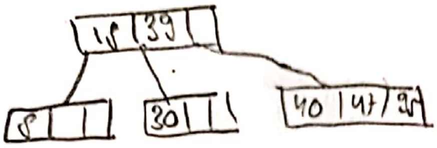
B tree removal



1) 20 removing

replace with largest of the left subtree

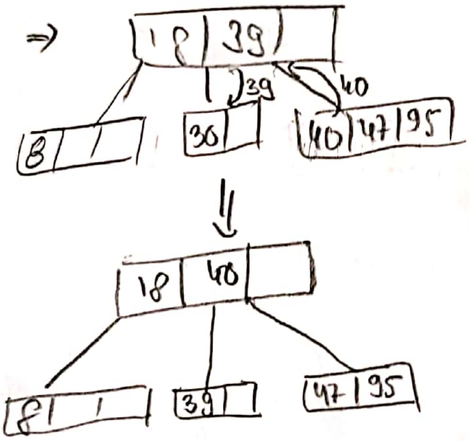
=)



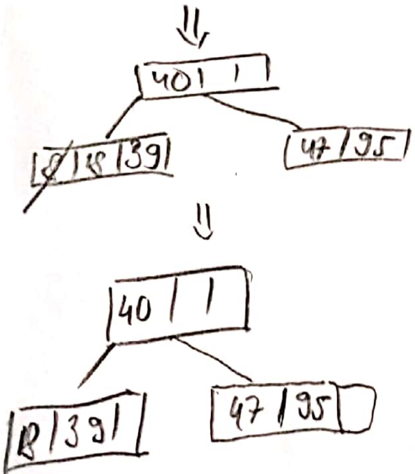
2) 30 removing

removing it makes its node have 0 keys and it needs at least 1, so since it is leaf borrow from immediate right through parent since right sibeing has more than minimum children

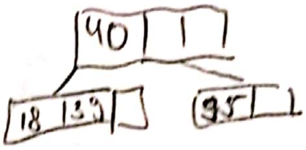
=)



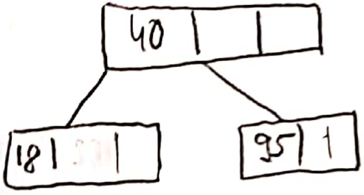
3) 8 (try borrowing from left or right but since it is not possible =) merge with sibling and shared parent(18) and than remove



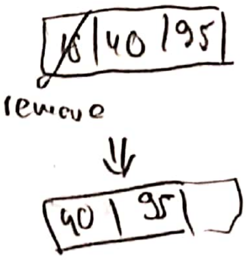
4) 47 (just remove since more than min childern and a leaf node



5) 39 (also leaf and more than min children)

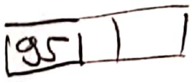


6) 18 (can't borrow from sibling =) merge with parent and remove



7) 40 (just remove since leaf and more than min children

=)



8) 95 (the only element + leaf =) remove)

=)

→ empty

AVL tree

171044095

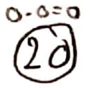
① BST

② |height of left Subtree| - |height of right subtree| = {-1,0,1} (Balance factor)

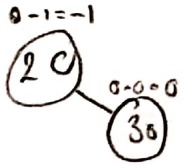
{Insertion}

20, 30, 8, 47, 39, 18, 40, 95

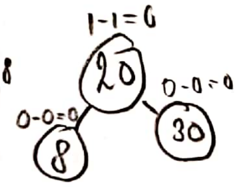
1) 20



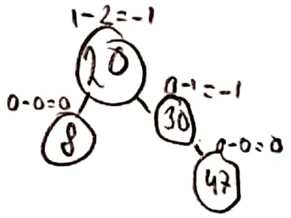
2) 30



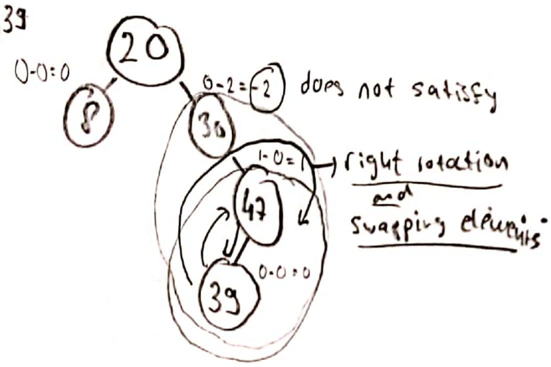
3) 8

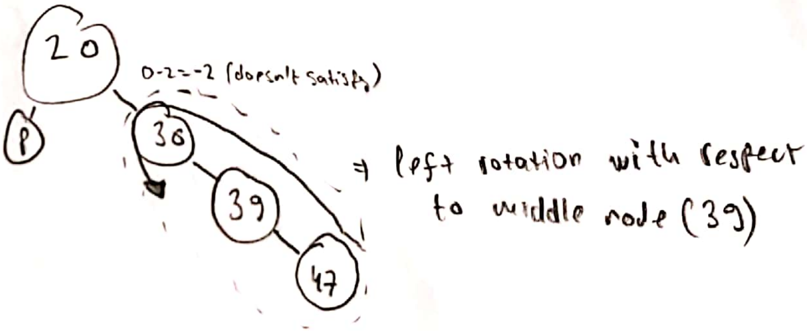


4) 47

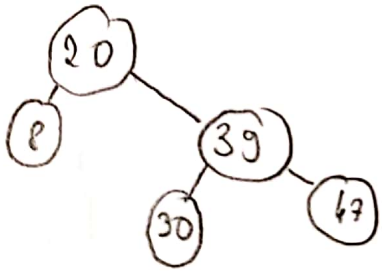


5) 39

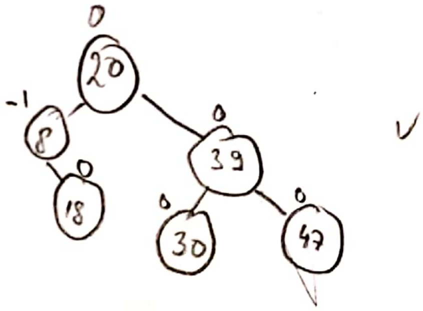




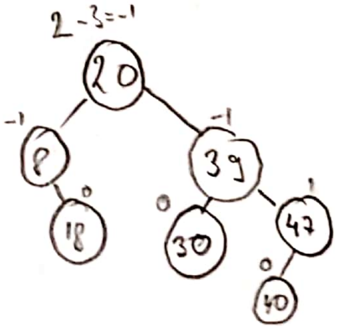
5)



