UNIT-2

ADVANCED DATA STRUCTURE

Data structure means how to organize the data in the primary memory. There are different approach to organize the data in memory.

Now we will discuss advanced data structure which are used for secondary memory and designing the operating system.

There are 4 advanced data structure —

1) B-tree
2) Binomial Heap
3) Fibonacci heap
4) Red black tree.

1) B-tree :

B-tree are used for secondary storage device & B-tree is a branching factor tree. and B-tree is also a rooted tree. Every node contains the following properties —

i) a) $n(x)$ indicates the no. of key stored in x.

b) Keys are stored in increasing order

c) leaf x has a boolean value, if it is true then x is a leaf otherwise x is a internal node.

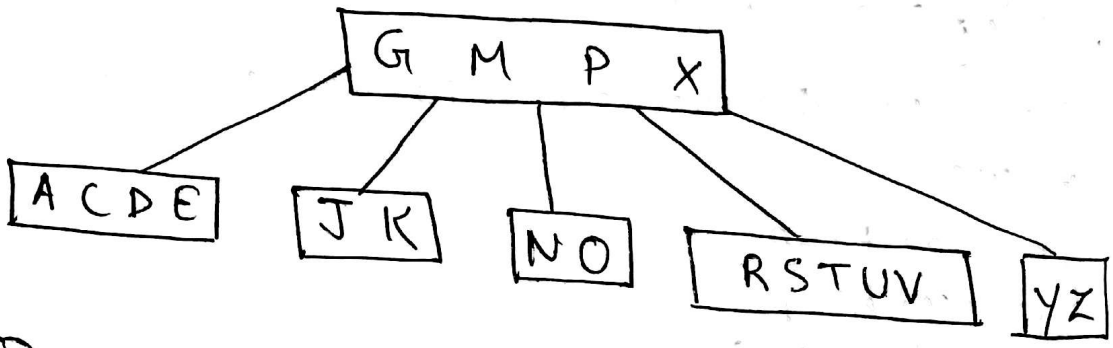
ii) Each internal node 'x' contain $n(x)+1$ child or pointers.

iii) All leaf have the same depth, it means B-tree is balanced tree.

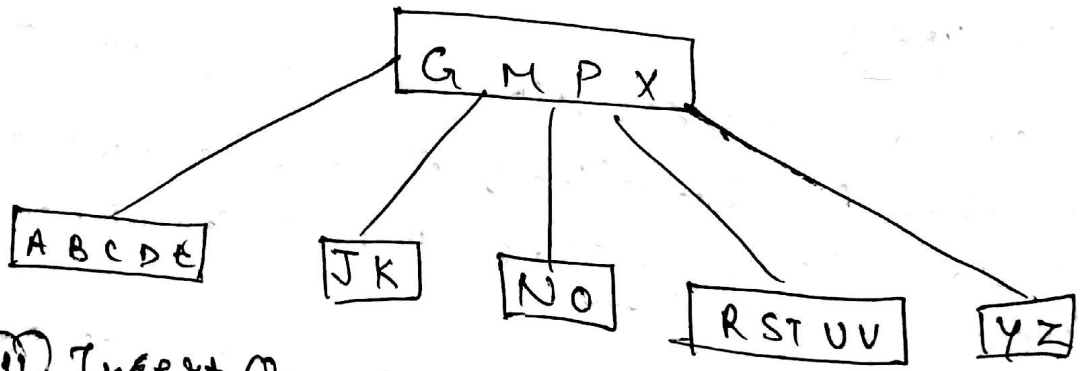
iv) If degree 'p' is given with condition $p \geq 2$ then atleast key would be $p-1$ except the root. and maximum keys would be $2p-1$

Insertion in B-tree

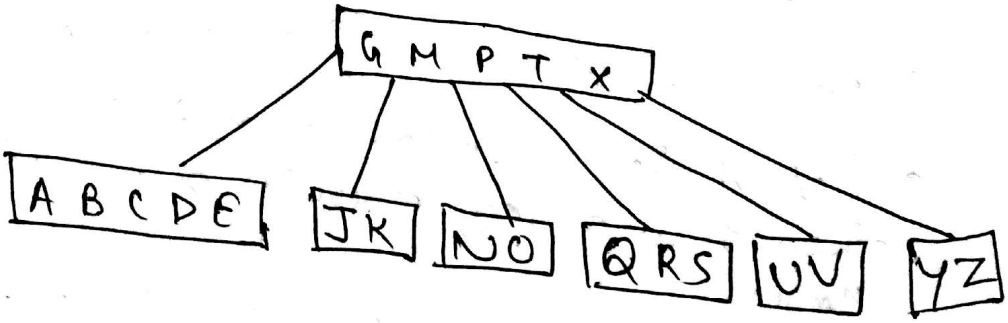
If degree = 3 then atleast keys = 2
max. keys = 5



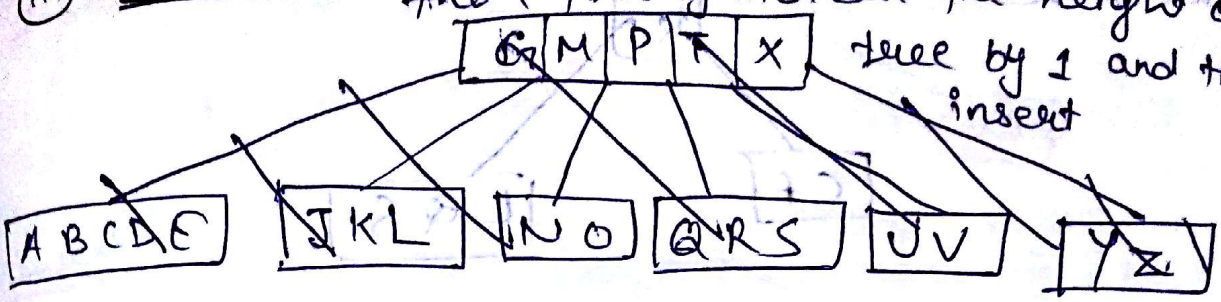
(i) Insert B

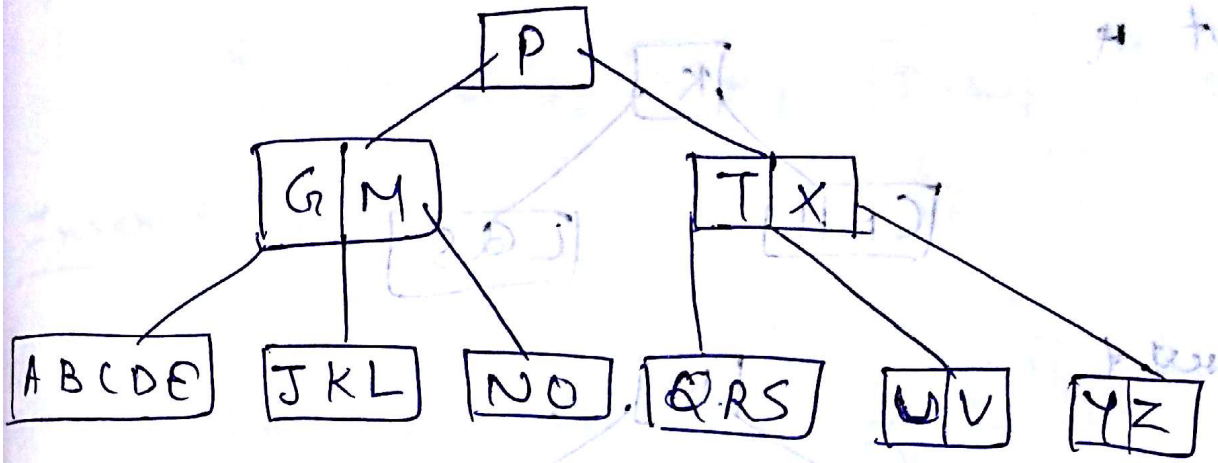


(ii) Insert Q



(iii) Insert Z — If root is full & insertion is perform then firstly increase the height of tree by 1 and then insert





Q Insert the element into B-tree:

F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E, G, I & Degree = 3

Sol

At least keys = 2
max keys = 5

~~F S Q K C~~

Insert F

F

4 S

F S

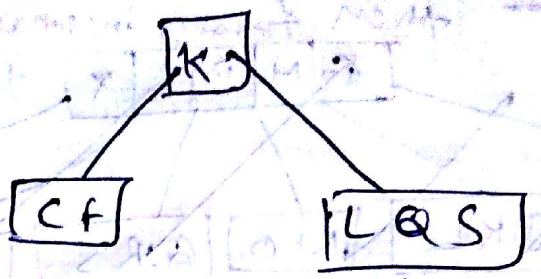
4 Q

F Q S

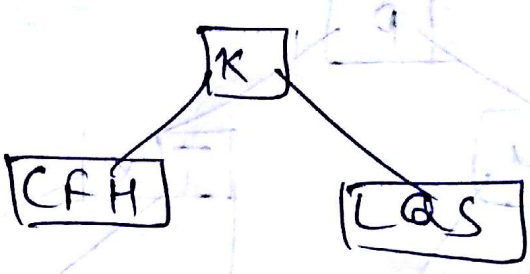
1 K

C F K Q S

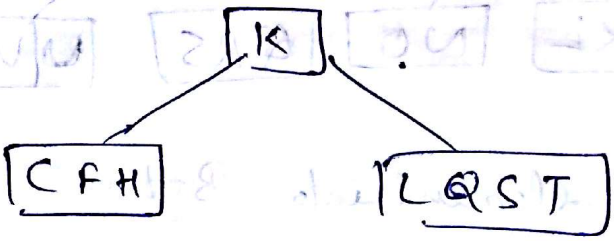
Insert L



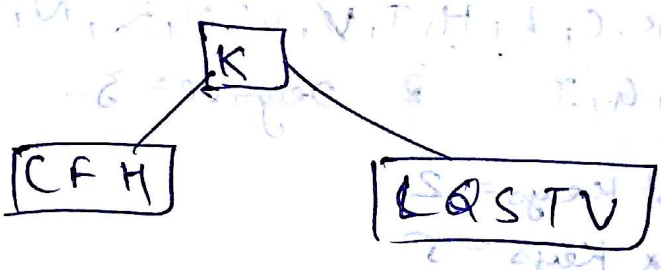
Insert H



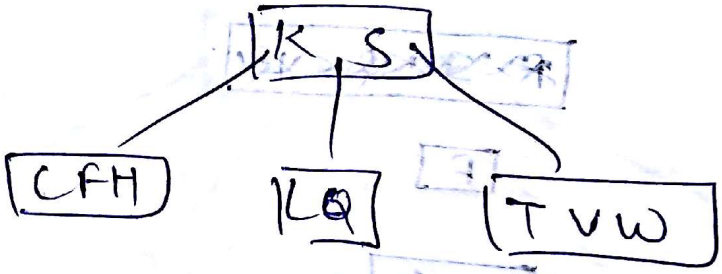
Insert T



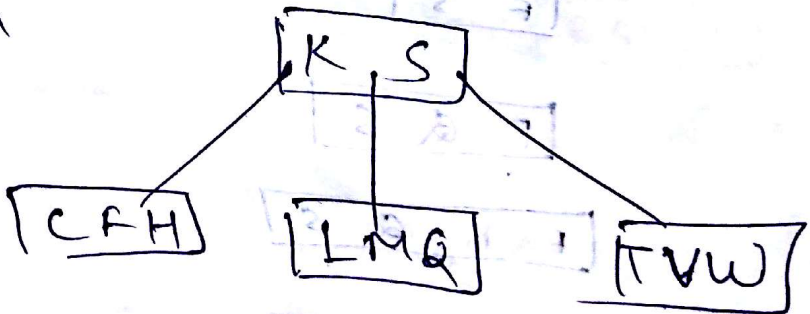
Insert V.



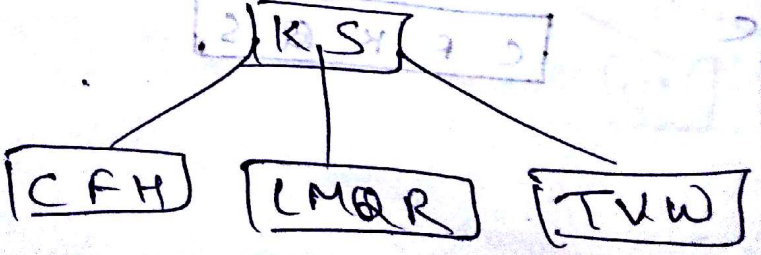
Insert W.



