PAGE No.
DATE / /

# Module 2. Advance Data Structures

## ✱ B-tree

→ Balanced m-way tree
→ It is generalization of BST in which a node can have more than one key and more than 2 children.
→ maintains sorted data.
→ All leaf node must be at same level.
→ B-tree of order [m] has following properties.
  → every node has max 'm' children.
  → min. children
    leaf → 0
    root → 2
    internal nodes → $\lceil \frac{m}{2} \rceil$ ← ceiling

    if m=5 $= \lceil \frac{5}{2} \rceil$

    $= \lceil 2.5 \rceil = 3$ if floor the 2

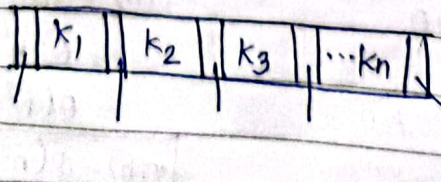
→ Every node has max (m-1) keys if (5-1) = 4
→ min keys :- root node → 1
    all other nodes → $\lceil \frac{m}{2} \rceil - 1$

    $\lceil \frac{5}{2} \rceil - 1$

    $= 3 - 1$

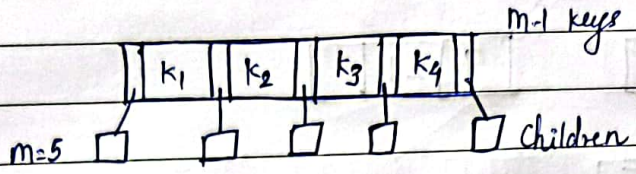
    $= 2$

node of B-tree



PAGE No.
DATE / /

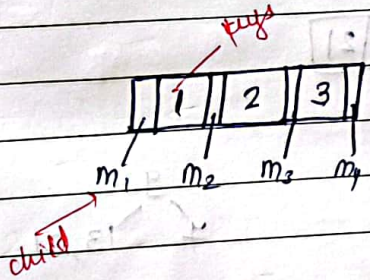
if m = 5
node can have max. 5 children




* Insertion → leaf node

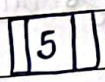
① Construct a B-tree of order 4 with foll. set of data.
5, 3, 21, 9, 1, 13, 12, 7, 10, 12, 4, 8

→ m = 4
keys = m-1
= 4-1
= 3

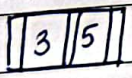



max child = 4 min child = 4-1 = 3
max key = 3 min keys = $\lceil \frac{m}{2} \rceil - 1$
$= \frac{4}{2} - 1$
$= 1$

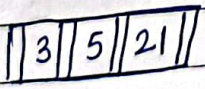
Insert 5



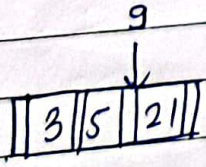

Insert 3

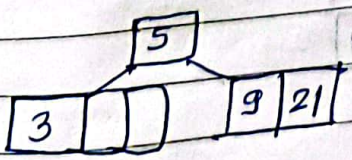



Insert 21



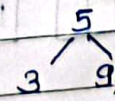

Insert 9



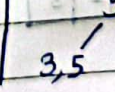

when order is even
either 5 or 9

3, 5, 9, 21




left biased

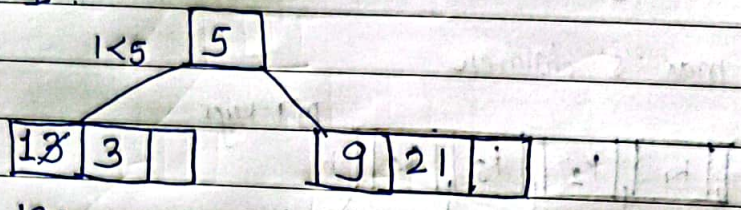
3, 5, 9, 21



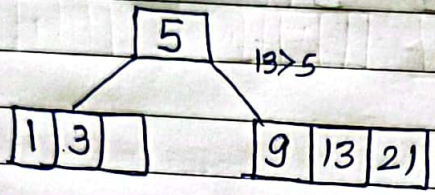

right biased

PAGE No.
DATE / /

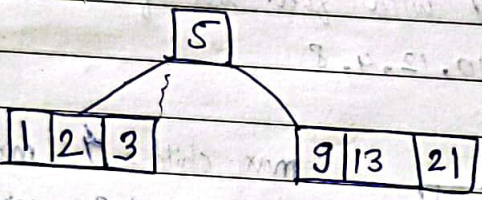
Insert 1:



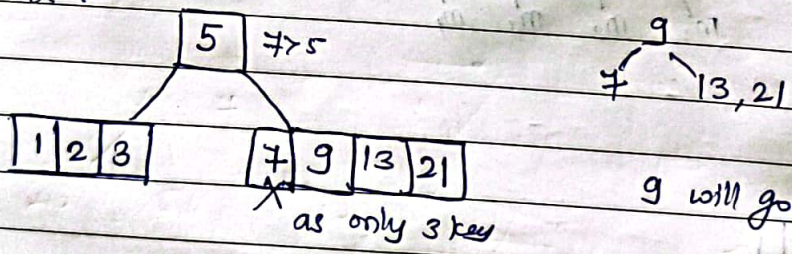

Insert 13:

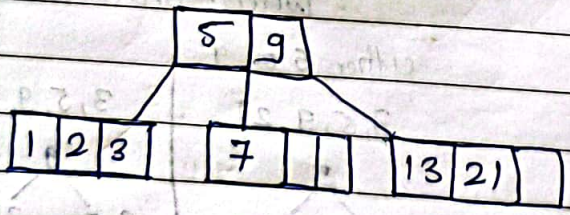



Insert 2

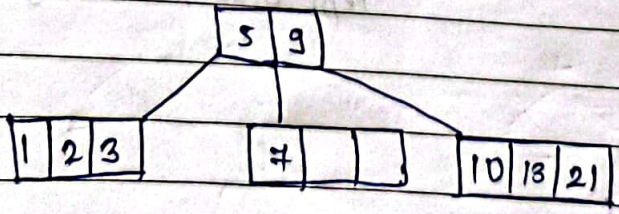



Insert: 7



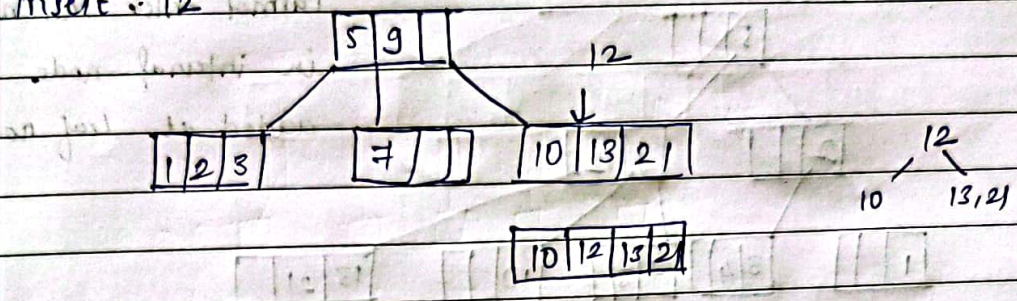



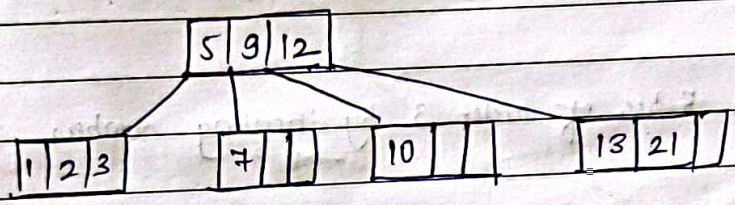

Insert: 10



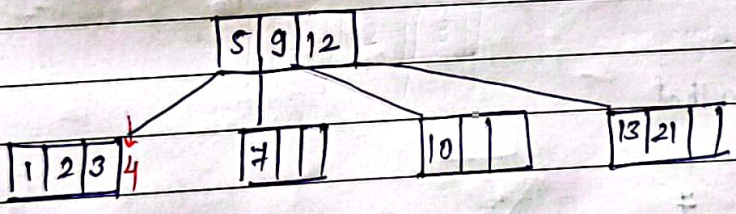
PAGE No.
DATE / /

Insert : 12

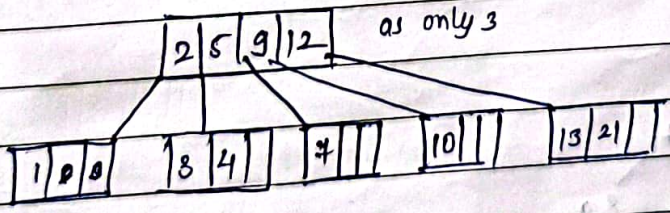


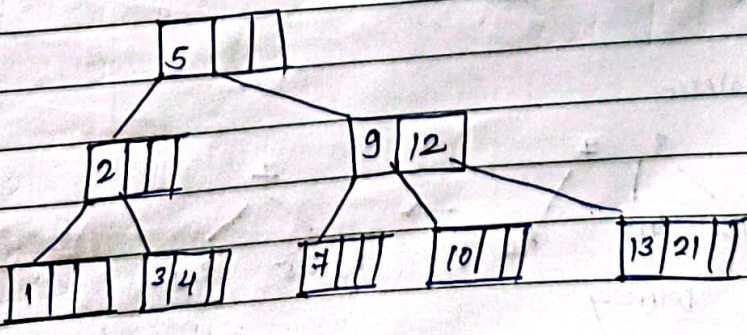




Insert : 4



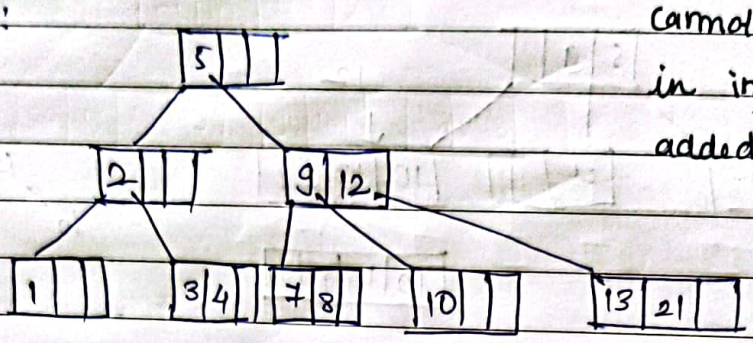

1 2 3,4 then 2 5 9,12



PAGE No.
DATE / /

Insert 8:

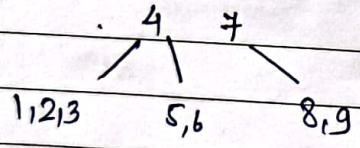



cannot enter the new data in internal node only added at leaf node

Q. Construct a B-tree of order 3 by inserting numbers from 1 to 10

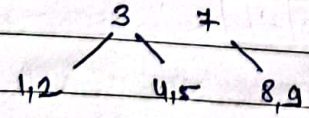
## * Deletion in B-tree

### ① Borrow method

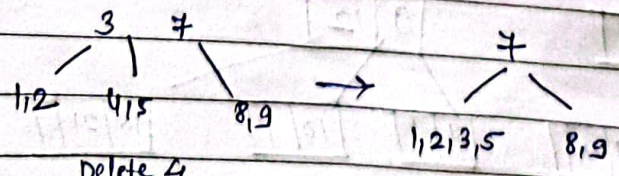



max child 4 min child 3
max key 3 min key 1

Delete 6

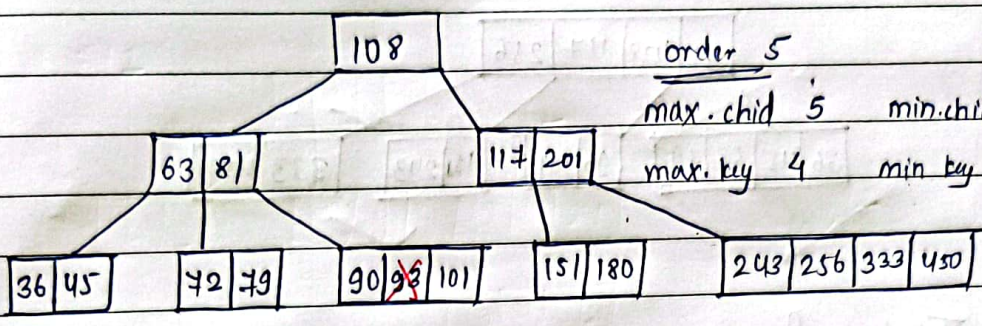



### ② Coalesce




Delete 4

PAGE No.
DATE / /



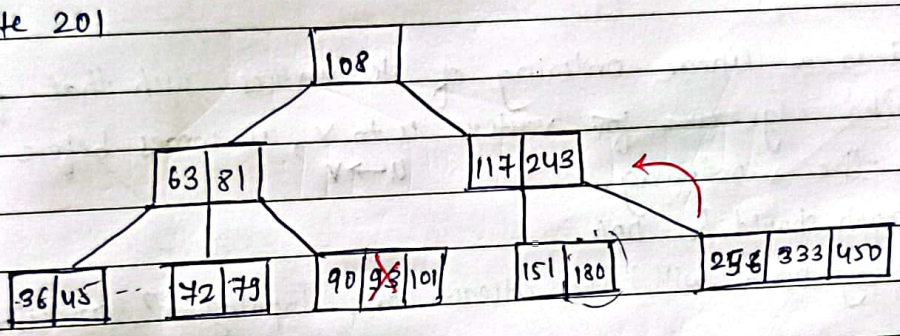

order 5

max. chid 5 min. chid 4

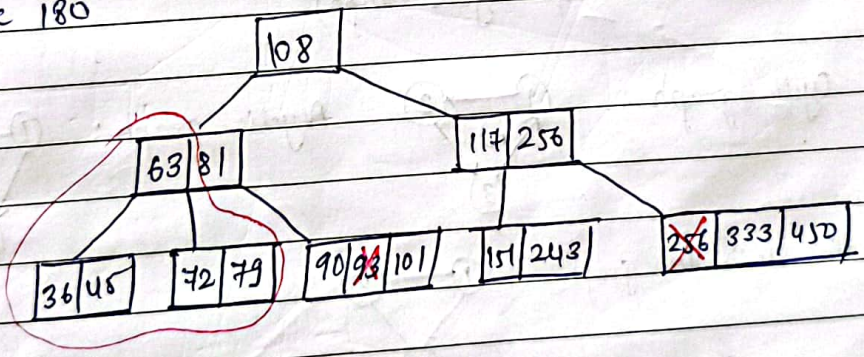
max. key 4 min key 2

Delete 93

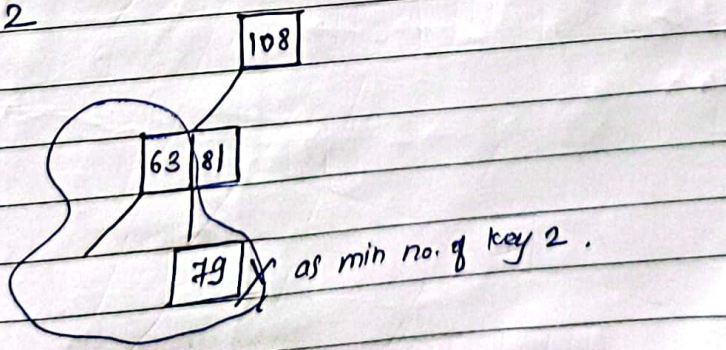
Delete 201

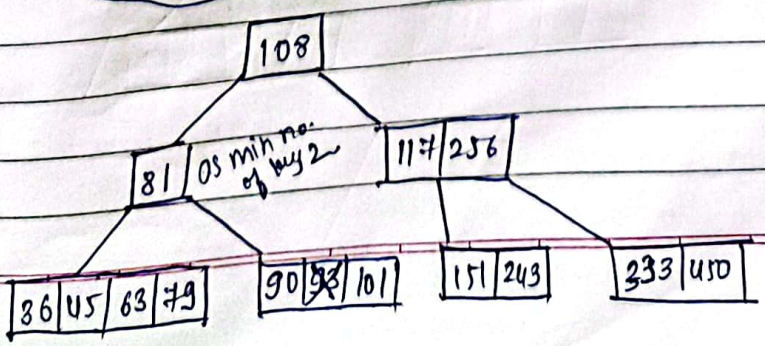


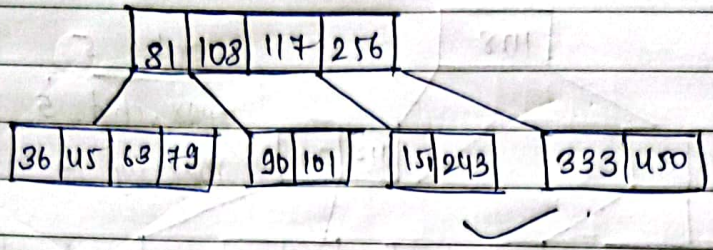

Delete 180




Delete 72



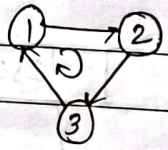



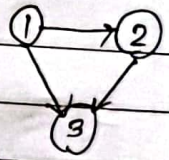


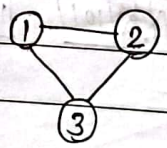

## * Topological sort.

- It is a linear ordering of its vertices such that for every directed edge uv for vertex u to v, u comes before vertex v in the ordering. $u \rightarrow v$
- Graph should be DAG
- Every DAG will have atleast one topological ordering.

### * DAG (Directed Acyclic Graph)

Cyclic graph : 

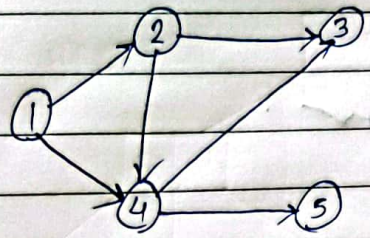

Acyclic graph : 


* Undirected graph: 


- Is directed ✓
- Any cycles ✗

PAGE No.
DATE / /

①



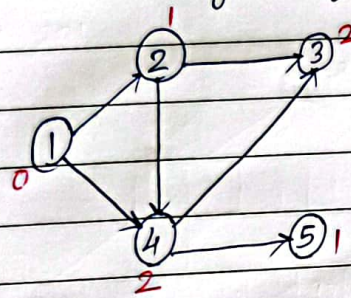

Is directed ✓ DAG.

Any cycles ✗

in-degree → incoming edges

out degree → outgoing edges

Step 1: Find in-degree of each vertex

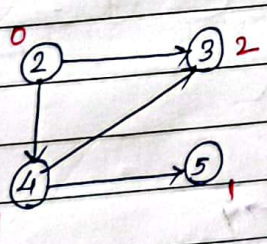



Delete 1 as in-degree 0

Step 2: Writing topological ordering with the vertex having in-degree 0.

| 1 |
|---|

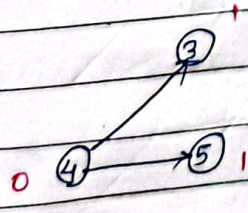
Step 3: Next step is to delete (1) and its edges.




Again calculate in-degree

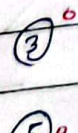
| 1 | 2 |
|---|---|

Delete 2 in-degree 0




| 1 | 2 | 4 |
|---|---|---|

Delete 4




∴ Two topological ordering

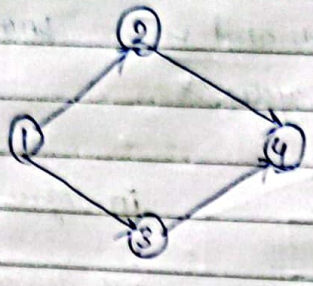
1 2 4 3 5 & 1 2 4 5 3

1 → 2 → 4 → 3 → 5

1 → 2 → 4 → 5 → 3

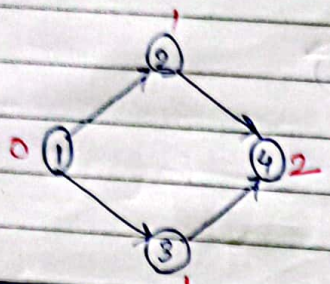
PAGE No.
DATE / /

(2)

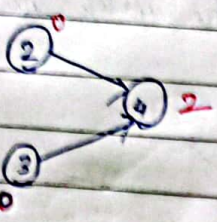


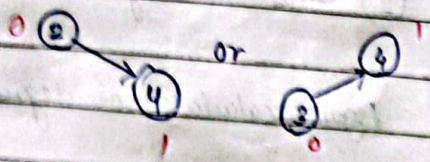

DAG ✓

→




| 1 | |
|---|---|




| 1 | 2 | 3 | 4 |
|---|---|---|---|

1 → 2 → 3 → 4

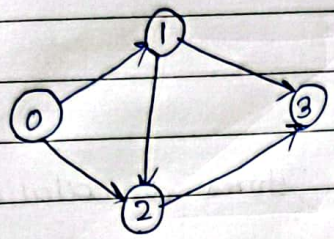
or

| 1 | 3 | 2 | 4 |
|---|---|---|---|

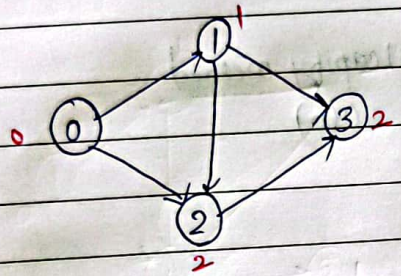
1 → 3 → 2 → 4

PAGE No.
DATE / /

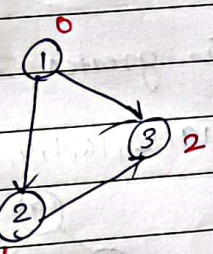
(3)