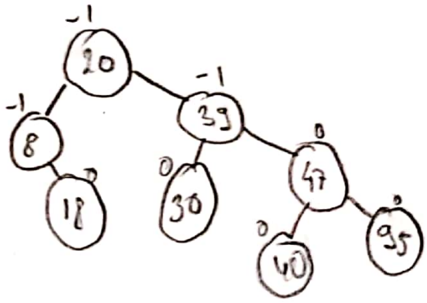
6) 18



7) 40

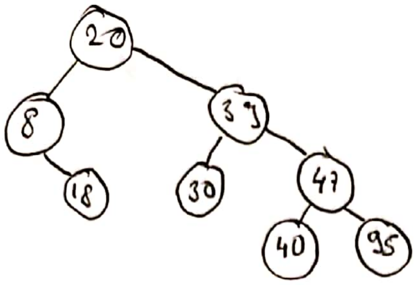


8) 95

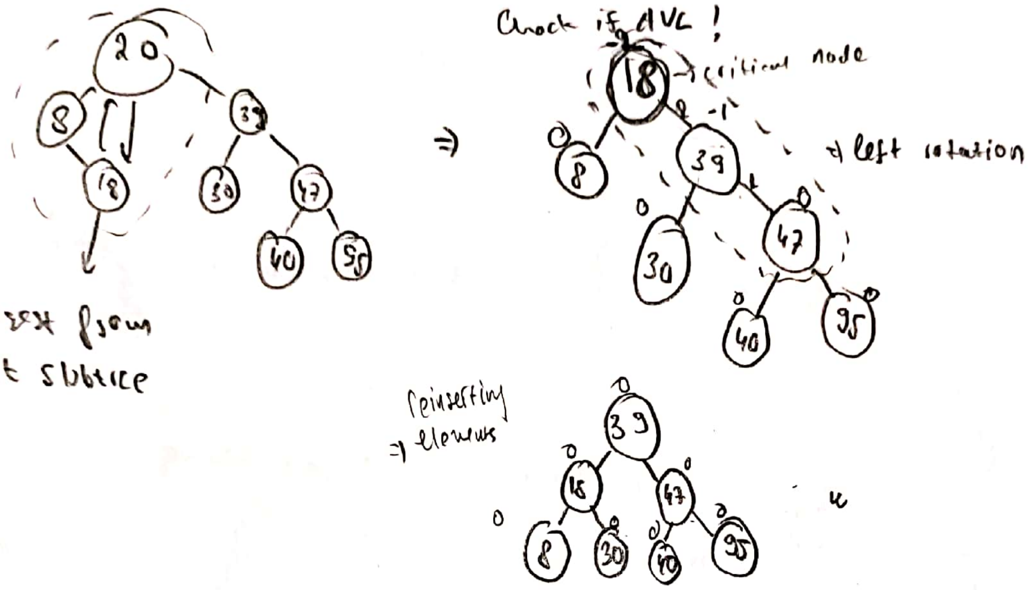


Scanned with CamScanner

AVL removal



1) 20 ⇒ BST Removal


largest from left subtree

Check if AVL!

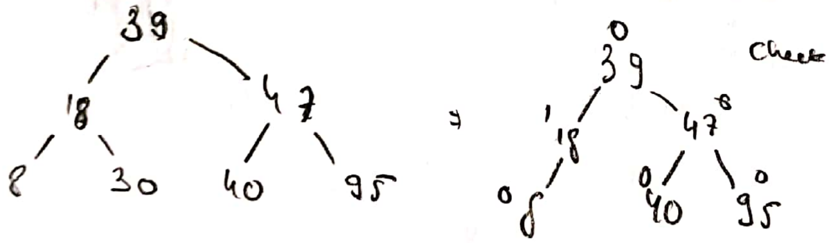
critical node

→ left rotation

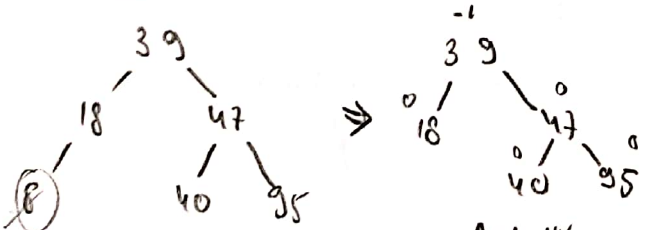
reinserting ⇒ elements

✓

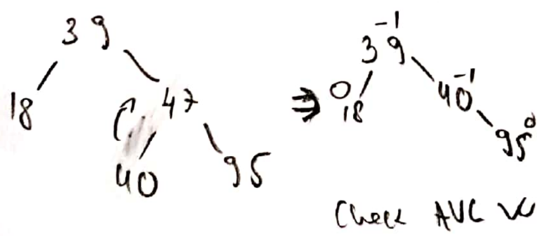
2) 30 removing (leaf node ⇒ just remove it for BST


Check AVL → good

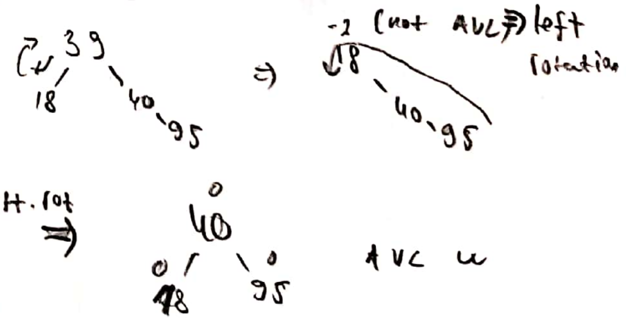
3) 8 removing (Leaf ⇒ just remove)


AVL ✓

4) 47 removal.. (replace with largest in left subtree)


Check AVL ✓

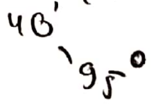
5) 39 removing (largest left subtree


-2 (not AVL ⇒ left rotation

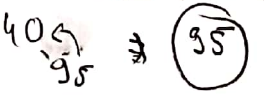
left. rot ⇒

AVL ✓

6) 18 removal (leaf ⇒ just rm)


