MODULE 2

TREE

B-Tree

* B-Tree is a specialized M-way tree created by Rudolf Bayer & Ed Creight that is widely used for disk Access
* A B-tree of order m can have a maximum m-1 keys & m-pointers to its Subtree.
* It may contains a large number of key values and pointers to its subtrees.
* Storing a large number of keys in a single node keeps the height of the tree relatively small

Properties of B-trees

* Every node in the B-tree have almost M-children
* Every node in the B-tree Except the root node and leaf Node has atleast minimum m/2 children
* Root Node has atleast two children if it is not a terminal leaf Node.
* All leaf Node are at the same level.
* B-tree is designed to store data, insertion, deletion operations performed in logarithmic amortized time.

Scanned by CamScanner

Insertion Operations

(1) Search the B-tree to find the leaf node where the new Key should be inserted

(2) If the leaf Node is not full, that it Contains less than (m-1) key values, then insert New Element in the node keeping the Nodes Elements Ordered

(3) If the leaf node is full that is the leaf node already contains (m-1) key values then

→ Insert the new Value in order into Existing set of keys

→ Split the node at its Median into two nodes.

→ Push the median Element upto its parent's node. If the parent's node is already full then split the parent's node by using the following some steps.

Scanned by CamScanner

Question :- Create a B-tree of order 5 by inserting the following Elements.

3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25 & 19

Order = 5

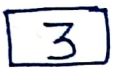
m = 5

m-1 keys

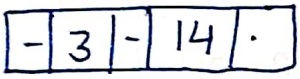
5-1 = 4 keys

5 m pointer.

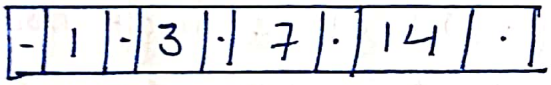
Start pt (3)



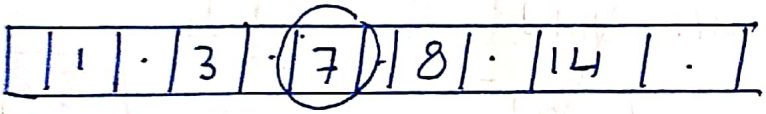
Insert (14)



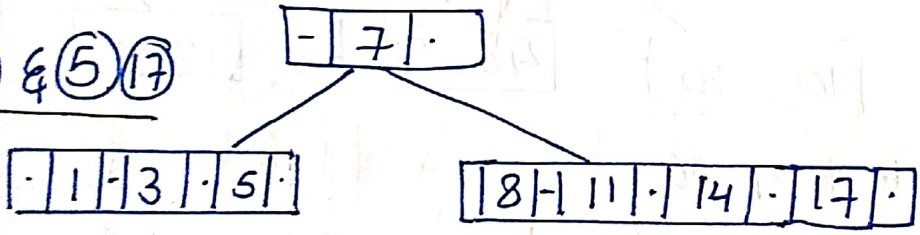
insert (7 & 1)



Insert (8)

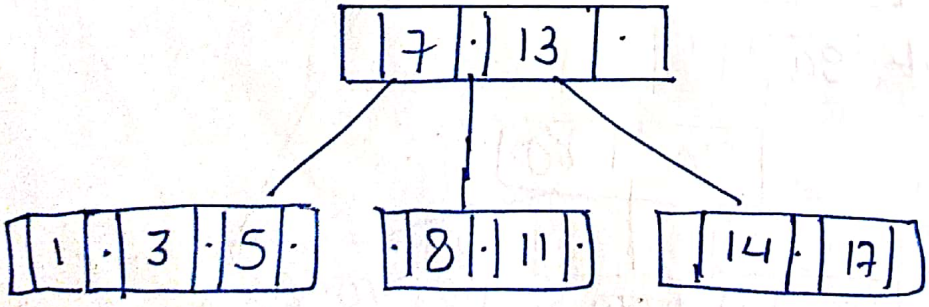


Insert (11) & (5) (17)



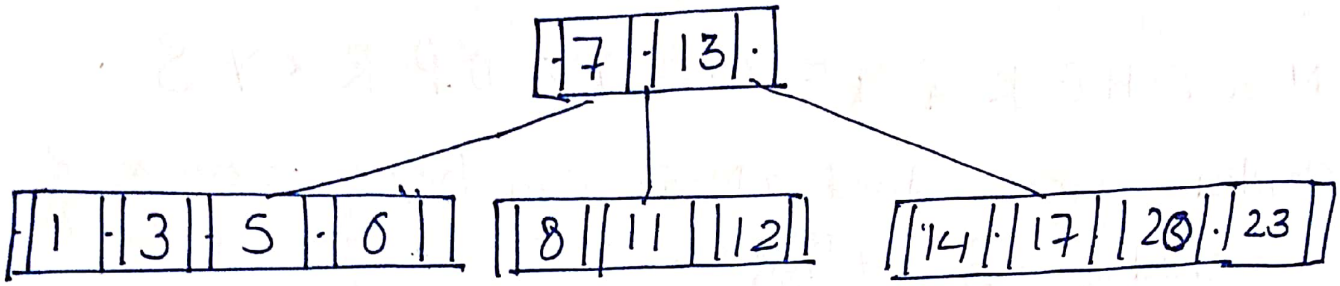
8 11 (13) 14 17

Insert (13)

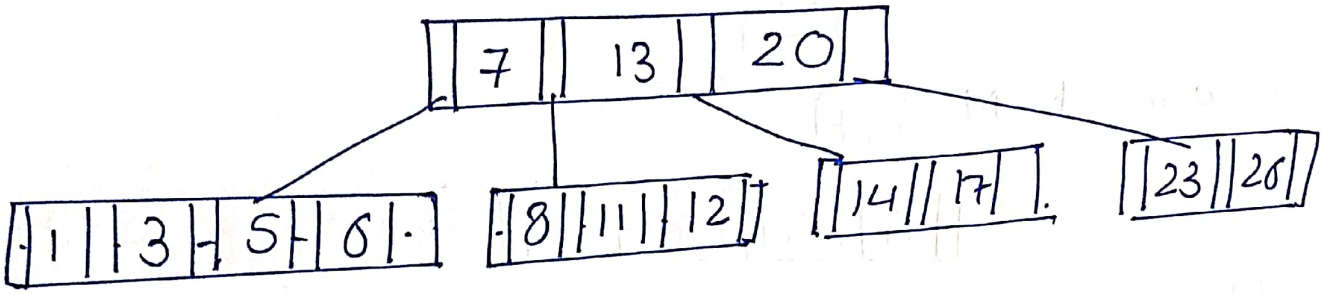


Scanned by CamScanner

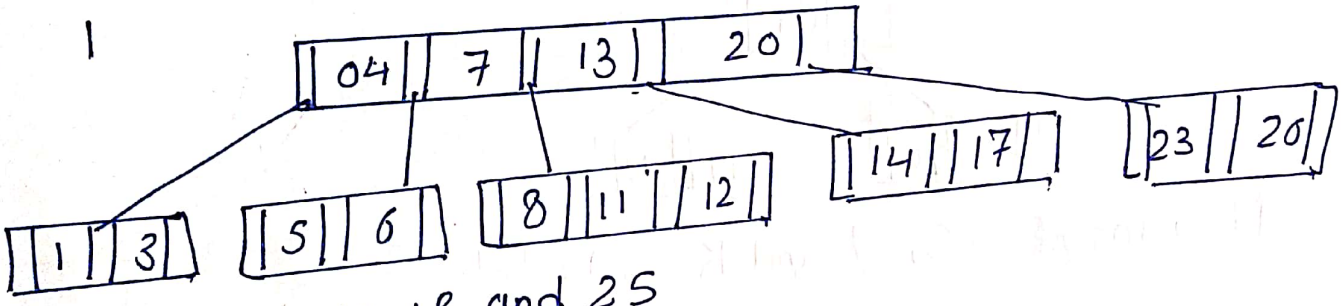
Now insert 6, 23 & 12 and 20



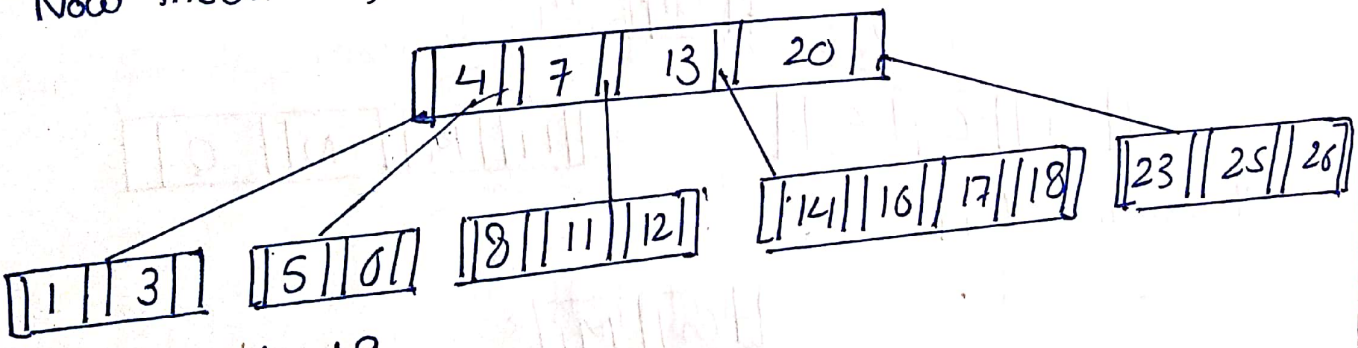
Now when we insert 26 then it will split:-



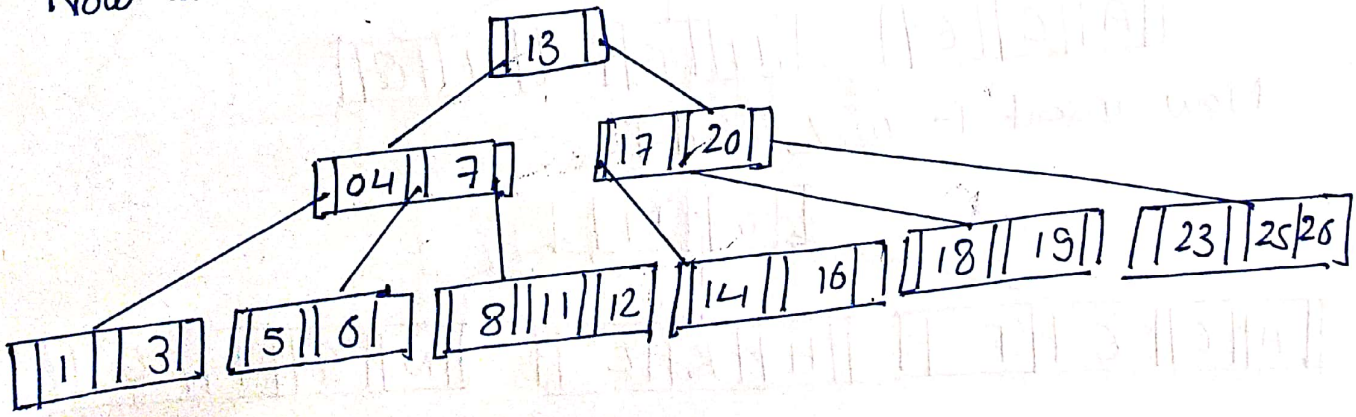
Now insert 4



Now insert 16, 18 and 25



Now insert 19



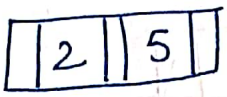
Scanned by CamScanner

Construct B tree to insert the following key Elements. (Order of tree is 3)

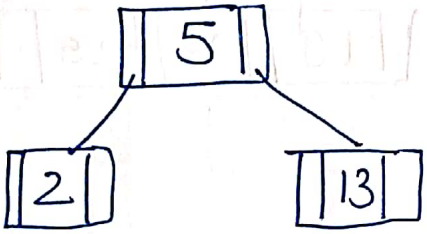
5, 2, 13, 3, 45, 72, 4, 6, 9, 22

First insert 5

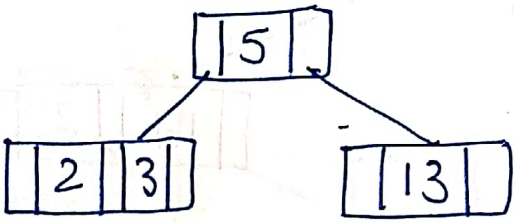
Insert 2



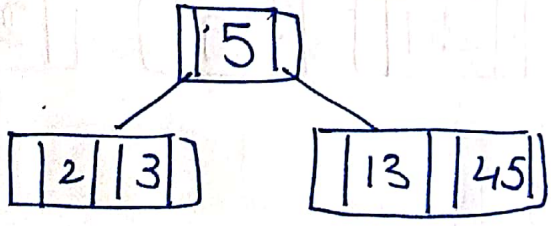
Now insert (13)



