Analysis of recursive Algorithm.

BST = O(n)

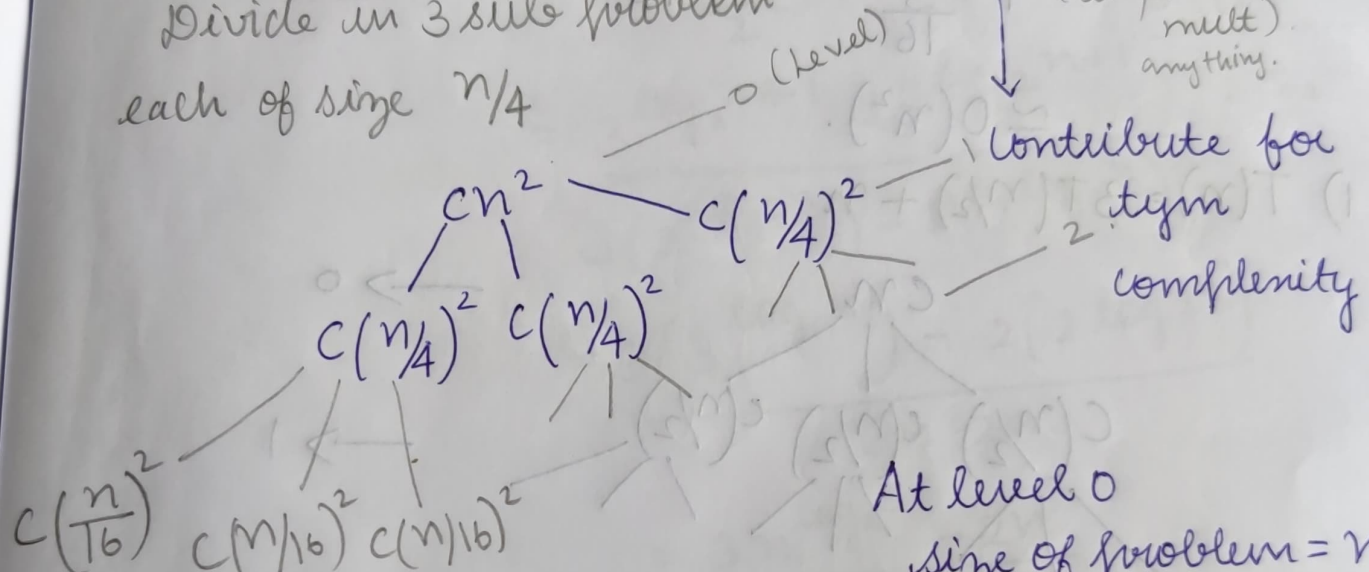
Recursion tree

$T(n) = aT(n/b) + f(n)$

$f(n)$ = basic operation (comparison or mult) anything.

eg: $T(n) = 3T(n/4) + n^2$

Divide in 3 sub problem each of size $n/4$



[Until ther size become = 1] recursion occurs.

$\frac{n}{4^i} = 1$

$n = 4^i$

[$\log_4 n = i \longrightarrow$ levels]

At level 0
size of problem = n

level 1
size = $n/4$

level 2
size = $\frac{n}{4^2} = \frac{n}{16}$

[level i
size = $\frac{n}{4^i}$]

No. of nodes

Level 0 $\Rightarrow cn^2$.

1 $\Rightarrow \frac{3}{16} cn^2$

i $\Rightarrow \left(\frac{3}{16}\right)^i cn^2$

$$T(n) = cn^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \cdots + \left(\frac{3}{16}\right)^i\right)$$

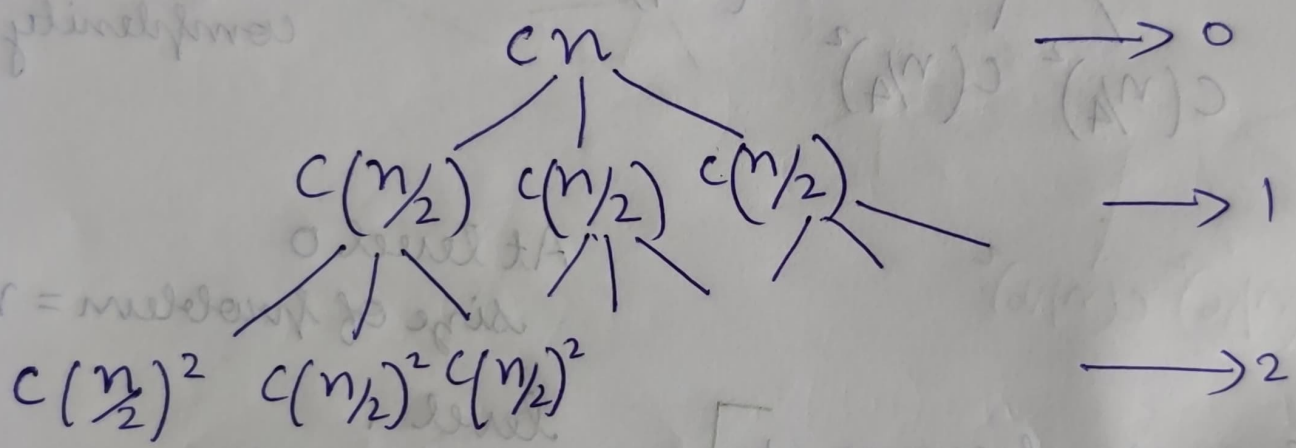
(total running tym)

G.P

$$S_\infty = \frac{1}{1 - \frac{3}{16}} \Rightarrow \frac{16}{13} = \text{___} = cn^2(\ )$$

$$\simeq O(n^2).$$

1) $T(n) = 3T(n/2) + n$



Level 0 $\Rightarrow$ size $= n$.

1 size $= n/2$.

2 size $= \frac{n}{2^2}$.

i size $= \frac{n}{2^i}$

$i = \log_2 n$

$n > 1$.

$$cn\left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \cdots \left(\frac{3}{2}\right)^i\right) \cdot \frac{n^{n+1} - 1}{n - 1}$$

$$n\left(\frac{1}{1 - \frac{3}{2}}\right) \Rightarrow n\left[\frac{3}{2}\right.$$

$O(n^{\log_2 3})$

$$n\left(1+\frac{3}{2}+\left(\frac{3}{2}\right)^2+\cdots\left(\frac{3}{2}\right)^{\log_2 n}\right)$$

$\log_2 n = b$.

$$\frac{r^{n+1}-1}{r-1} = \frac{\left(\frac{3}{2}\right)^{\log_2 n+1}-1}{\frac{3}{2}-1}$$

$$\frac{\left(\frac{3}{2}\right)^{b+1}-1}{\frac{3}{2}-1}$$

$$\frac{3^{b+1}-1}{2^{b+1}} \quad \frac{3-2}{2\cdot 2^b}$$

$$= \frac{3^{\log_2 n+1}-2^{\log_2 n+1}}{3(2^{\log_2 n})-2(2^{\log_2 n})}$$

$$= \frac{3(3^{\log_2 n})-2(2^{\log_2 n})}{3n-2n}$$

$$= \frac{3n^{\log_2 3}-2n}{n}$$

$$= \frac{n\left(3n^{\log_2 3}-2n\right)}{n}$$

$$= \simeq O\left(n^{\log_2 3}\right)$$

$-1 < |r| < 1 \Longrightarrow \frac{a}{1-r}$

$|r| > 1 \Longrightarrow \frac{r^{n+1}-1}{r-1}$

The Master Method

$$T(n) = aT(n/b) + f(n)$$

$a \geq 1$, $b > 1$ & f = +ive.