AVL ✓

7) 40 removal (one child ⇒ swap with that child)


AVL ✓

8) 95 (only node ⇒ just remove it

∅ → empty tree

Red- Black tree

These steps are followed

1) Empty tree => create **black** root node
2) Not empty => new node as leaf node that's **Red**

3) Parent of new node black => **exit**
4) Parent red => check color of parent's **sibling** :

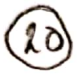
a) black or null => do AVL rotation & recolor

b) Red => recolor & check if parent's parent of new node is not root node => recolor it & recheck

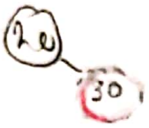
20, 30, 8, 47, 39, 18, 40, 95

1) 20

empty => root is black

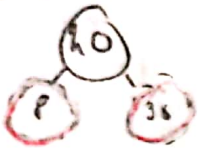


2) 30 (add as red comp to BST)



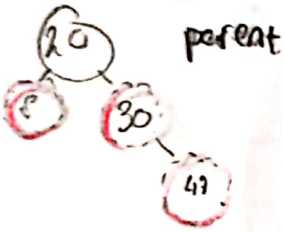
black parent => exit

3) 8 (add as red exit)

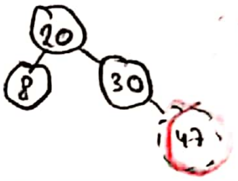


parent Black => ok

4) 47

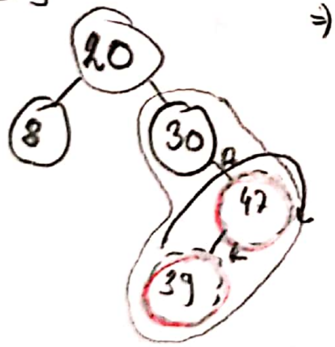


parent red => Check sibling
=> it's red
=> recolor parent & sibling =>

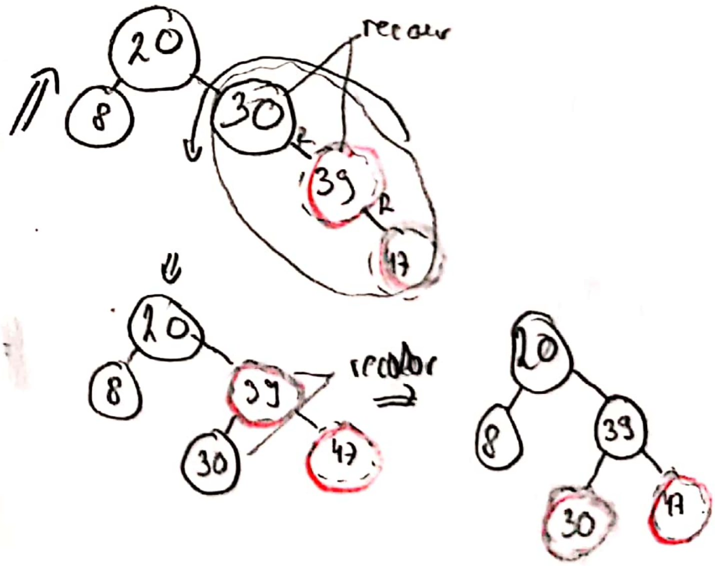


parent's parent root => ok

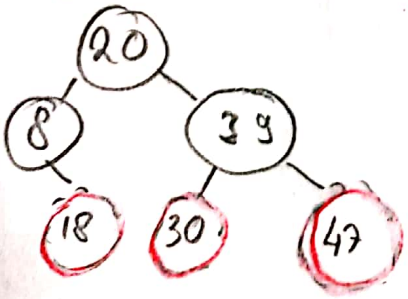
5) 39



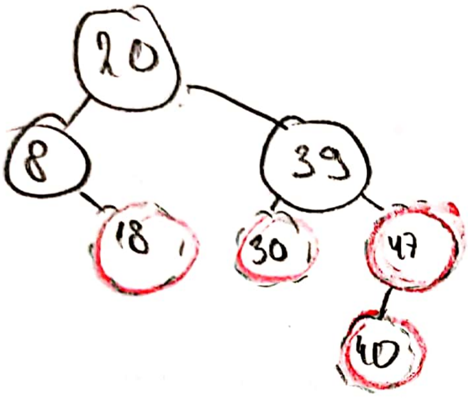
=> parent red => Check sibling
=> null
=> do AVL rotation & recolor



6) 18 (parent Black => it's okay



7) 40 (add as red leaf)



=) 40's parent red

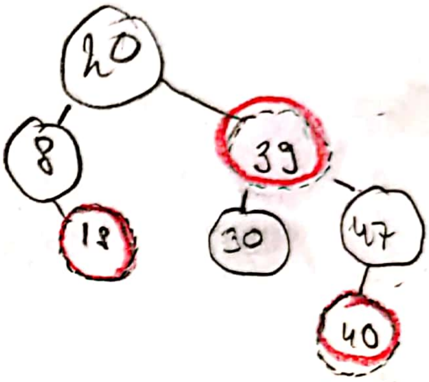
=) check sibling

=) red

=) recolor =)

=) parent's parent not root => also recolor

=) recheck

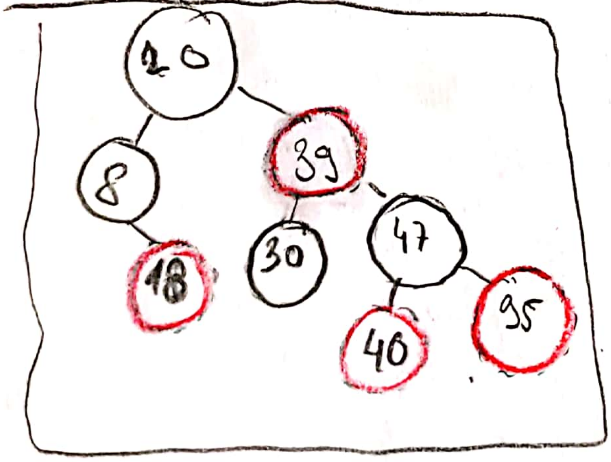


okay

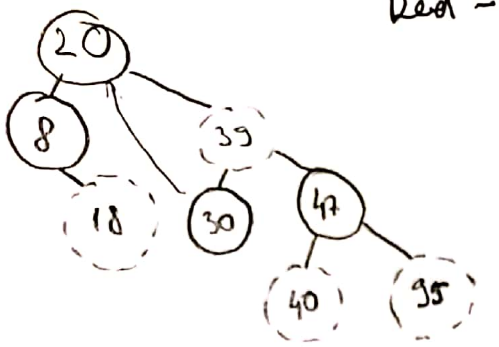
recheck

=) 39's parent black => exit it's okay ...