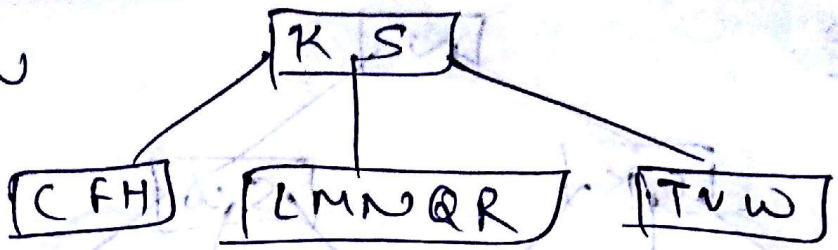
Insert M



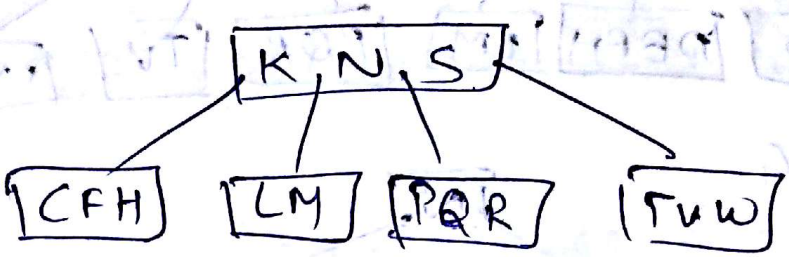
Insert R



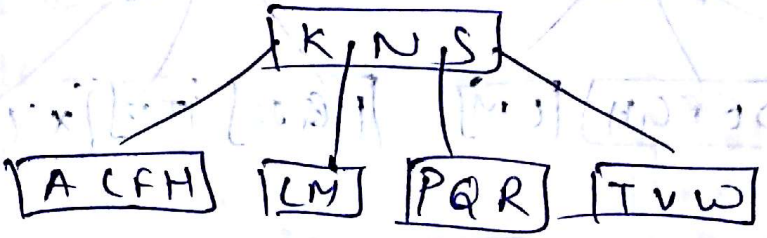
Insert N



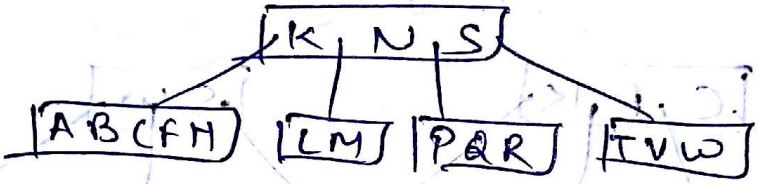
Insert P



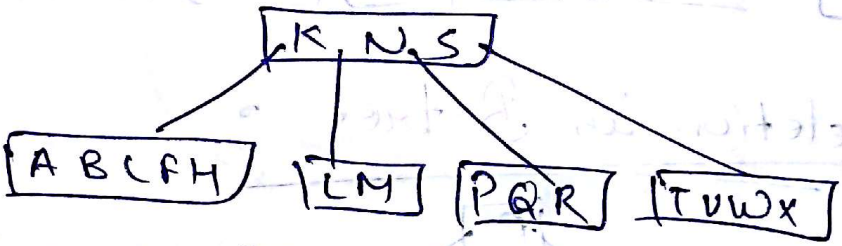
Insert A



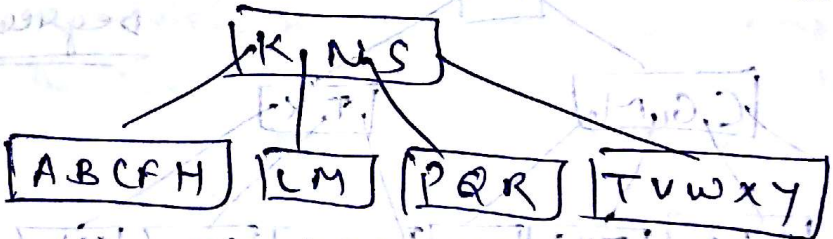
Insert B



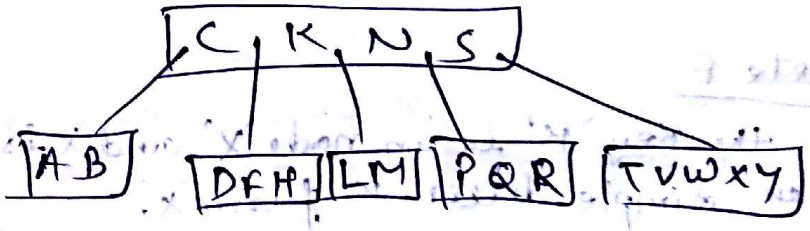
Insert X



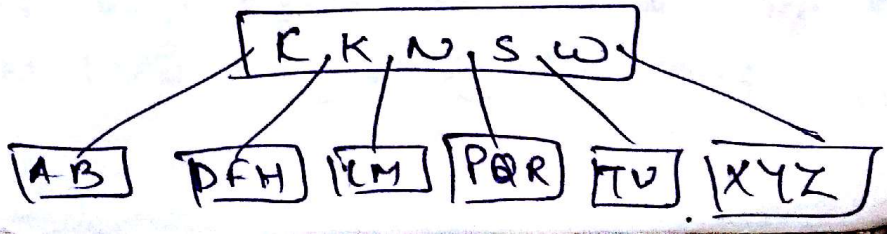
Insert Y



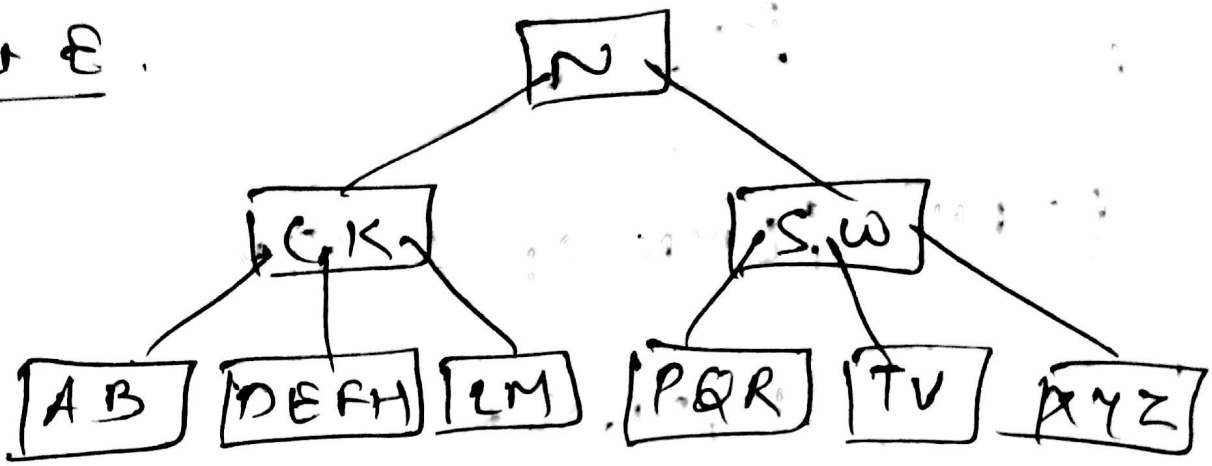
Insert D



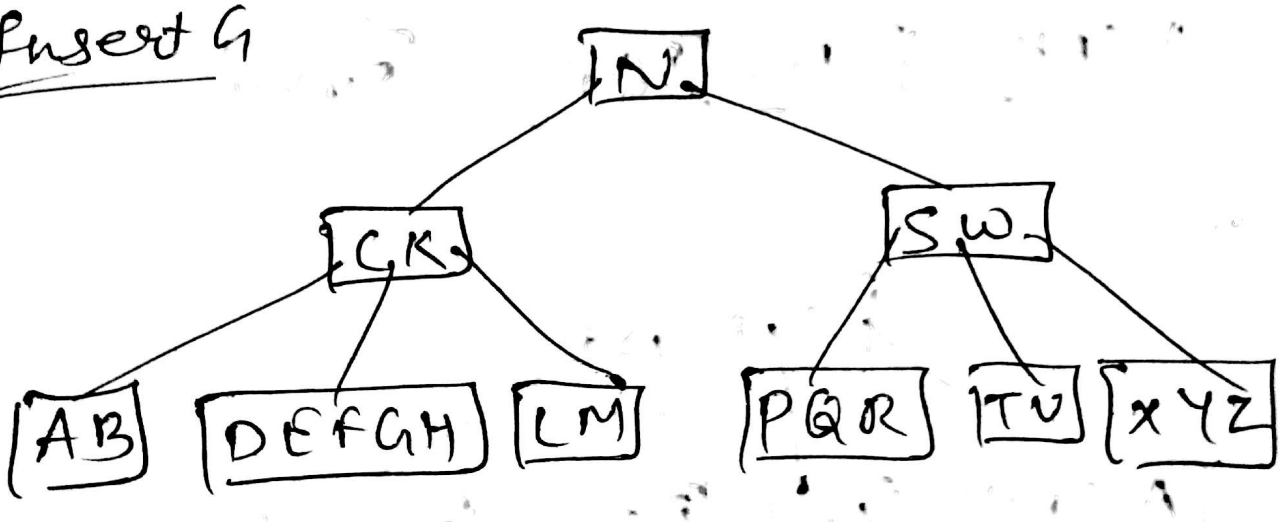
Insert Z



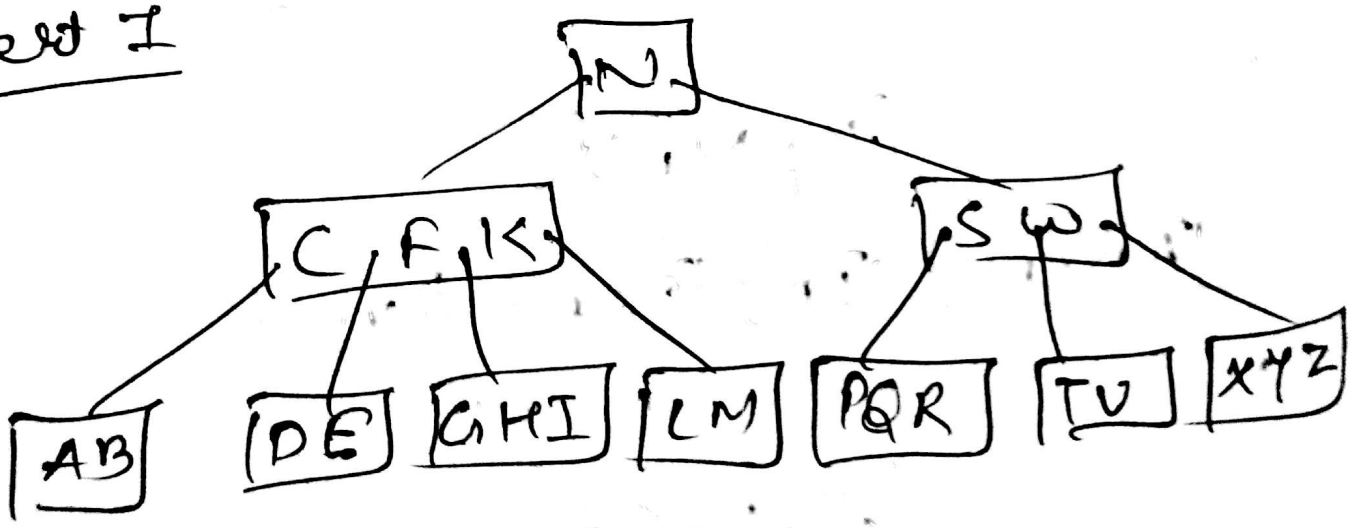
Insert E



Insert G



Insert I



Q. when Order is given: [B-tree]

$M \geq 2$.

then min. keys $= \left\lceil \frac{M}{2} \right\rceil - 1$

and Max. keys $= M - 1$

Q. Insert in Key in B-tree when order is 5:
25, 31, 37, 76, 5, 60, 38, 8, 30, 15, 35, 17, 23, 53, 27, 43, 65, 48

Sol

⇒ min. keys = 2.

Max. keys = 4

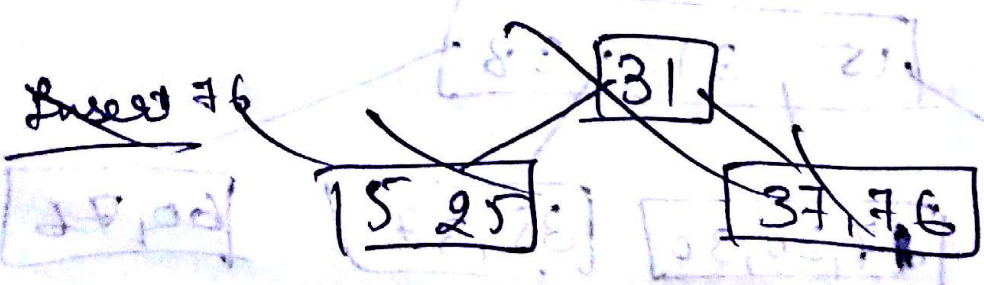
~~Insert 5~~ 5

Insert 25 | 5 25 |

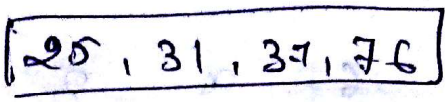
" 31 | 5 25 31 |

" 37 | 5 25 31 37 |

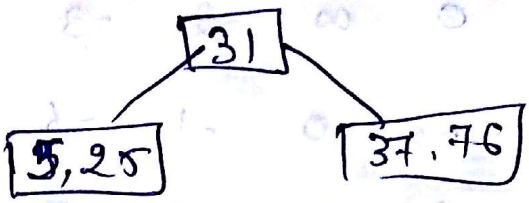
Insert 76



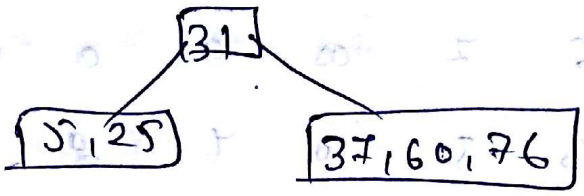
Insert 76



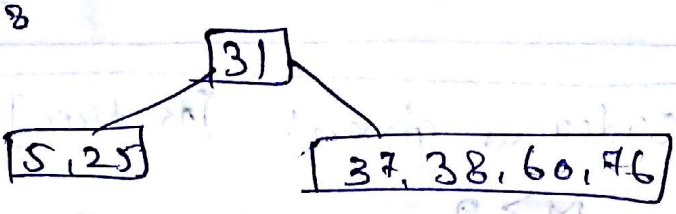
Insert 5



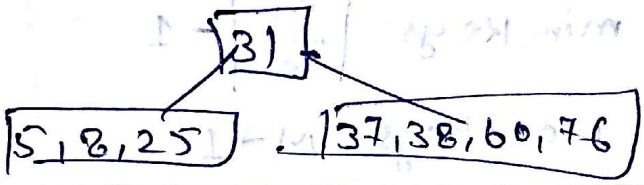
Insert 60



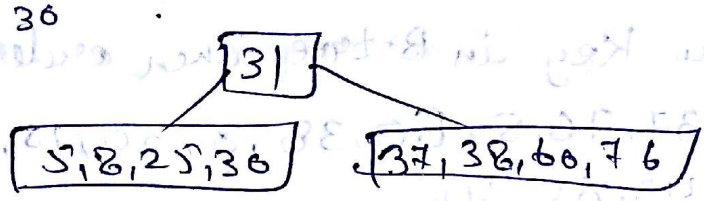
Insert 38



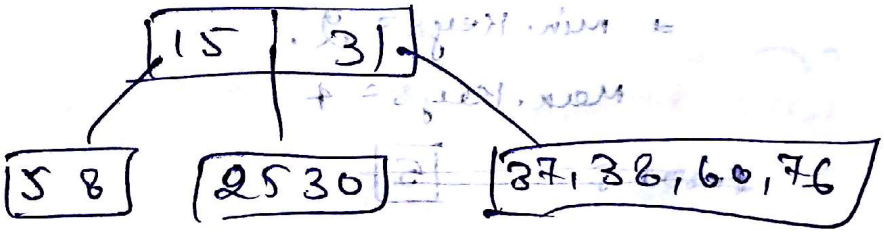
Insert 8



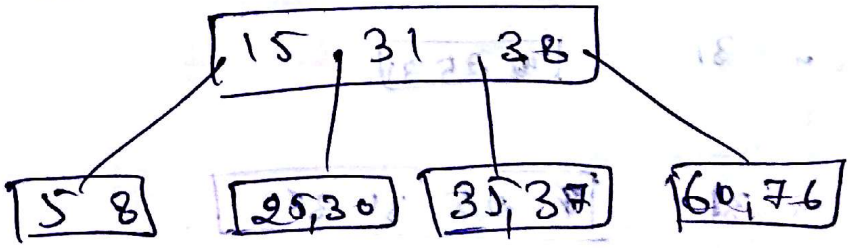
Insert 36



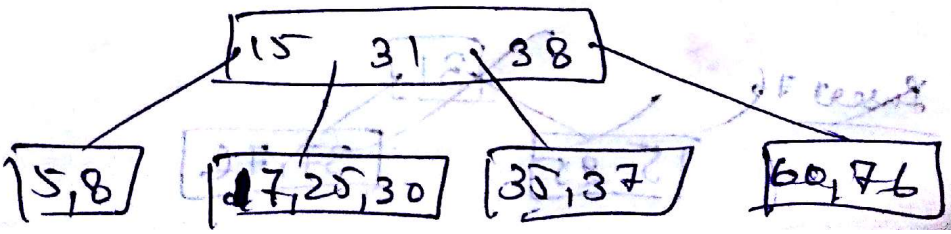
Insert 15



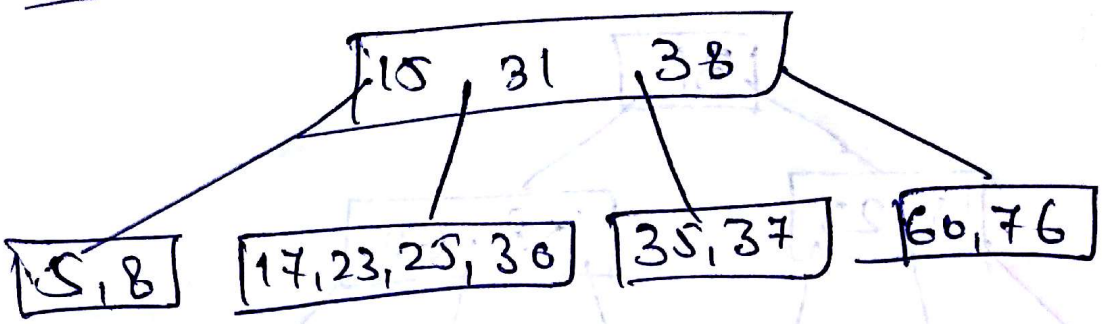
Insert 35.



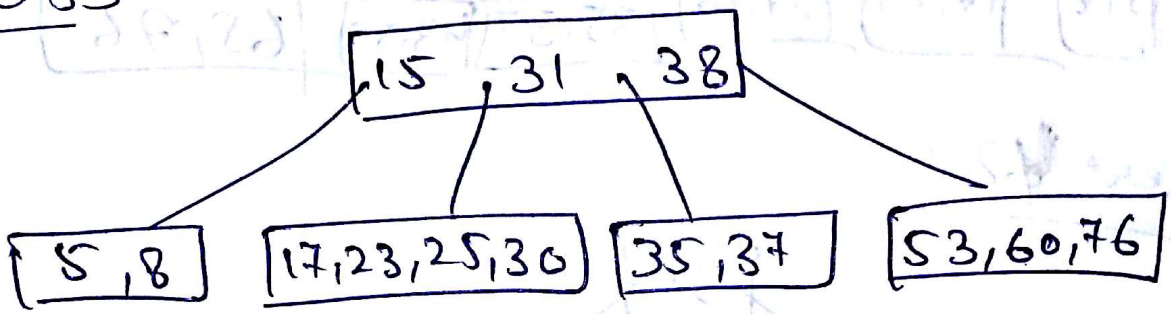
Insert 17



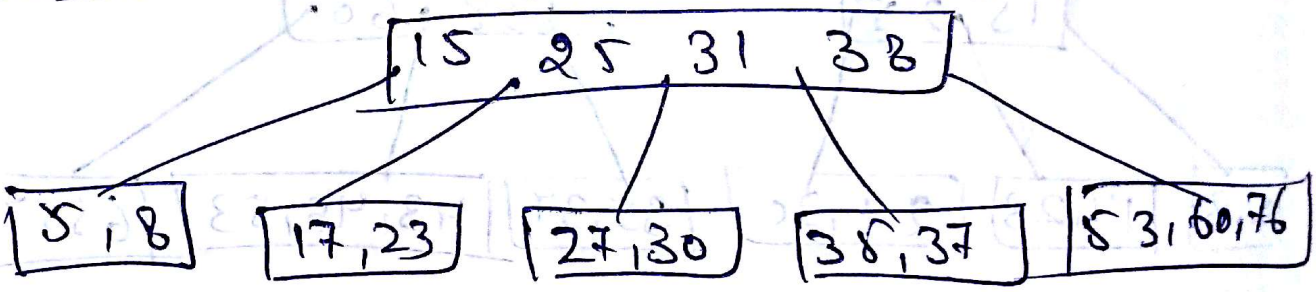
Insert 23



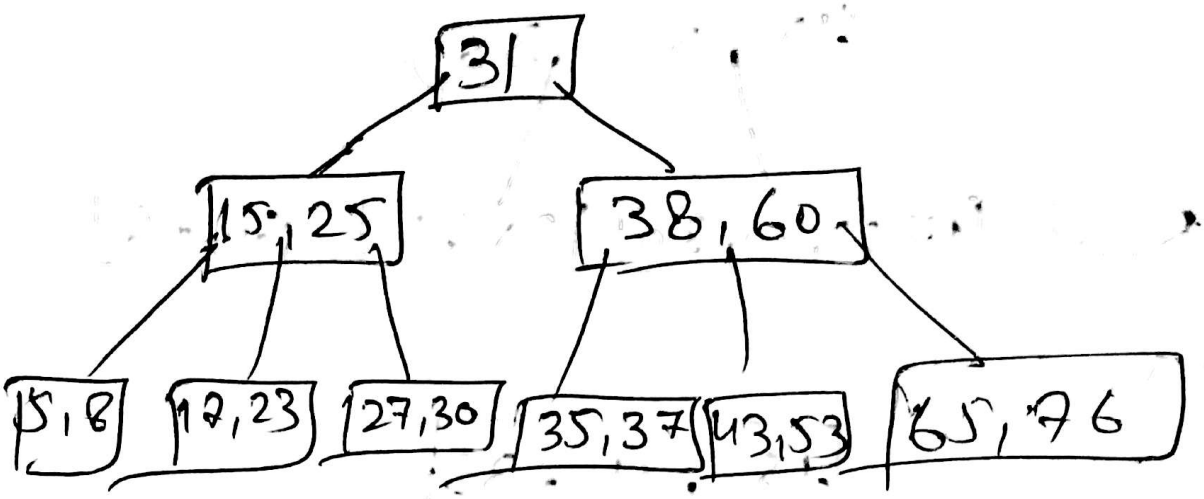
Insert 53



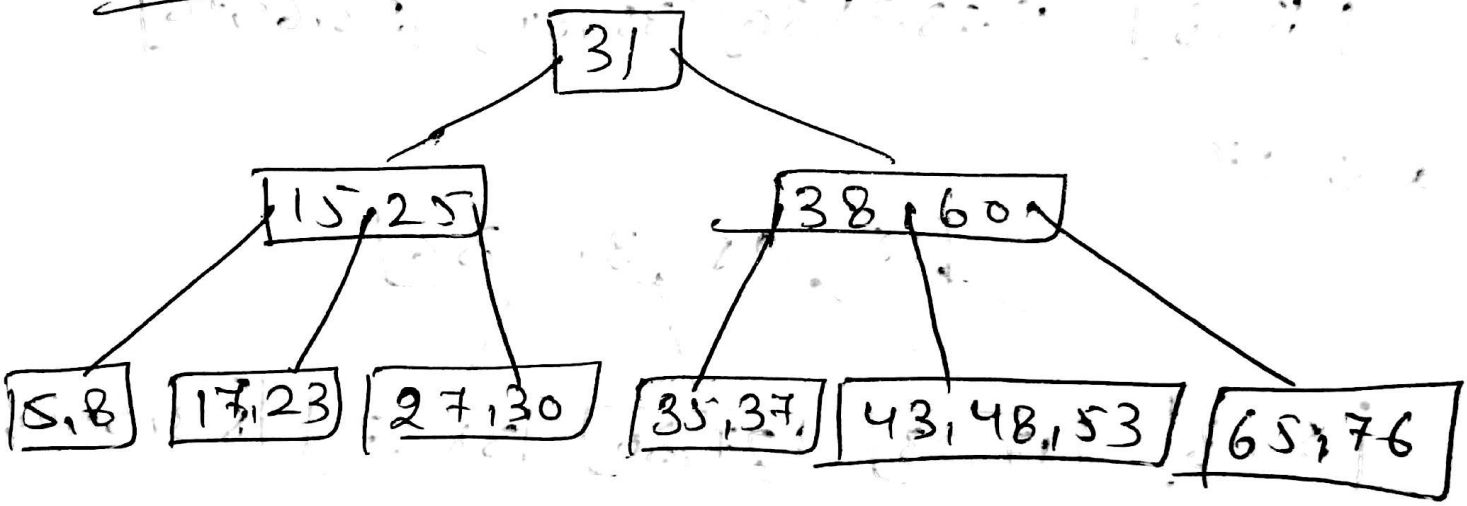
Insert 27



Insert 43 & 65

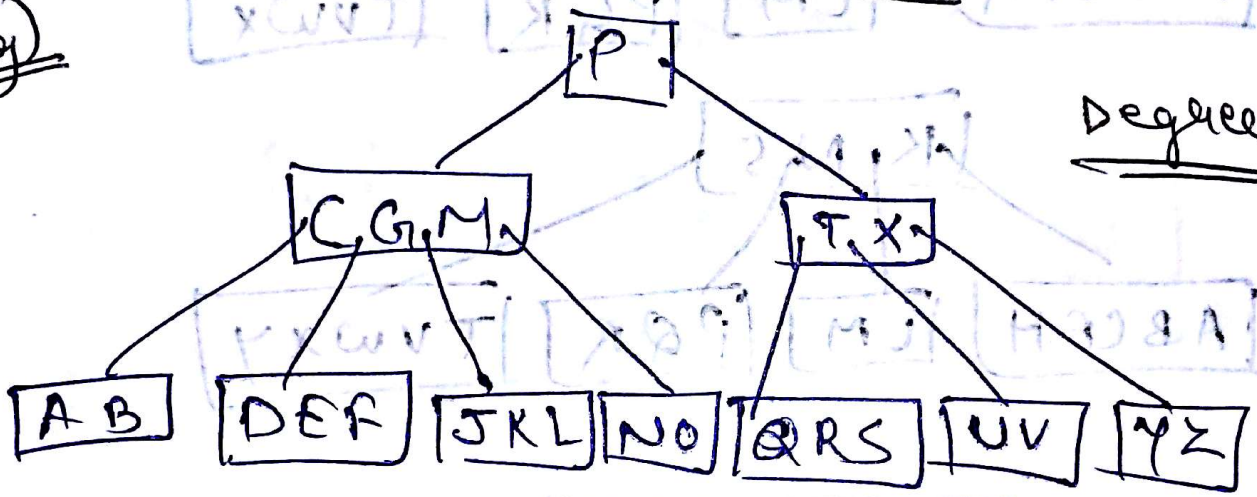


Insert 48



Deletion in B-tree :

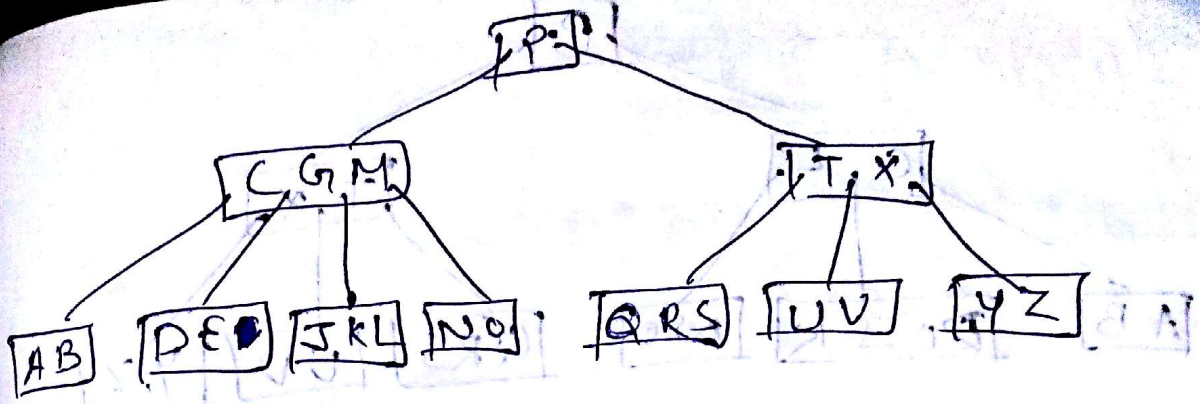
eg)



Degree = 3

case) Delete F

[If the key 'k' is in node 'x' and x is a leaf then simple delete 'k' from 'x'.]