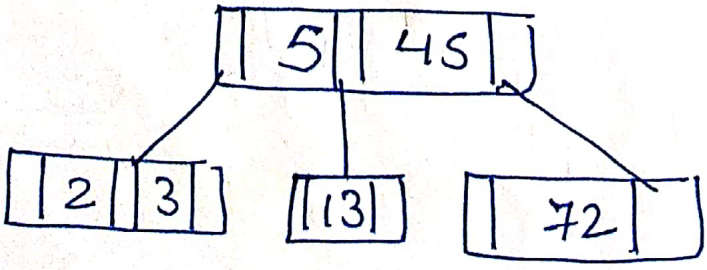
Now insert 3



Now insert 45

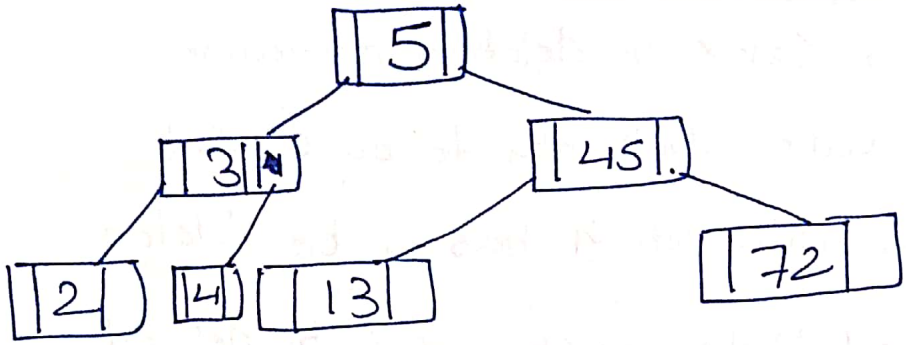


Now insert 72

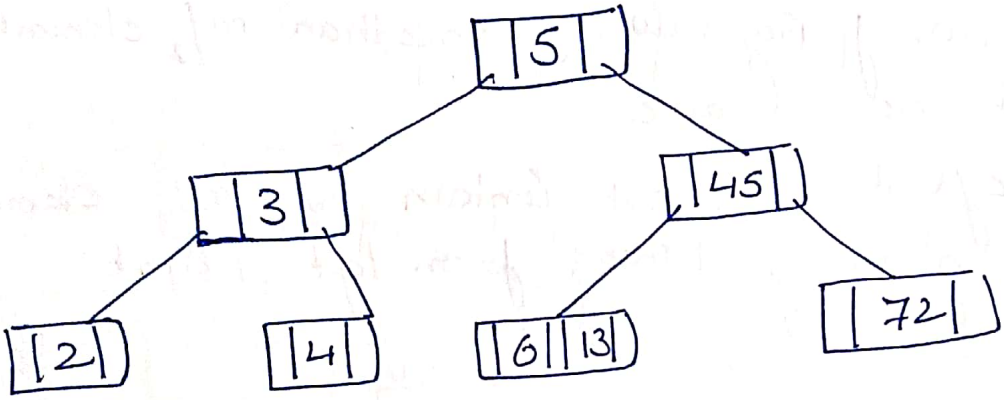


Scanned by CamScanner

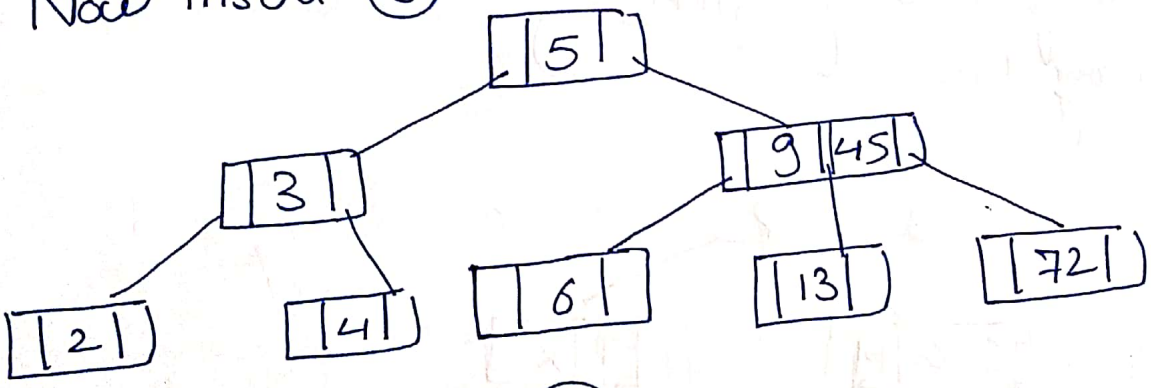
Now insert (4)



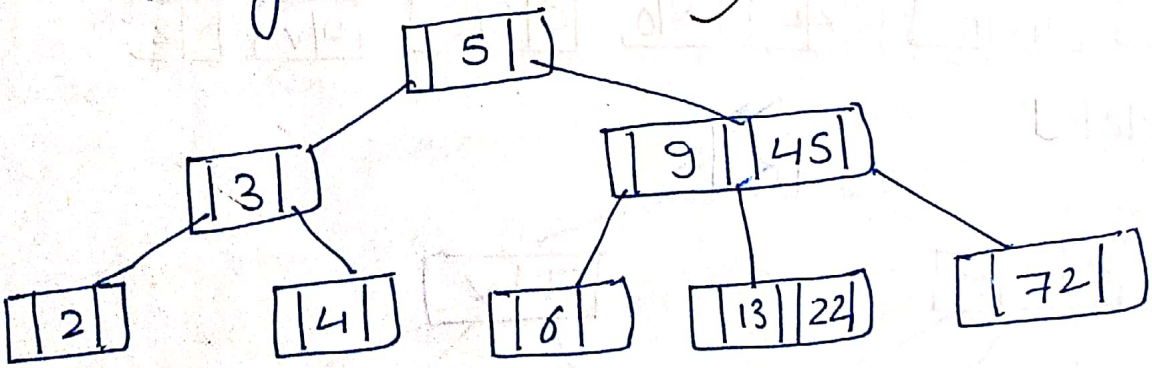
Now insert (6)



Now insert (9)



Now finally insert (22)



Scanned by CamScanner

Deletion Operation

There are two Cases in deletion operation :-

(*) find a leaf Node which has to be deleted

(*) An internal node which has to be deleted.

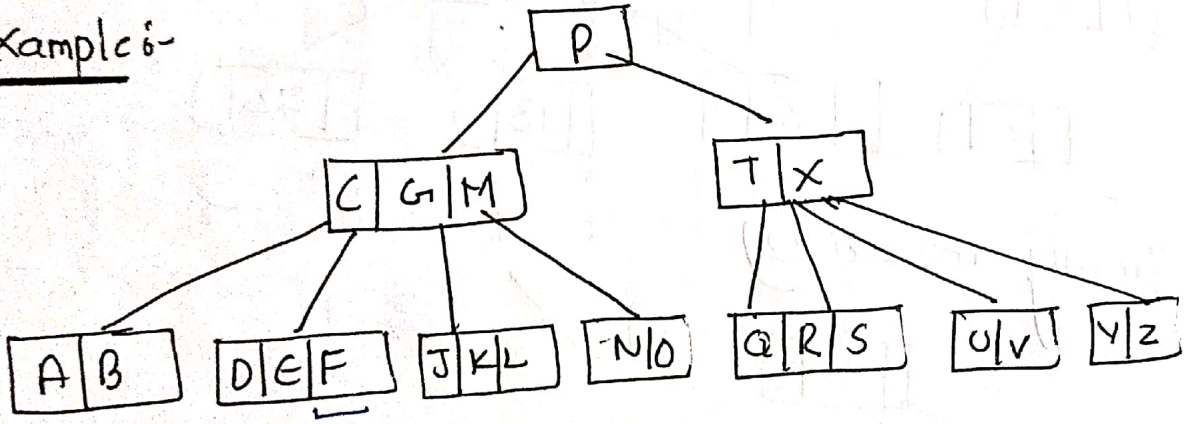
(1) locate the leaf Node which has to be deleted.

(2) If the leaf Node which has to Contain More than the Minimum no. of Key value (More than $m/2$ elements) then deleted the value.

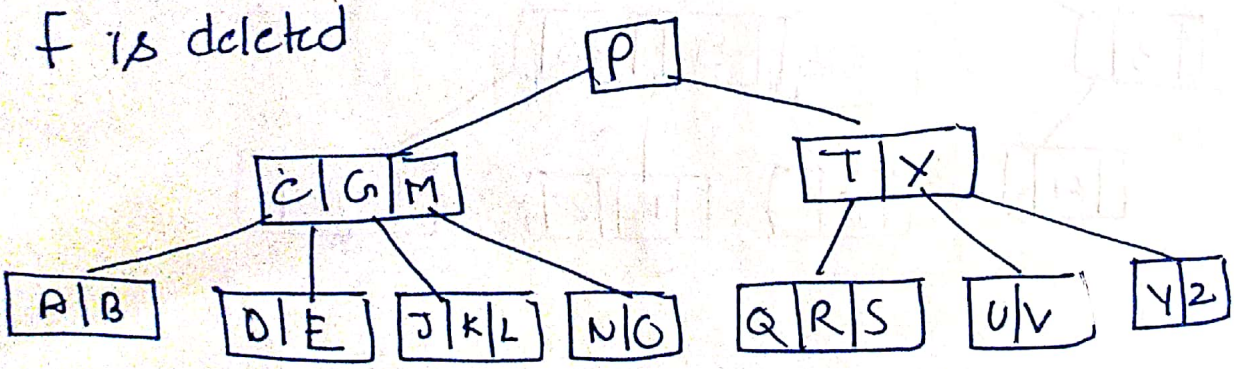
(3) Else if the leaf Node does not Contain Even $m/2$ elements then fill the node by taking from left & Right Sibling.