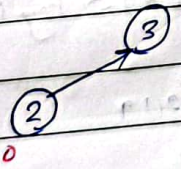

→ Calculate in-degree




| 0 | | | |
|---|---|---|---|



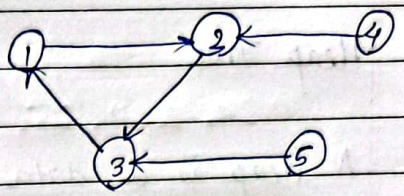

| 0 | 1 | | |
|---|---|---|---|



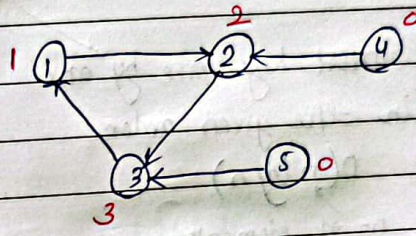

| 0 | 1 | 2 | 3 |
|---|---|---|---|

∴ 0 → 1 → 2 → 3

(4)



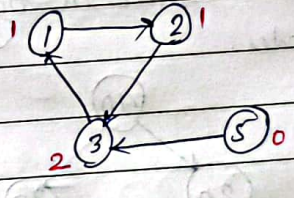

→ Calculate in-degree



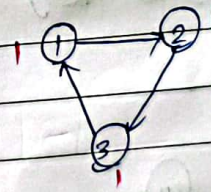

Select 4 or 5

selecting (4)

| 4 | | | |
|---|---|---|---|




| 4 | 5 | | |
|---|---|---|---|




Cycle present.

No vertex with in-degree 0 and is the cyclic graph.

Binary Heap

max heap  min heap

PAGE No.
DATE / /

## * Heap Tree

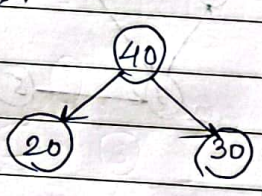
A heap is a data structure that stores a collection of objects.

Heap Tree construction

- Insert key one by one in the given order — $O(n \log n)$ (n = no. of elements)
- Heapify method — $O(n)$

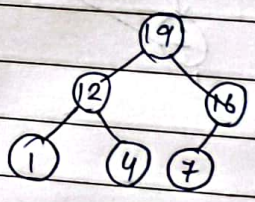
### * max Heap

- If value stored in any node is greater than or equal to its child nodes.



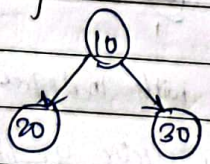

- Root element is larger.

19, 12, 16, 1, 4, 7

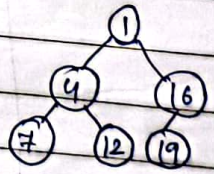



key (child) ≤ key (parent)

### * min Heap




1, 4, 16, 7, 12, 19




key (parent) ≤ key (child)

PAGE No.
DATE / /

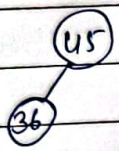
## * Insertion

Q. Build max Heap 45, 36, 54, 27, 63, 72, 61, 18
Also draw memory representation of the heap

Insert 45

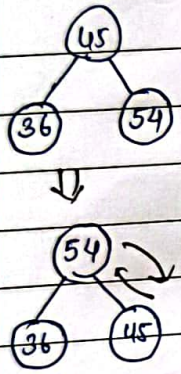

Insert 36

check nodes

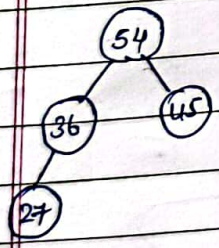



Insert 54

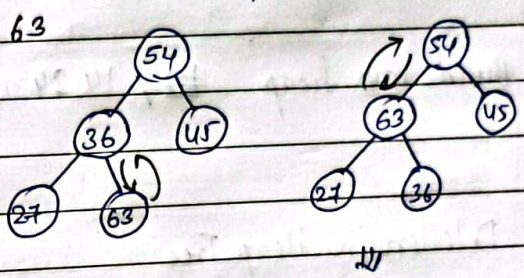
45 < 54
hence swap

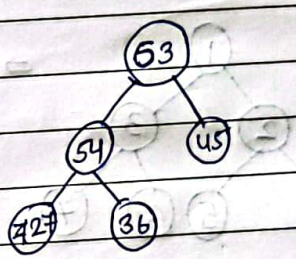



Insert 27

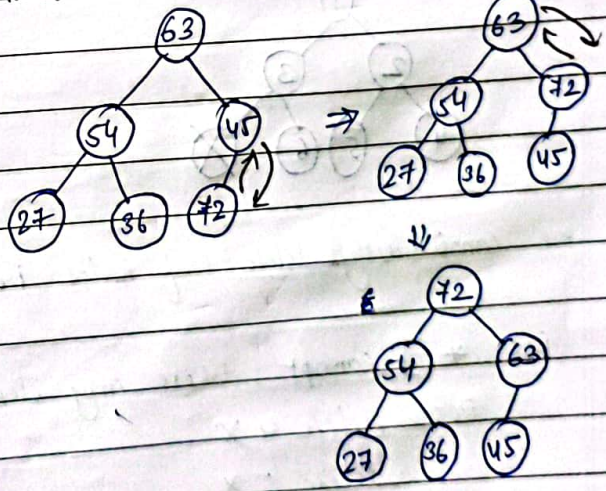



Insert 63

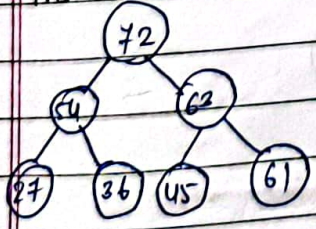


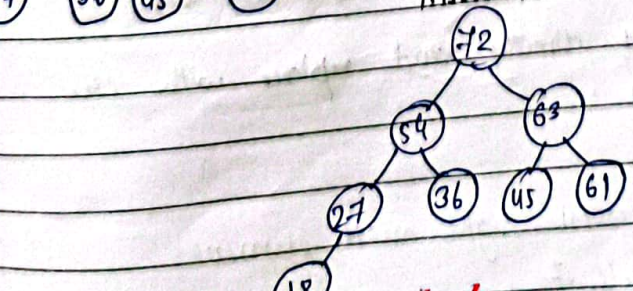

Insert 72




Insert 61




Insert 18




| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 8 | 72 | 54 | 63 | 27 | 36 | 45 | 61 | 18 |