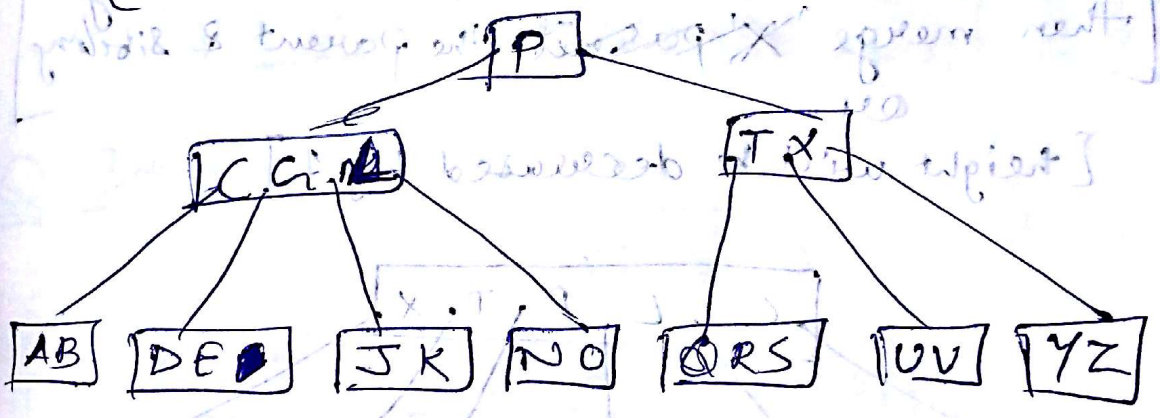


Case-2:

If the key 'k' is in node 'x' and x is internal node.

a) Delete the M

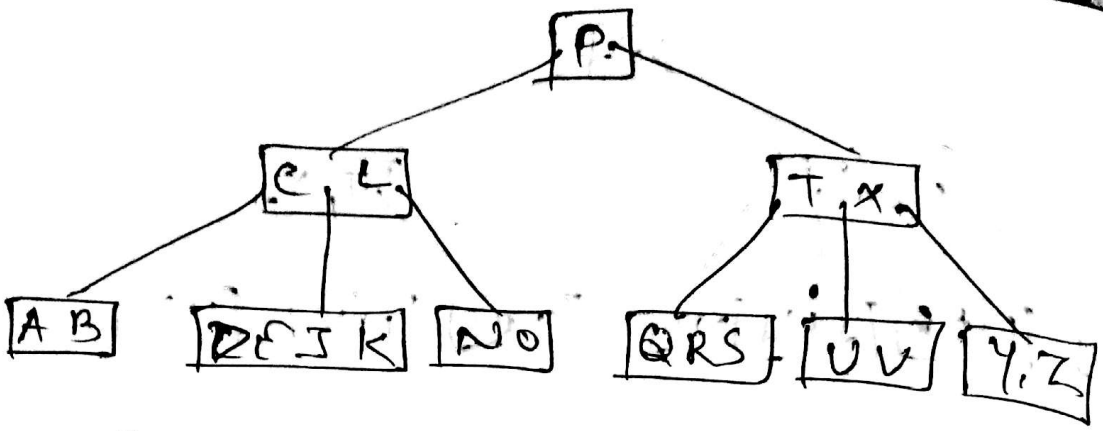
If the delete node is internal node then see the left child having 't' keys than remove predecessor of child & add that into their parent.



b) If the delete node is internal node then see the right child having 't' keys then remove the successor of child & add that into their parent

c) Delete the G

If the deleted node is in between the internal node then see the left & right child of having 't-1' keys then merge both the left & right child and remove that node.



Case-3

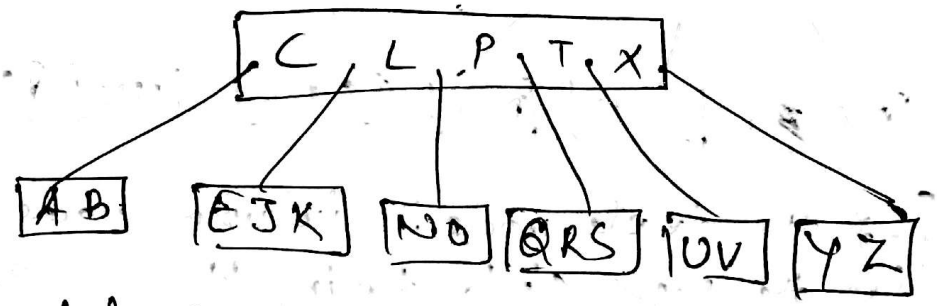
If the key 'k' is not present in internal node 'x', Determine the root $C_i(x)$ [parent] and $C_i(x)$ has only 't-1' keys.

b) Delete D :

[If parent contain 't-1' keys and check the sibiling node, if it also contain 't-1' key then merge ~~'x' parent~~ the parent & sibiling

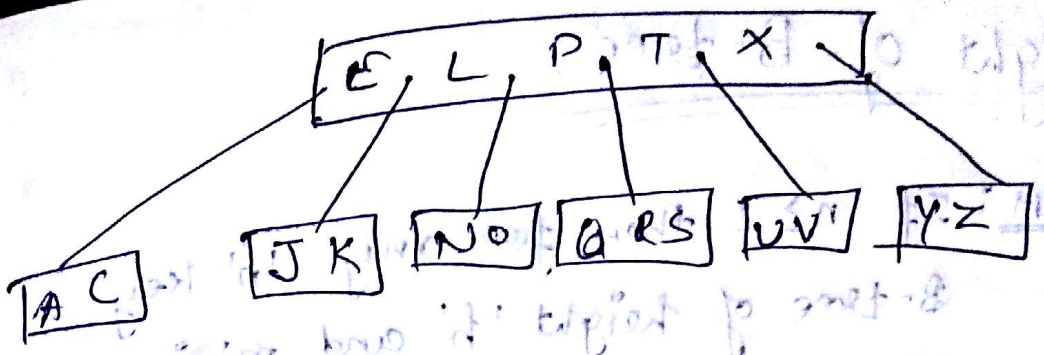
or

[height will be decreased by 1]]

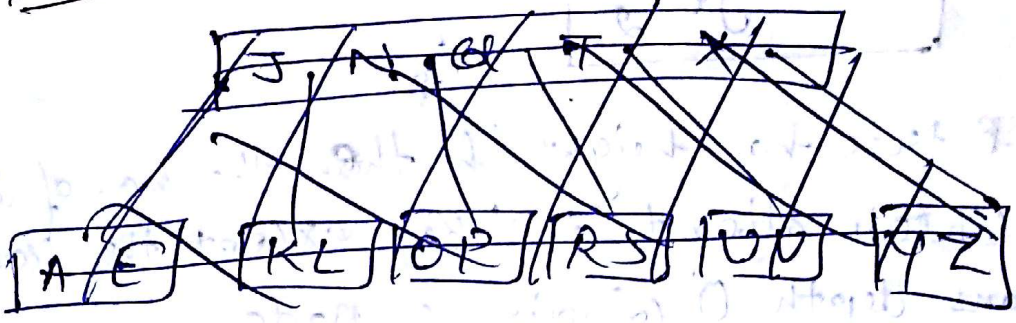


a) Delete B

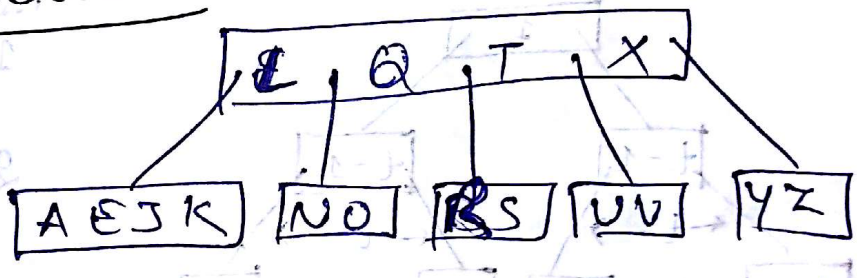
[If parent has only 't-1' key but immediate sibling has 't' key then give a extra key to the parent from child moving a key from parent left or right sibiling into x.]



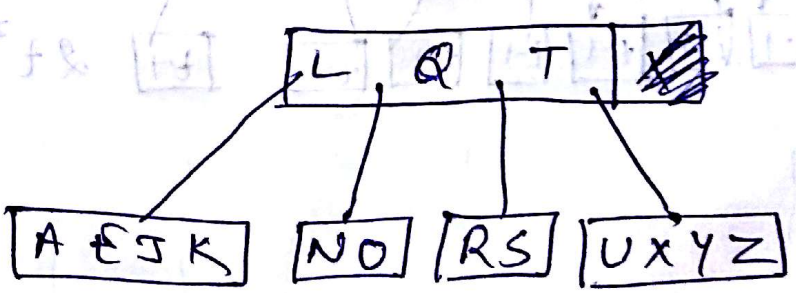
Q Delete the 'C'



Q Delete P



Q. Delete V

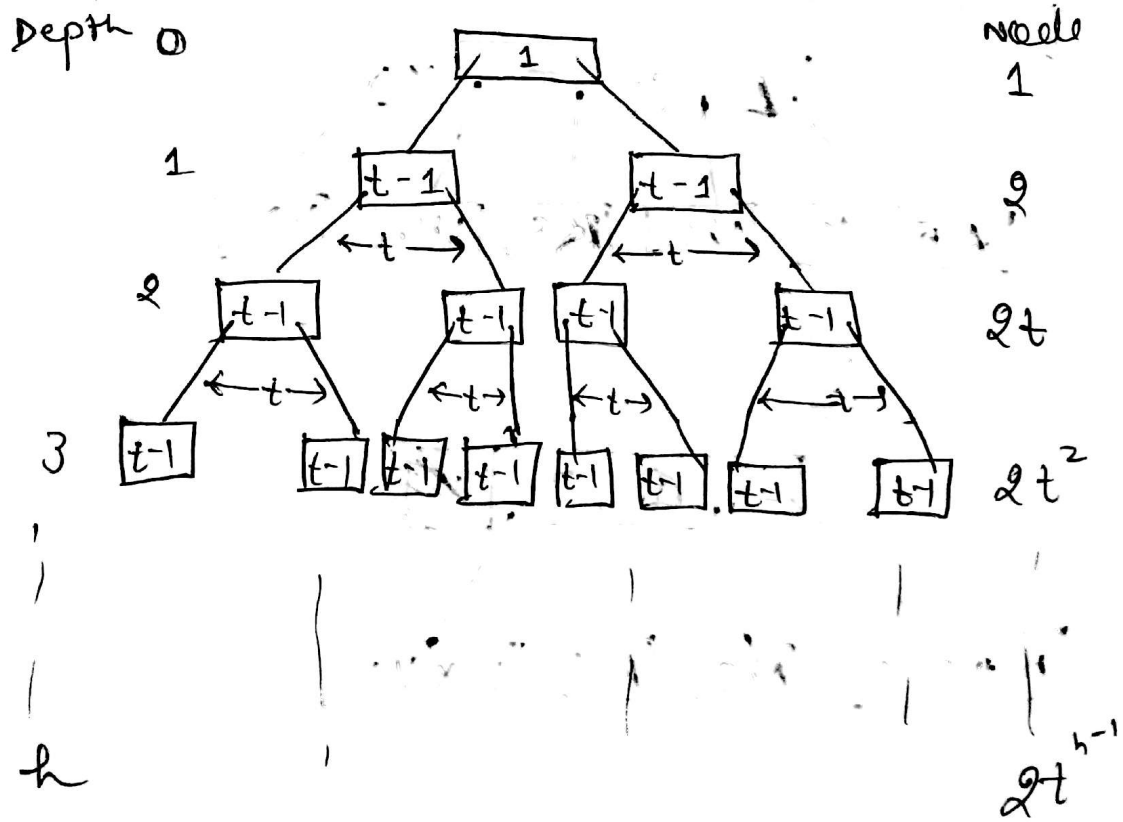


Height of B-tree:

THEOREM: If $n \geq 1$ then for any 'n' key, B-tree of height 'h' and minimum degree $t \geq 2$ then the height of the tree is

$$h \leq \log_t \frac{n+1}{2}$$

If a B-tree has height 'h' then the no. of it node contain atleast 't-1' key except the root.
It means depth '0' contain 1 node.
and depth '1' contain 2 node
depth '2' contain $2t$ node
depth '3' contain $2t^2$ node ... and so on until the depth 'h' contain $2t^{h-1}$ node.



$$n \geq 1 + (t-1)\left[2 + 2t + 2t^2 + \ldots + 2t^{h-1}\right]$$

$$n \geq 1 + 2(t-1)\left[1 + t + t^2 + t^3 + \ldots + t^{h-1}\right]$$

$$n \geq 1 + 2(t-1)\left[\frac{t^h - 1}{t-1}\right] \quad \ldots (GP)$$

$$n \geq 1 + 2(t^h - 1)$$

$$n \geq 2t^h - 1$$

$$\frac{n+1}{2} \geq t^h$$

$$t^h \leq \frac{n+1}{2}$$

$$h \log t \leq \log\left(\frac{n+1}{2}\right)$$

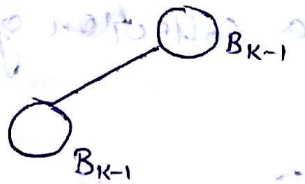
$$h \leq \frac{\log\left(\frac{n+1}{2}\right)}{\log t}$$

$$\boxed{h \leq \log_t \frac{n+1}{2}}$$

Binomial Heap

Binomial tree: It is an ordered tree. It is indicated by B_K where K is order. $\boxed{B_K = B_{K-1} + B_{K-1}}$

i.e.,

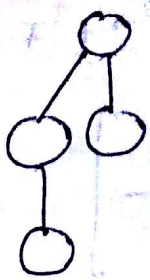


No. of nodes $= 2^K$

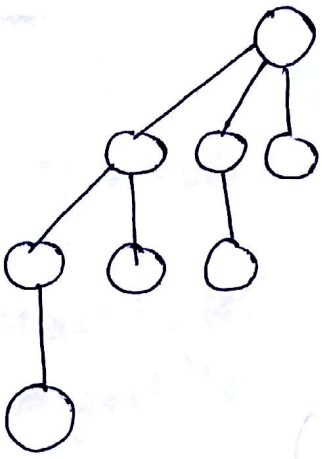
B_0

B_1

B_2



B_3



Properties of Binomial tree:

1) It has 2^k nodes.
2) Height of tree is 'k'.
3) There are exactly $\binom{k}{i}$ nodes at depth i.

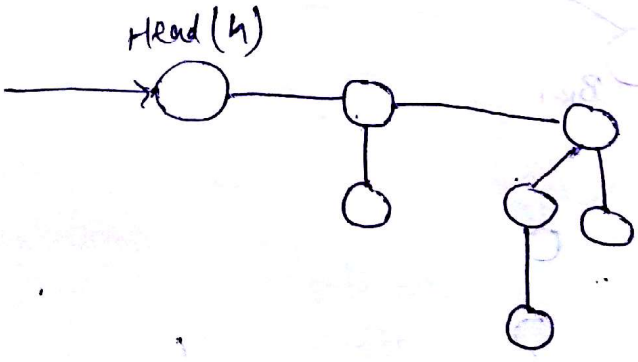
ie, $$\frac{k!}{(k-i)!\,i!}$$

4) The root has degree 'k'.

* Binomial heap:

Binomial heap is a collection of binomial tree.