Max no. of node in Binary tree $(2^{n+1}-1)$

Example :-

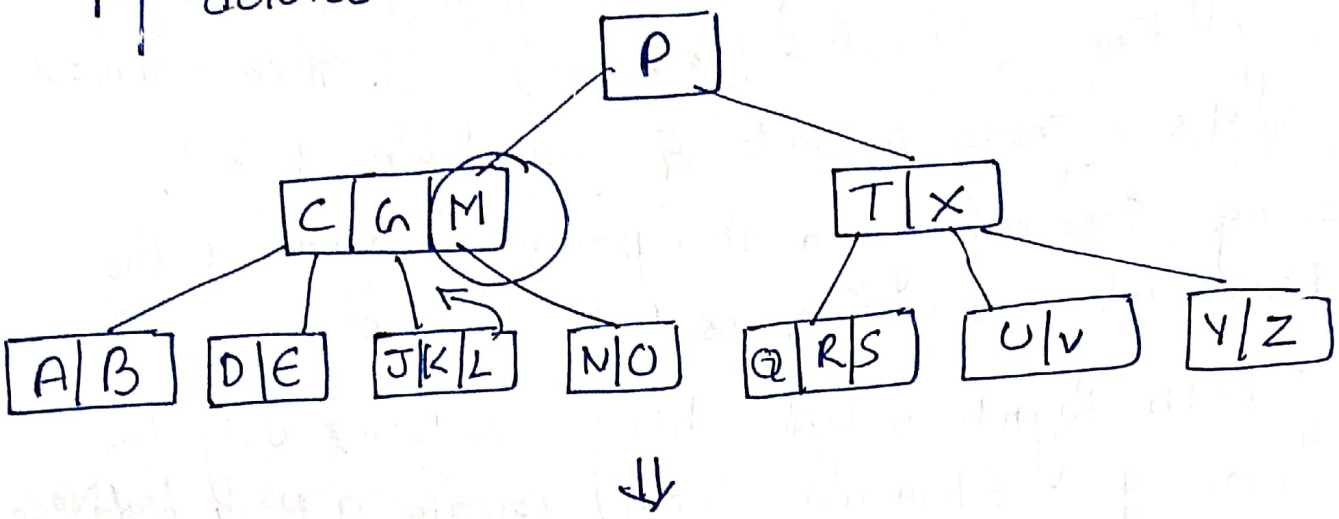


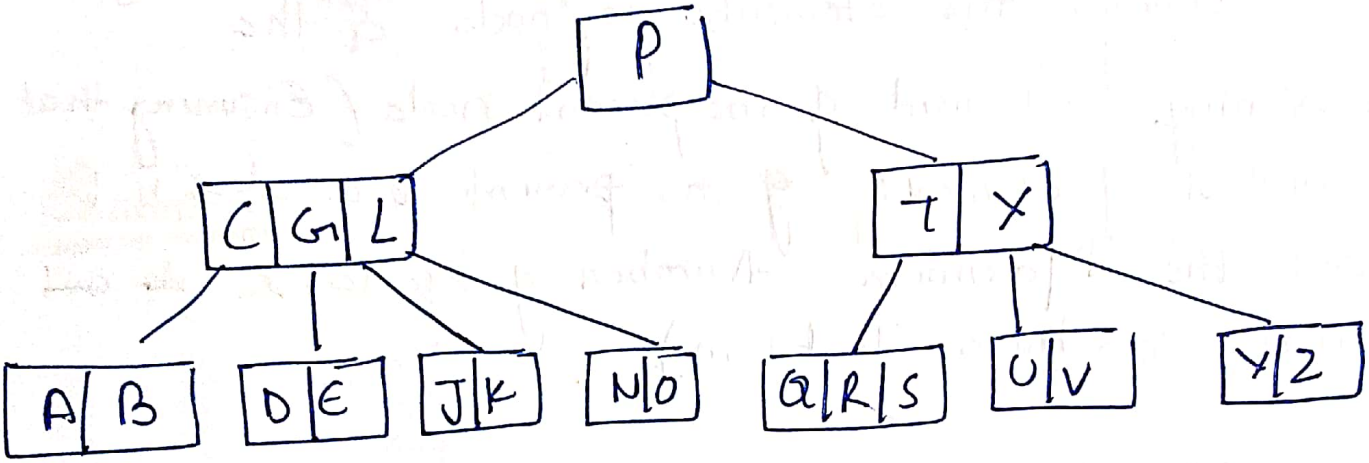
f is deleted



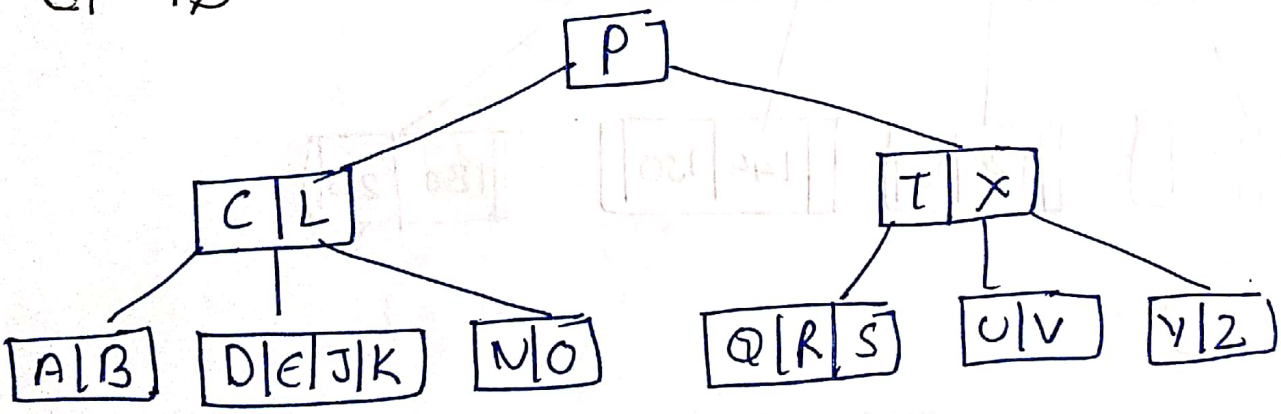
Scanned by CamScanner

M deleted





G is Deleted



3@

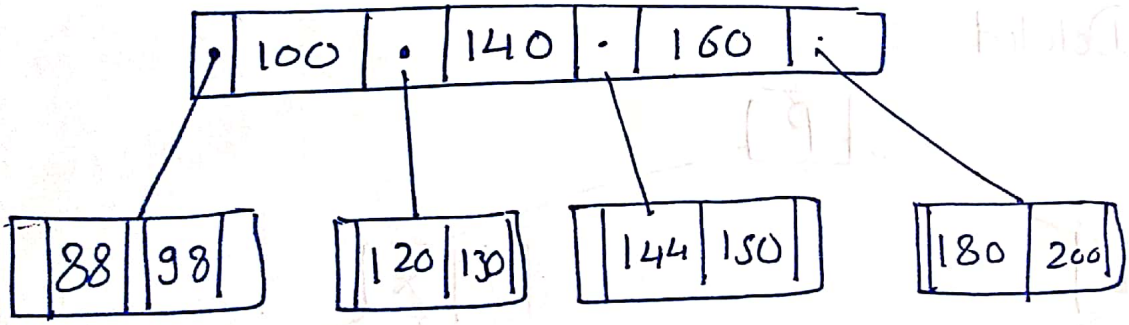
⇒ If the left sibling has more than the Minimum Number of Key value (Eliminate). Puts the largest Key into parent's node & down the intening Element from the parent node to the leaf Node where the Key is to be deleted.

Scanned by CamScanner

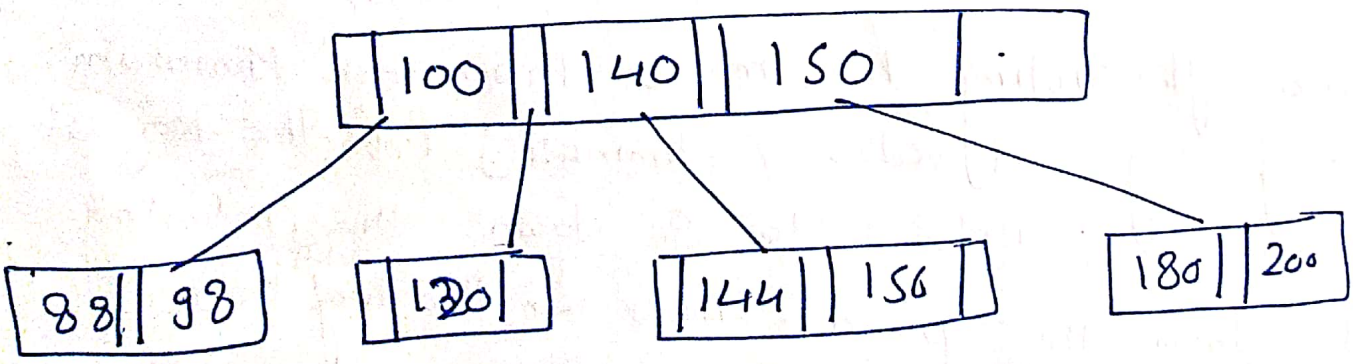
(5) If the Right sibling has more than the min number of Key value (Element) push it to smallest Key into it's parent's Node & pull down the intervening Element from the parent's Node to the leaf Node where the Key is to be deleted.

(4) Else if both Right & left sibling Contains only the Min. No. of Elements then Create a New leafNode by Combining the ~~Elements~~ leafnode & the intervening Element of the parent node (Ensuring that the number of Elements of the ~~parent Node~~. does not Exceed the Maximum Number of elements. ~~do not~~ a node can have ~~that~~ is m).

⇒



Delete (130)



Scanned by CamScanner

Red Black Tree

A Red Black Tree is a type of self-balancing binary Search Tree, in which Every node is colored with a red or Black. The Red black tree satisfies all the properties of the binary search tree but there are some additional properties which were added in Red Black Tree.

Properties of Red Black Tree

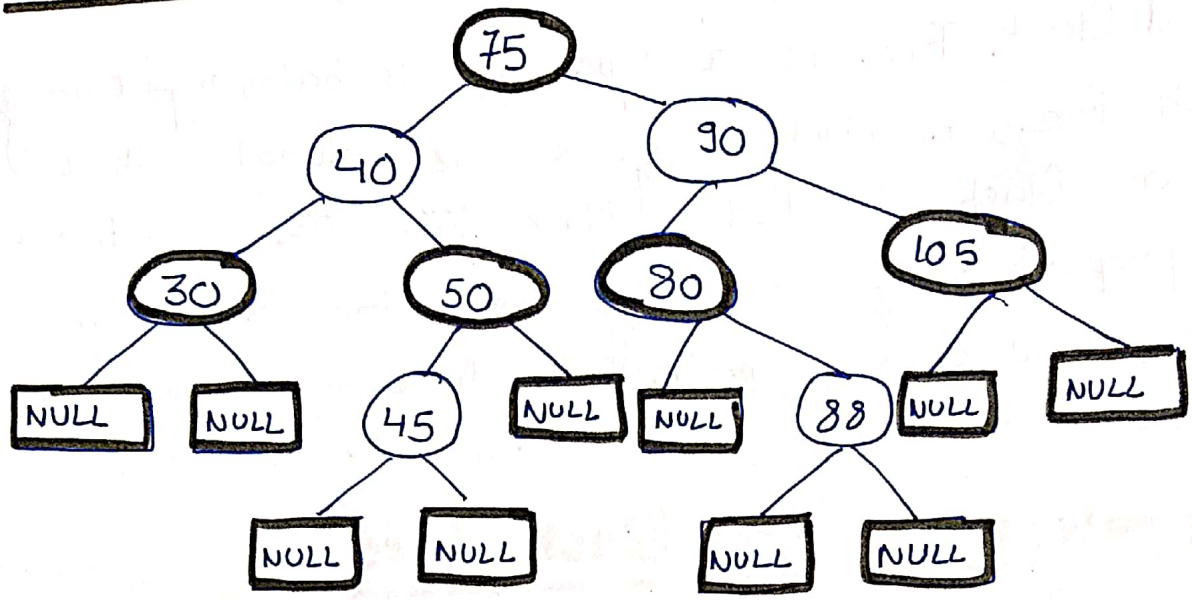
- The root node should always be black in color.
- Every child node of a node is black.
- Every node is Either Red or black.
- If a node is red the both its children are black.
- Every simple path from the root node to the (downward) leaf node contains the same number of black nodes.

A red black tree's node structure would be :-

```
Struct_red_black_node {
enum { red, black } colour;
  void * item;
  Struct tred_black_node *left,
                  * right
                  * parent;
}
```

Scanned by CamScanner

Representation of Red Black Tree



Advantages of Red Black Tree

- Red Black trees are useful when we need insertion & deletion Relatively frequent.
- Red black trees are self-balancing so these operations are guaranteed to be O(log n)
- They have Relatively low constants in a wide variety of scenarios.

Note

- A Red black tree with n internal nodes has height at most 2log (n+1)
- It can always be searched in O(log n) time
- A Rotation is a local operation in a search tree that preserves in-order traversal key ordering.
- No adjacent nodes are Red

Scanned by CamScanner

Insertion in Red Black Tree

* Every node which needs to be inserted should be marked as Red.
* Not Every insertion causes imbalancing but if imbalancing occurs then it can be Removed depending upon the configuration of tree before the New insertion is made.
* In Red Black tree if imbalancing occurs then for Removing it two Methods are used that are :—
  - → Recoloring
  - → Rotation

To understand insertion operation ; let us understand the keys required to define the following Nodes :—

* Let N is Newly inserted node
* p is the parent node of N
* g is the grandparent node of N
* U is the uncle node of N.

Scanned by CamScanner

Case ① New Node N is always Red

We have to Enter a Node and which is ~~Red~~ the Root Node. Then That root Node is simple inserted like

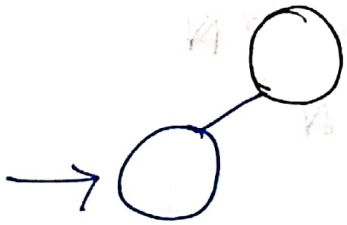
→ It is a Red Colour but the Root Node is always black a Red Black Tree. So we have to Recolour it.

Restructuring or Rotating

→ Right
→ left
→ Right-left
→ Left Right

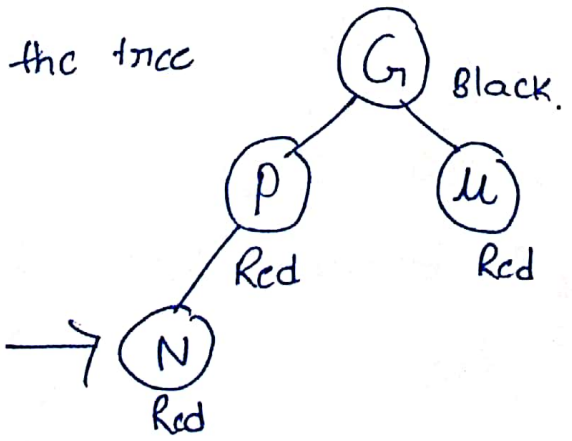
Case ② New Node N is inserted & whose Parent is Black.

The New Node is Entered which is Red in Colour



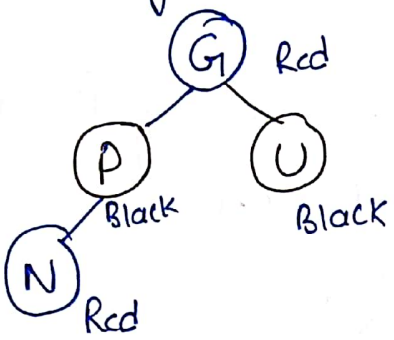
Case ③ New Node N is inserted whose parent(P) is not black & uncle U is also Red

Suppose the tree



Scanned by CamScanner

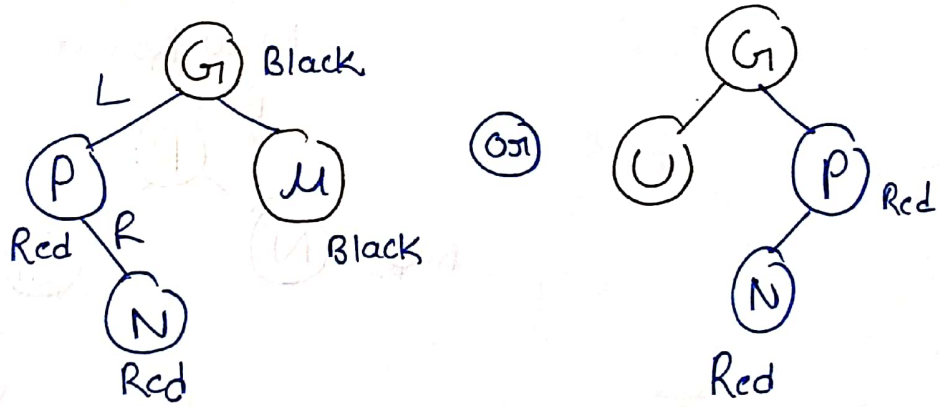
⇒ when we Entered the Node N then Parent P & child both are Red which violates the property of BR tree. So change the Colour of G, P & U



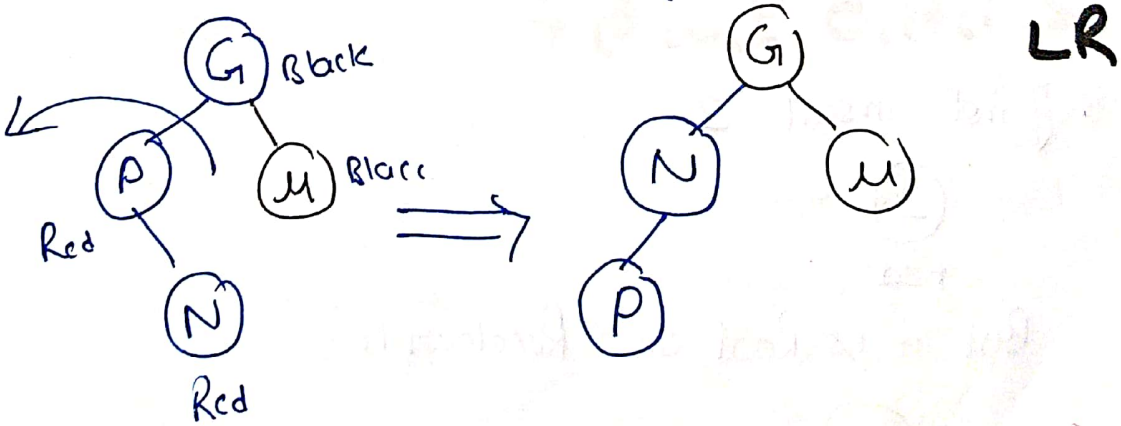
But if G is root then then it is black.

Case (4) N is added to right of left child of grandparent
or N is added to left of right child of grandparent

→ P is Red & Uncle (U) is black.



So then we have to perform Rotation.