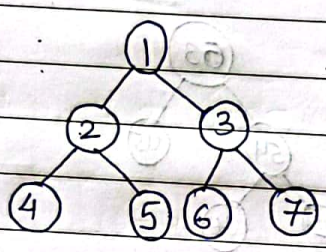
Heap[1] Heap[2] ... Heap[8]

PAGE No.
DATE / /

Q. Build max heap for the following data.
14, 24, 12, 11, 25, 8, 35.

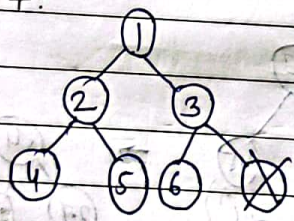
Q. Build min heap ~~#~~, 14, 24, 12, 11, 25, 8, 35

## * Deletion in Heap Tree




⇒ min heap

Delete 7:

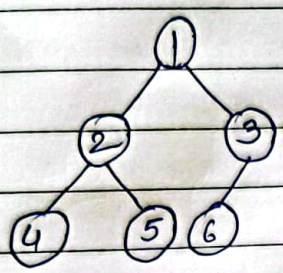



We cannot directly delete leaf nodes i.e deletion of last node.

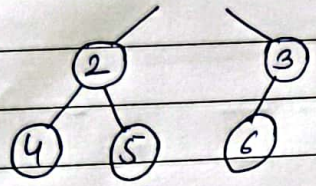
i.e * I cannot delete any element directly
i.e. delete 4 ✗
delete 5 ✗

* We cannot delete root directly and replace with the lowest node from right.

* We can delete the lowest right most element.
i.e. we can delete 7

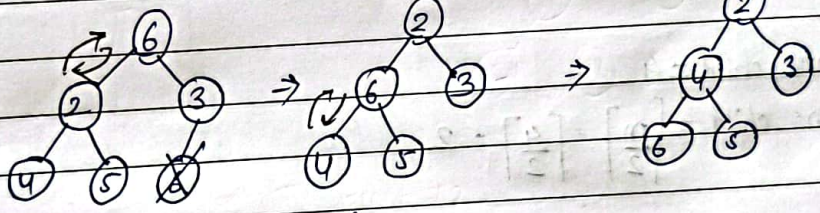



Delete 1 :




lowest level right most element and make it root.

↓




not mean heap hence swap ~~the~~ child nodes.

PAGE No.
DATE / /

* B+ tree.

- It grows towards the root like B-tree.
- The data is stored only in leaf node.
- Leaf connected with each other.

- B-tree
  leaf as well as internal nodes.

Q. 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42

m=4.

max. child = 4

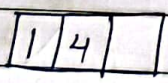
min. child = $\lceil \frac{m}{2} \rceil = \lceil \frac{4}{2} \rceil = 2$

max. key = (m-1) = (4-1) = 3

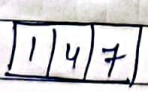
min. key = $\lceil \frac{m}{2} \rceil - 1 = 2 - 1 = 1$

→ Inset 1

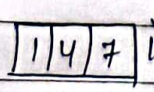

Inset 4

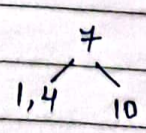



Inset 7

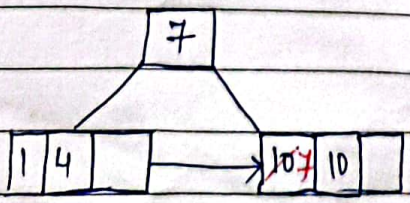



Inset 10

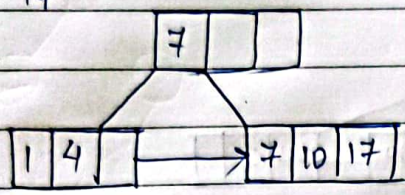


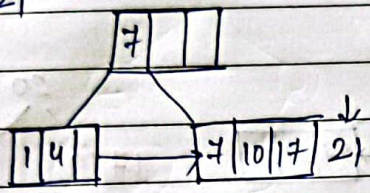

Right biased.