8) 95 (parent black => it's okay)



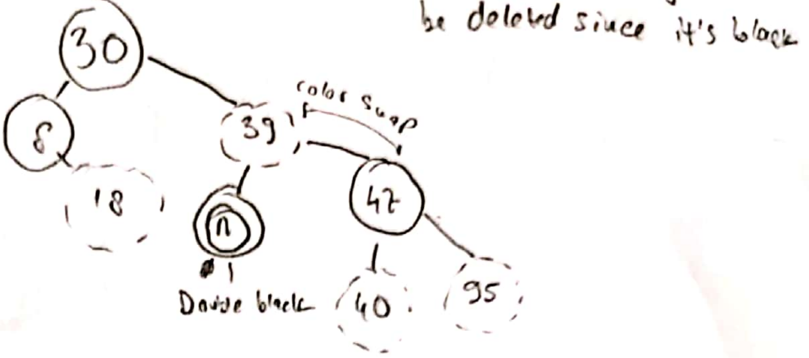
Red-Black remove   ◌ red   ○ black



1) 20 remove

=) BST removal (smallest of the right subtree).

=) swap 20 and 30 (30's node can't just be deleted since it's black



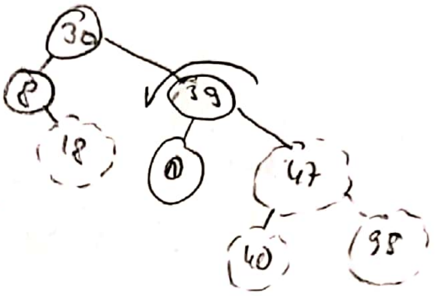
=) far child is red

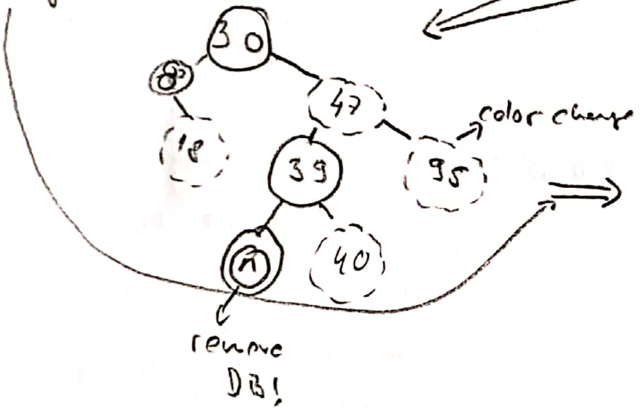
=) swap color of parent and sibling =)

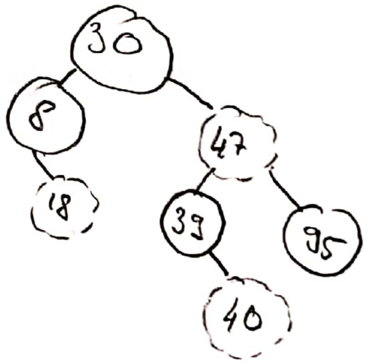
=) rotate parent in DB's direction

=) remove DB

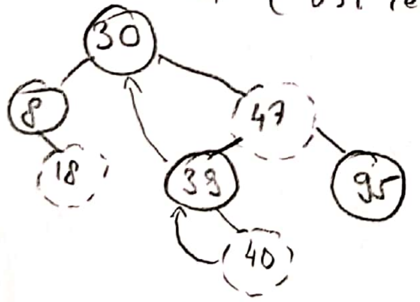
=) Change color of red child to B.





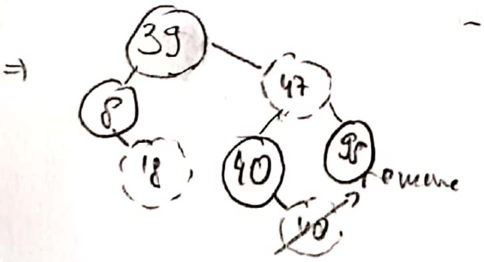


2) 30 remove (BST removal first)

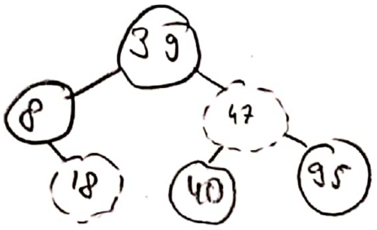


- Smallest from the right subtree (39 in 30
- 39's node 1 child =) just put that child instead of 39
- 40's node to be deleted
- Since 40 is a Red node =) no need for anything else

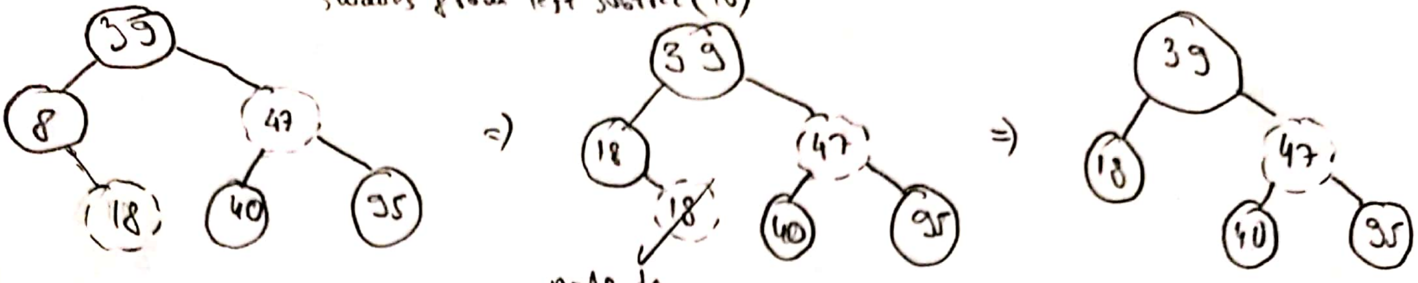
=)