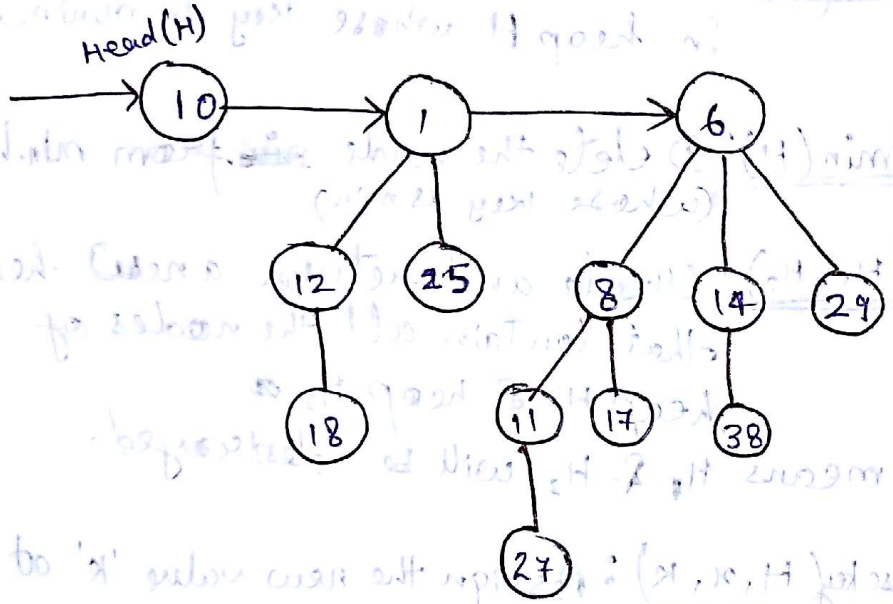
Eg)



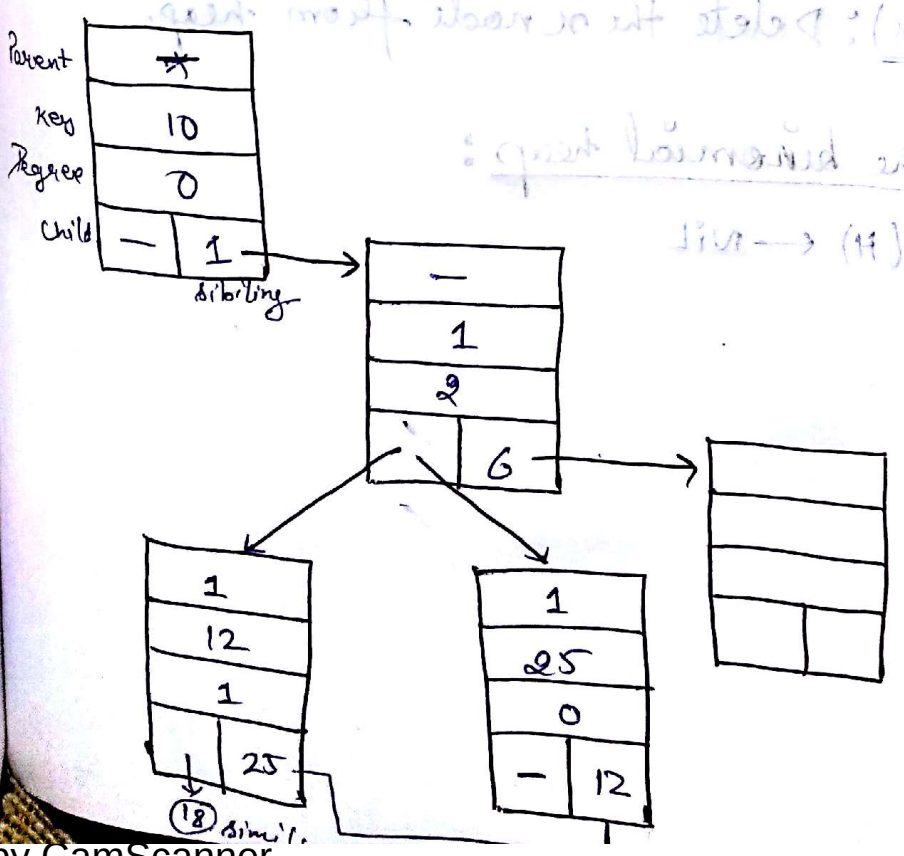
Properties of Binomial heap:

1) It contain the min. heap property
2) For any non-negative integer 'k' (order) there is atmost one binomial tree whose root has degree 'k'.

Representation of Binomial heap



Graphical representation:



Operation of Binomial heap

There are seven operations —

1) **Make heap**: Create the heap which contains no elements.

2) **Insert (H, x)**: Insert a node 'x' in binomial he

3) **Minimum (H)**: Returns a pointer to the n in heap H whose key is minimum

4) **Extract min (H)**: Delete the node from min-h (whose key is min)

5) **Union (H_1, H_2)**: Create and return a new he that contain all the nodes of heap H_1 & heap H_2 Ot means H_1 & H_2 will be destroyed

6) **Decrease key (H, x, k)**: Assign the new value 'k' to node x

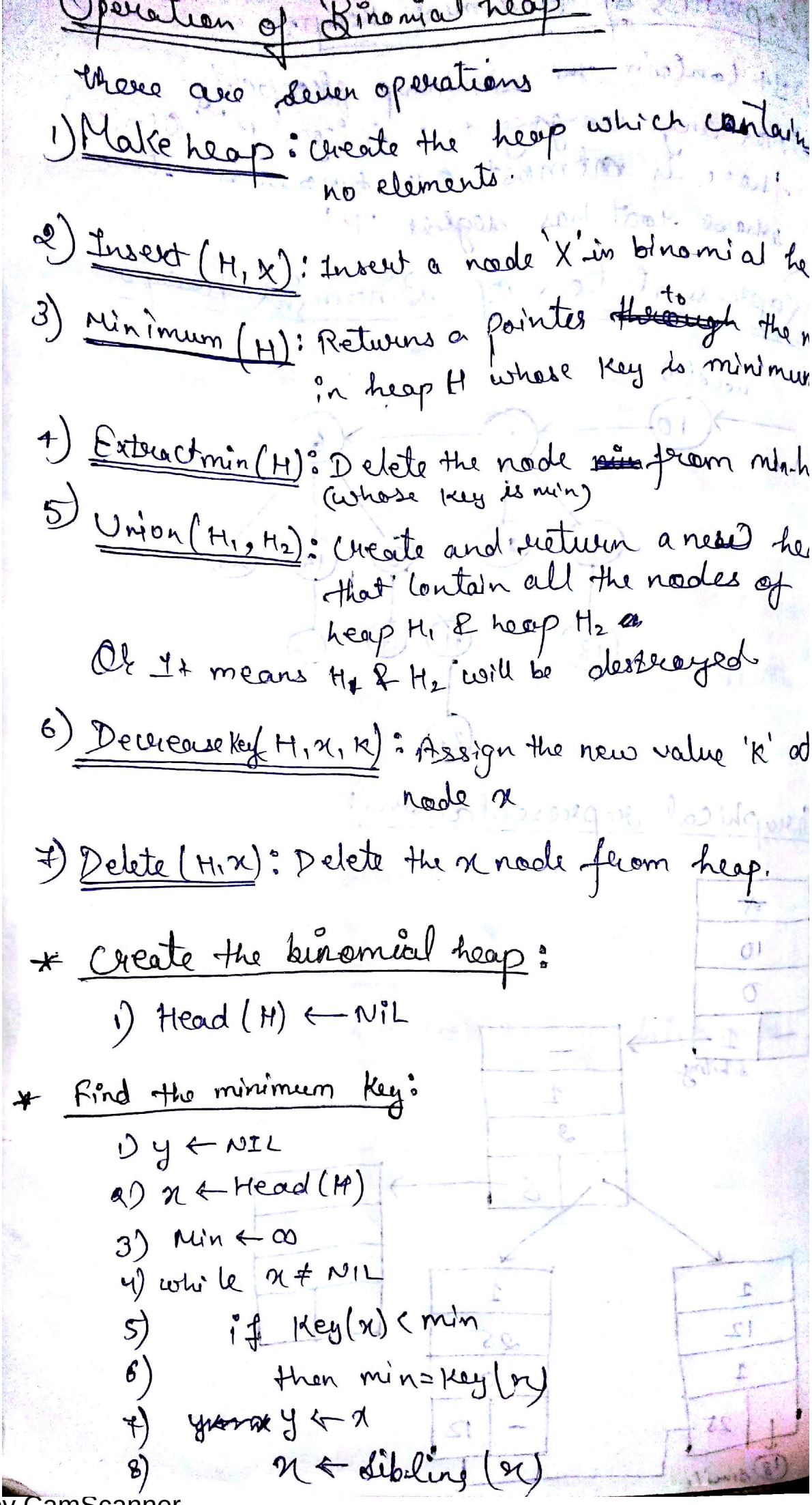
7) **Delete (H, x)**: Delete the x node from heap.

* **Create the binomial heap**:

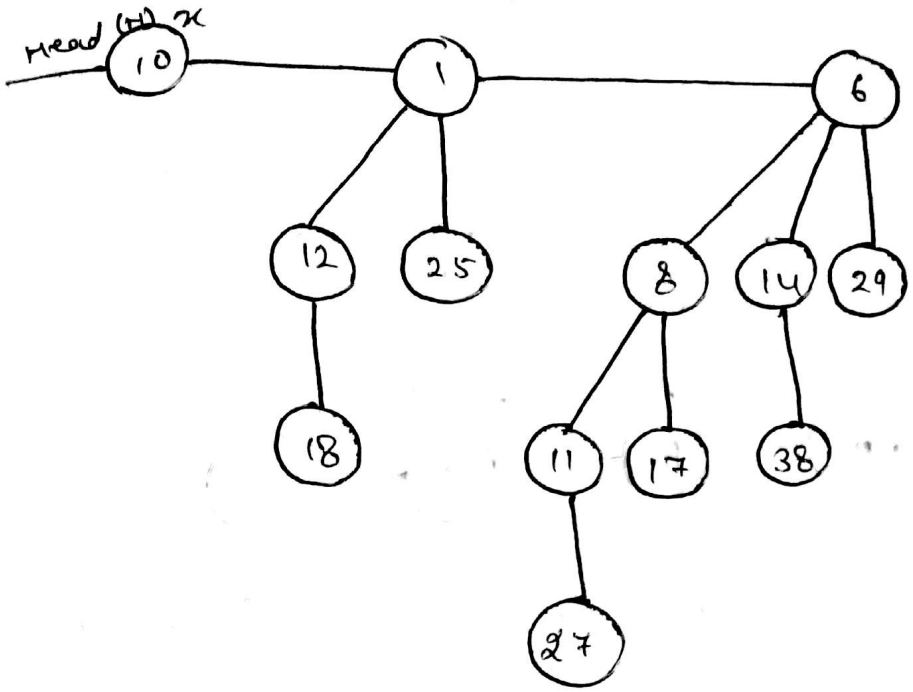
1) Head (H) ← NIL

* **Find the minimum key**:

1) y ← NIL
2) x ← Head (H)
3) Min ← ∞
4) while x ≠ NIL
5) if key(x) < min
6) then min = key(x)
7) y ← x
8) x ← sibling (x)

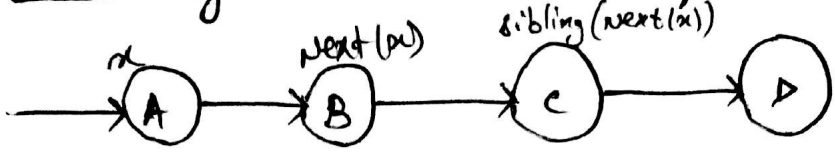


(9) return y



1) $y \leftarrow NIL$
2) $x \leftarrow Head(H)$
3) $min \leftarrow \infty$
4) while $x \neq NIL$ (true)
   5) if $10 < \infty$
      6) $min \leftarrow key(x)$ ⇒ $min \leftarrow 10$
   7) $y \leftarrow Head(H)$
   8) $x \leftarrow sibling(x)$ ⇒ $x \leftarrow 1$ (1 is key)
4) while $x \neq NIL$ (true)
   5) if $1 < 10$
      6) $min \leftarrow key(x)$ ⇒ $min \leftarrow 1$
   7) $y \leftarrow 1$
   8) $x \leftarrow sibling(1)$ ⇒ $x \leftarrow 6$
4) while $x \neq NIL$ (true)
   5) $6 < 1$
   7) $y \leftarrow 6$
   8) $x \leftarrow NIL$
4) while $x \neq NIL$ (false.

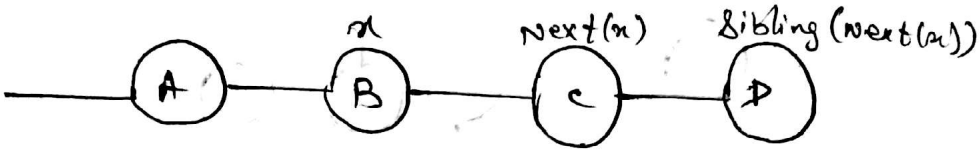
Union of two binomial heap:



Case-1: It occurs when degree of 'x' ≠ degree of next(x)

i.e, degree(x) ≠ degree(next(x))

then forward the x.



Case-2: if degree(x) = degree(next(x)) = degree(sibling(next(x)))

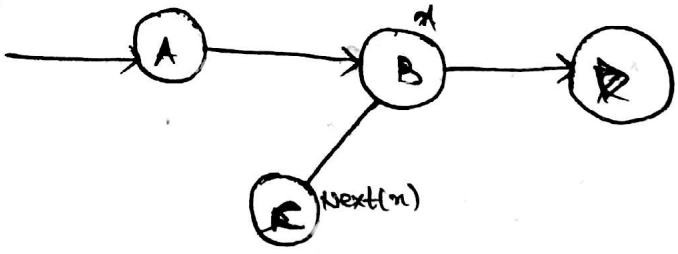
then forward the x.

case-3: ~~Case-4~~: It occurs when

degree(x) = degree(next(x)) ≠ degree(sibling(next(x)))

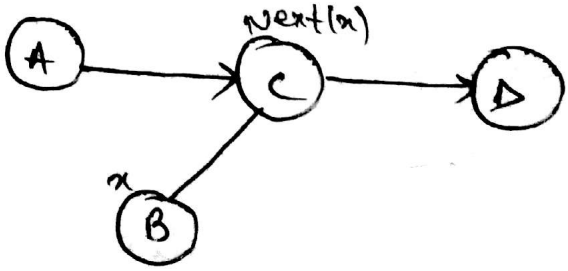
Case-3: it occurs when key(x) ≤ key(next(x))

then:



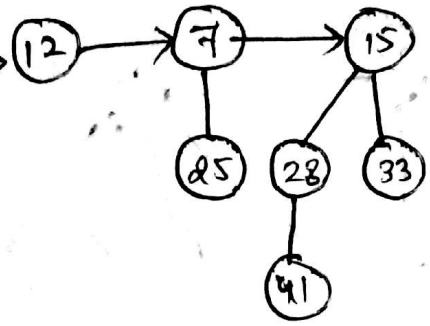
Case-4: It occurs when key(x) ≥ key(next(x))

then

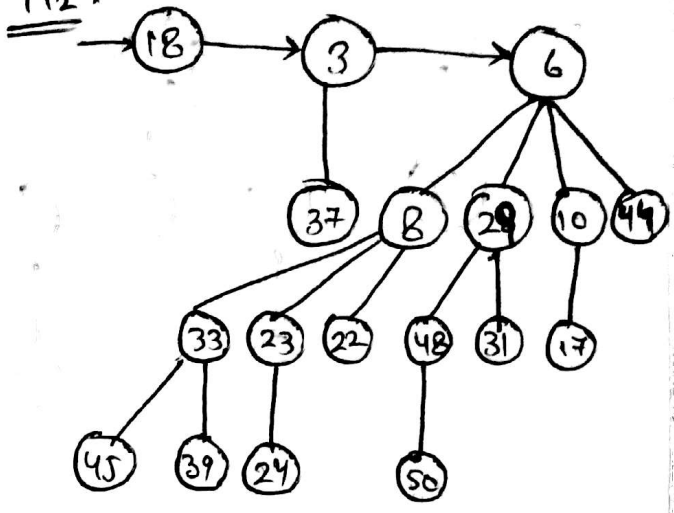


eg)

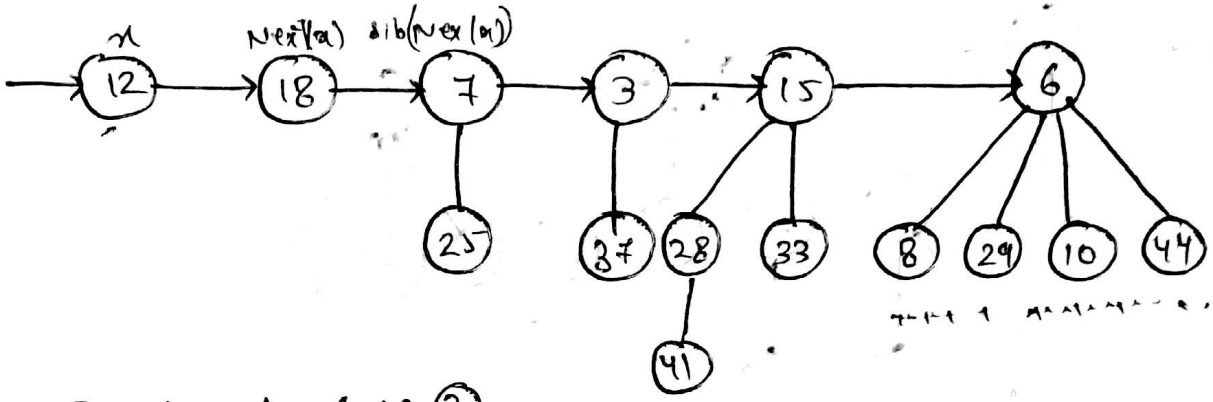
H_1 :



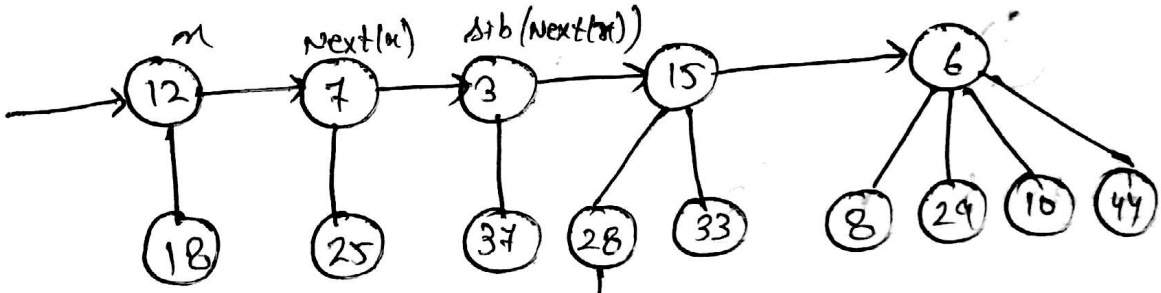
H_2 :



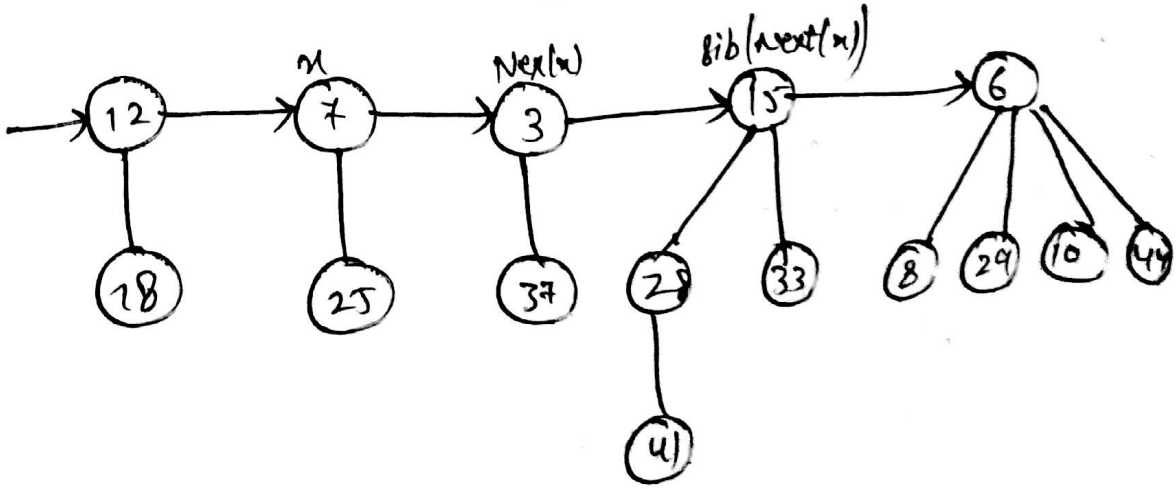
Sol :