Tetha

$f(n) \in \Theta(n^d)$ $\quad d \geq 0$

$T(n) = \Theta(n^d)$ $\quad a < b^d$

$T(n) = \Theta(n^d \log_b n)$ $\quad a = b^d$

$T(n) = \Theta(n^{\log_b a})$ $\quad a > b^d$

Example

1. $T(n) = 9T(n/3) + n$

$a = 9$, $b = 3$, $f(n) = n$. $\Rightarrow d = 1$.

$T(n) = \Theta(n^{\log_b a})$ $\quad 9 > 3^1$

$n^{\log_3 9} = n^2$

$\boxed{T(n) = \Theta(n^2)}$

2. $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1$, $b = \frac{3}{2}$, $d = 0$ $\quad f(n) = 1$.

$T(n) = \Theta(n^d \log_b n)$ $\quad 1 = \left(\frac{3}{2}\right)^0$

solve: $= n^0 \log_{\frac{3}{2}} n$

$\boxed{T(n) = \Theta \log_{\frac{3}{2}} n}$

3. $T(n) = 4T(n/2) + n$

$a = 4, b = 2, f(n) = n \quad d = 1.$

$4 > 2.$

$T(n) = \theta(n^{\log_b a})$

$n^{\log_2 4}$

$T(n) = \theta(n^2).$

4. $T(n) = 4T(n/2) + n^2$

$a = 4, b = 2, f(n) = n^2, d = 2.$

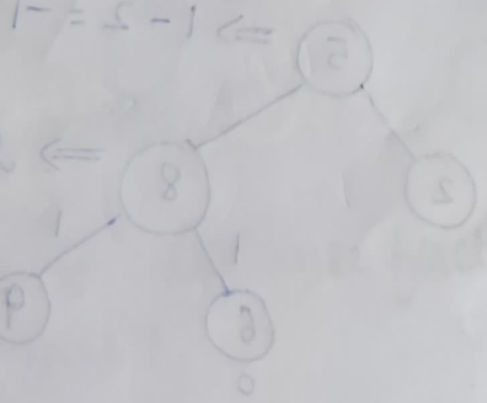
$4 = 4.$

$T(n) = \theta(n^d \log_b n).$

$T(n) = \theta(n^2 \log_2 n)$

5. $T(n) = 4T(n/2) + \frac{n^2}{\log n}$ can't apply master's theorem

$f(n) = \theta\, n^d.$



Binary search tree

Binary tree in which the keys are distinct.

* Tree can be empty.
* If not empty Lside should be less than root and Rside should be greater than root.

Disadvantage:

Skewed BST (no balance). worst case : $O(n)$.
best case : $\Theta(\ln(n))$

AVL Trees

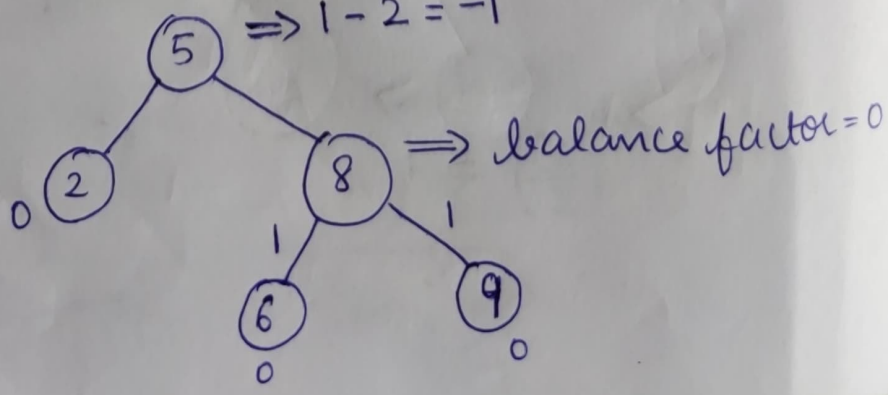
* A binary search tree that is <u>balanced</u>
* every node has <u>balance factor</u>.

balance factor = $h_L - h_R$.

h_L = height of left subtree.

h_R = height of Right subtree

eg:



* if balance factor is +1, -1, 0 then its AVL tree.

Maintain balance

insert a node → height increase by 1

delete a node → height decrease by 1

so we need to do <u>rotation</u>.

Rotation — LL, RR ⇒ single rotation

LR, RL ⇒ double rotation

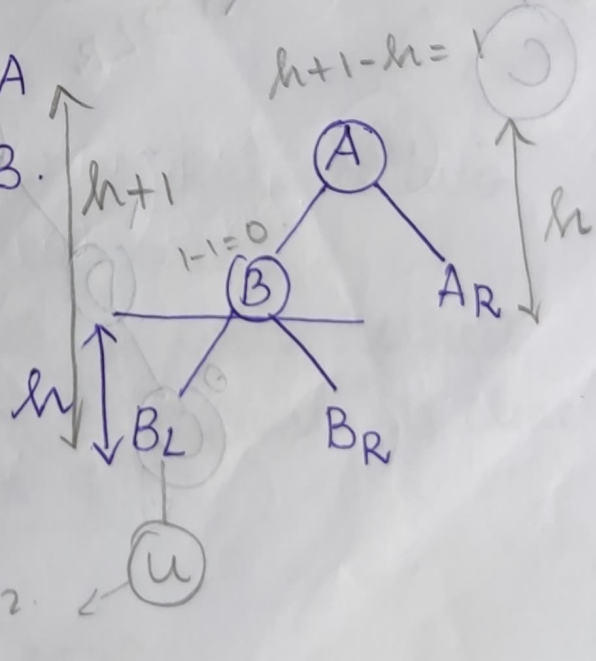
Insertion may involve

any 1 of 4 rotation is used to maintain balance.

* LL ← — left subtree of left subtree of A

A_R = right subtree of A

B_R = right subtree of B.



When you add "u" balance factor of A becomes 2 and B becomes 1

On rotation

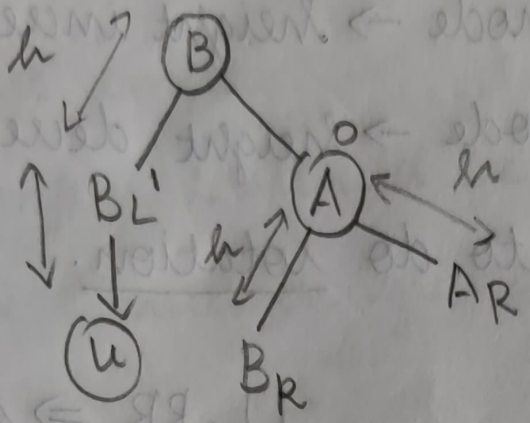
(i) Pull B

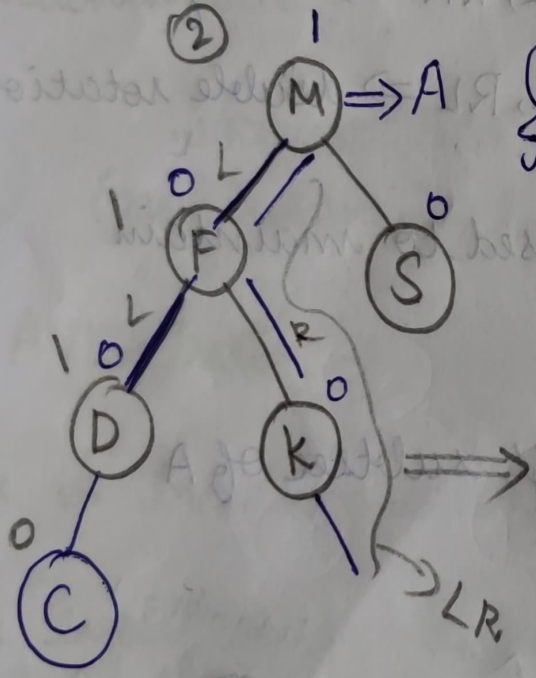
$A > B_R > B$

$A_R > R$.