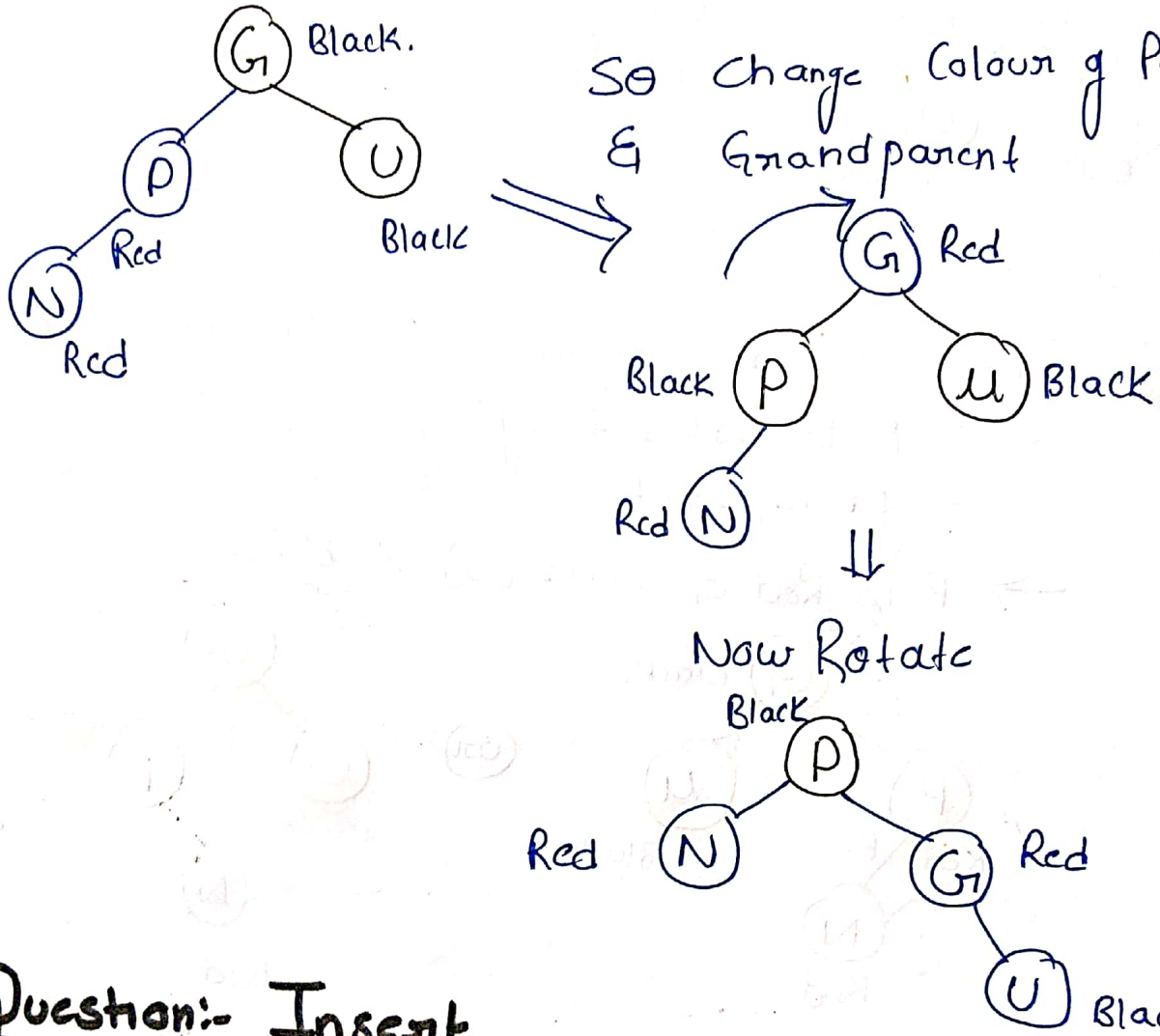


But N & P are Red so check Case (5)

Scanned by CamScanner

Case ⑤ N is added to left of left child of Grandparent or N is added to right of child of grandparent

→ P is red & U is black



Question:- Insert

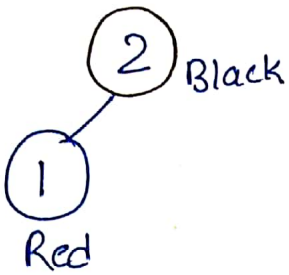
2, 1, 4, 5, 9, 3, 6, 7

Step ① First insert 2

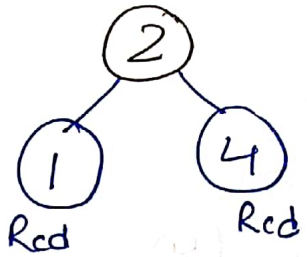
But it is Root so Recolour it

Scanned by CamScanner

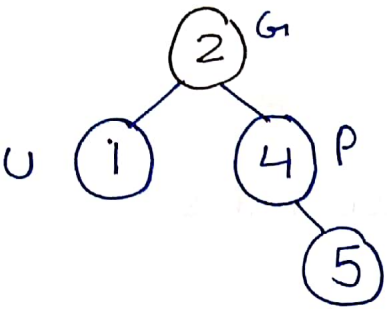
Step (2) Insert 1



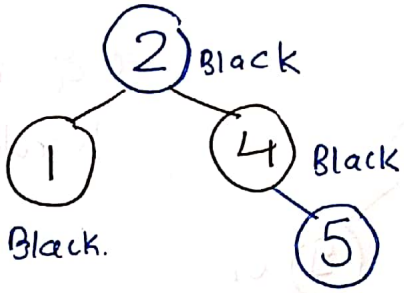
Step (3) Insert 4



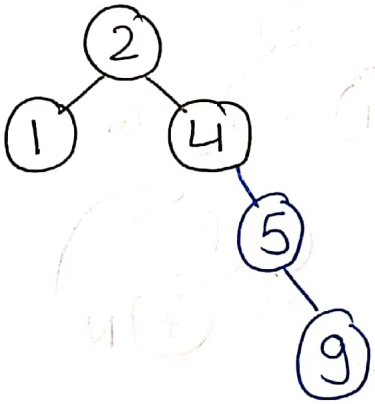
Step (4) Insert 5



4 & 5 both are Red so Recolor 1 & 4

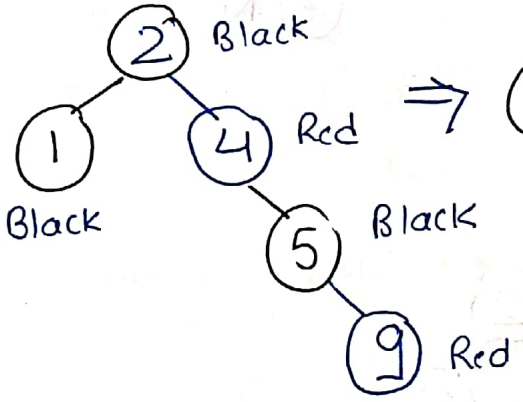


Step (5) Insert (9)

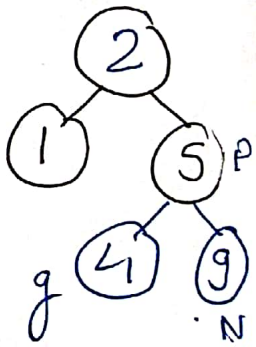


5 & 9 both are Red

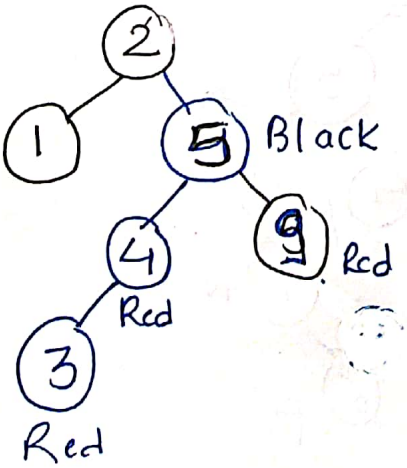
Recolor 4 & 5



⇒

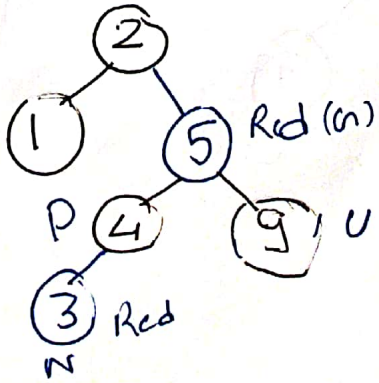


Step (6) Insert (3)



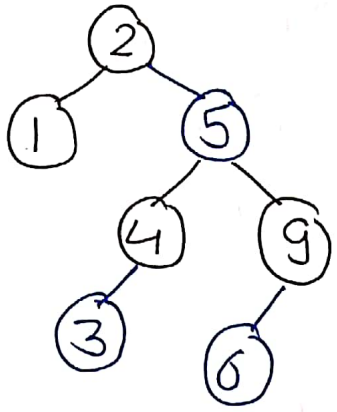
⇒

4 & 5 adjacent Red so Recolor 5, 4

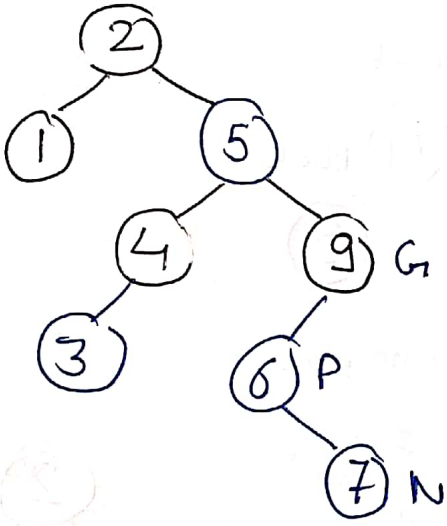


Scanned by CamScanner

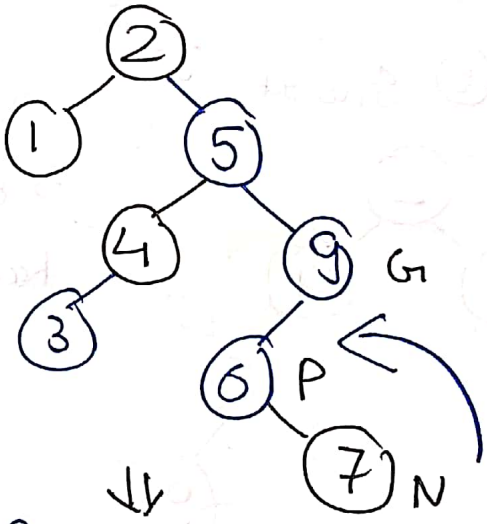
Step (7) Insert (6)



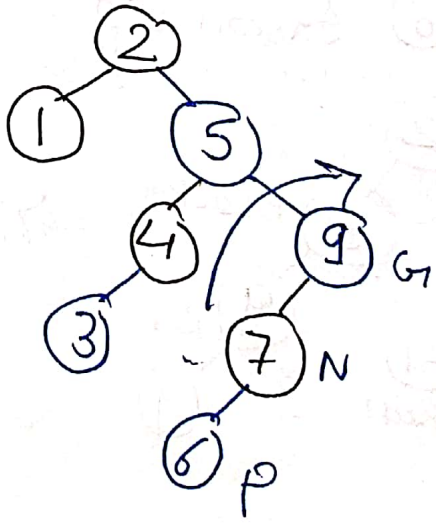
Step (8) Insert (7)



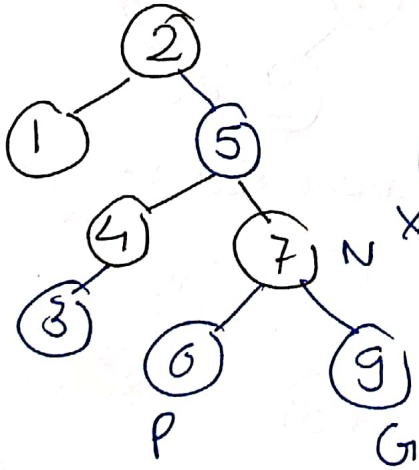
6 & 7 adjacent are Red so Recolor 7, 6, 9



Rotate N about P



Rotate X about G

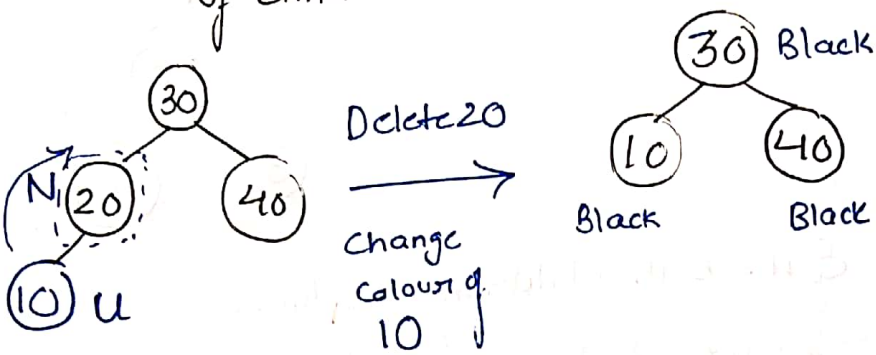


Scanned by CamScanner

Deletion in Red Black Tree

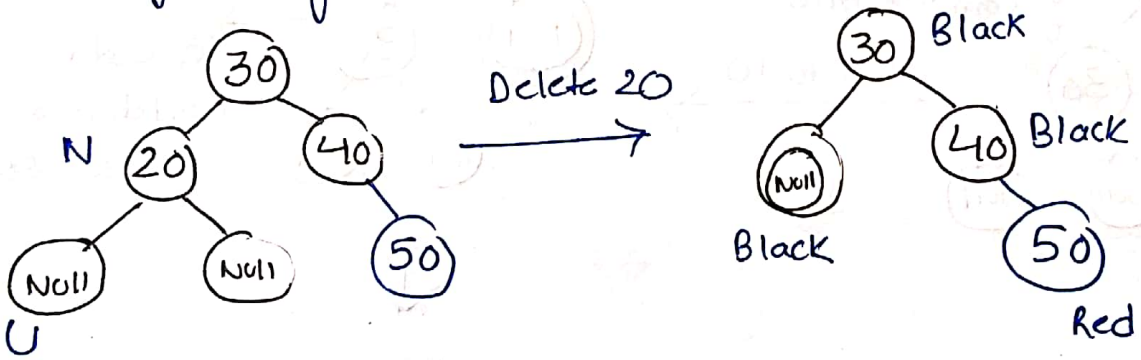
→ Deletion of node With two Children
→ Deletion of Node with almost one Children