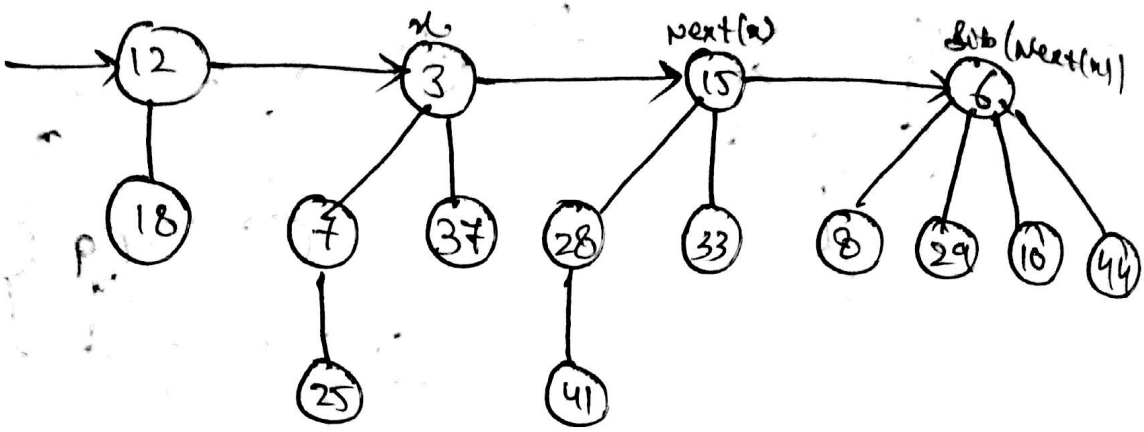
~~Case 3:~~ Acc. to case ③



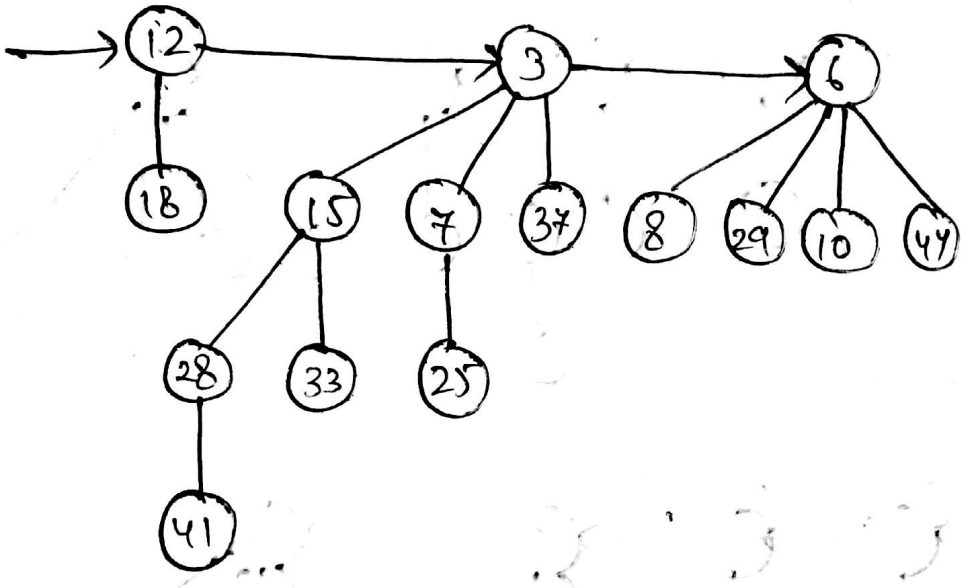
Acc. to case. ②



Acc to Case 4)



Acc. to Case 3:



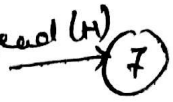
* Insert a node:

Binomial-Heap-insert (H, x)

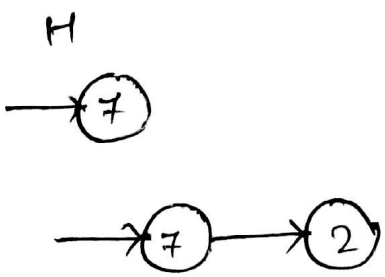
1. H' ← Make a Binomial Heap
2. P(x) ← NIL
3. child (x) ← NIL
4. degree (x) ← 0
5. Sibling (x) ← NIL
6. Head (H') ← x
7. Binomial Heap Union (H, H')

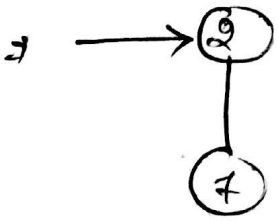
Q| Insert the element in the binomial heap.
7, 2, 4, 17, 1, 11, 6, 8, 15, 10, 20

Sol: Insert 7.

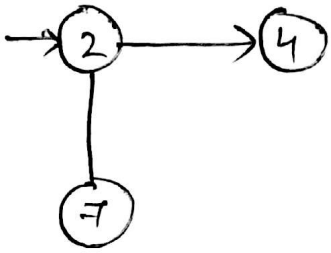


Insert 2

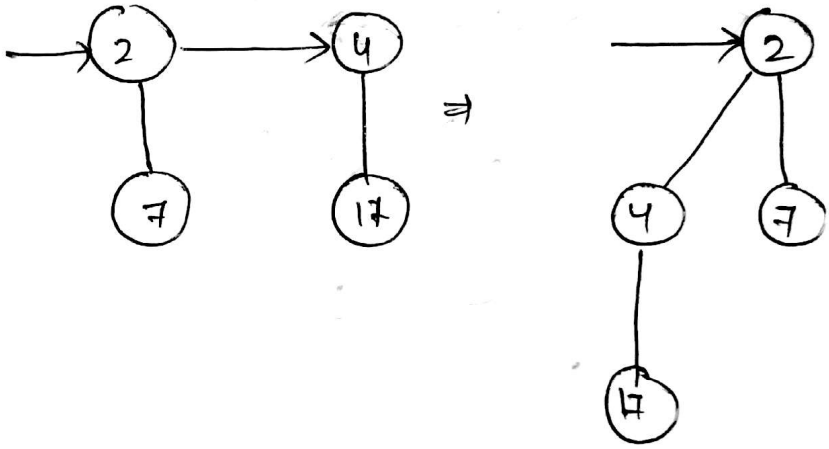




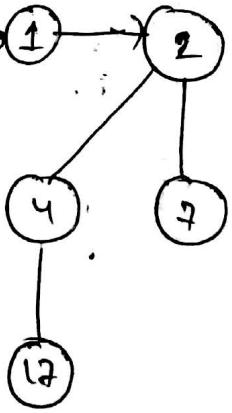
Insert 4



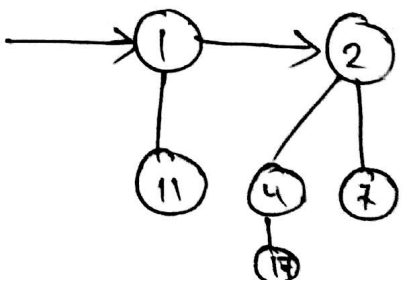
Insert 17



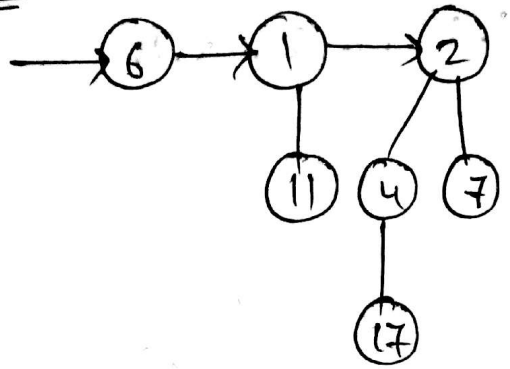
Insert 1



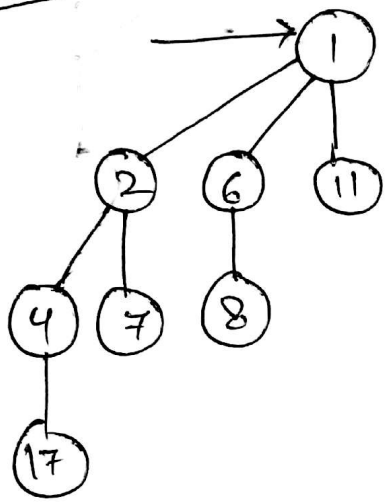
Insert 11



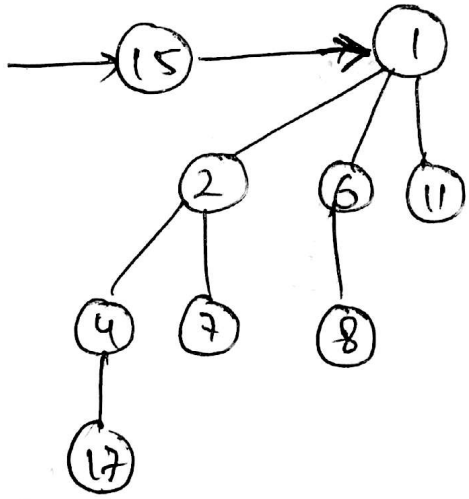
Insert 6



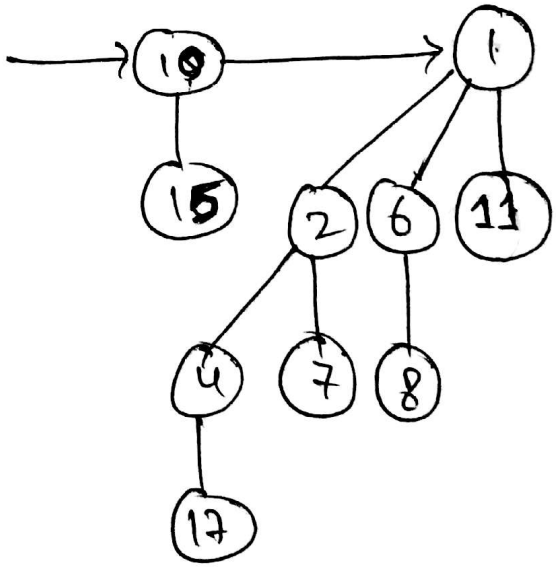
Insert 8



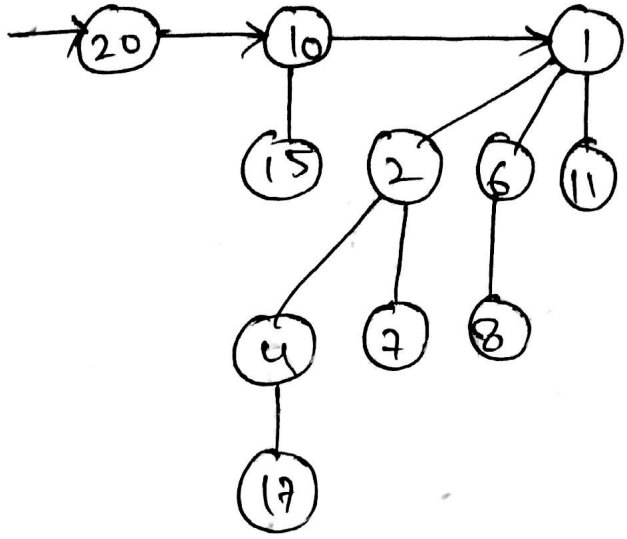
Insert 15



Insert 10



Insert 20

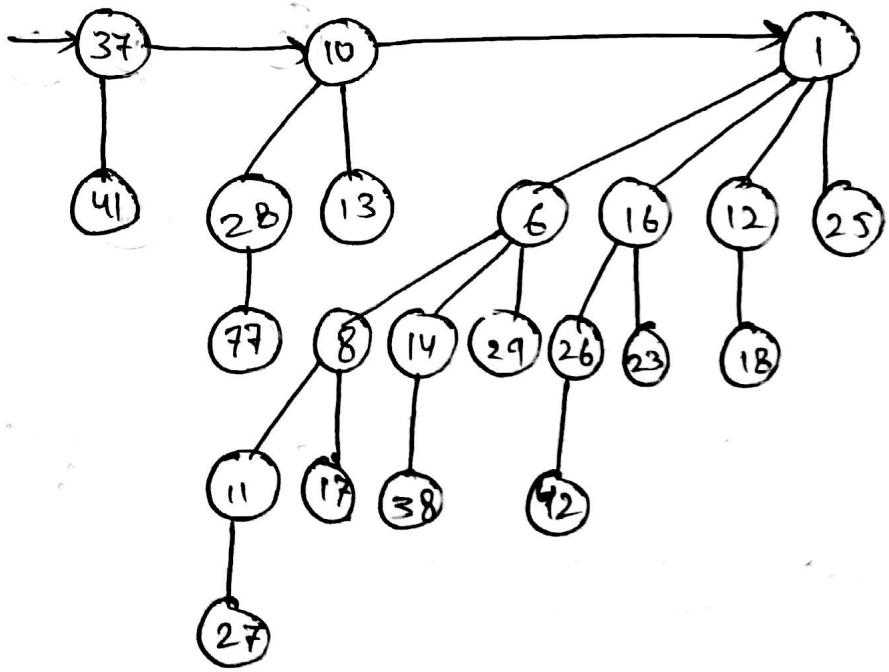


* Extract the node with minimum key:

Binomial-Heap-extract-min(H)

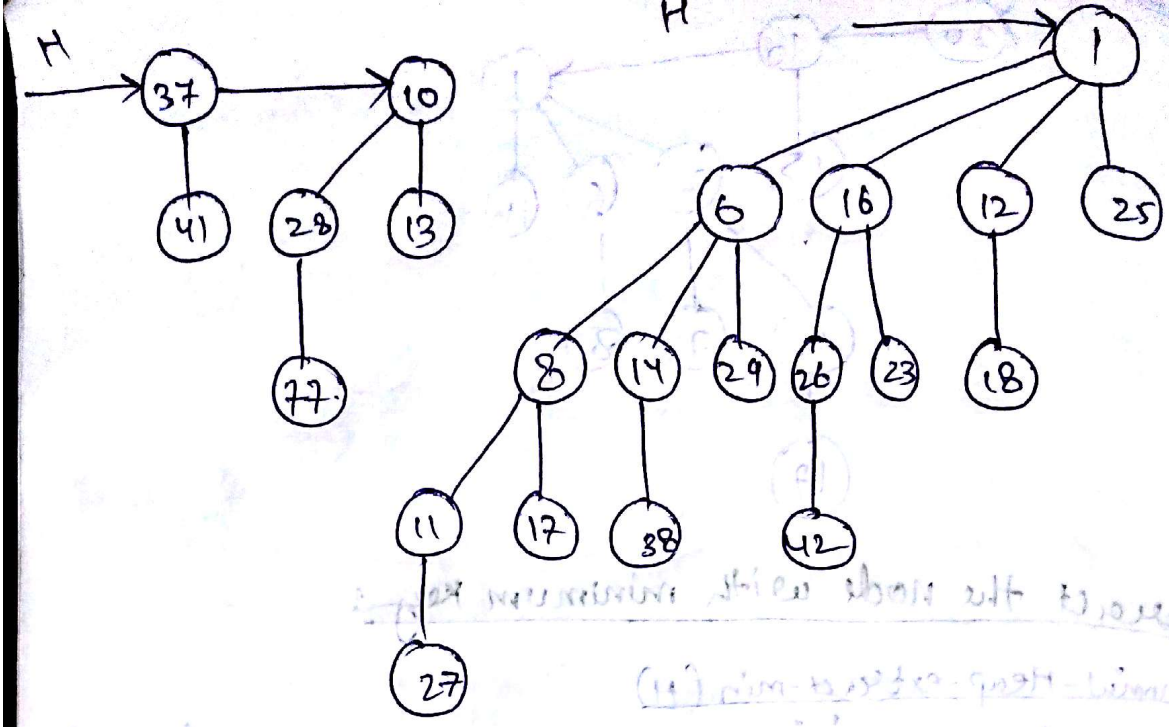
1. Find the root x with the minimum key in H & remove from the heap
2. H' ← make a new binomial heap.
3. Reverse the order of Child(x) and set head H'
4. Binomial-heap-Union(H, H')
5. return x

Eg)



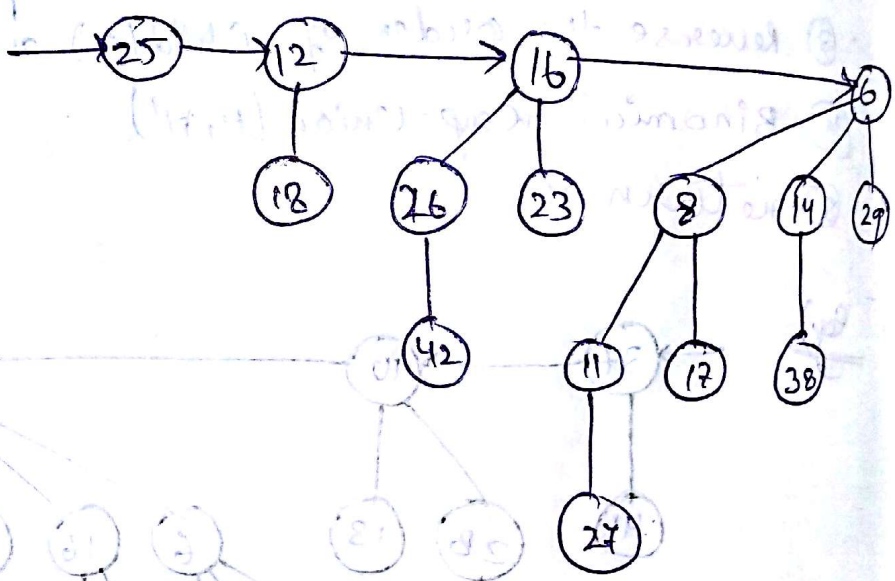
Sol:

1. Find the minimum Key, and break heap.
2. ~~H' ← make a new binom~~

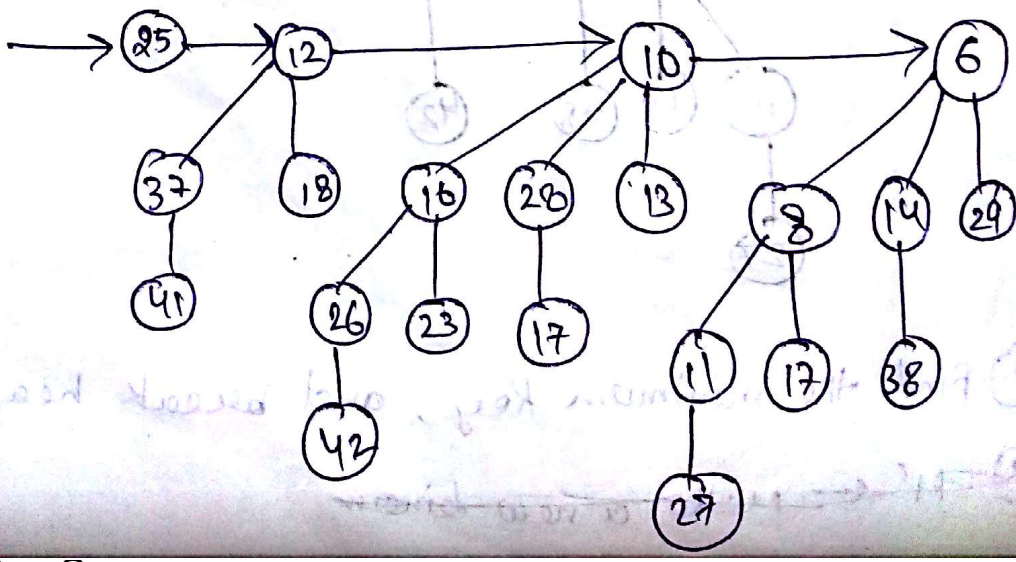


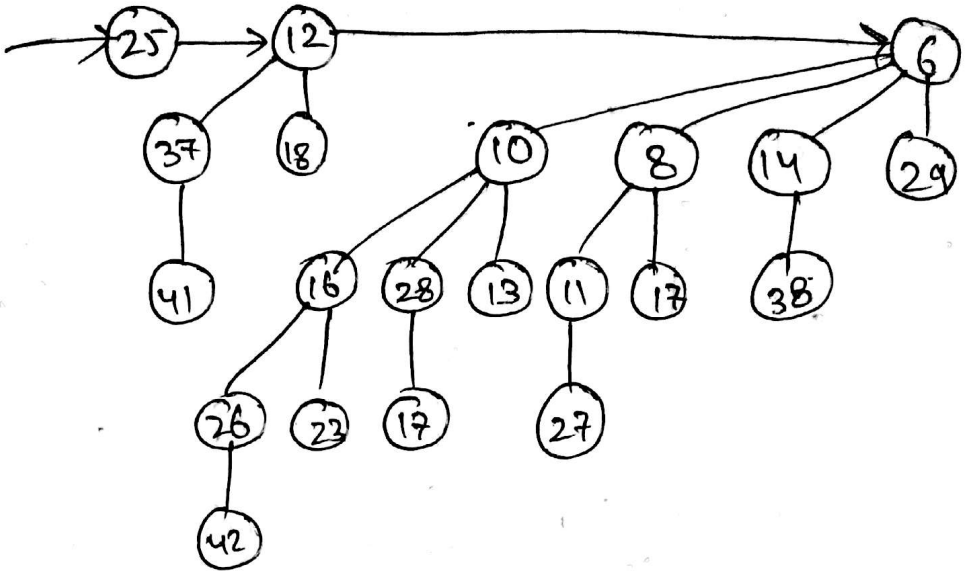
(2) H' ← make a new binomial heap.