$h+1-h-1=0$.

So:



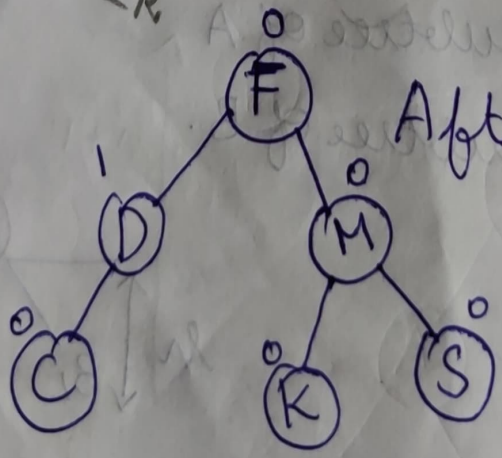


(LL Rotation)

Insert C. and pen with pencil

m is the node that is imbalanced.

A ⇒ Imbalance.



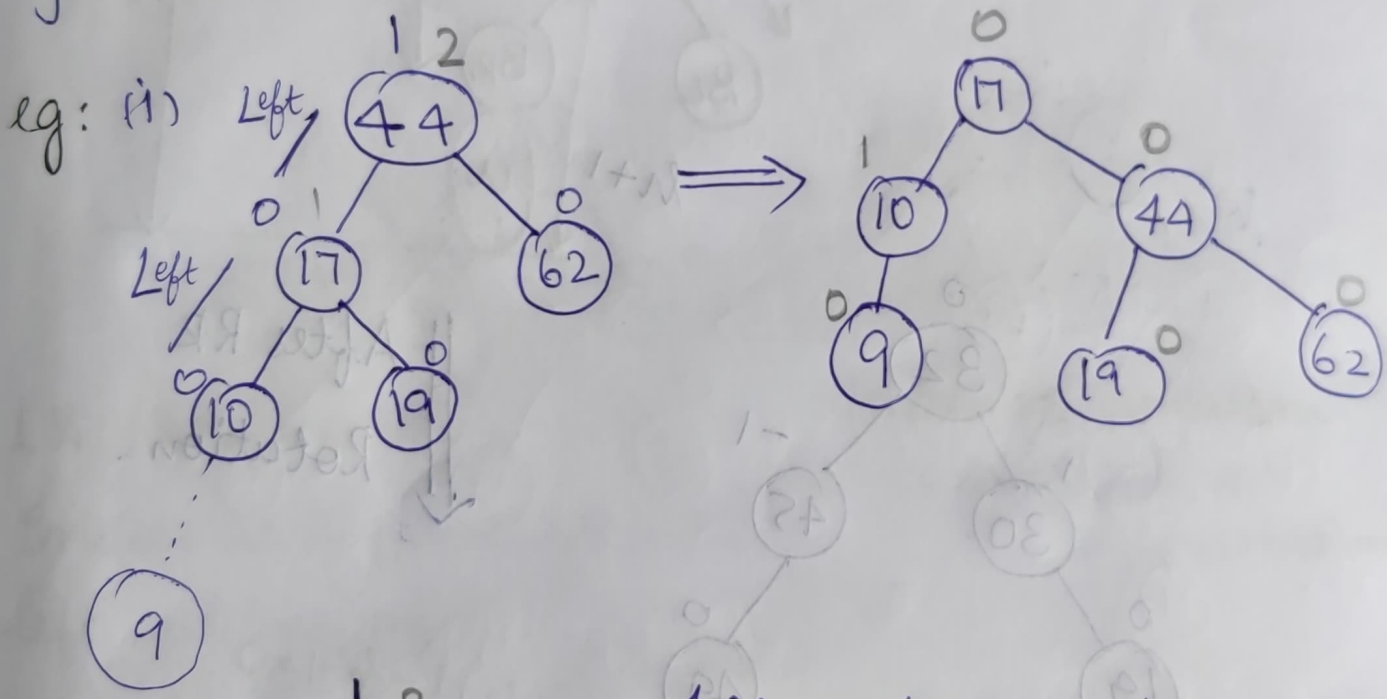
After Rotation.

B is represented as K. [2 changes in LL]

* B_R become left child of A.
* B becomes root node

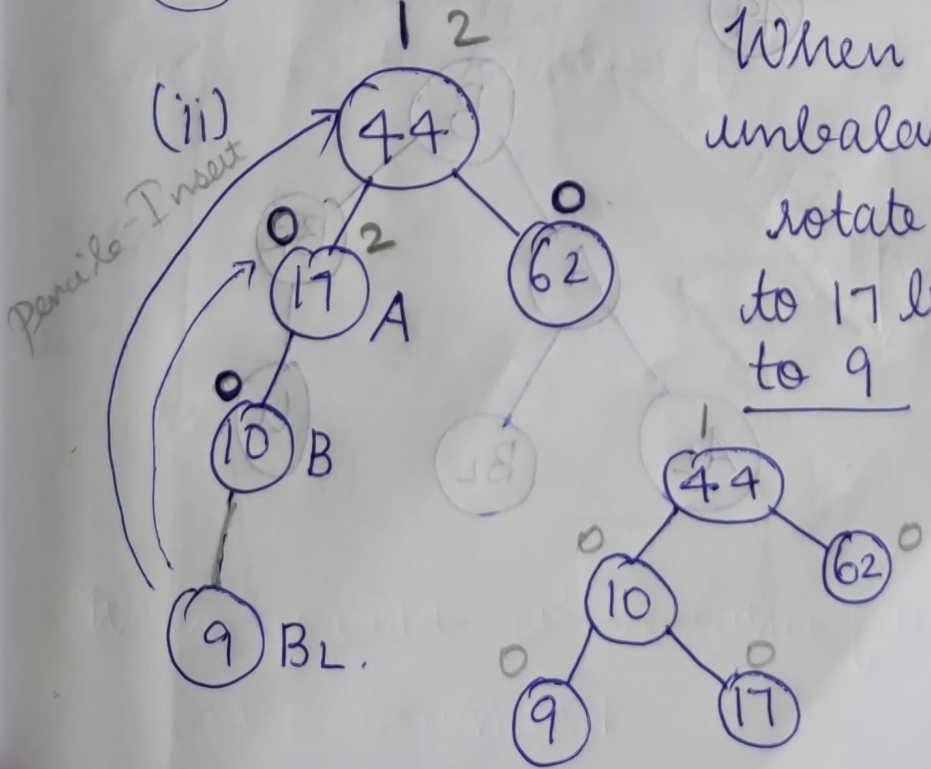
```
Node LL(Node A).
{
Node K = Left(A).
Left(A) = Right(K).
Right(K) = A.
return K.
}
```

eg: (i)