=)

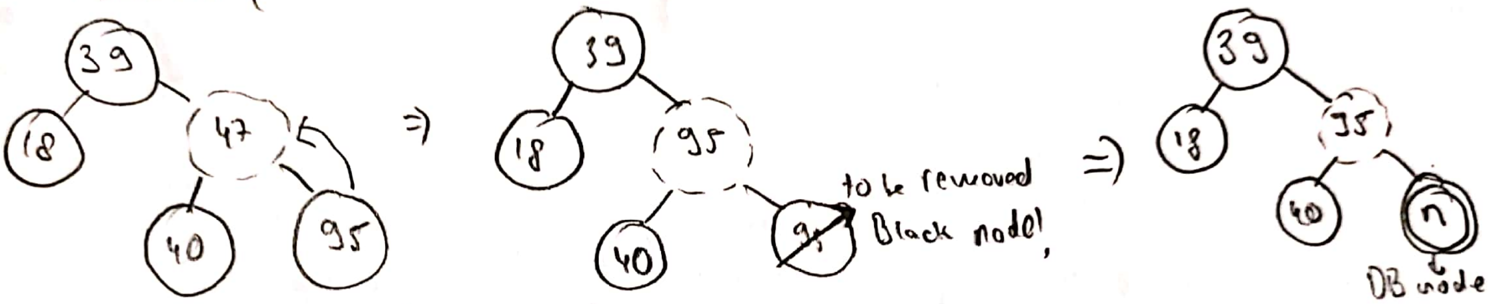


3) 8 removal (BST removal first)
smalles from left subtree(18)

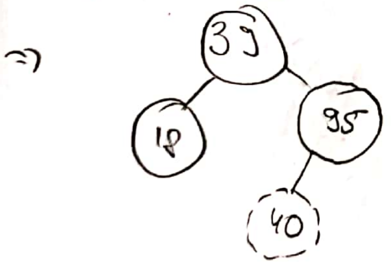


node to be removed is 18 → can safely be removed

4) 47 removal (

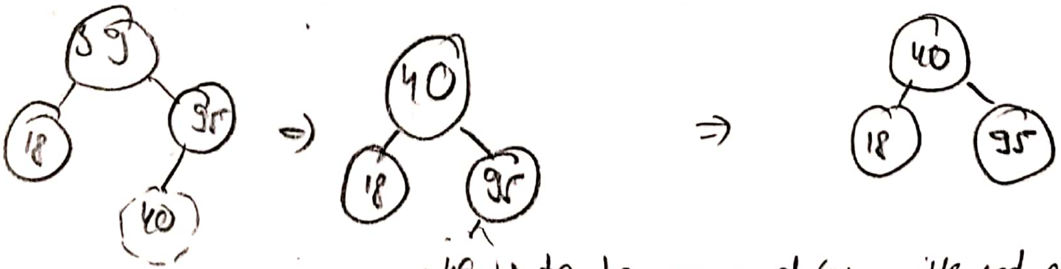


DB node's sibling and its children are black ⇒ change parent's colour and then change siblings colour



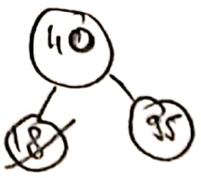
⇒ removed !

5) 39 remove (swap 40 and 39)

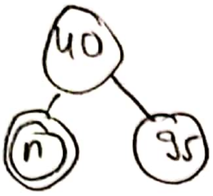


~~40~~ to be removed (since it's red can remeve it

6) 18 removal (child => remove)

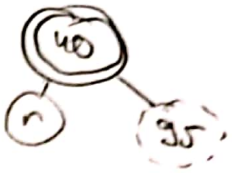


=>

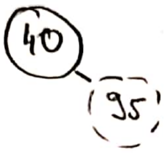


sibling and its children black
=> change parent to DB
and sibling to
red

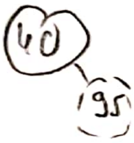
=>



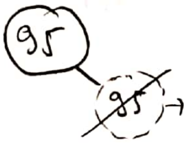
=> since DB is root
we can just
make it black =>



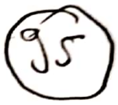
7) 40 (replace with smallest in right)



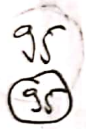
=>



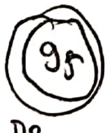
→ node to remove
is red => just remove it =>



8) 95 removal



=> no children => remove 95 =>



DB

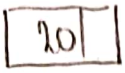
DB is root => just remove it

empty tree!

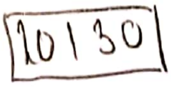
2-3 tree insertion

20, 30, 8, 47, 39, 18, 40, 95

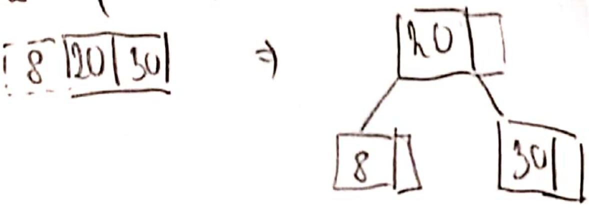
1) 20 (empty => just add)



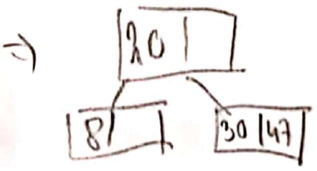
2) 30 (there is space so add next to 20)



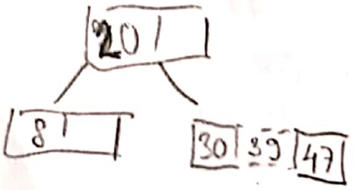
3) 8 (no space => split with middle el. (20))



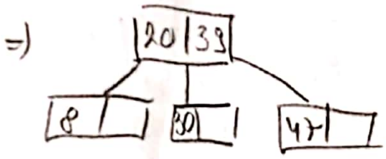
4) 47 (>20 => next to 30)



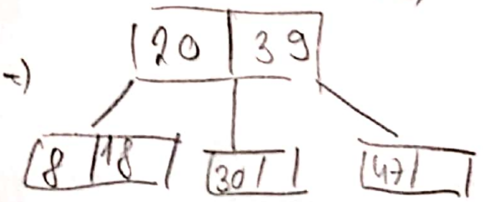
5) 39 (BST insert...)