(3)



(4) union:

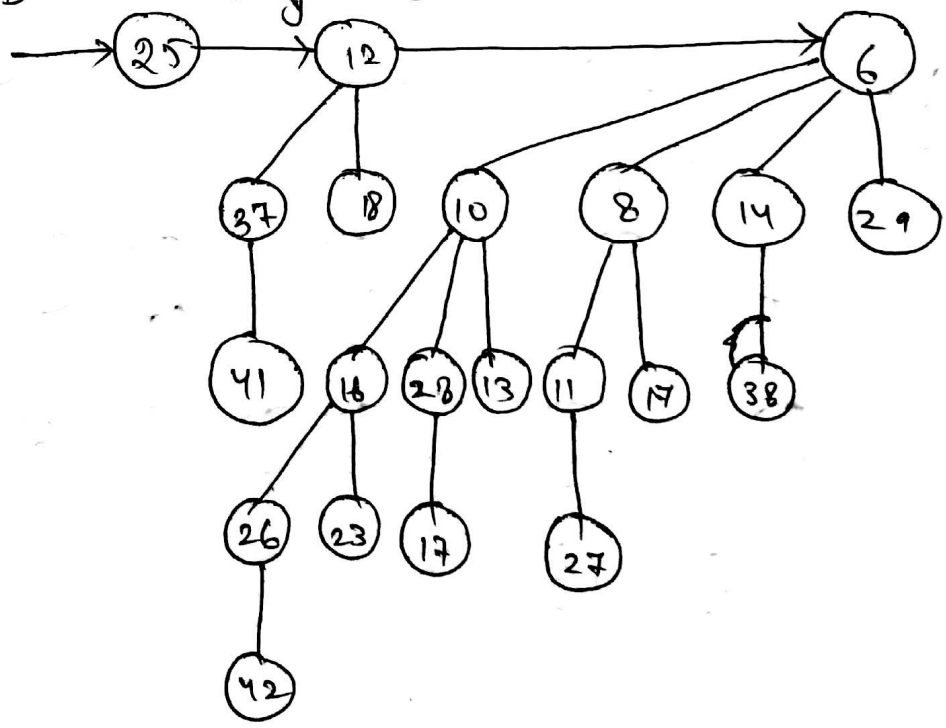




* Binomial-Heap-Decrease (H, x, k)

1) if $k \geq key(x)$
2) then error.
3) $key(x) \leftarrow k$
4) $y \leftarrow x$
5) $z \leftarrow parent(y)$
6) while $z \neq NIL$ and $key(y) < key(z)$
7) $key(y) \leftrightarrow key(z)$
8) $y \leftarrow z$
9) $z \leftarrow parent(y)$

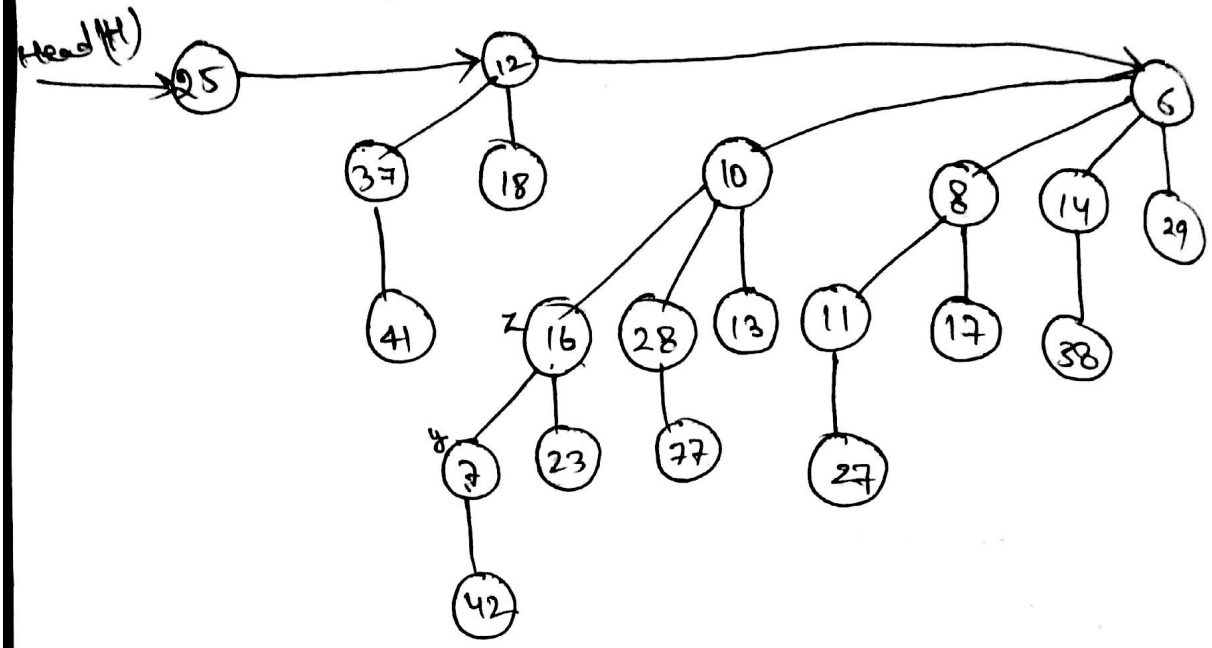
eg:- Decrease key '26' to '7'.



Sol

① if $7 \geq Key(x)$ → $7 \geq 26$ (false)

③ $Key(x) \leftarrow 7$



④ $y \leftarrow x$

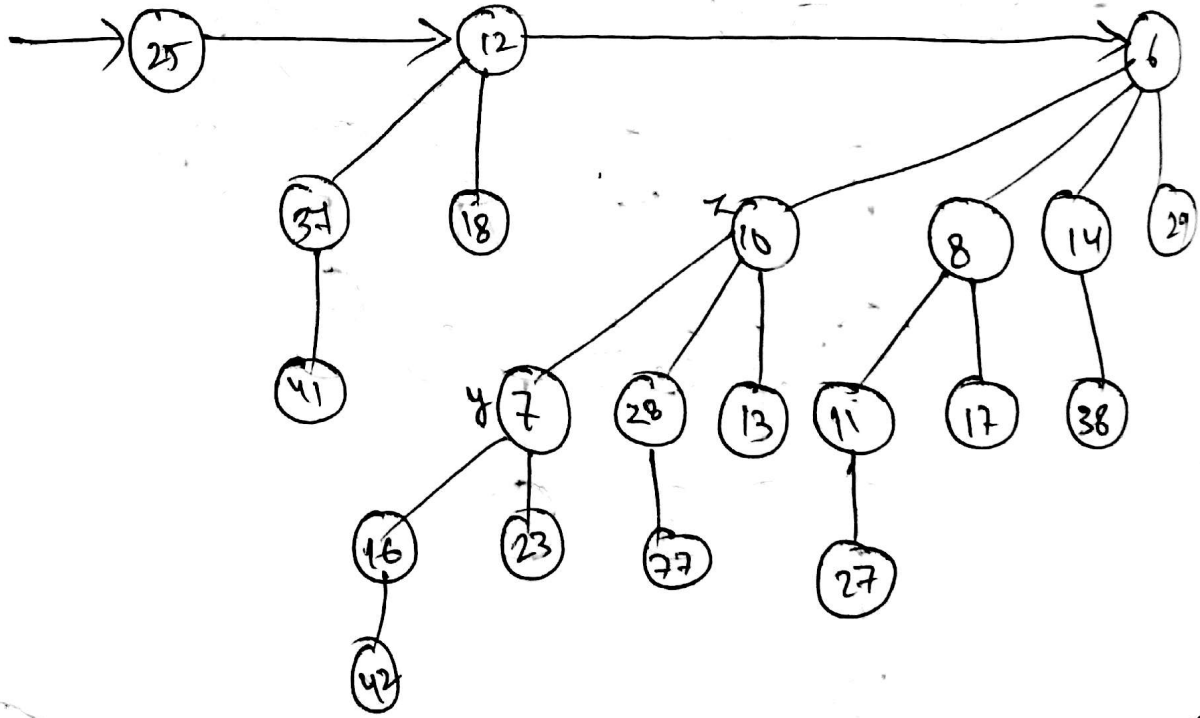
⑤ $z \leftarrow parent(x)$

⑥ while $z \neq NIL$ and $Key(y) < Key(z)$ (true)

⑦ $Key(y) \leftrightarrow Key(z)$ $z \leftrightarrow 16$

⑧ $y \leftarrow z$

⑨ $z \leftarrow parent(y)$

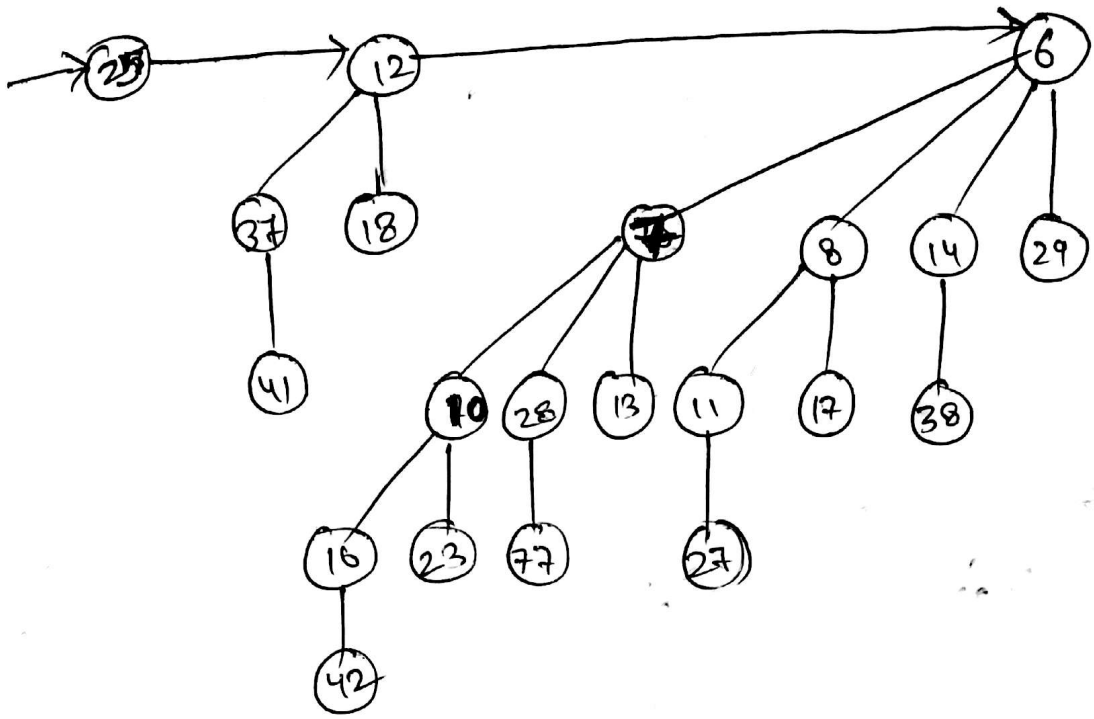


(6) while $z \neq NIL$ and $key(y) < key(z)$ (true)

(7) $key(y) \leftrightarrow key(z)$ $7 \leftrightarrow 10$

(8) $y \leftarrow z$.

(9) $z \leftarrow parent(y)$

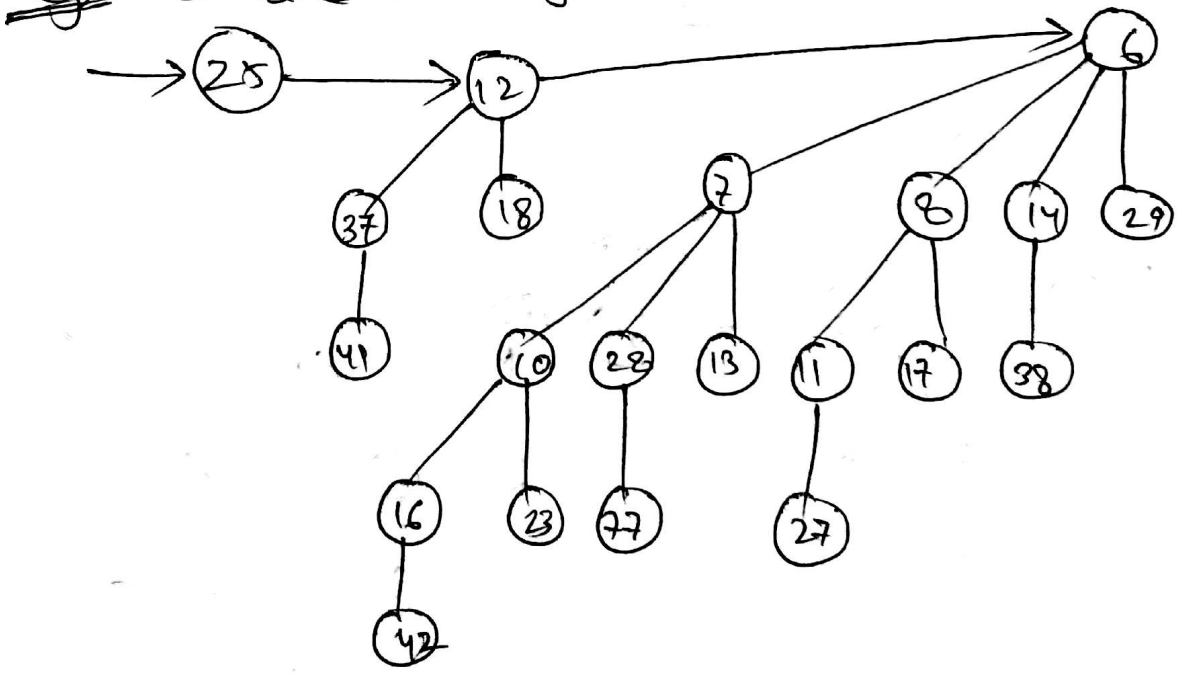


(6) while $z \neq NIL$ and $key(y) < key(z)$ (false)

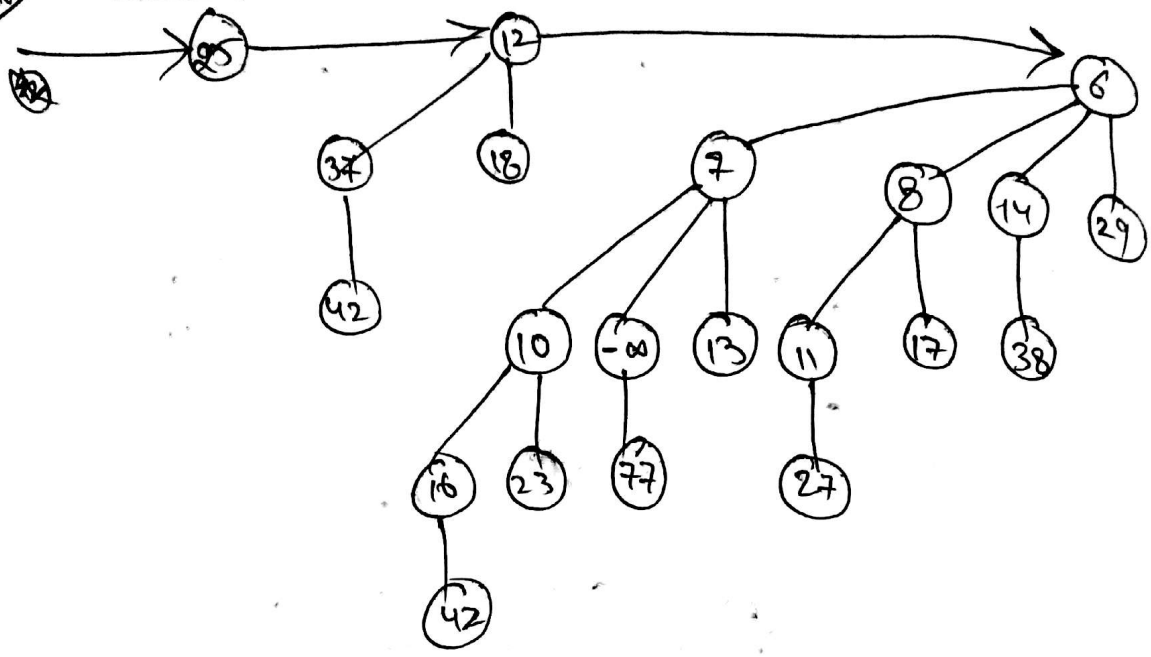
Binomial-Heap-delete (H, x)

1) Binomial-Heap-decreaseKey$(H, x, -\infty)$
2) Binomial-Heap-extract-min(H)

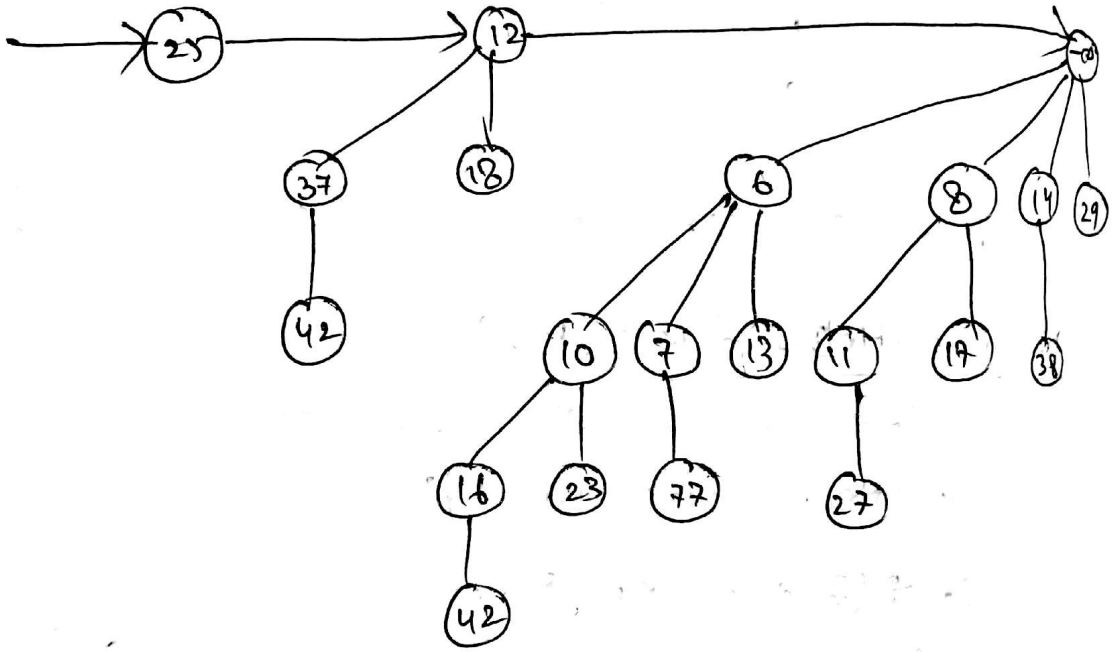
eg) Delete the Key '28'.