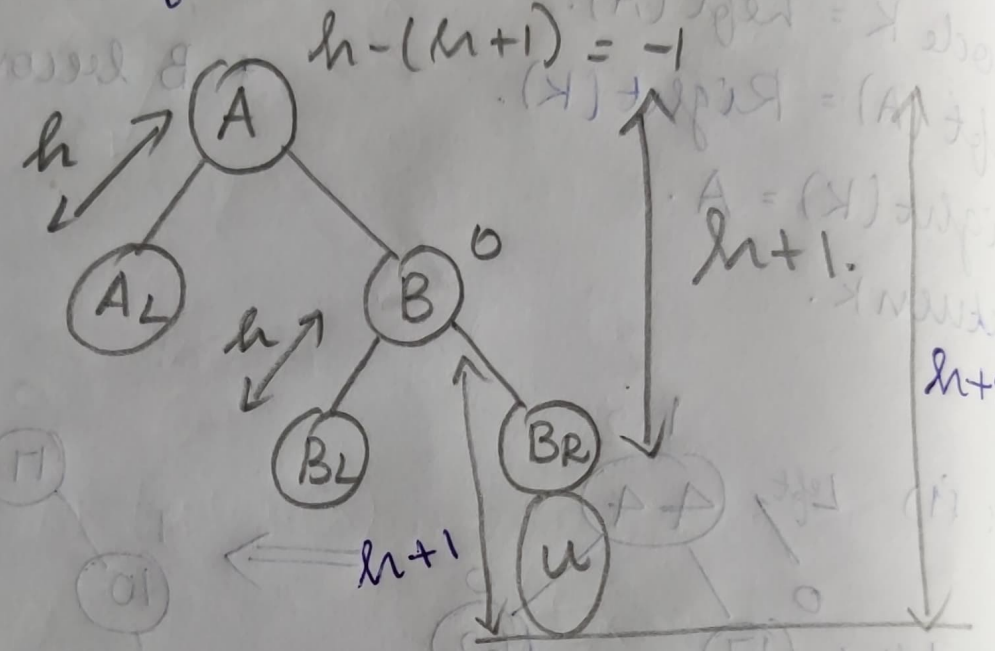
When there is two unbalance rotate with respect to 17 because its close to 9.

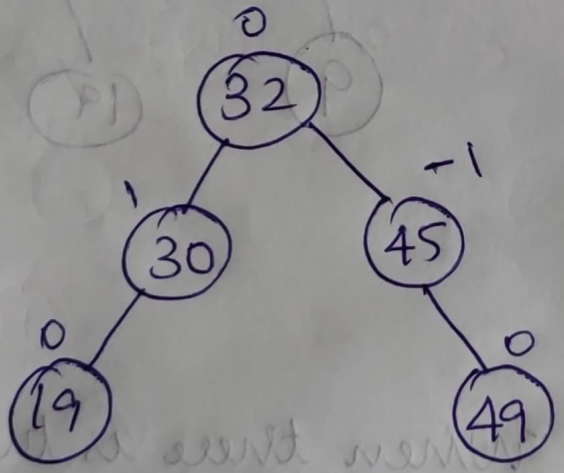
(ii)



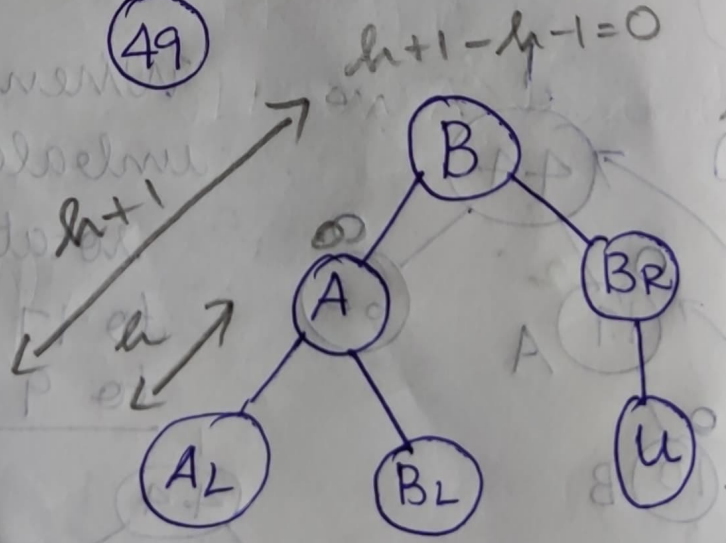
RR - Rotation

Insert a node in the right subtree of rightsubtree of A.

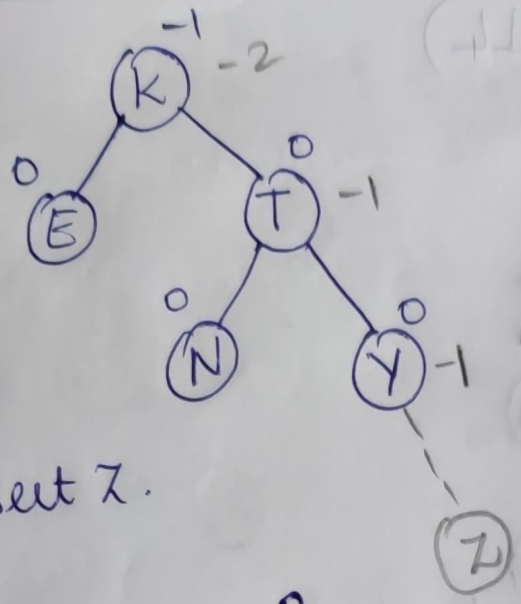




After RR Rotation.

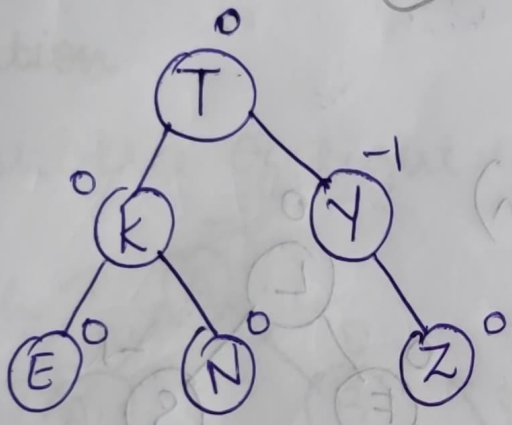


eg:1



Insert Z.

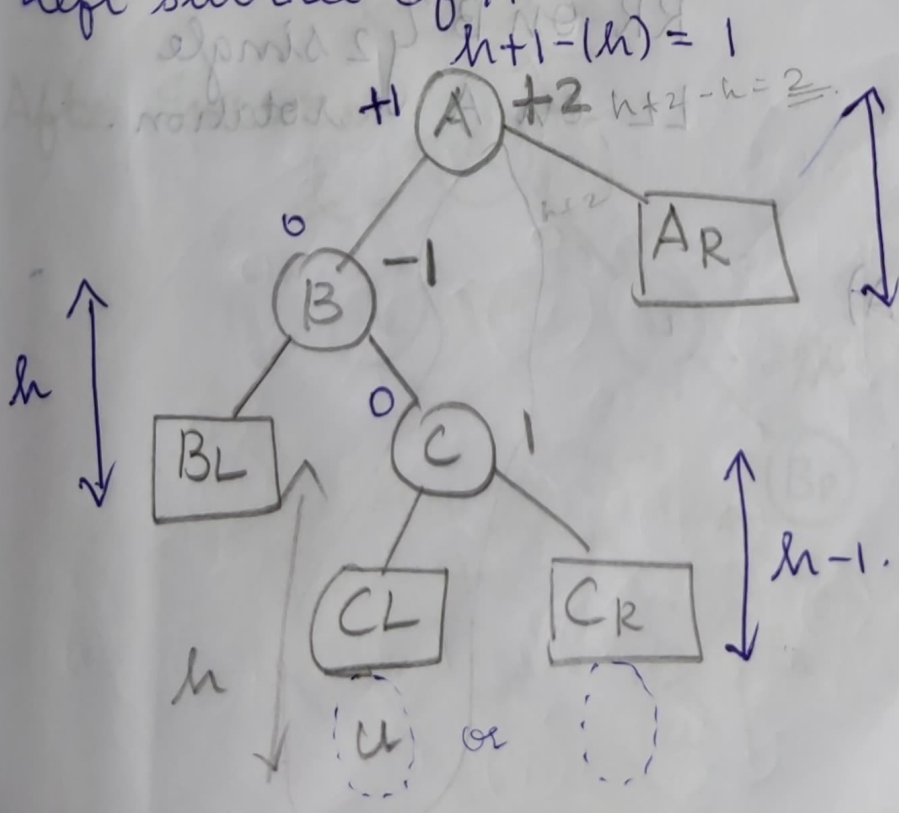
```
Node RR(Node A)
{
Node K = right(A)
right(A) = left(K)
left(K) = A
return K
}.
```