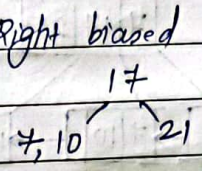
Insert 17

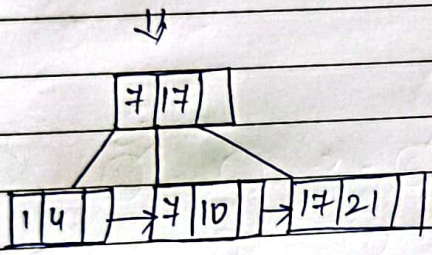



Insert 21

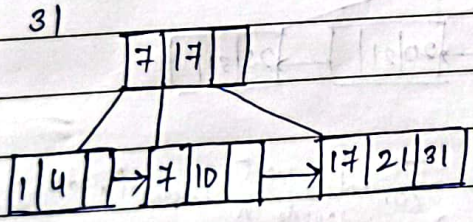



Right biased

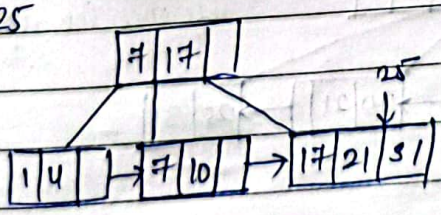


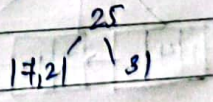



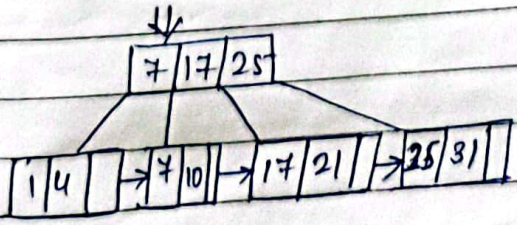

Insert 31




Insert 25

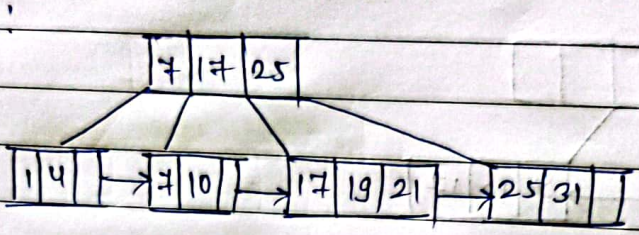


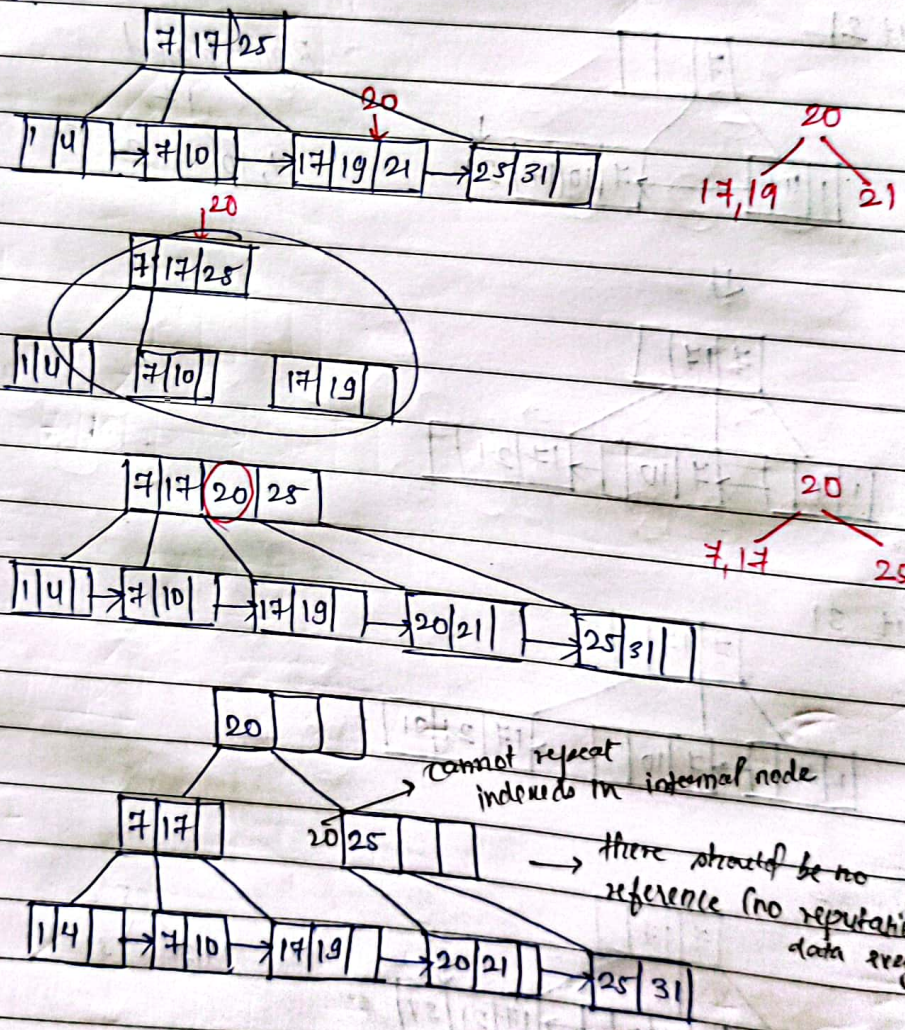



PAGE No.
DATE / /

Insert 19:

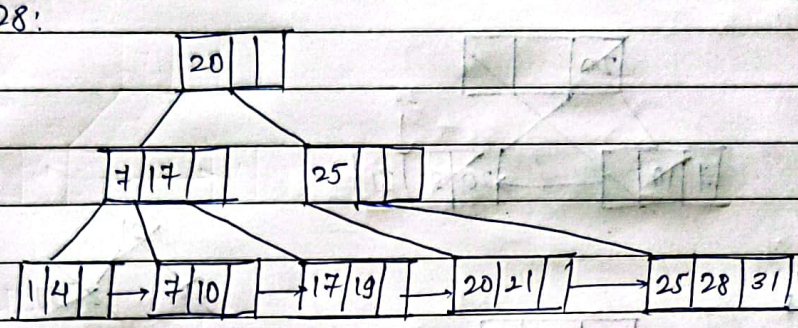



Insert 20:



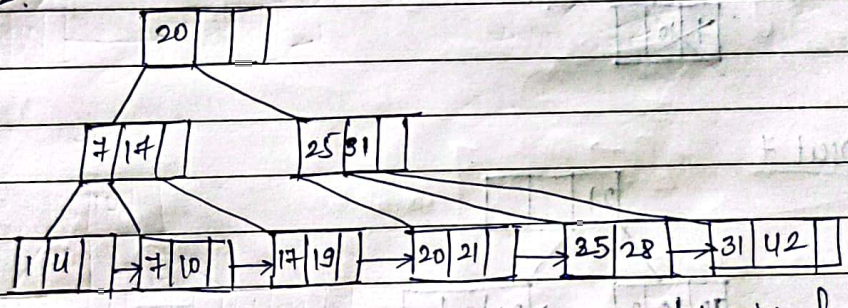
PAGE No.
DATE / /

Insert 28:




25,28 31 42

Insert 42:




↑ original full data is present!

Q. Construct B+ tree 21, 31, 20, 10, 7, 25, 42

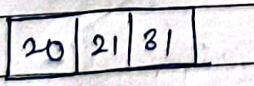
order of 4

i.e. m=4

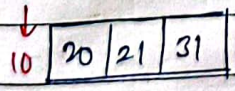
max. child = 4 max. keys = 4-1 = 3

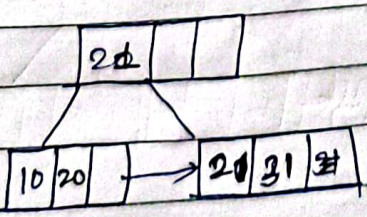
min. child = 2 min. keys = 1

Insert 21,31,20

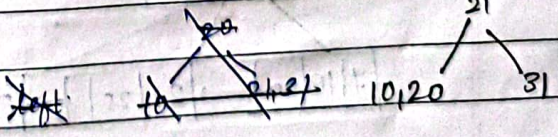


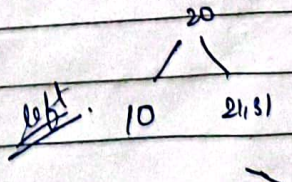

Insert 10




right biased.