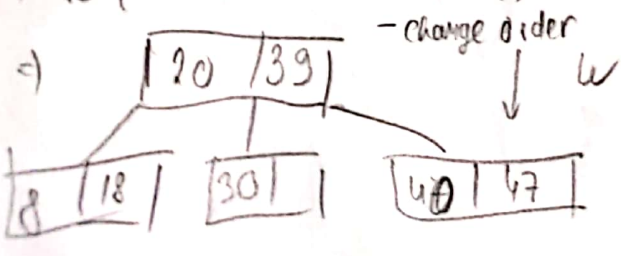
=> 3 keys => split
39 goes up



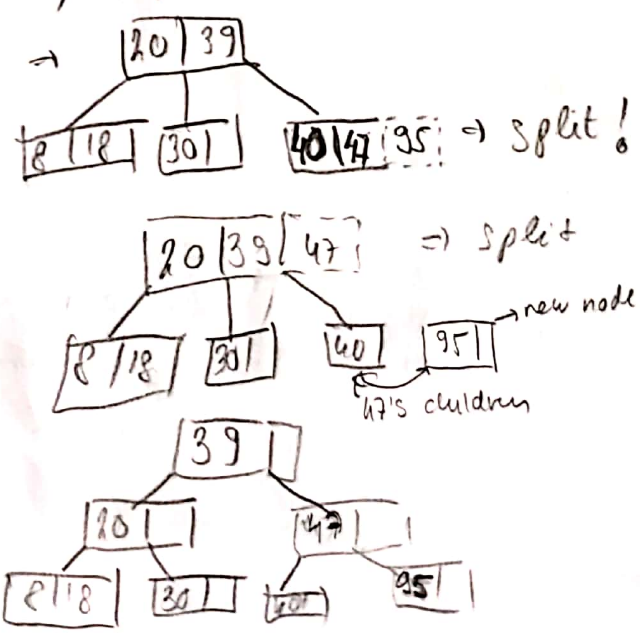
6) 18 (<20 => next to 8)



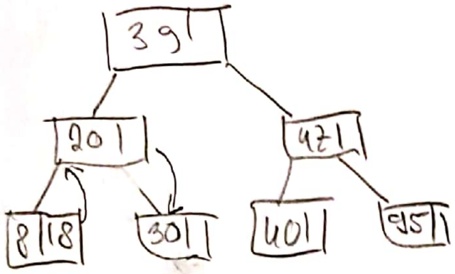
7) 40 (>39 => next to 47)



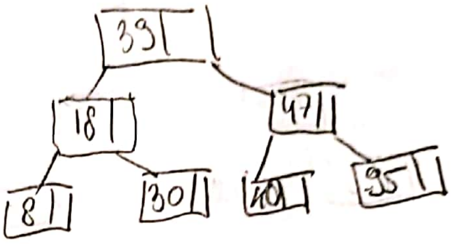
8) 95 (>39 => with 40,47



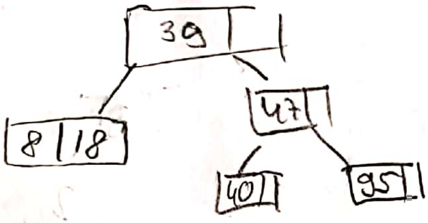
2-3 tree removal



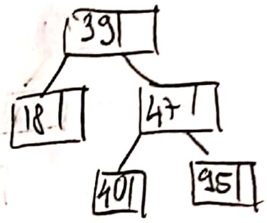
1) 20 => largest from left subtree => swap with 18 and remove 18



2) 30 => merge with its parent and its sibling and remove 18 than

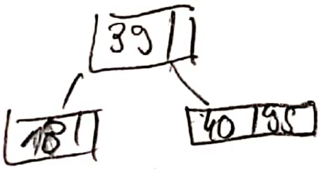


3) 8 (leaf => just remove...

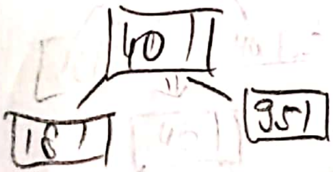


4) 47 (merge with children and than remove it...)

→ [40|47|95] => [40|47]



5) 39 => replace with smallest from right subtree(40)



6) 18

=> can't Borrow => merg with parent and remove it

[~~18~~|40|95] => [40|95]

7) 40 (leaf => just remove it)

=> [95|]

8) 95 (only element => just remove

[|] => empty tree

Skip list insertion

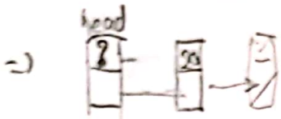
20, 30, 8, 47, 39, 18, 40, 95

cap = $2^1 - 1 = 1$

size = 1

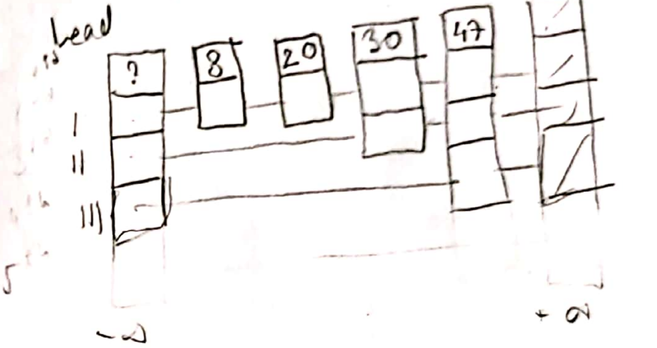
1) 20 empty list → insert at beginning
random level = 1 ...

cap = 1
lvl = 1



4) 47
comparing and reaches 30
⇒ moves to 30's node checks for smaller again (can't find
⇒ will be inserted after 30) links of 30 and head updated
Random leve → 3

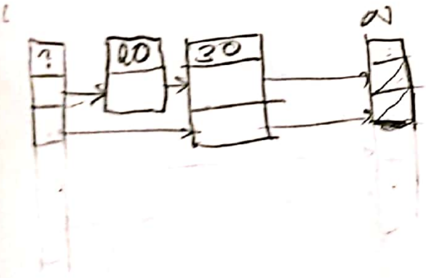
cap = 7
Lvl = 3



2) 30 (Checks from last level until it reaches 20, than 30 > 20
⇒ move
after that since it is at the tail it will just be inserted after 20 with leve 2 now (random