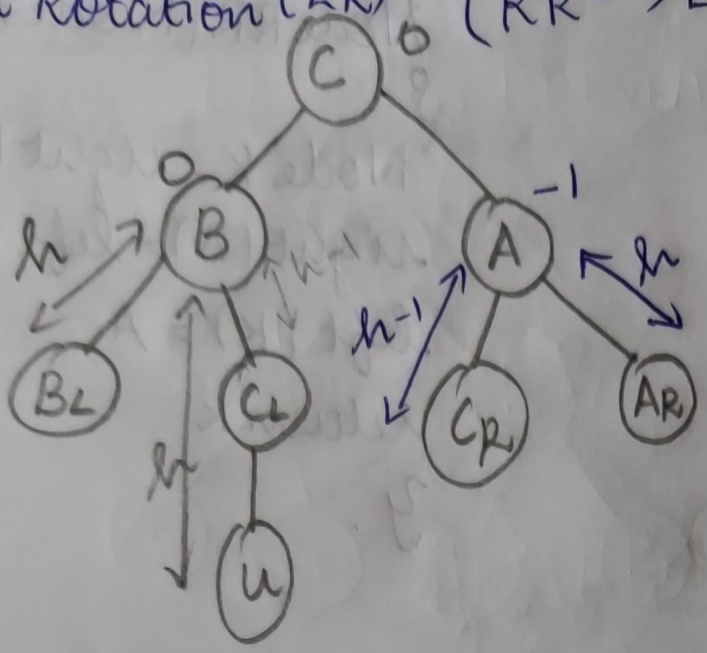
LR Rotation

Insert at right subtree of left subtree of A.

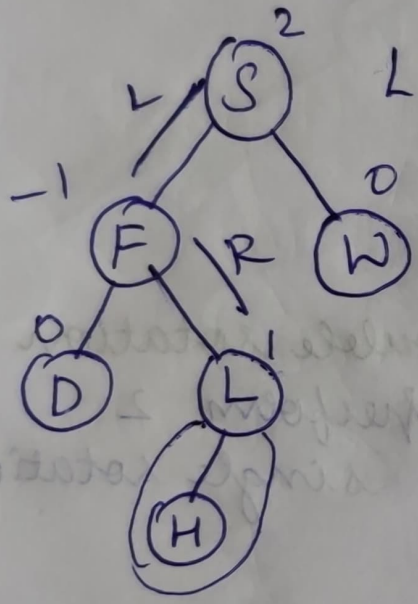
Double rotation (perform 2 single rotation)



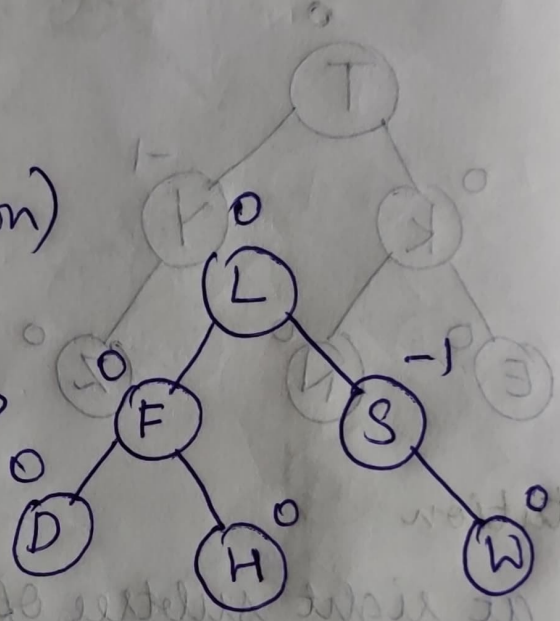
After Rotation (LR) (RR → LL)



i) Insert H

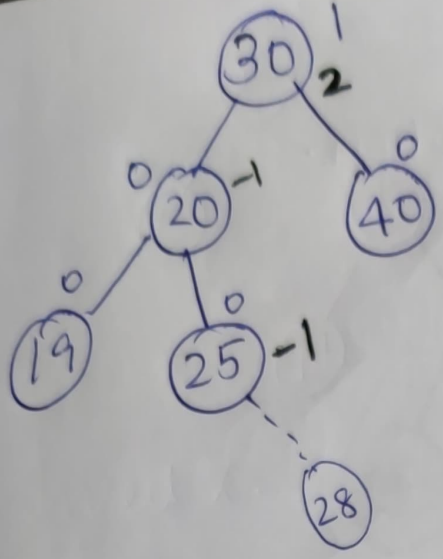


LR (Rotation)

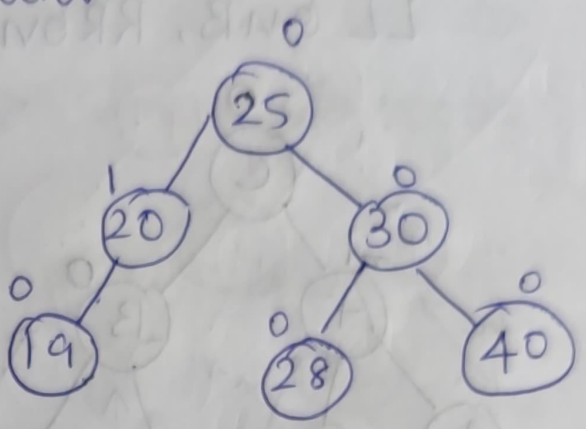


RR on B } 2 single
LL on A } rotation.

```
Node LR(NodeA).
{
  Left(A) = RR(leftA)
  node(K) = LL(A)
  return K
}.
```

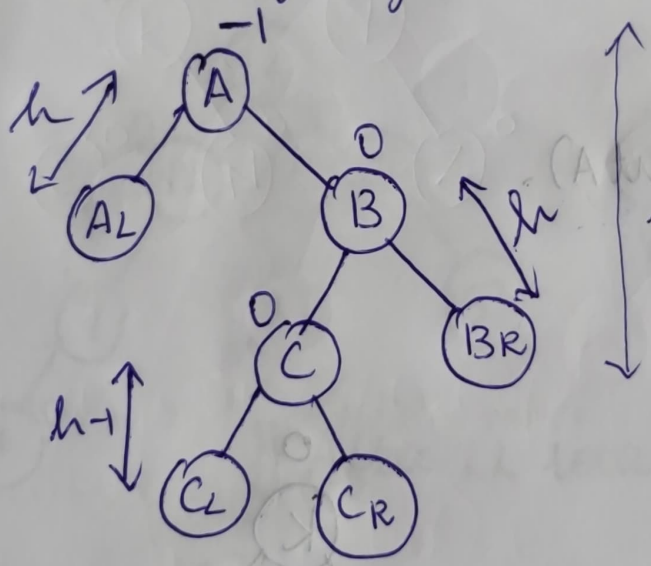


LR rotation.