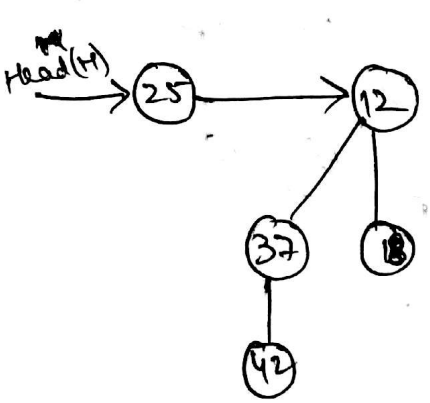
(8) Decrease key:

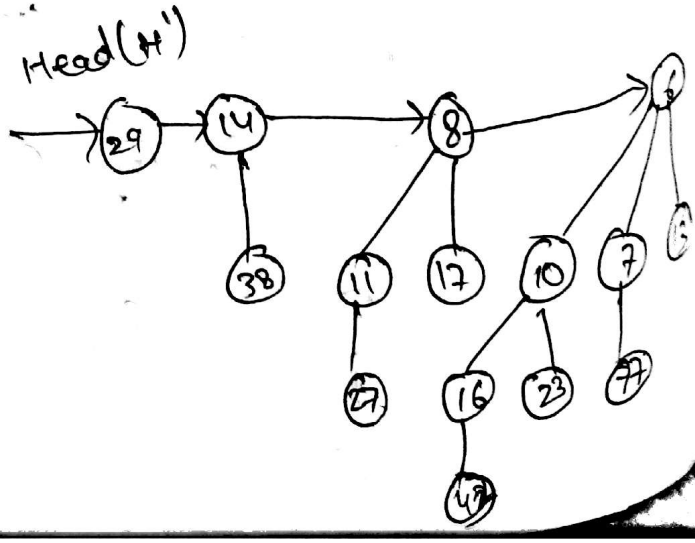


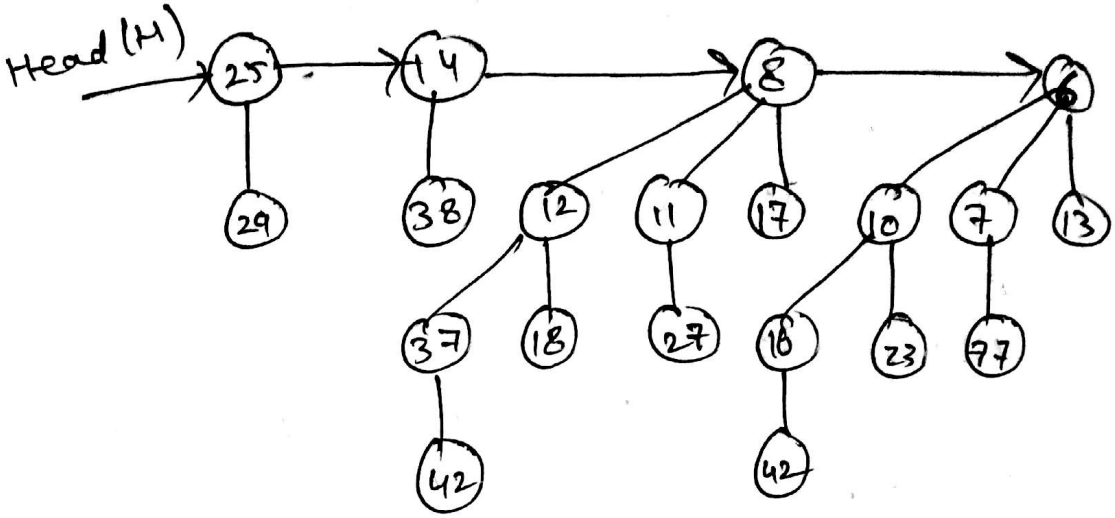
~~Decrease key.~~

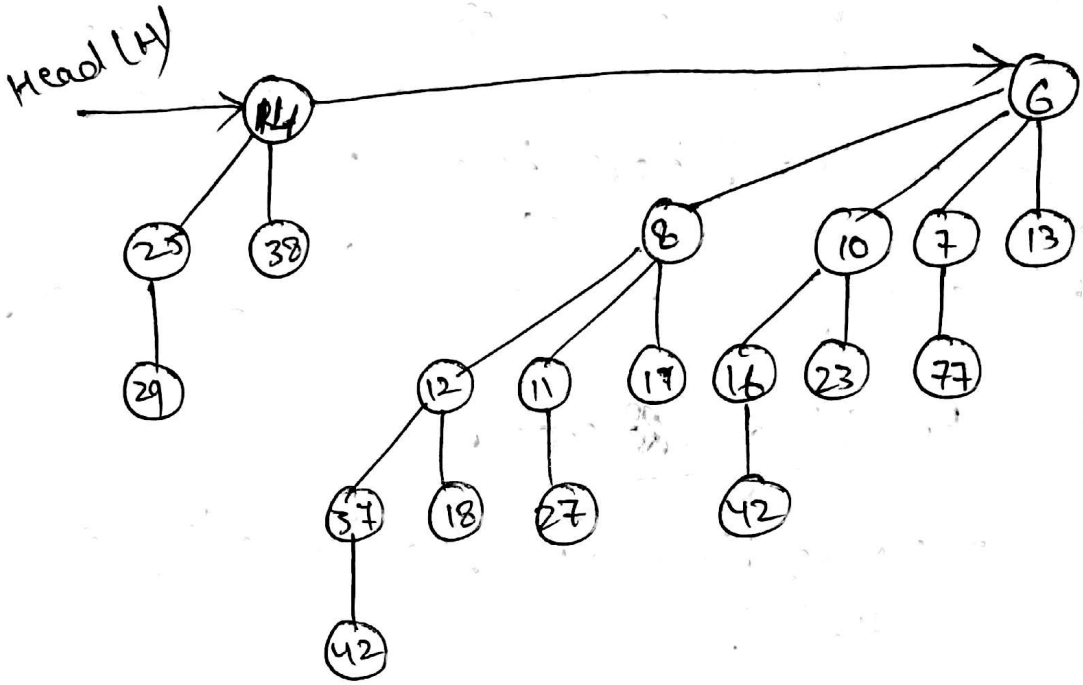


(9) Extract Min:









Red-Black tree :

A red-black tree is a binary search tree with extra bit of storage per node for an attribute "colour", which is either red or black.

Each node contain some fields -

i) Colour
ii) Key
iii) ~~Left~~
iv) Right
v) Parent

In ~~consider~~ R-B tree, no roots to the leaf ~~leaves~~ path is more than twice as long as any other. so, RB tree is approximate balanced tree.

Properties of Red-black tree :

i) Every node is either red or black.
ii) Root is always black.
iii) Every leaf is black and every leaf node is NIL pointer.
iv) If a node is red the both its children are black.
v) For each node every path from the node to leaf contains the same no. of black nodes

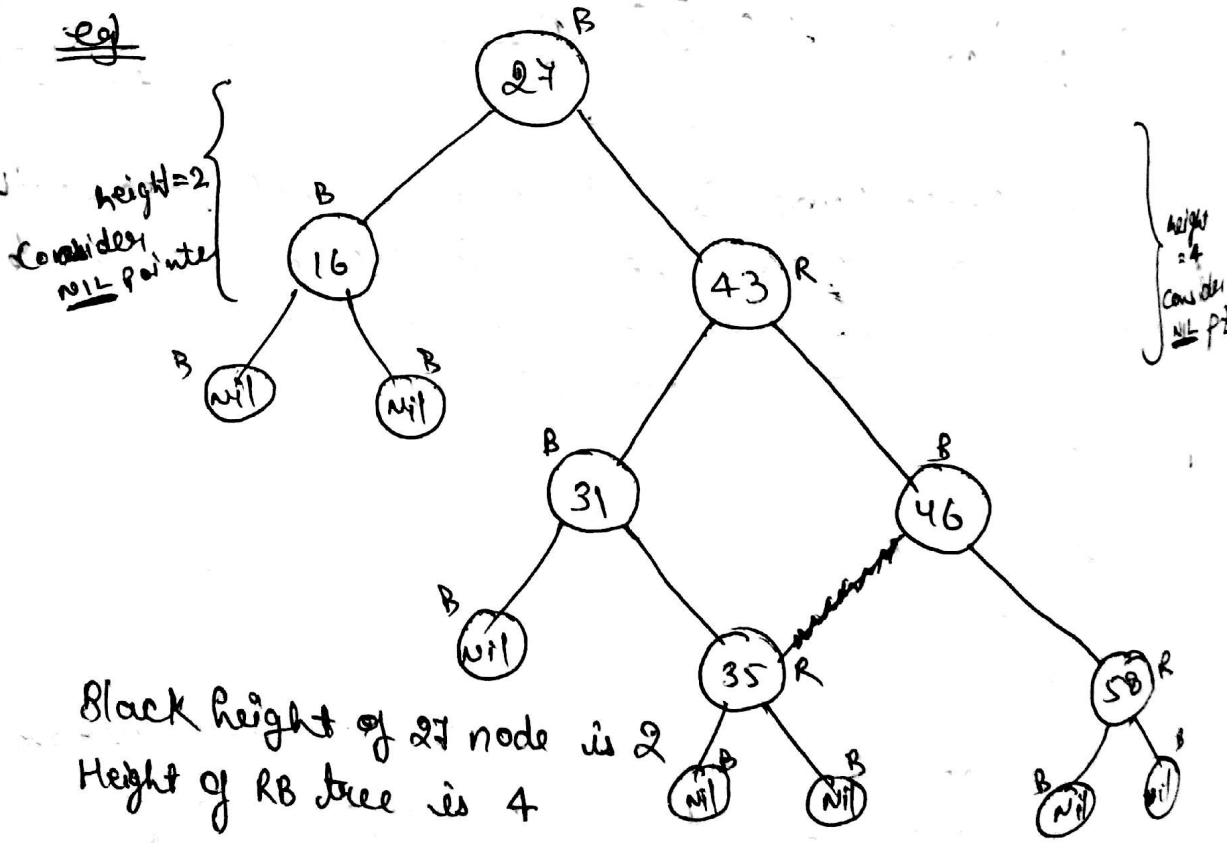
Height of Red-Black tree :

H(x) = no. of edge in a longest path to a leaf.

Black height of a tree :

bh(x) = No. of black node from 'x' to leaf excluds the 'x'.

eg



Black height of 27 node is 2
Height of RB tree is 4

Theorem:

A Red-Black tree with n-internal nodes has height atmost $2\log(n+1)$

***we can prove this by induction method**

Let no. of internal node $= 2^{bh(x)} - 1$

step-1) if x is a leaf node (Nil ptr.) then black height of x is 0

Now $2^{bh(x)} - 1$

$= 2^0 - 1 = 1 - 1$

$= 0$ (and also no. of internal node = 0)

~~step-2) if x is an internal node, then height of red-black tree is bh(x)-1 now, no. of internal nodes is =~~

step-2) If x is an internal node then total no. of internal node that x contain 2 children and each child has a black height either bh(x) or bh(x) - 1, its depends on the colour of the child

So, total no. of internal node $= 2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1 + 1$

$= 2^{bh(x)} - 1$ (which we assume)

step-3) If x is a root node then atleast half the nodes on any path from the root to leaf not include the root.

So, bh(x) is atleast $h/2$.

then total no. of internal nodes $= 2^{h/2} - 1$

$\Rightarrow n \geq 2^{h/2} - 1$

$$n+1 \geq 2^{h/2}$$

$$\log_2(n+1) \geq \frac{h}{2} \log_2 2$$

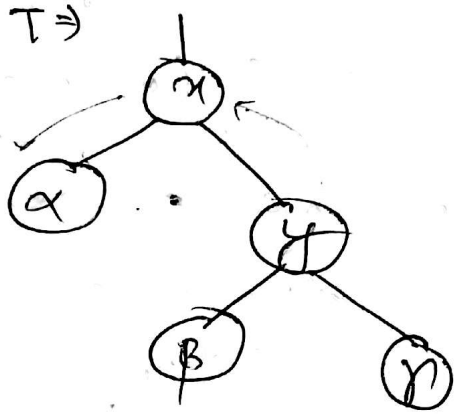
$$\log_2(n+1) \geq \frac{h}{2}$$

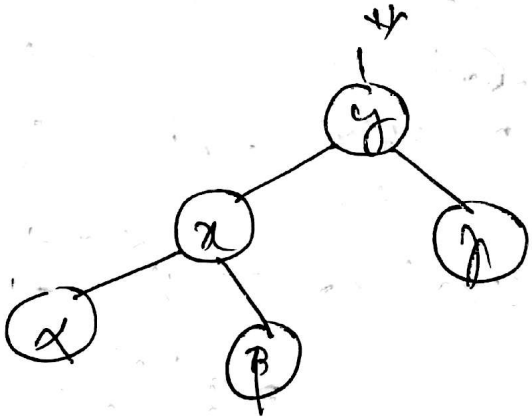
$$\Rightarrow \boxed{h \leq 2\log_2(n+1)}$$

Rotation: there are two types of rotation:-

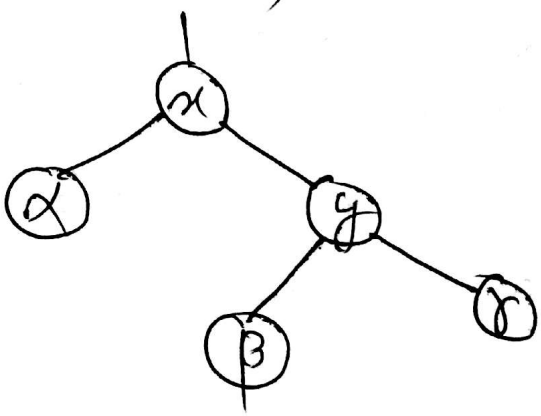
1) **left Rotation:** T ⇒

left (T, x)

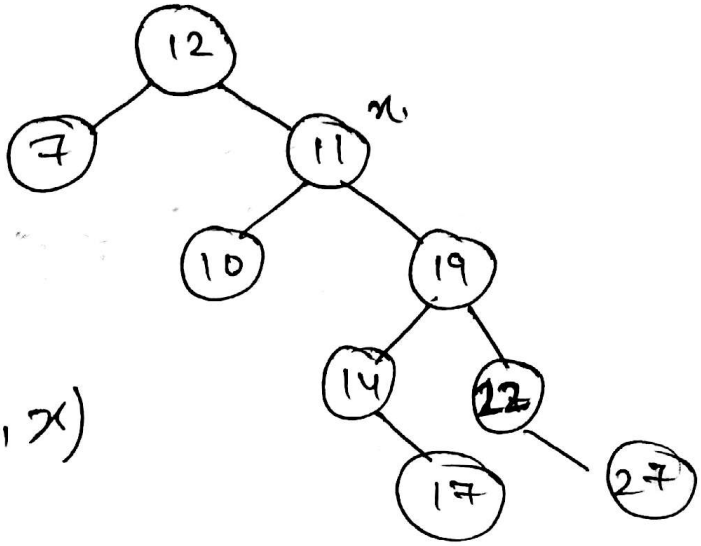




2) **Right rotation:** Right (T, y)

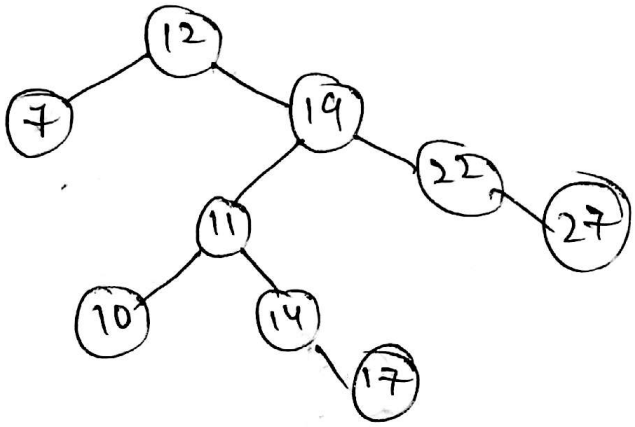


Q:



i) Left (T, x)

Sol: Left (T, 11)



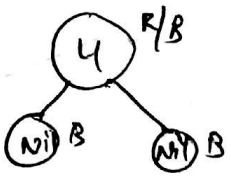
Insertion in Red-black Tree

Q Insert into RB tree

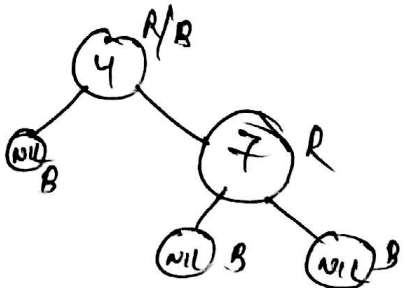
4, 7, 12, 15, 3, 5, 14, 18, 16

Sol:

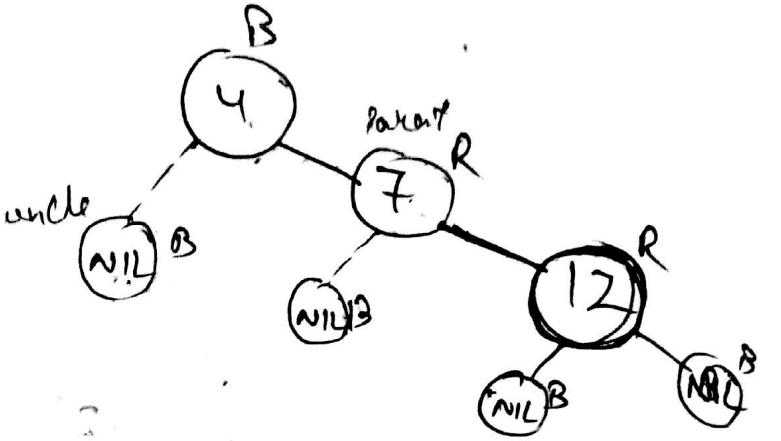
Insert 4



Insert-7

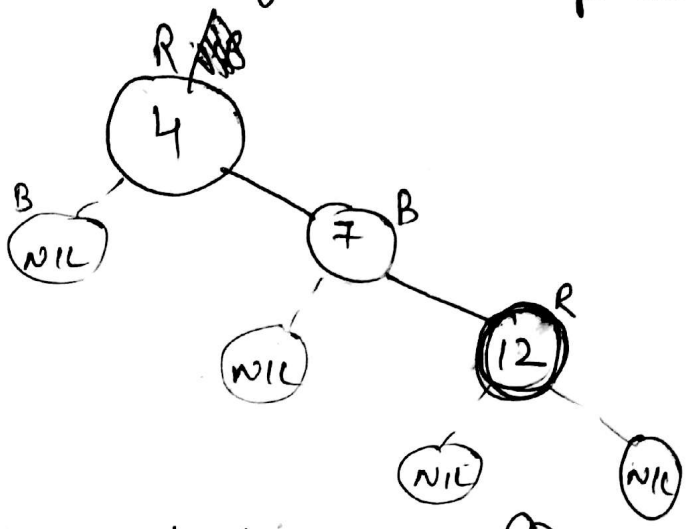


Insert - 12

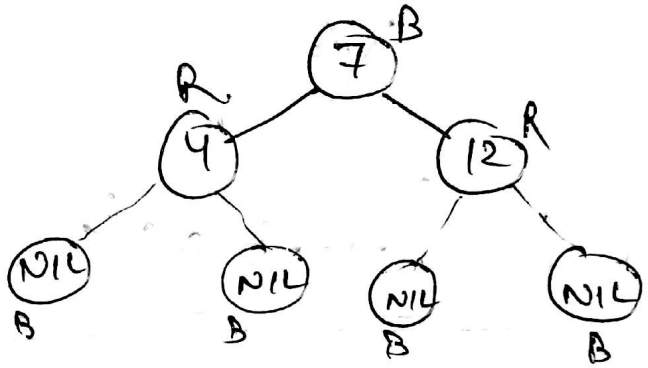


~~B left rotation of 4~~

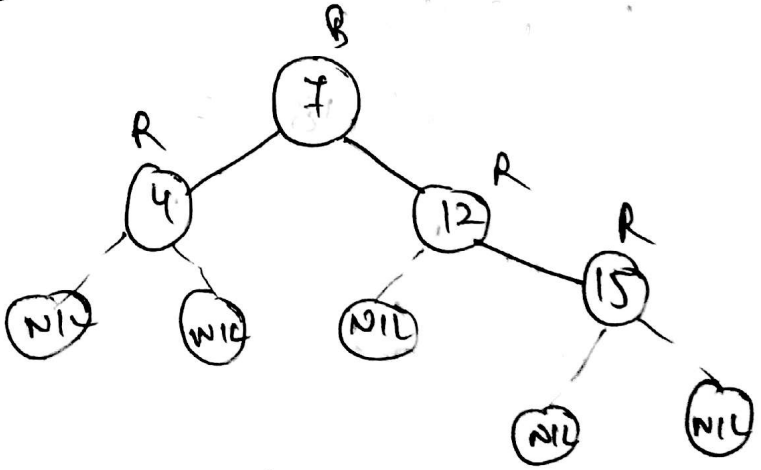
recolouring.