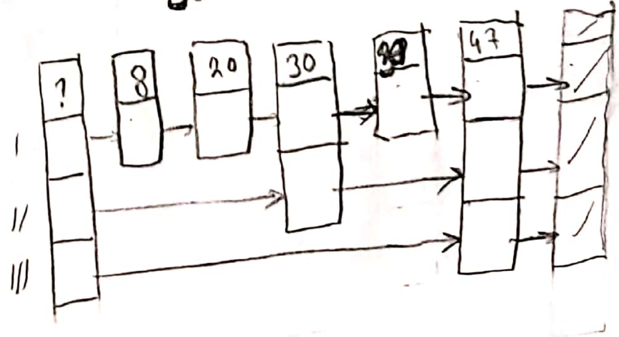
cap = 2

cap = 3

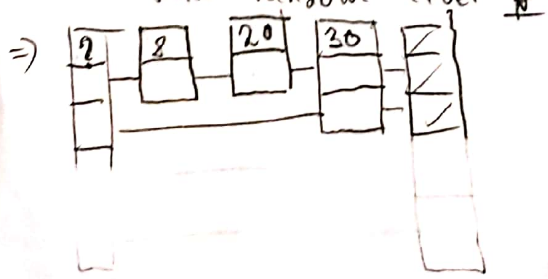


5) 39 – checks 4th and 3th level, but since both barger doesn't move, than on 2rd level 30 is smaller so moves to 30. Checks until the 1st level and inserts after 30th's node. (leve 2 again

5 < 7
lvl = 3



3) 8 ⇒ it would check with all links and since none are smaller it will be inserted at the beginning with random level 1



6) 18 Check's 4th, 3th, 2rd level
but all are larger

at 1rd 8 is less than 18
=> moves to 8

5<7
1<=3

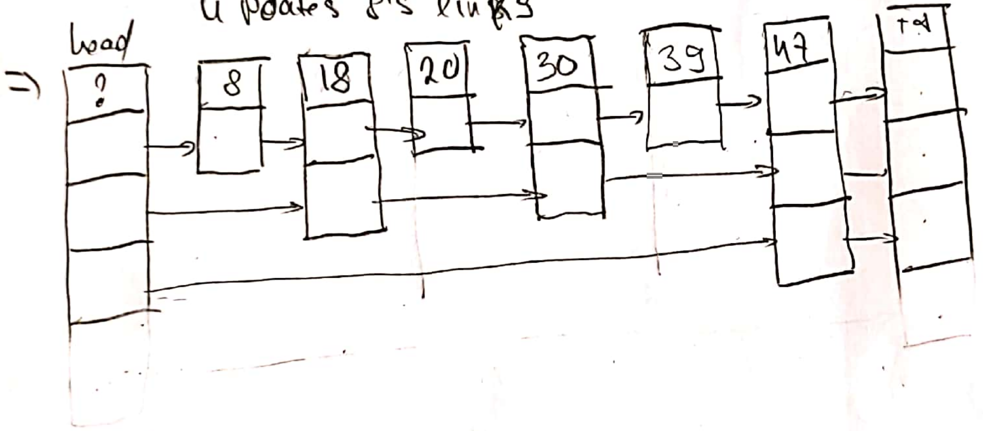
1

---

than checks only node
8 is pointing to
20 is greater than 18
and it's the last level
=> insert after 8
let random level be 2
updates 8's links

level
4
3
2
1

=>

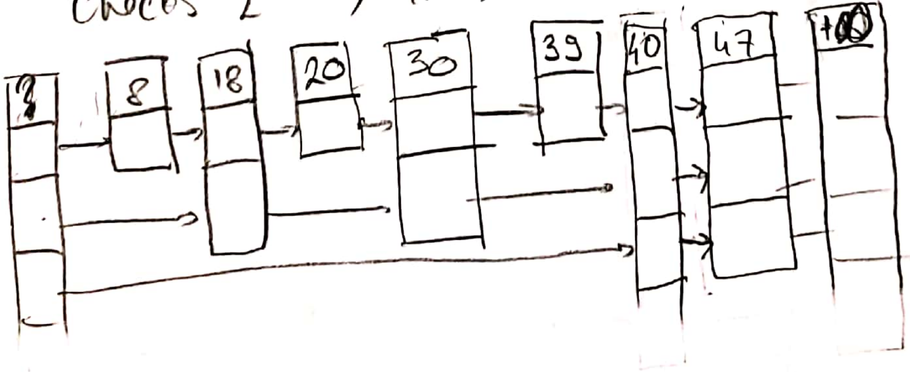


7) 40

5

at 5th and 4th level larger
3rd level 18 => Smaller => moves to 18
checks 3rd => 30 => smaller => moves to 30
Checks 3rd => 47 => stays
checks 2nd => 39 => moves to 39
Checks 2nd => 47 => inserts after 39 with level 3

=>



8) 95

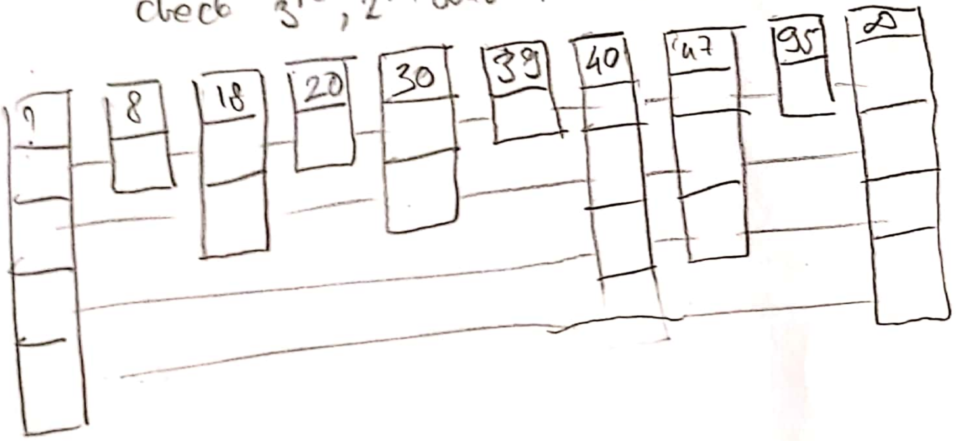
check 4th level => 40 => less than 95 => move to 40's node