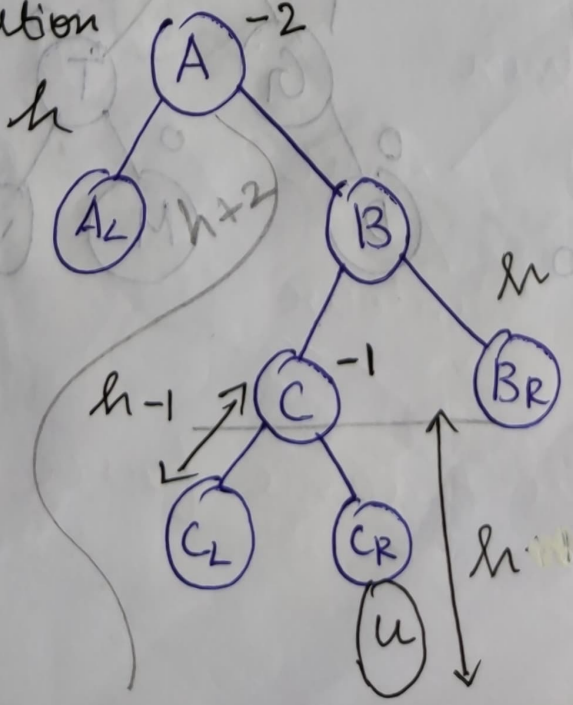


RL Rotation

left subtree of right subtree of A.



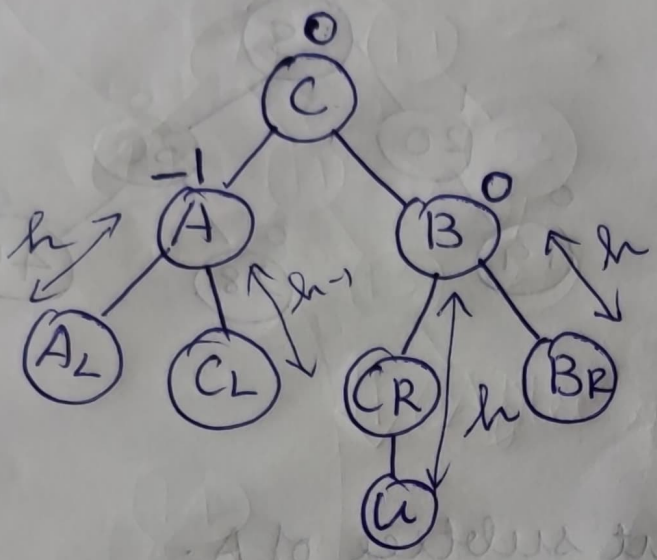
After Insertion



Insertion can be done in C_L or C_R

RL rotation.

LL on B, RR on A.



Node RL (Node A)

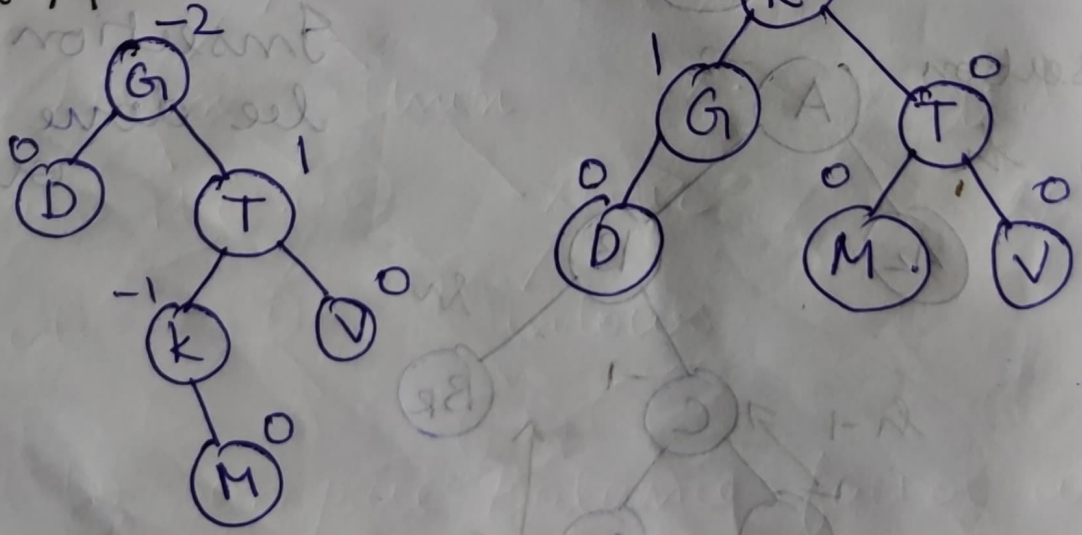
{ right (A) = LL (right A).

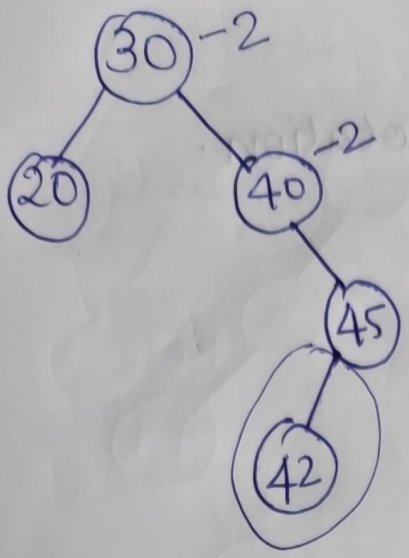
node (k) = RR (A)

return k

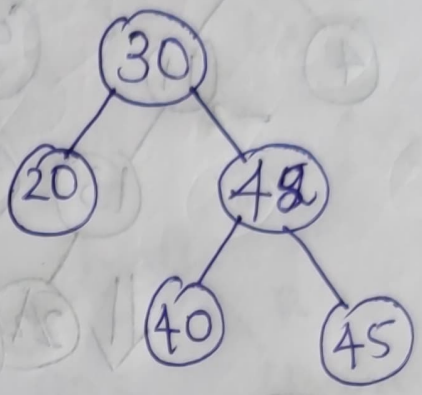
}.

Insert M

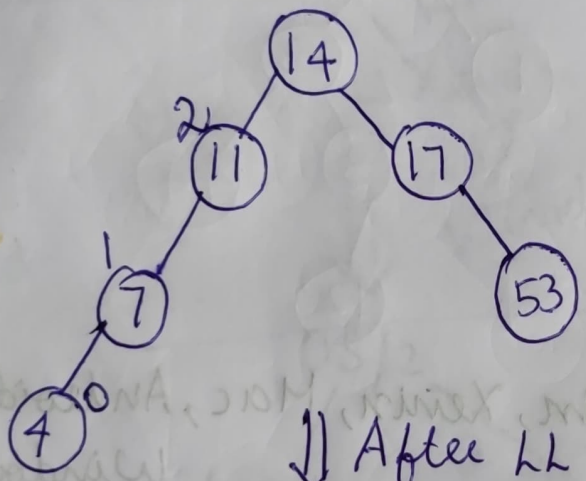




$\Longrightarrow$ which imbalance node is close to inserted node.