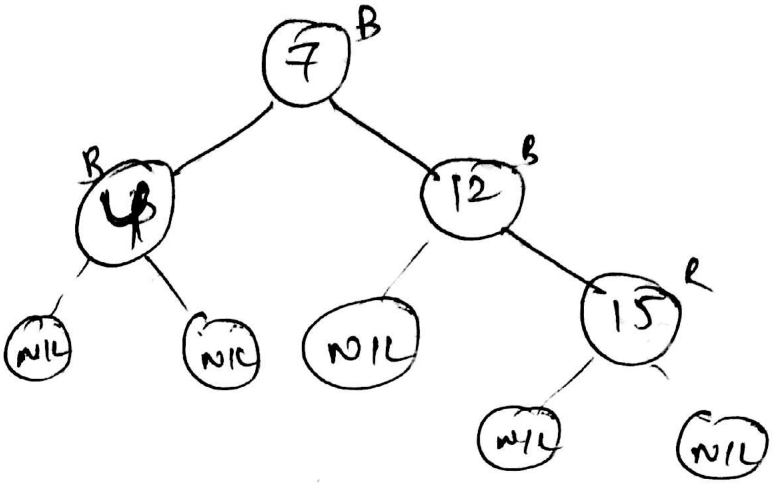
left rotation with (4). (Grand parent)



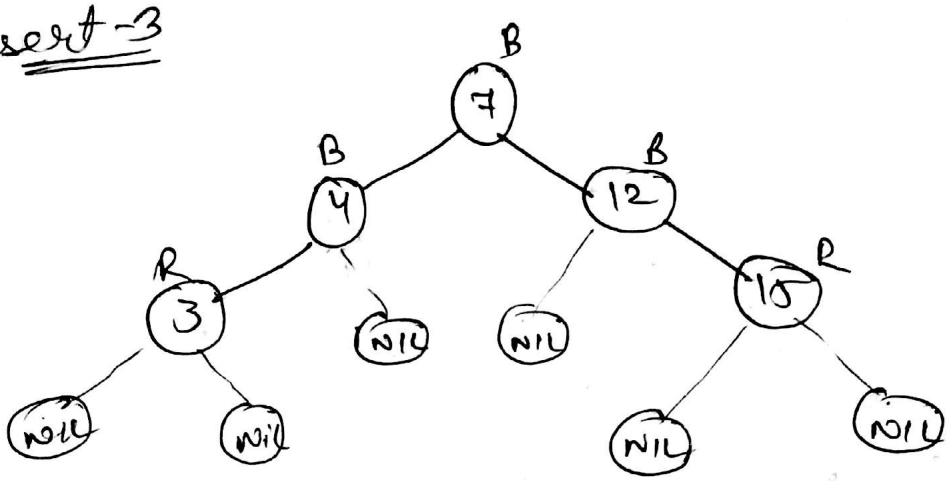
Insert · 15



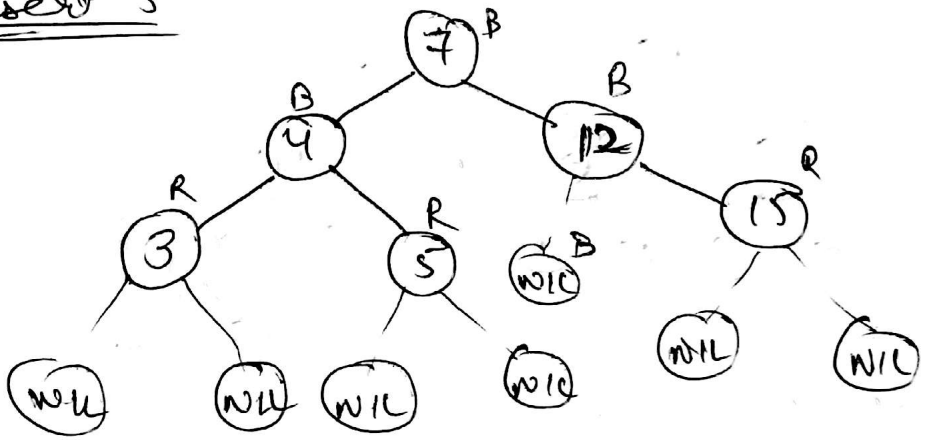
recolouring



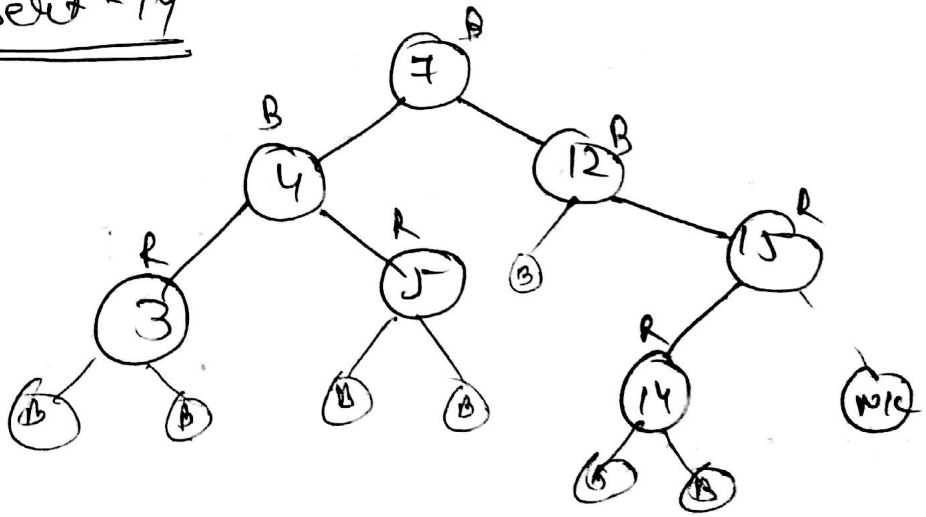
Insert -3



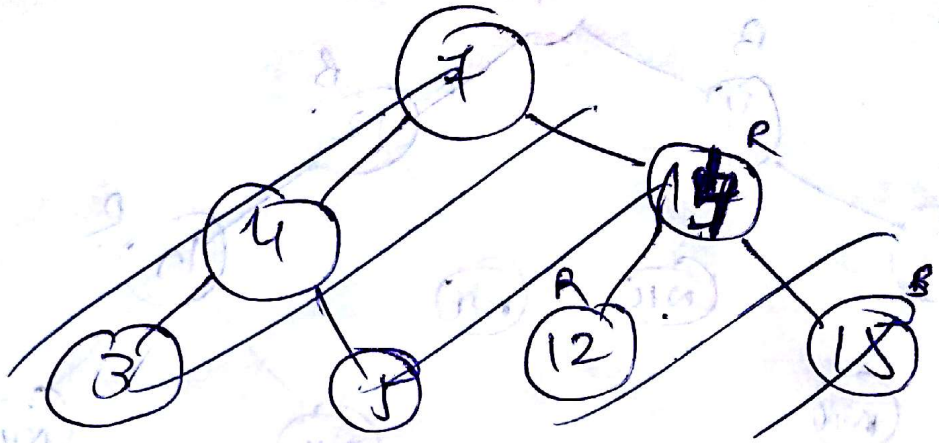
Insert 5



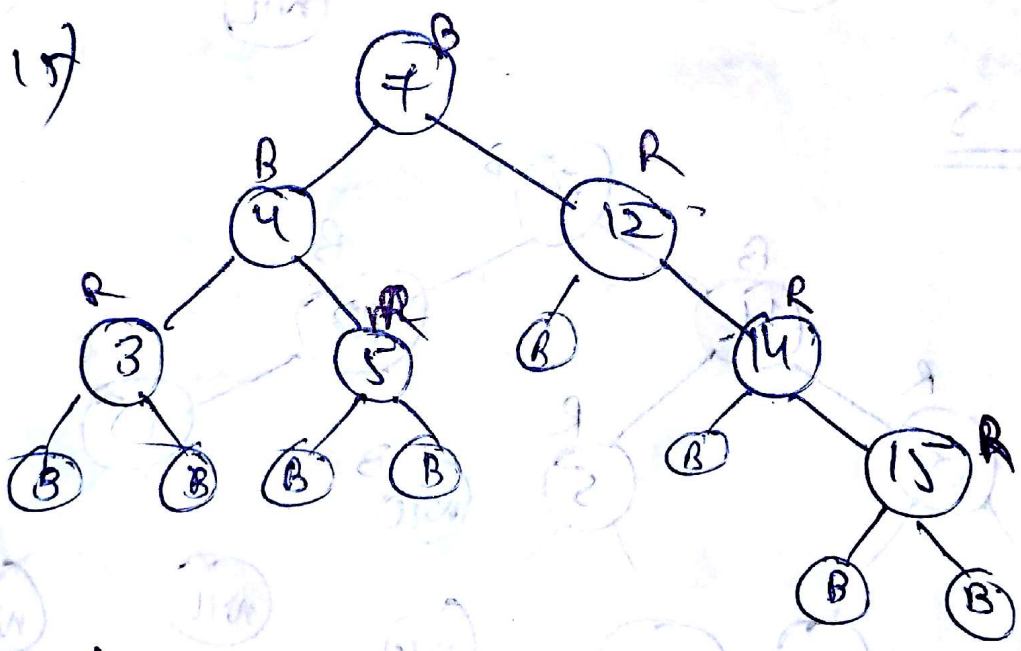
Insert -14



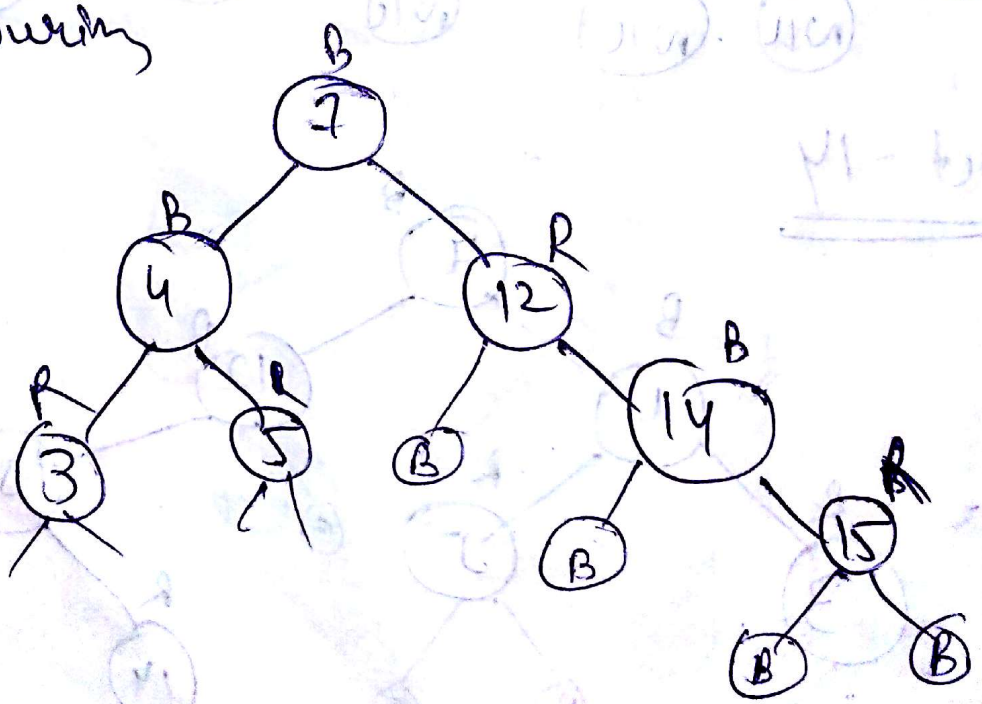
(R=) First right on parent & left on grandparent



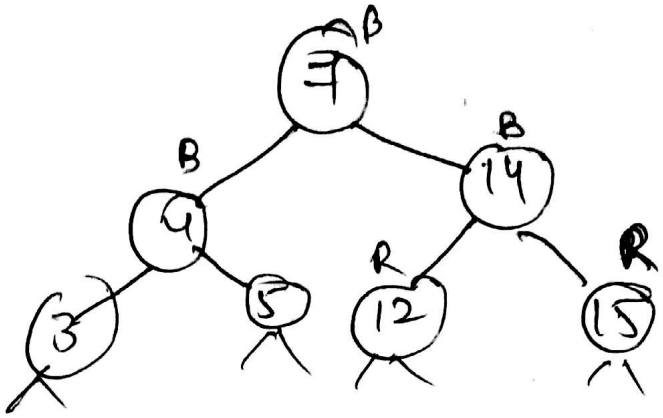
Right (T, 15)



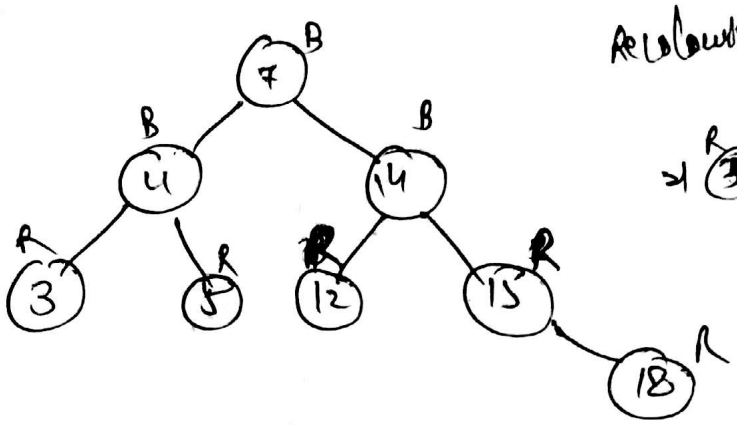
Recolouring



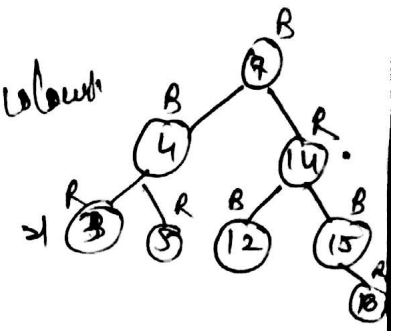
Left (T, 12)



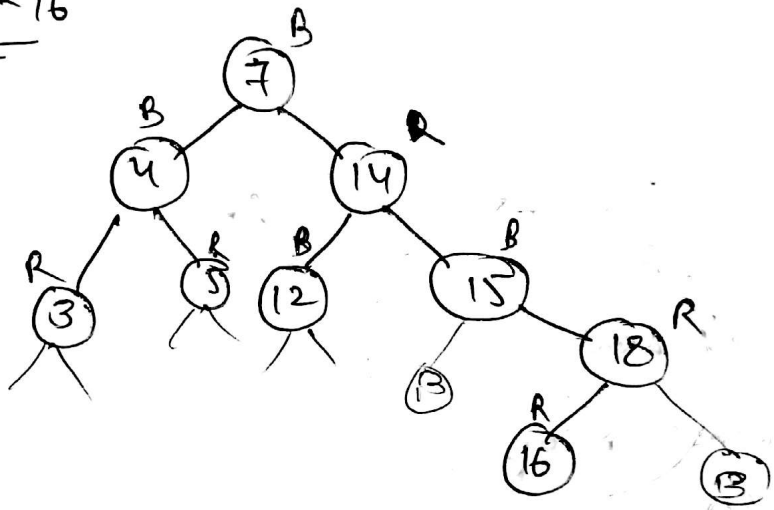
Insert - 18



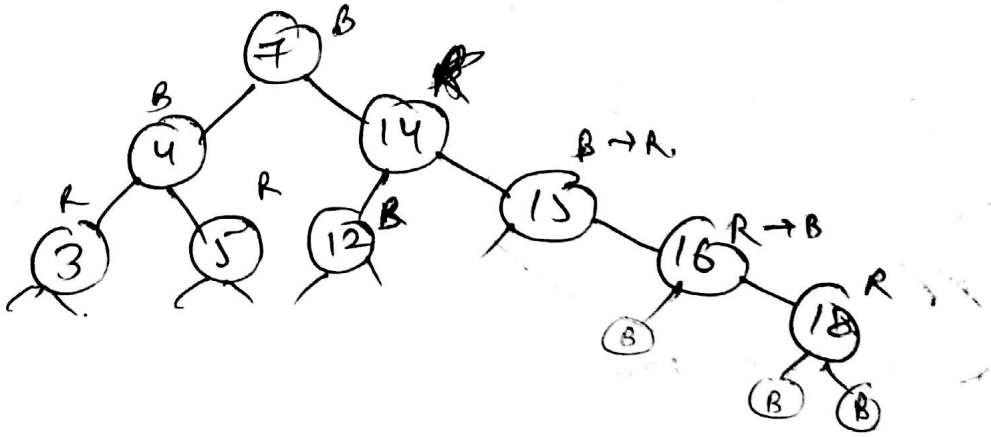
Recolour



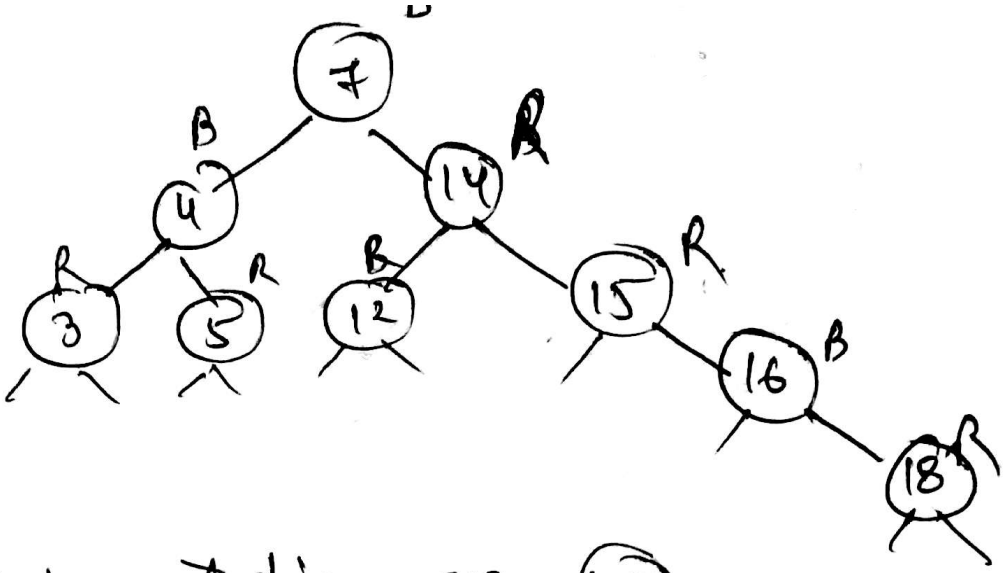
Insert - 16



First right on 18



Recolour



Left rotation on (15)