Case ① Let N be the deleted Node & U is the child
If Either U and N is red

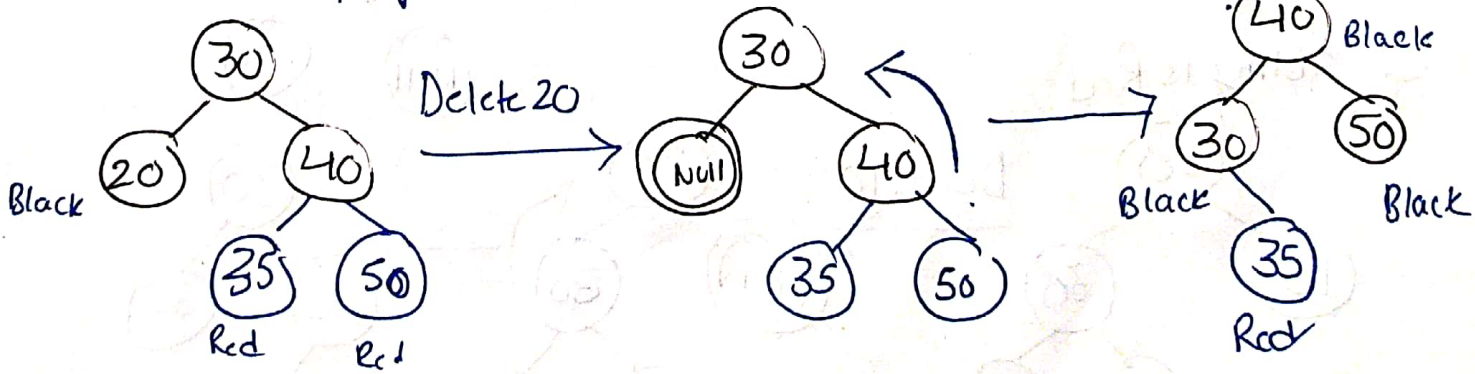


Case ② If Both N and U are Black

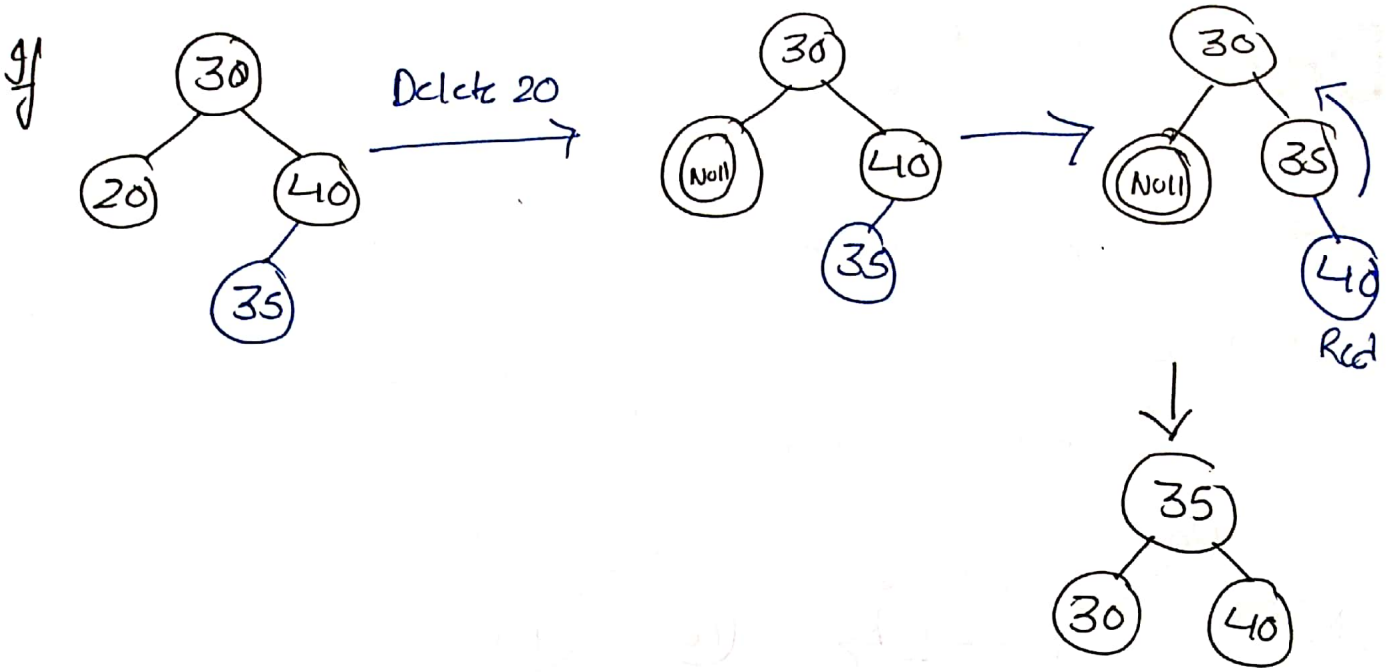
→ If Color of U is double black so convert it into single black.



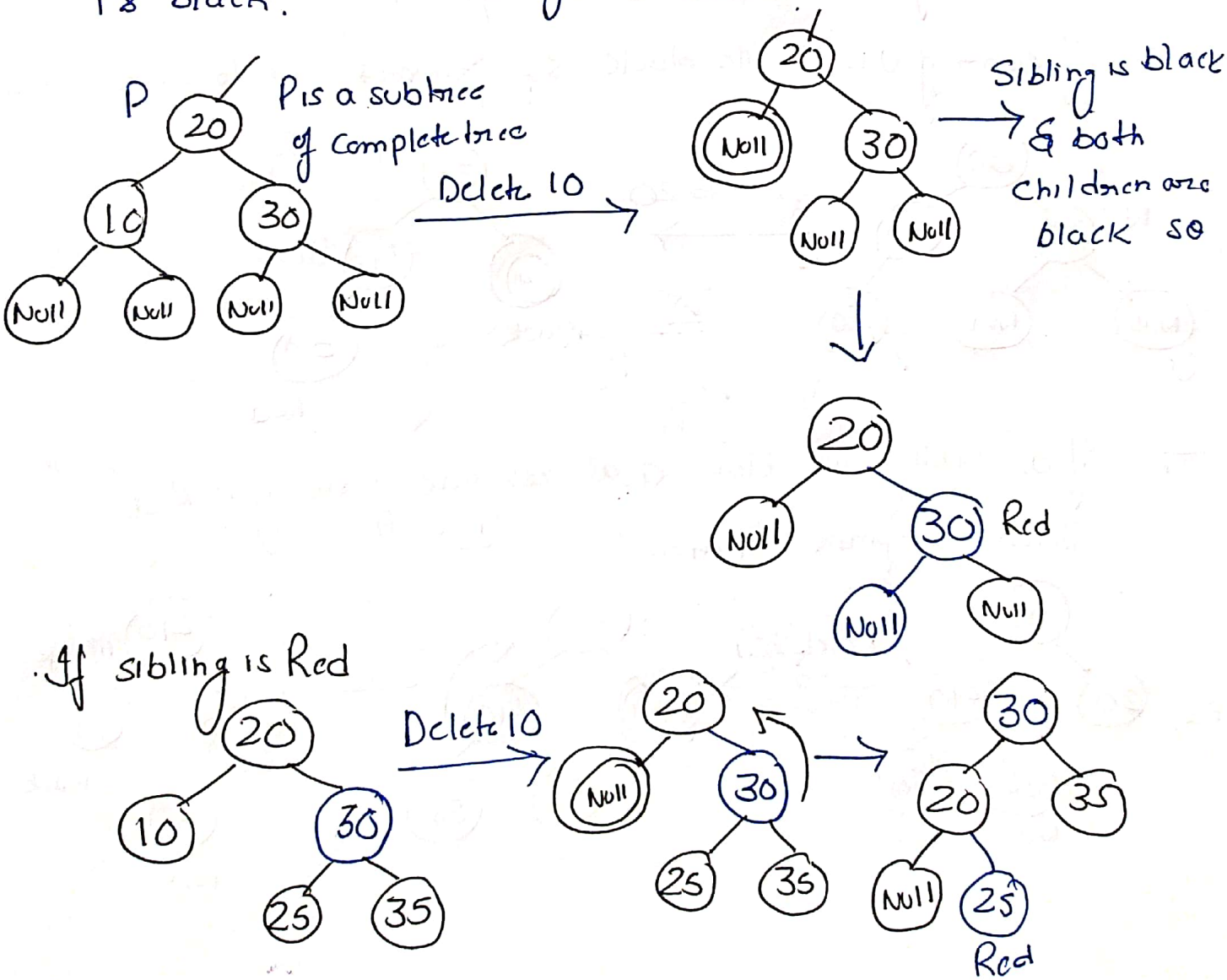
→ If a sibling is black & atleast one of sibling is Red
Then perform Rotation.



Scanned by CamScanner



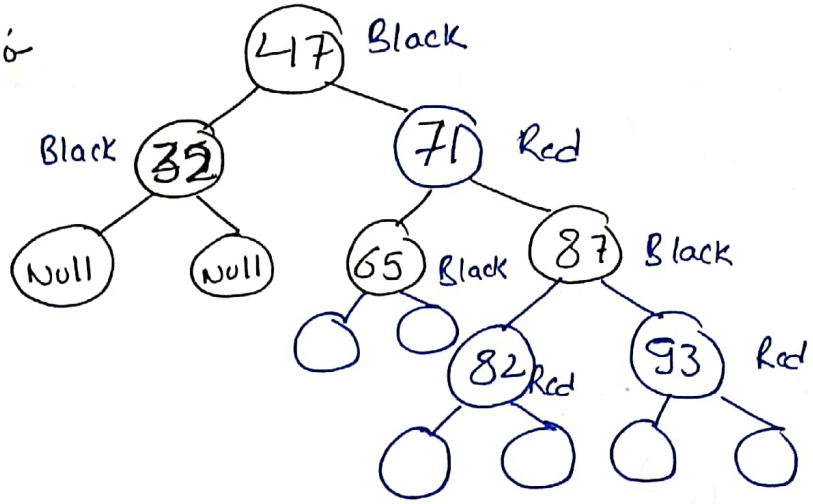
If sibling is black & its both children are black.
Then Perform Recoloring & recur for the parent if parent is black.



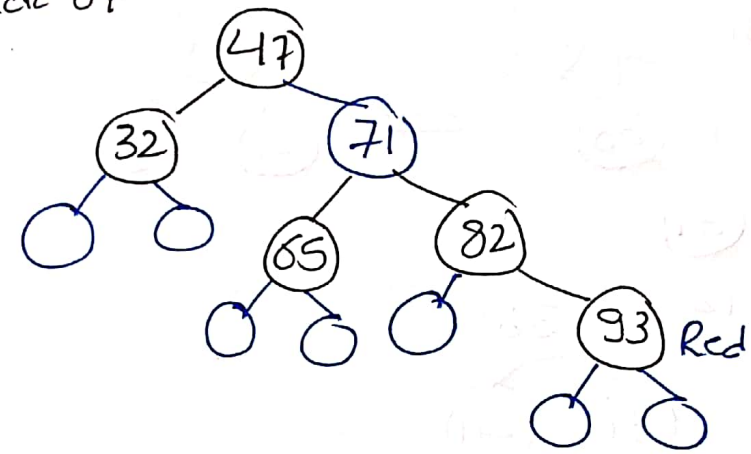
If sibling is Red

Scanned by CamScanner

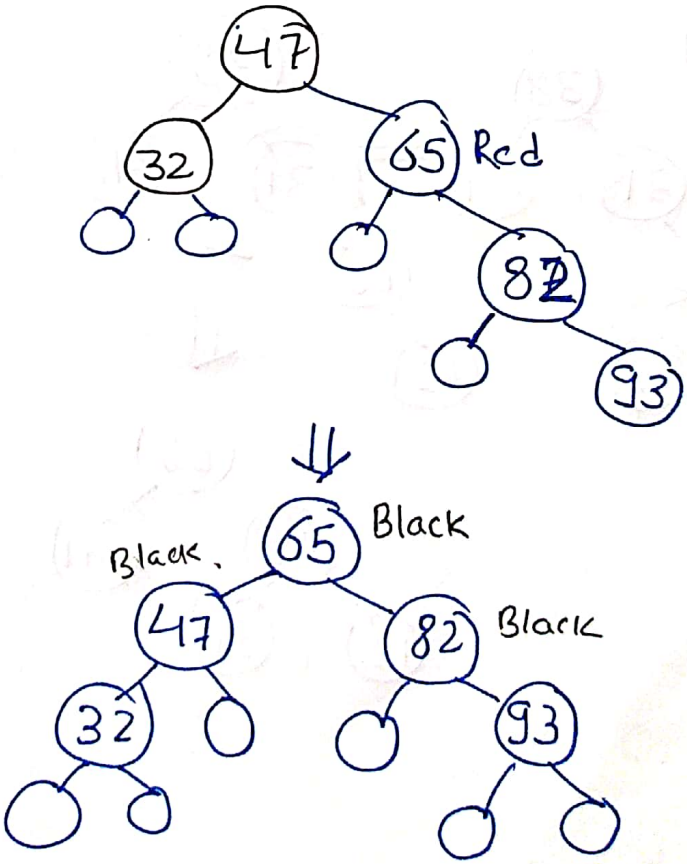
Example



Step ① Delete 87



Step ② Delete 71



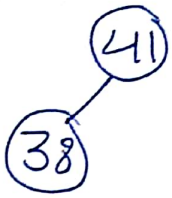
Scanned by CamScanner

Question:-

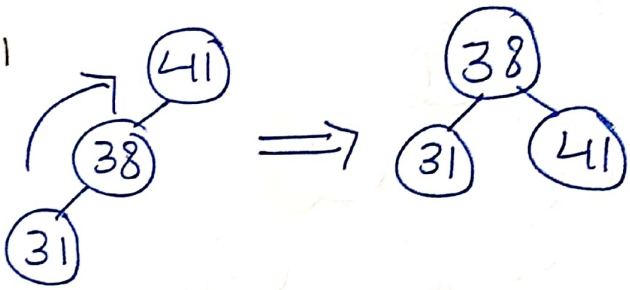
Show steps of inserting the Keys 41, 38, 31, 12, 19, 8 into initially Empty Red Black Tree.