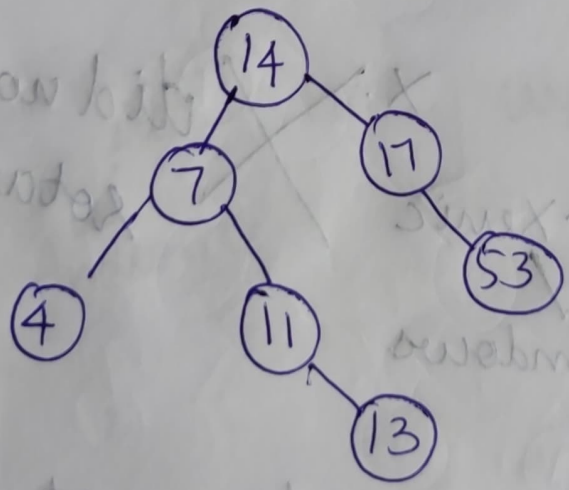
Ex1: 14, 17, 11, 7, 53, 4, 13



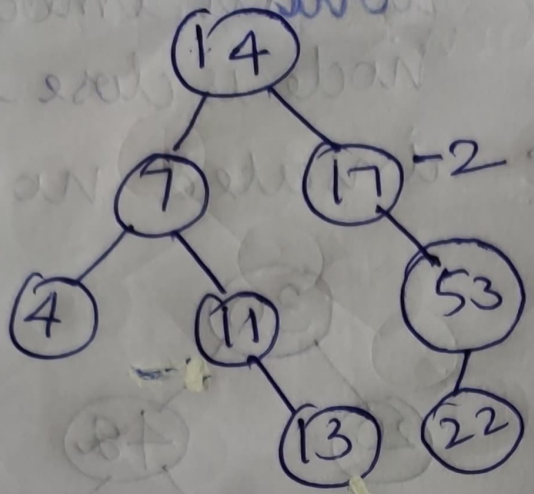
each tym check the balance factor

How to reach 4 from 11 (LL rotation)

$\Downarrow$ After LL rotation.

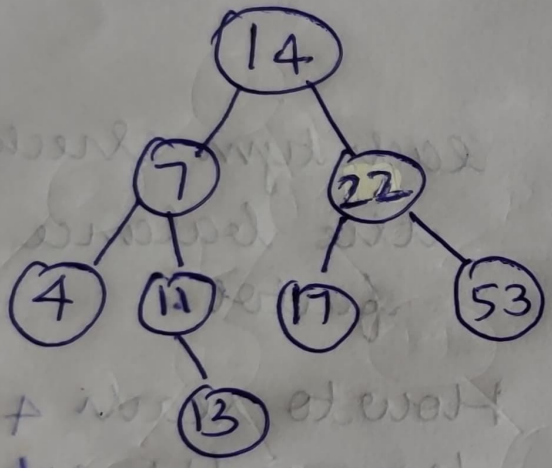


Now its an AVL tree



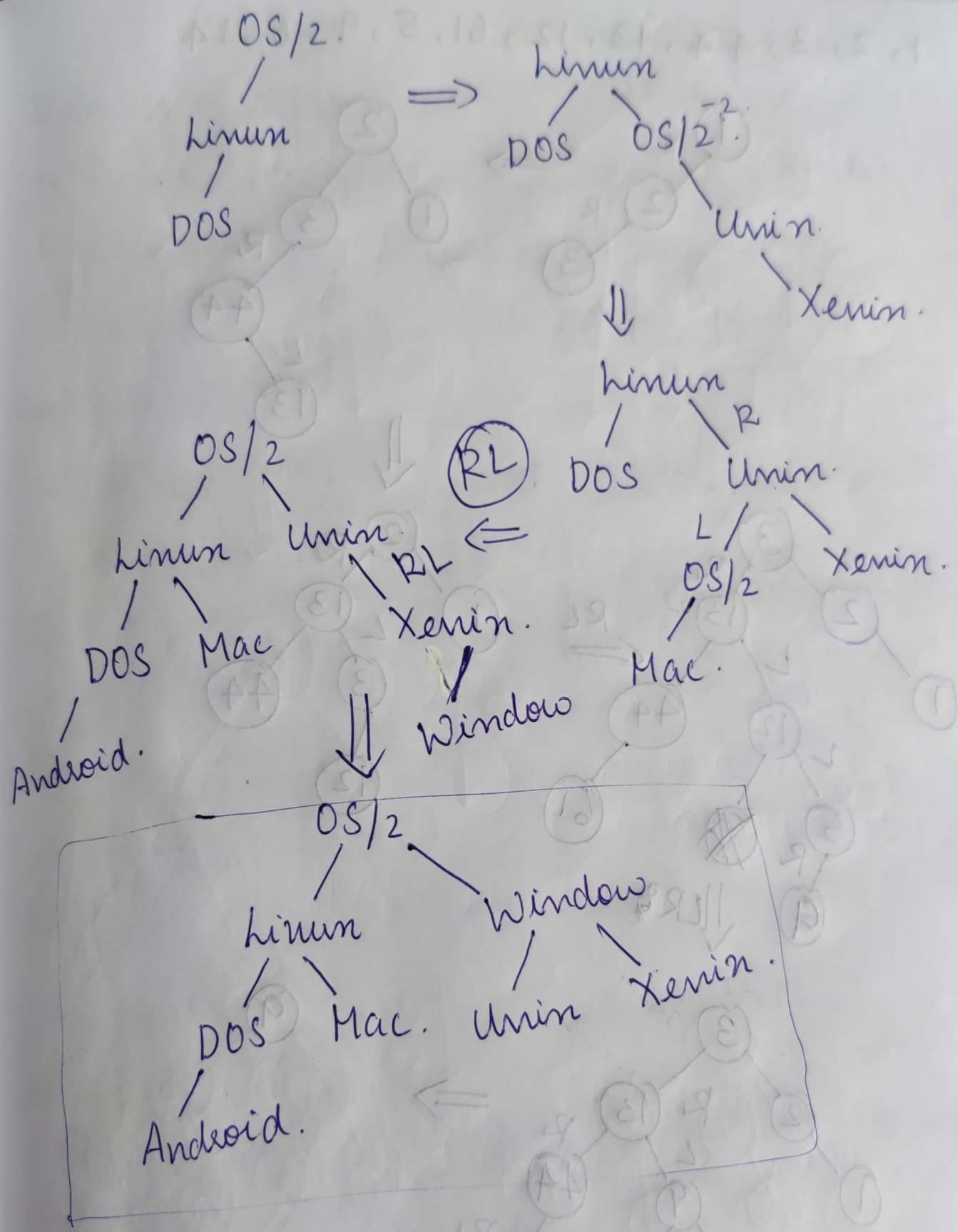
RL rotation.

⇓ After rotation.