check its 4th => bigger => stay

check 3rd => 47 => less => move to 47

check 3rd, 2nd and 1st level => all larger => insert after 47 and update links

8 > 7
cap = 15
lvl = 4



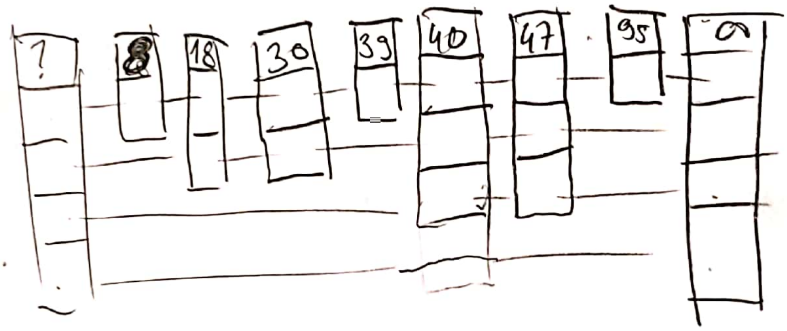
let level be 2

Skip-list removal

1) 20

find it!

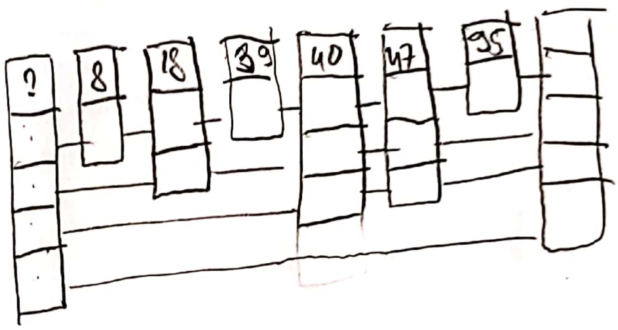
levels
4: 20 < ∞ => stay
(null)
3: 20 < 40 => stay
2: 20 > 18 => move to 18
2: 20 < 30 => stay
1: 20 = 20 => 18 points to 30
=> remove 20



2) 30

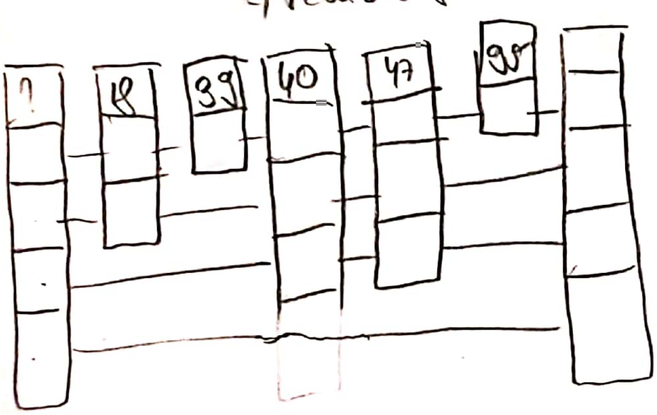
levels
4: 30 < ∞ => stay
3: 30 < 40 => stay
2: 30 > 18 => move to 18
2: 30 = 30 => 18's 2nd points to 40
18's 1st points to 39

=> remove



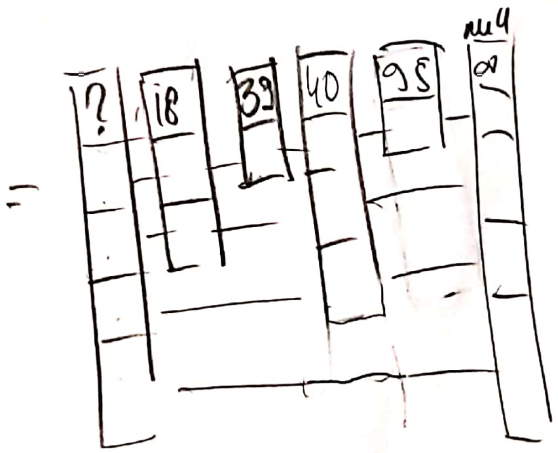
3) 8

4: 8 < ∞ => stay
3: 8 < 40 => stay
2: 8 < 18 => stay
1: 8=8 => head's 1st level points to 18 now
=> remove 8



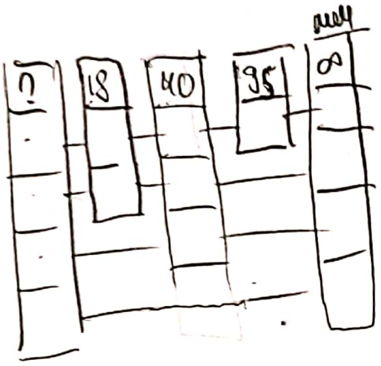
4) 47

4: 47 > ∞ => move to 40
null
4: 47 < ∞ => stay
or null
3: 47 = 47 => 40's 3rd pointing to
∞ or null
2nd also
1st to 95



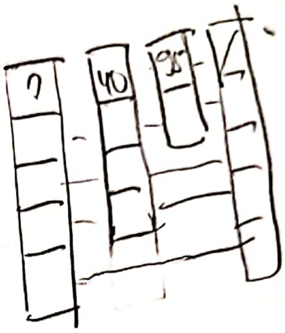
5) 39

4: 39 < ∞ ⇒ stay
3: 39 < 40 ⇒ stay
2: 39 > 18 ⇒ move to 18
2: 39 < 40 ⇒ stay
1: 39 = 39 ⇒ 18's 1st points to 40
remove 39



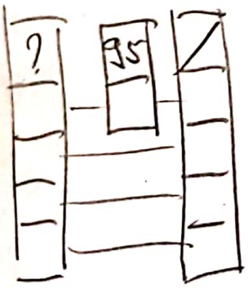
6) 18

4: 18 < ∞ ⇒ stay
3: 18 < 40 ⇒ stay
2: 18 = 18 ⇒ head's 2nd point to 40
1st -||-
remove 18



7) 40

4: 40 < ∞
3: 40 = 40
⇒ head's 4th points to ∞/null
3rd -||-
2nd -||-
1st points to 95



8) **95**

4: 95 < ∞ ⇒ stay
3: 95 < ∞ ⇒ stay
2: 95 < ∞ ⇒ Stay
1: 95 = 95 ⇒ remove and head points to null/∞
remove 95

⇒ empty list