Step(1) Insert 41

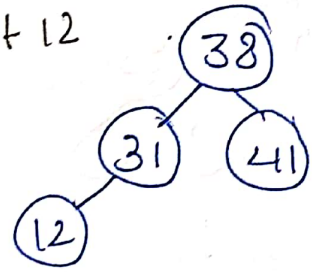
Step(2) Insert 38



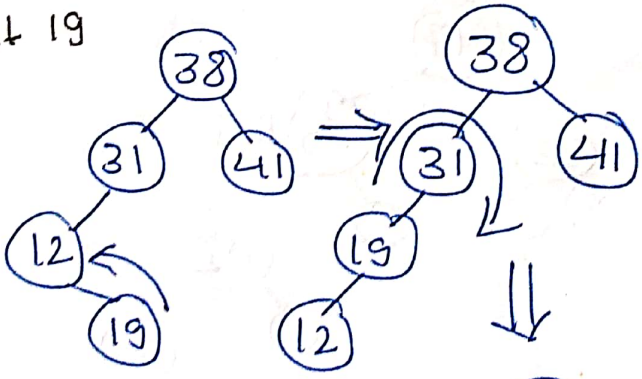
Step(3) Insert 31



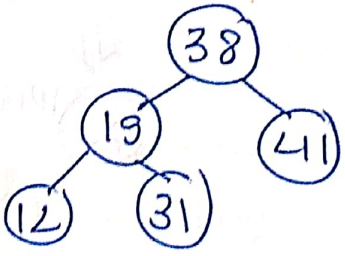
Step(4) Insert 12

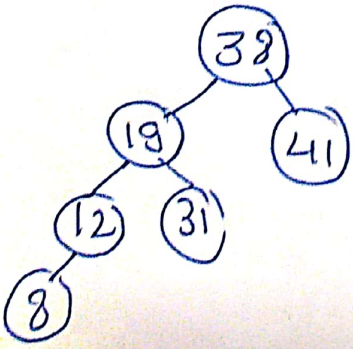


Step(5) Insert 19



Step(6) Insert (8)

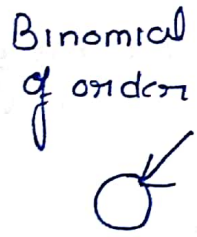




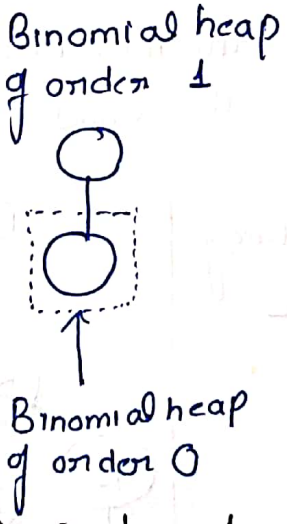
Scanned by CamScanner

Binomial Heaps

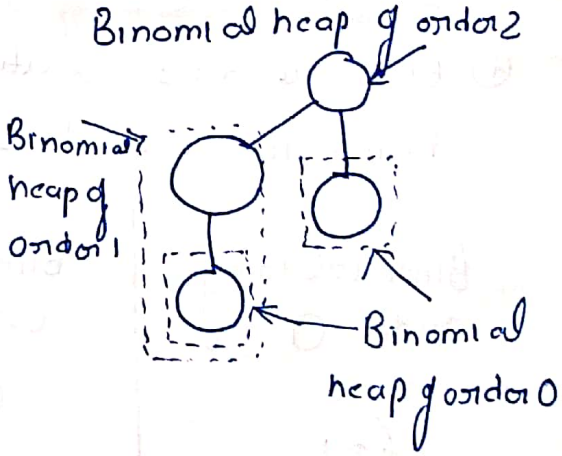
* Binomial heap is analogues to binary heap with Quick Merging facility. Binomial heap can be constructed as forest of binomial trees.
* Recursive defination of binomial tree is given as —
  - → Binomial tree with Zero order has only one Node.
  - → Binomial tree of order K has a root with children being Roots of binomial trees of order K-1, K-2, K-3 .... 2, 1, 0 in same order.



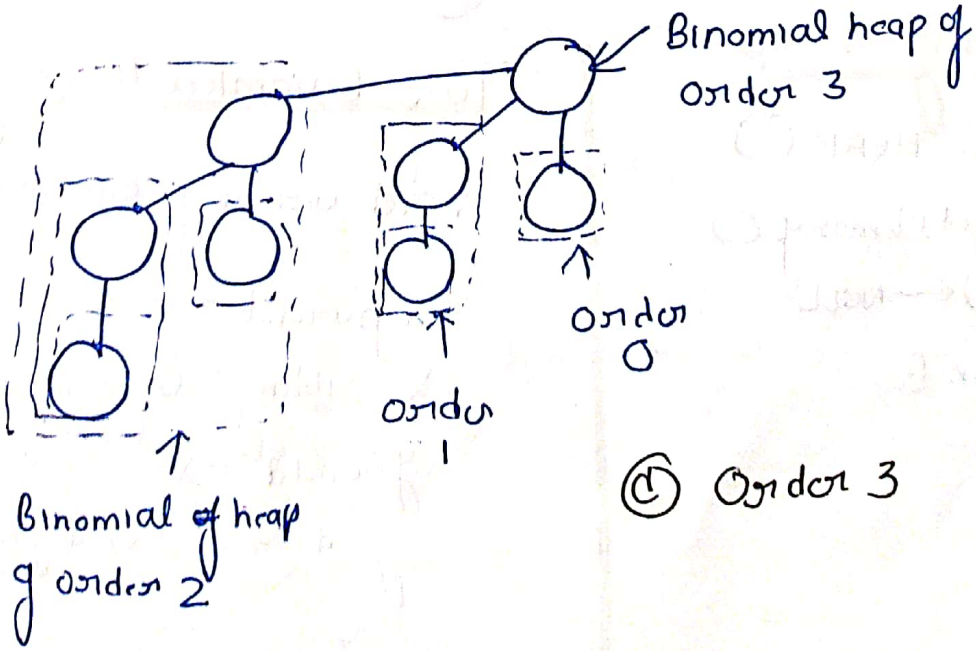
(a) Order 0



(b) order 1



(c) Order 2



(d) Order 3

Scanned by CamScanner

Properties of Binomial Heap

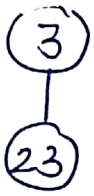
- Binomial tree of order K has 2^K nodes.
- Binomial tree of order K has height K.
- Binomial tree of order K-1 can be merged to form a tree of order K, by attaching one of the tree as a left most child of the root of another tree
- Two Binomial trees with same order cannot Exist in binomial heap.
- Binomial tree of order K contains $\binom{K}{d}$ nodes at depthd.

Structure of Binomial Heap

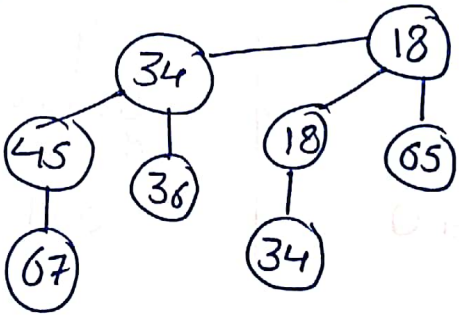
- Binomial tree satisfies the min-heap property. i.e Key of node must be less than its child.

Binomial tree of order 0

Binomial tree of order 1



Binomial tree of order 3



For Creating Heap

```
CREATE_HEAP()
B ← getMemory()
head(B) ← NULL
return B.
```

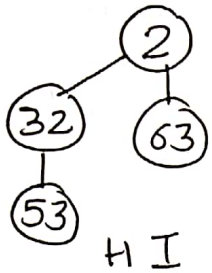
Two Binomial Heap of same Degree

```
void uniting-binomial Heap
{
  x.parent = y
  x.sibling = y.child
  y.child = x
  y.degree = y.degree+1
}
```

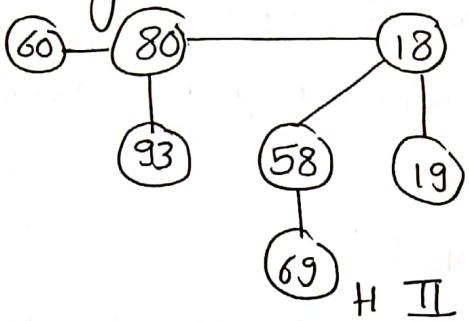
Scanned by CamScanner

Union / Merging of Heaps

Question: Merge the following heaps



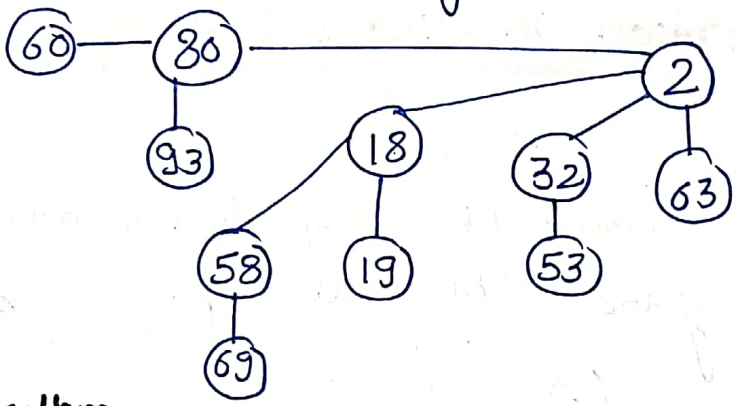
H I



H II

Key of root heap H1 is smaller than Key root of heap HII

Hence (2) will be Root of heap.



Algorithm

MERGE_HEAPS (H, H')

Create an array B with index i pointing to Hi

```
Carry ← 0
i ← A[i]
while i < logn do
  if H == NULL & H' == NULL then
    A[i] ← Nill
  end
  if H == NULL or H' == NULL then
    A[i] ← NULL
    Carry ← 0
  end
  if H ≠ NULL & H' ≠ NULL then
    A[i] ← NIL
    Set Carry bit to links of two heaps H & H'
  end i ← i+1
end
```

Scanned by CamScanner

Insertion Operation in Binomial Heap

To insert the Element with value x do the following

→ Create heap of order 0 for x and call it H'

→ Merge Existing heap H & Newly created heap H'

Algorithm

BH_INSERTION ()

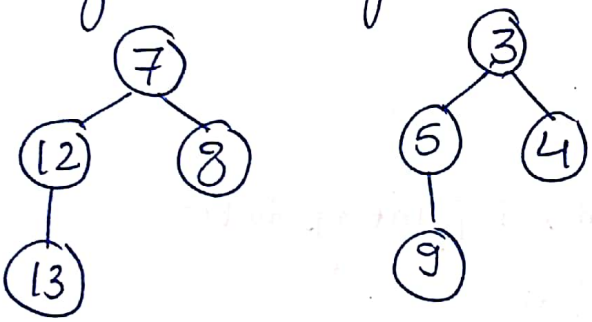
H' ← Create heap of order 0 with element x

MERGE_HEPS (H, H')

Deletion Operation in Binomial Heap

→ Find Minimum

To find Minimum Element of the heap, find the minimum among the roots of the binomial heap.



Compare roots from both tree. 3 is smallest so minimum is 3

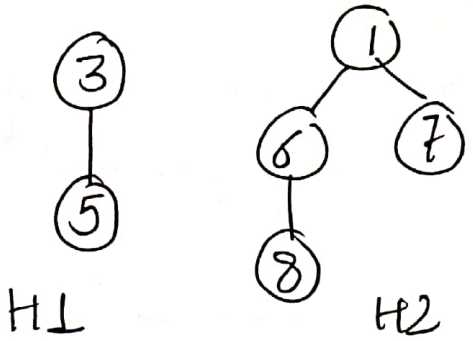
Deletion of Minimum

To delete the minimum Element from the heap, first find the Element, remove it from its binomial tree & obtain a list of its subtrees Then transform this list of subtrees into a separate binomial heap by re-ordering them from smallest to largest order.

Scanned by CamScanner

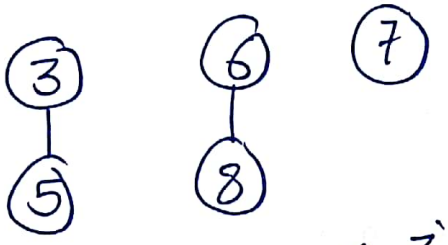
Then Merge these heap with the original heap.

Example :-

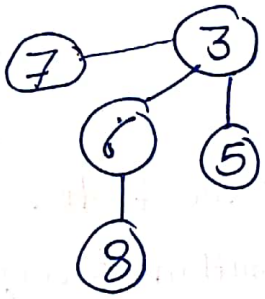


H1 H2

Minimum Element 1, so Remove it



Now Merge all 3 order of 1st & 2nd is same



Algorithm

EXTRACT_MIN(H)

X ← head(H)

Remove root of H & create forest of Heaps

Collect roots of H in reverse order H'

for i=1 to size(H') do

MERGE_HEAPS(H, Hi') // Hi' is the binomial with 1th root in list H'.

end

return X

Scanned by CamScanner

Fibonacci Heap

A Fibonacci heap is a specific implementation of the heap data structure that makes use of Fibonacci numbers.