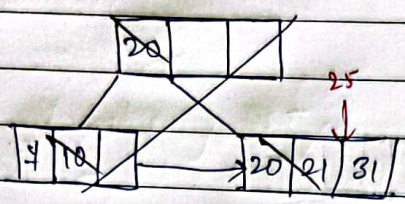


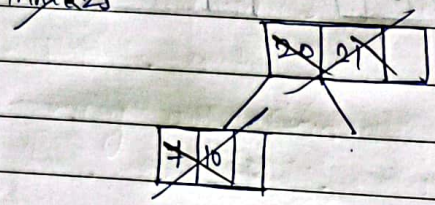

=>

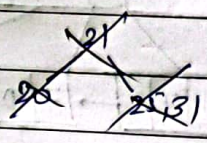
PAGE No.
DATE / /

Insert 7

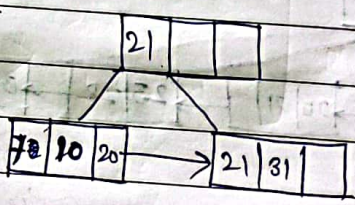



Insert 25

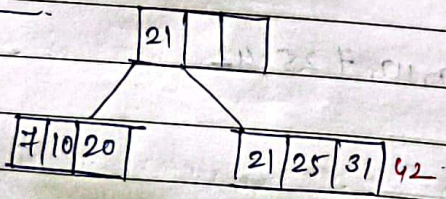


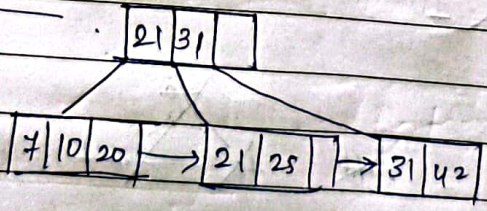

Insert 7

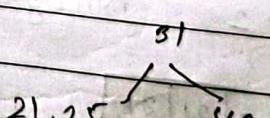



Insert 25




Insert 42



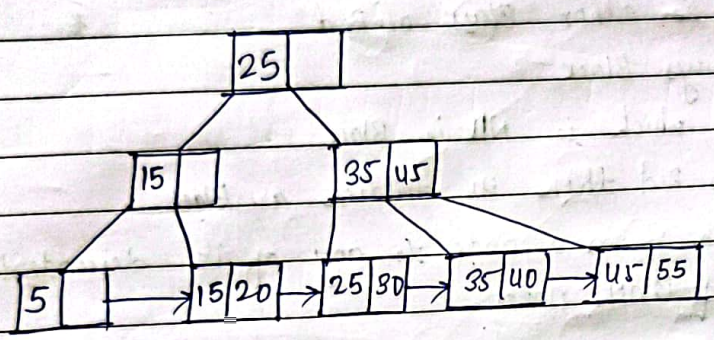



PAGE No.
DATE / /

## Deletion in B+ tree.

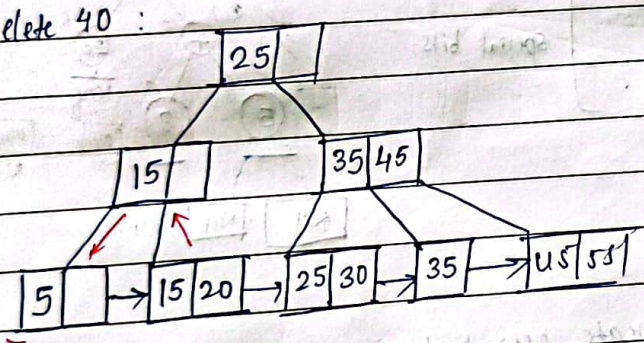
max child = 3 min child = 2

40, 5, 45, 35, 25, 55 order m = 3 max key = 2 min. key = 1



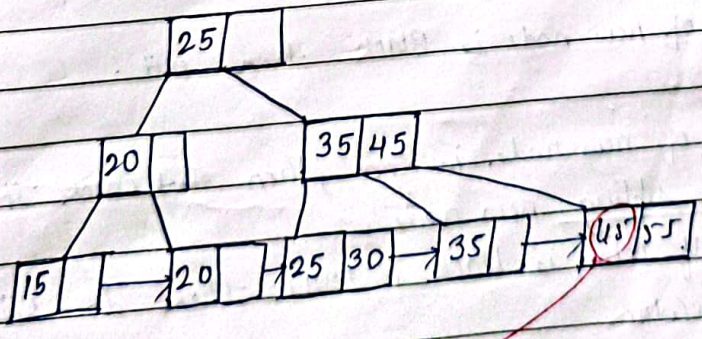

Delete 40 :

We can directly delete 40 and min. key is 1.




Delete 5

If minimum key 1 is present then it takes from siblings. (borrow)



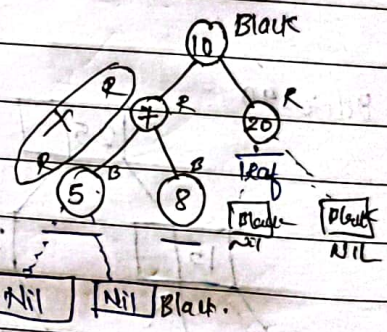

Delete 45

Directly delete 45

# Red-Black Tree

- It is a self-balancing BST
- Every node is either Black or Red.
- Root is always Black
- Every leaf which is NIL is Black.
- If node is Red then its children are Black.
- Every path from a node to any of its descendent NIL node has same no. of Black nodes.

0 → Black (zero)
1 → Red.
} special bits.




## Rules:

1) If tree is empty, create new node as root node with color Black
2) If tree is non-empty, create new node as leaf node with color Red
3) If parent of new node is Black then exit.
4) If parent of new node is Red, then check the color of parent's sibling of new node (uncle of new node)
   a) If color is black or NULL then do suitable rotation & recolor.
   b) If color is Red then recolor and also check if parent of new node is not root node then recolor it and recheck.

AVL trees are subset of red-black tree

PAGE No.
DATE / /

## Properties:

root = black

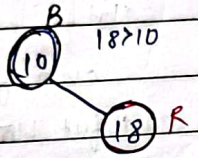
No two adjacent red nodes (constrain only on red node)

Count no. of black nodes in each path.

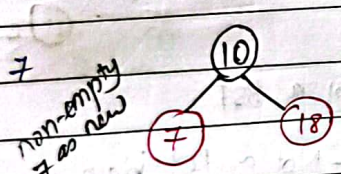
Q. 10, 18, 7, 15, 16, 30, 25, 40.

→ Insert 10

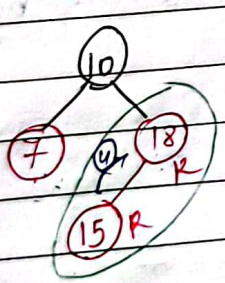

Insert 18




Insert 7




Insert 15




new node in red. (non-empty)

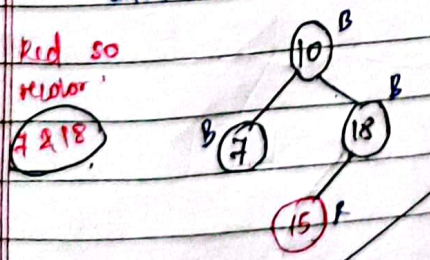
violating the property of No two adjacent red nodes (R-R)

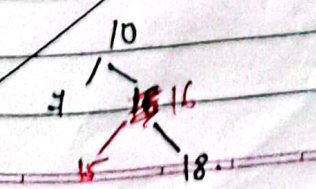
Ctr red.

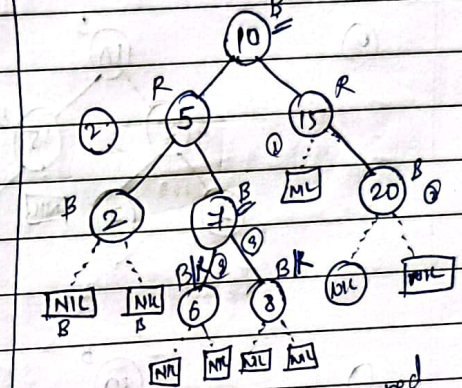
4th Rule 15 parent 18 and 7 is the uncle of 15

7 & 18 are siblings of Root node.

Red so recolor 7 & 18

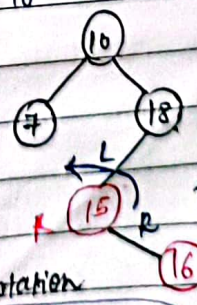



Not a red black as

Insert 16



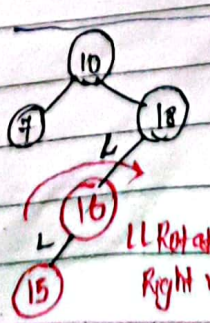

4.a. NIL Black.