Time Complexity of various Heap Operations

Find Minimum	O(1)
Delete Minimum	O(log n)
Insert	O(1)
Extract min	O(log n)
Union	O(1)
Makeheap	O(1)

* Fibonacci heaps are used to implement the Priority Queue element in Dijkstra's Algorithm giving the algorithm a very Efficient Running time.
* Fibonacci heaps have a faster running time than other heap types. Fibonacci heaps are similar to binomial heaps but Fibonacci heaps have less rigid structure.
* Binomial heaps Merge heaps immediately but Fibonacci heaps wait to Merge until the Extract-min function is called. While Fibonacci heaps have very good theoretical Complexities.

Scanned by CamScanner

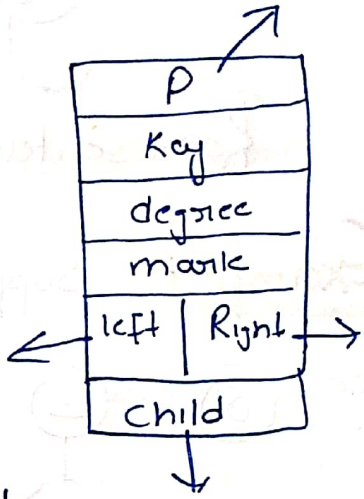
(*) Fibonacci heap is differ from binomial-heaps however in that they have more relaxed structure allowing for improved asymptotic time bounds work that maintains the structure is delayed until it is convenient to perform.

Structure of Fibonacci Heap

Each node x contains

→ A pointer p[x] to its parent

→ A pointer child[x] to one of its children



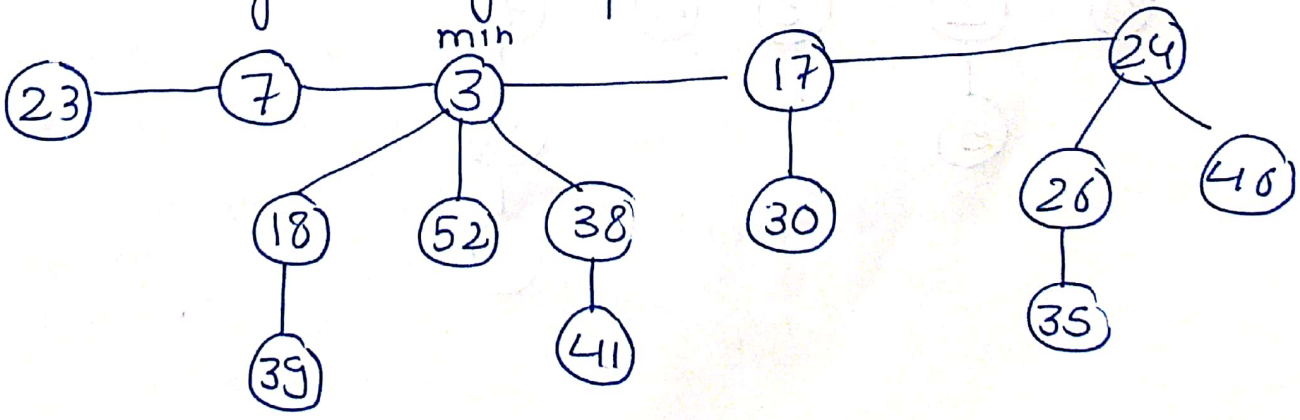
The children of x are linked together in a circular doubly linked list which is called the child list

→ Each child y in a child list has pointers left[y] & right[y] that point to y's left & right siblings

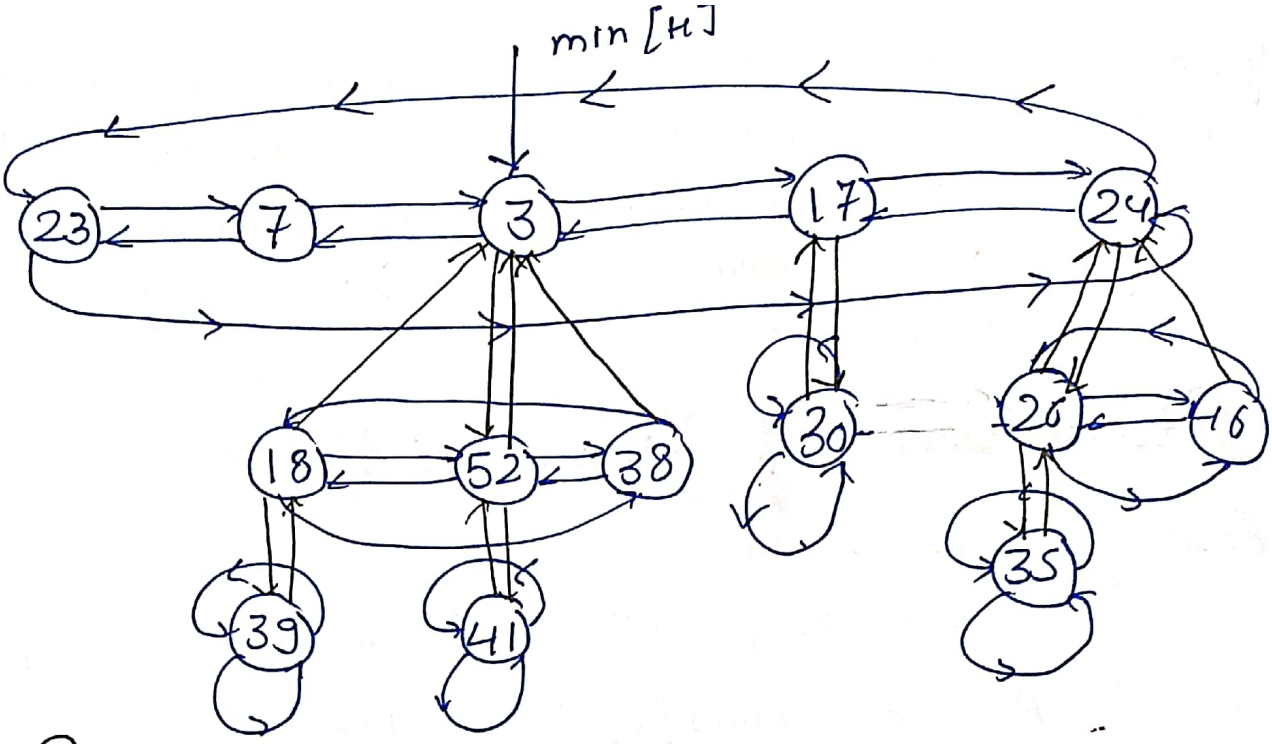
→ If y is only child then

$$left[y] = right[y] = y$$

→ The roots of all trees are linked together using their left & right pointers.

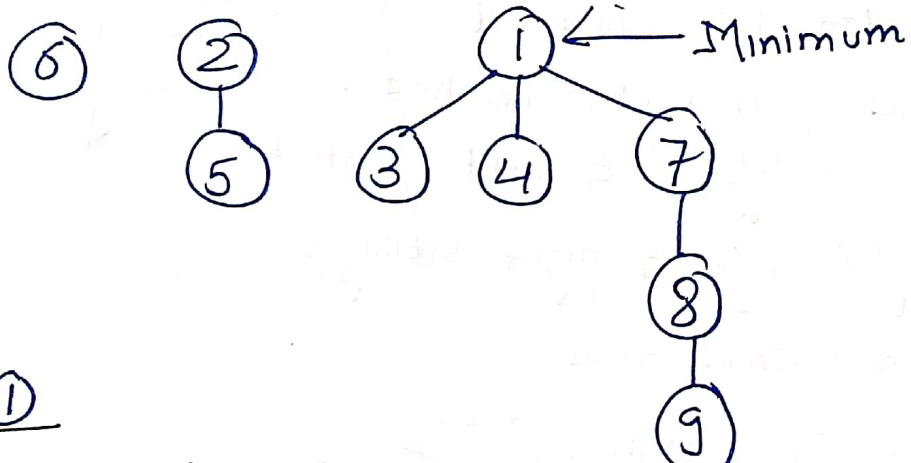


Scanned by CamScanner



Representation of Fibonacci Heap;

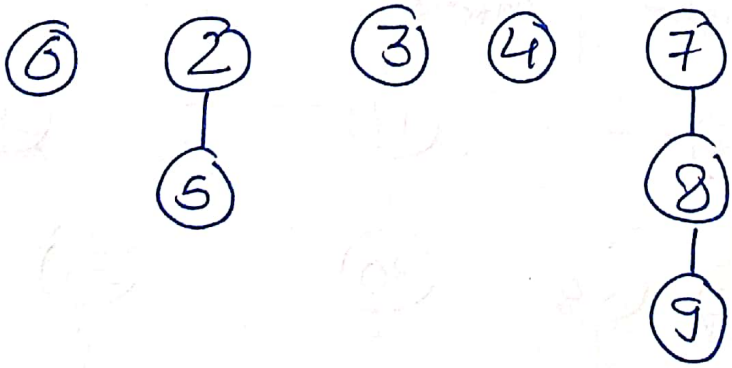
Example: Suppose these Trees are given



Step ①

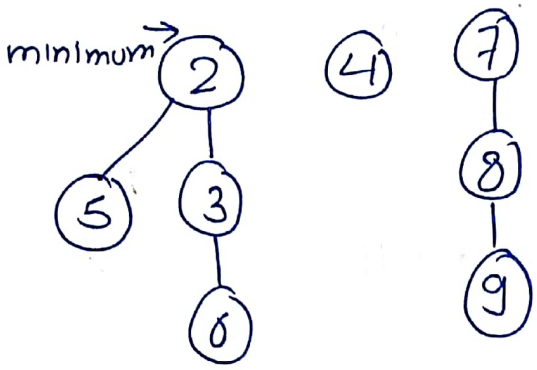
we have to Extract Minimum that is ①

Step ② Now Minimum Element is deleted



Scanned by CamScanner

Step (3) Now find Minimom from all subtrees
so (2) is minimum from all subtree



Various operations of Fibonacci Heap

Insert

Insertion to a fibonacci heap is similar to the insertion operation of binomial heap.

Remove

To delete an Element decrease the key using decrease key to negative infinity & then call the Extract-min. when the has has a value of -ve infinity. Since the heap is a min heap, it will become the Root of the tree

Algorithm to inserting a Node

```
FIB_HEAP_INSERT (H, X)
  degree[x] ← 0
    p[x] ← NIL
    child[x] ← NIL
    left[x] ← x
    right[x] ← x
    mark[x] ← FALSE
Concatenate the root list containing x with root list H
 if key[x] < key[min[H]] then
      min[H] ← x
 endif
      n[H] ← n[H]+1
 end
```

Scanned by CamScanner

Algorithm to Extracting the minimum

FIB_HEAP_EXTRACT_MIN (H)

$z \leftarrow min[H]$

If $z \neq NIL$

then for Each child x of z

do add x to the root list of H

$p[x] \leftarrow NIL$

remove z from the root list of H

If $z = right[z]$

then $min[H] \leftarrow NIL$

else $min[H] \leftarrow right[z]$

Consolidate (H)

$n[H] \leftarrow n[H] - 1$

return z.

Decrease Key

To decrease the value of any Element in the heap Two situations arise:-

→ If the heap properties are not violated then simply decrease X

→ If violation does occur, remove the node it's parent, If parent is not a root, mark it. If it has marked already it is removed as well & it's parent is marked and so on. Continue this process up the tree until Either the root or an unmarked node is Reached. Then Next set the minimum pointer to the decreased value if it is the New Minimum.

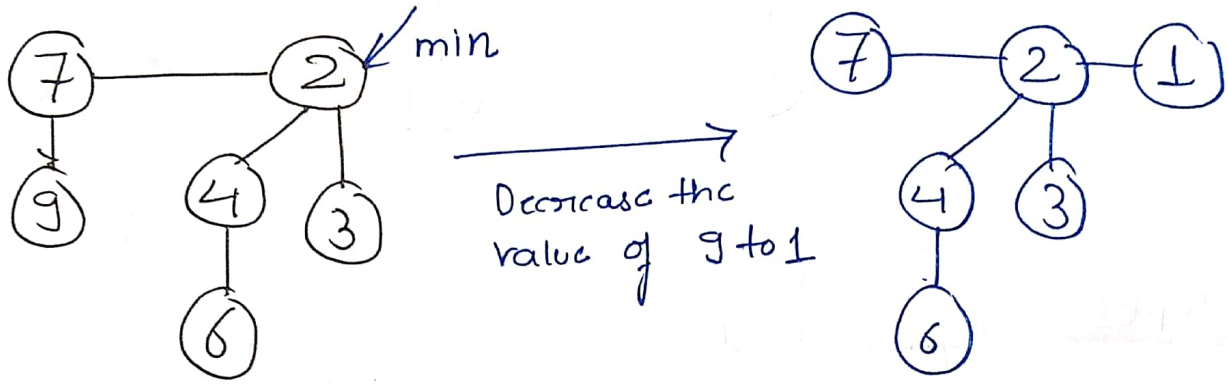
Scanned by CamScanner

Essentially the marking tracks if-

→ The node has had no children removed (unmarked)

→ The node has had a single child Removed (marked)

→ The node is about to have a second child removed (removed a child of a marked node)



Scanned by CamScanner

Algorithm For writing Two fibonacci Heap

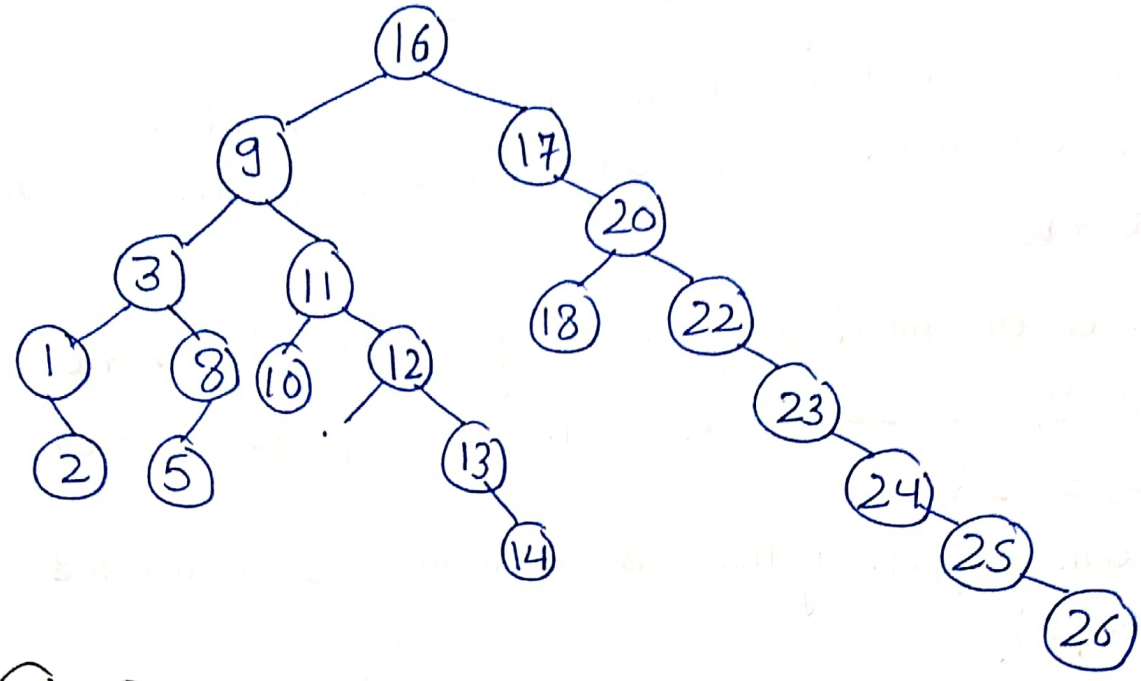
```
Void writing fibonacci_heap
  x = min (H1)
  y = Right (x)
  P = min (H2)
  Q = left (P)
  Right (x) = P
  Left (P) = x
  Left (y) = Q
  Right (Q) = y
  Concate (n)
n(Hi) = n(H1) + n(H2)
  if Key (x) < Kay (P)
    min (H) = x
    else
    min (H) = P
  }
```

Algorithm for finding Minimum Key in Binomial Heap

```
{ void_minimum key value
  { xx = head (H)
  min = x key
  x = x. sibling
  while (x! = NULL)
  { if x.kay < min
      min = x.Kay
      x = x.sibling
  }}
```

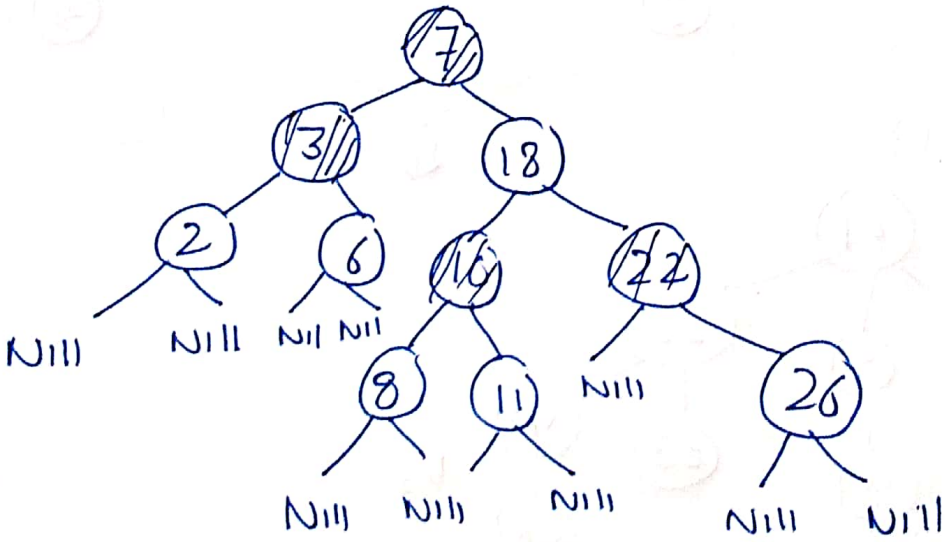
Scanned by CamScanner

Question:- (1) Show the Result of inserting the keys 16, 9, 17, 11, 3, 12, 8, 20, 22, 23, 13, 18, 14, 10, 1, 2, 24, 25, 26, 5 into an Empty Binary Search Tree.



(2) Draw Red Black Tree with 10 nodes.

Let Nodes: 7, 18, 3, 22, 26, 10, 11, 8, 2, 6



→ Black

→ Red

Scanned by CamScanner

(3) How many Binomial trees are there in a binomial heap with 19 Nodes? what are they.

In Binomial Heap no two trees are same, unique tree B_k is combination of $2(k-1)$ tree

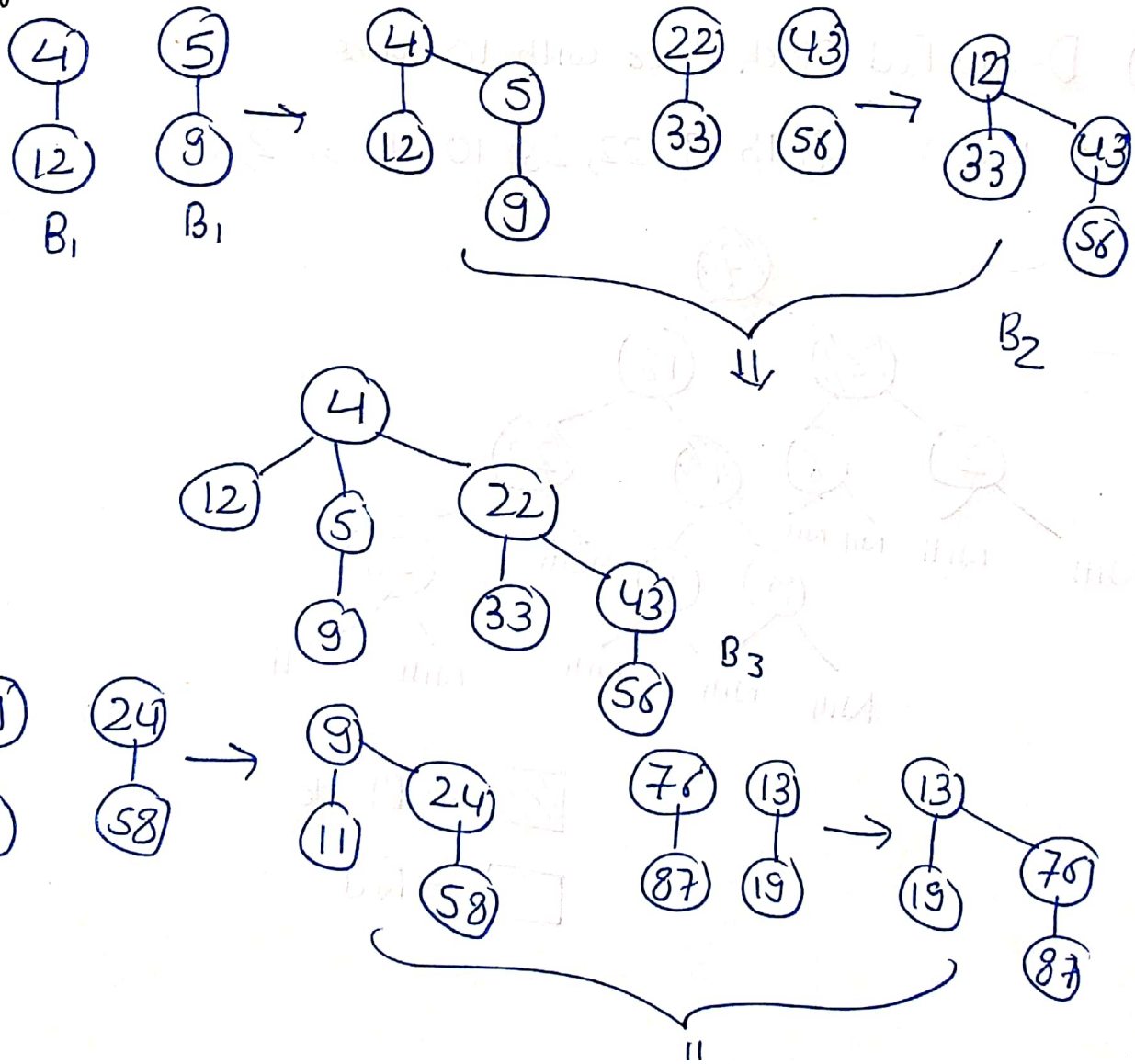
If there are 19 Nodes then = 16node +2node+1node

mean 8 binomial tree with 16 node & 2node & 1node

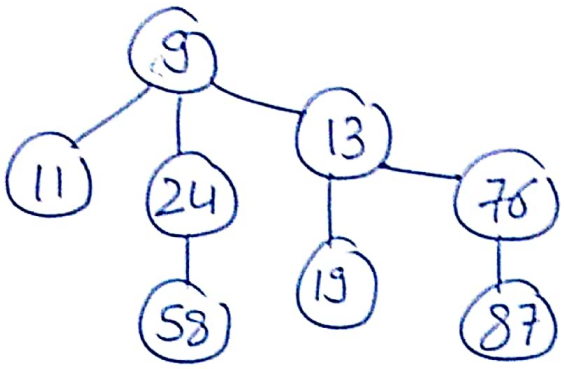
$B_4 + B_1 + B_0$

(4) Create a binomial heap with following set of data 12, 4, 5, 9, 33, 22, 56, 43, 9, 11, 24, 58, 76, 87, 19, 13, 21, 48, 92, 55, 28.

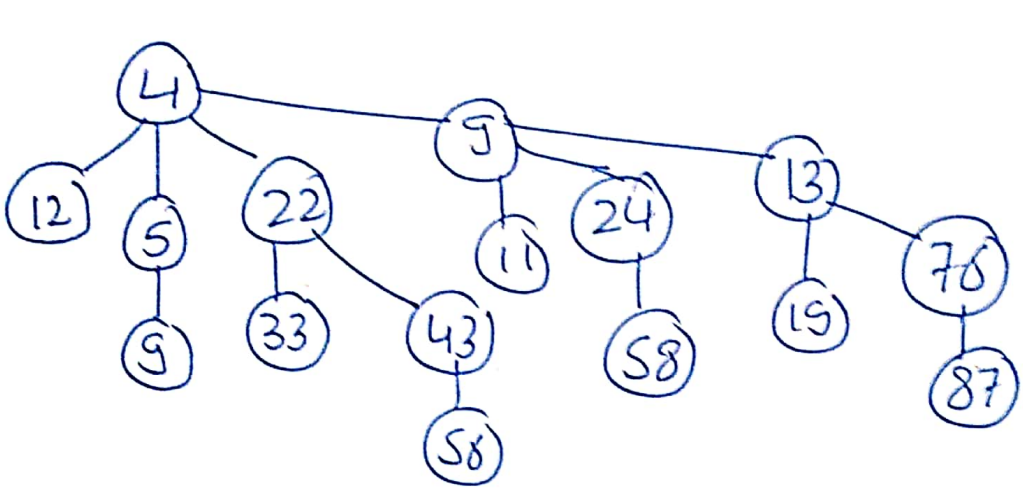
what will happen if the node containing 21 is deleted from heaps?

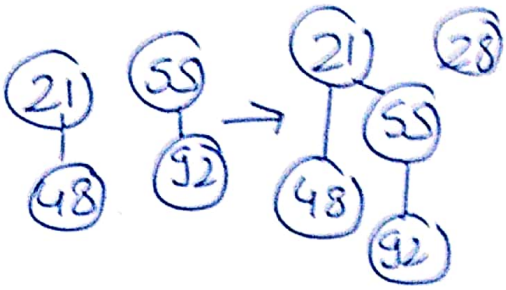


Scanned by CamScanner

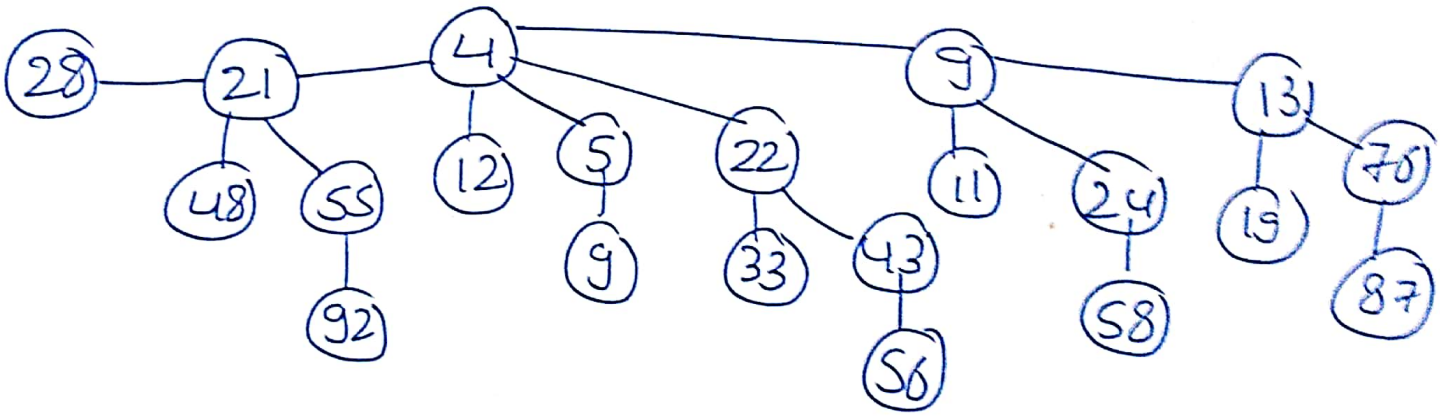


B_4





$B_5 \rightarrow B_3 + B_4$



Scanned by CamScanner