4.a. new node red.

R-R conflict

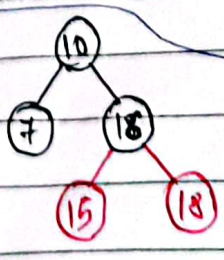
LR rotation

4th rule:



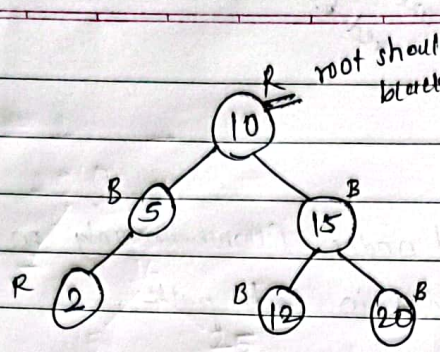

LL Rotation

Right rotation.



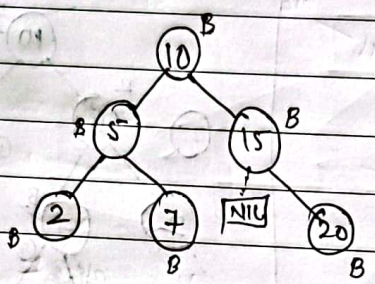
PAGE No.
DATE / /

(1)




not a red-black tree.

(2)

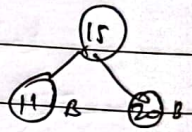



(2) Every node is either black or red.

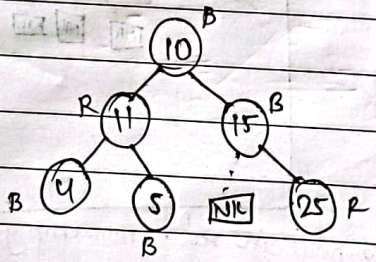
Not a red-black tree.

even if we don't have red.

Perfect binary tree. if



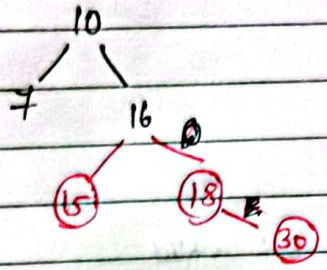

(3)



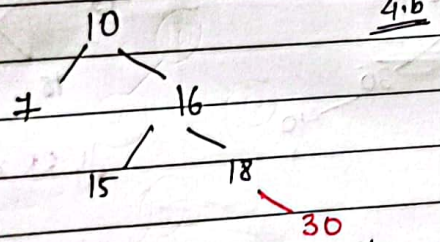

- Not a BST
- Not a Red-black tree.

PAGE No.
DATE / /

Insert 30

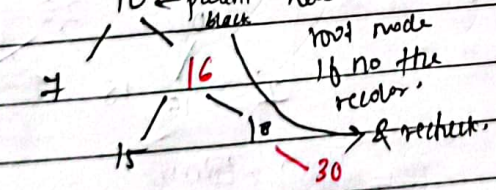



Red - Red conflict.
↳ check parent
check parents sibling → Red.
∴ Re-colour.
parent & sibling will recolor



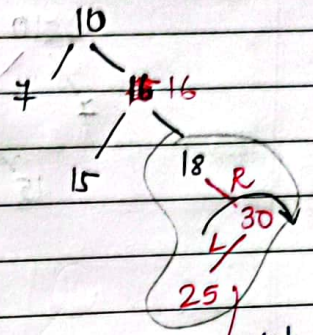

4.b.

and also check parent's parent of 10 ↔ parent (black) new node is root node If no then recolor & recheck.




RCB tree.

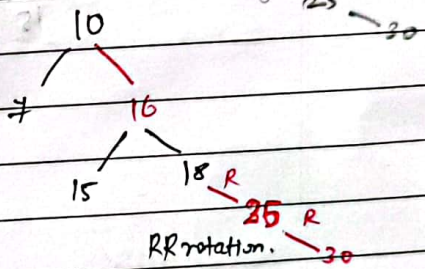
Insert 25



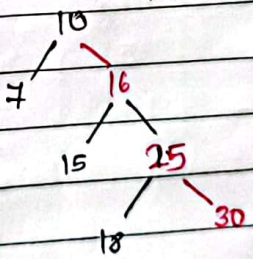

R-R conflict.
NULL - black. 4.a
4.b.

RL rotation.

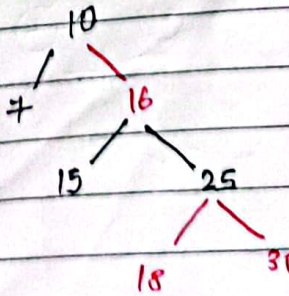
Right




RR rotation.

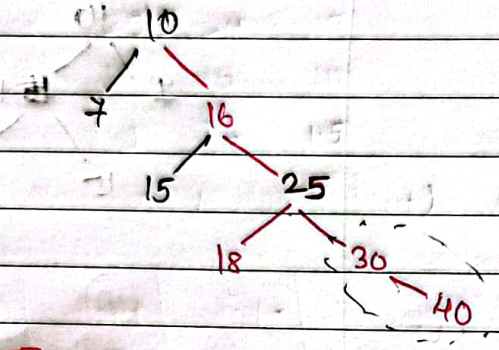



Now recolour.



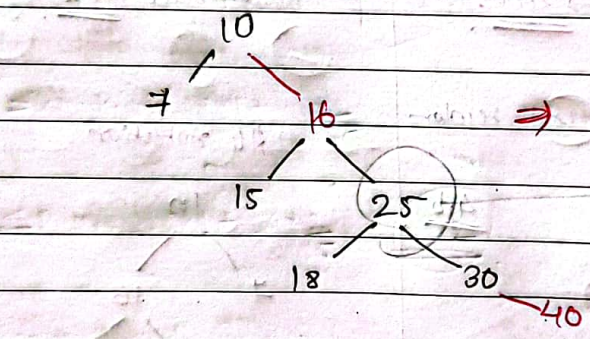
PAGE No.
DATE / /

Insert 40.



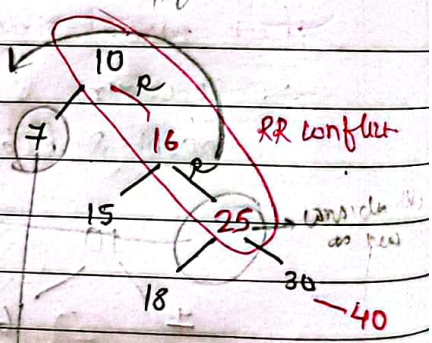

R-R conflict.

Just recolor.




⇒

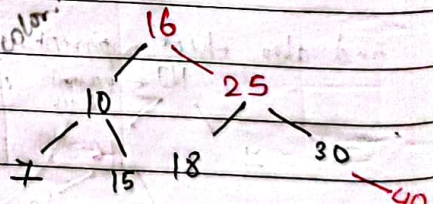
left rotation




RR conflict.

RR rotation.

Rotation & recolor.




Now recolour.