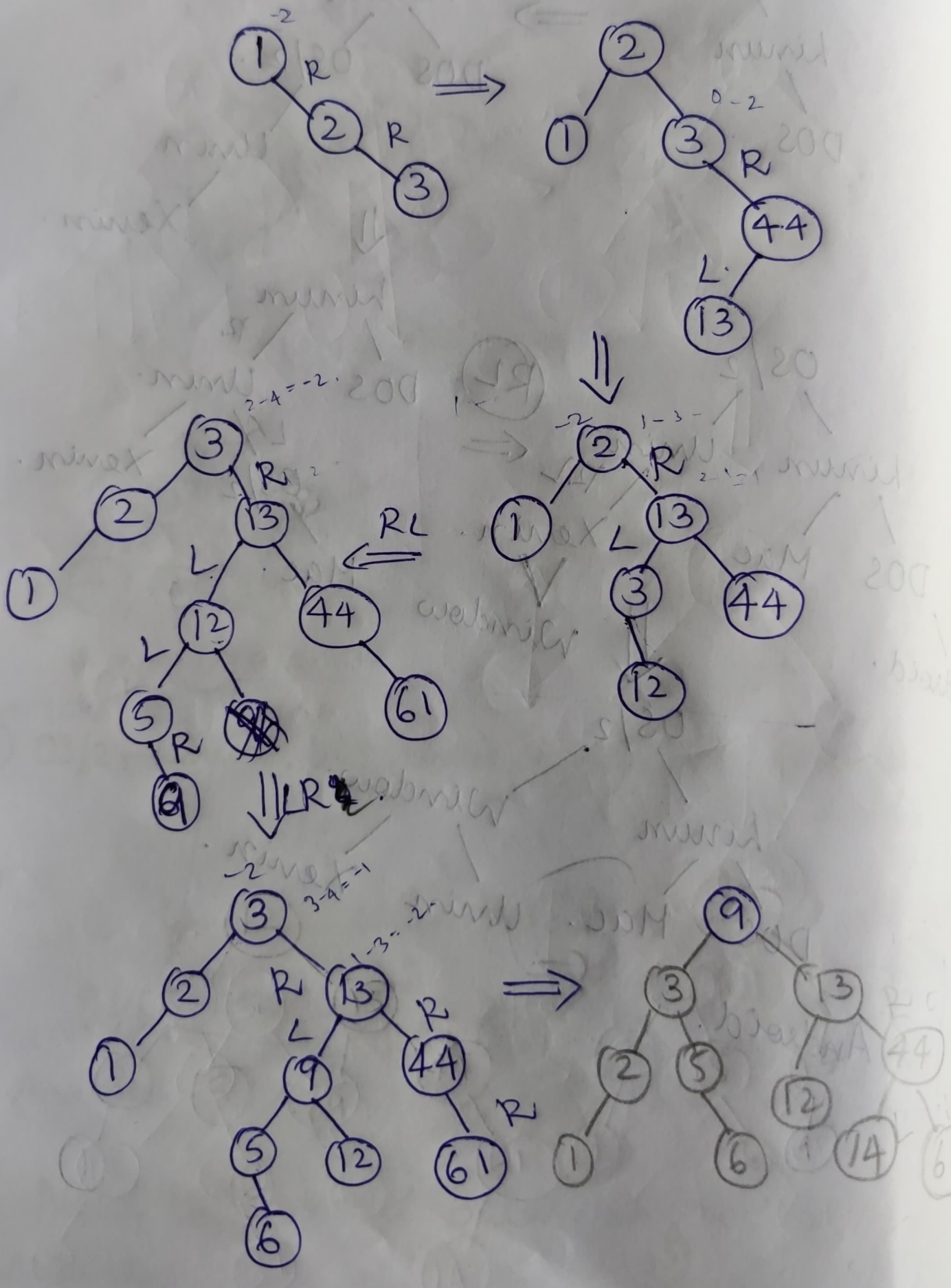


1) OS/2, Linux, DOS, Unix, Xenix, Mac, Android, Window

OS/2


OS/2
Linux
DOS
Linux
DOS
OS/2
Unix
Xenix
Linux
DOS
R
Unix
L
OS/2
Xenix
Mac
OS/2
Linux
Unix
RL
Xenix
DOS
Mac
Android
Window
OS/2
Linux
Window
DOS
Mac
Unix
Xenix
Android

1, 2, 3, 44, 13, 12, 61, 5, 9, 6, 14



Deletion

2 types of rotation.

1. R-type (Node deleted is a right subtree)
2. L-type (Node deleted is a left subtree

Node lies in the left subtree of the imbalanced node.

3 category for both rotation.

$R_0, R_1, R_{(-1)}$ and $L_0, L_1, L_{(-1)}$

R_0 - Left child of unbalanced node A has a balance factor 0.

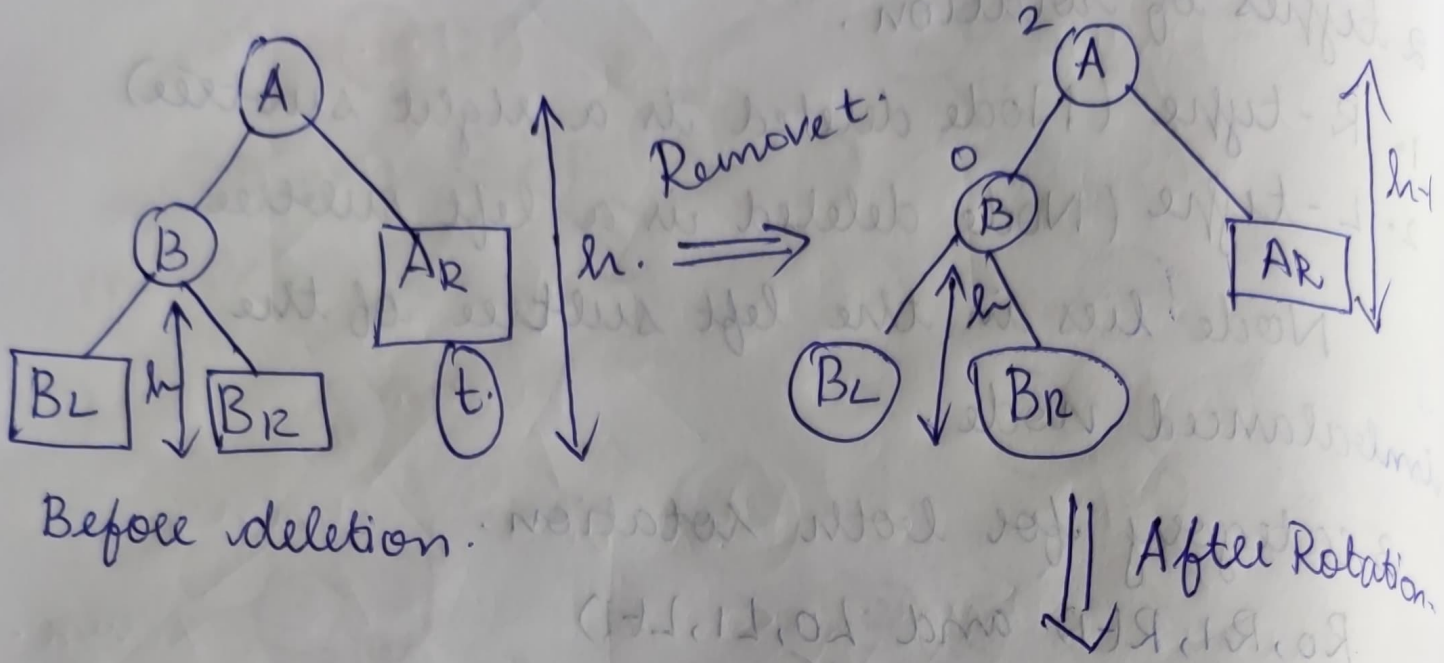
R_1 - Left child of unbalanced node A has a balance factor 1.

$R_{(-1)}$ Left child balance factor -1.

R type: deleted node is in right of unbalanced node (A)

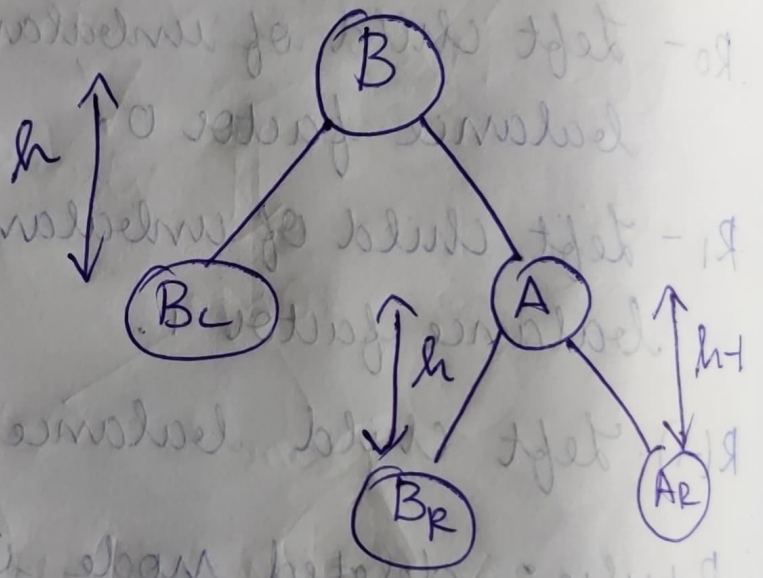
L type: deleted node is in left of unbalanced node (A).

R0 Generic Representation

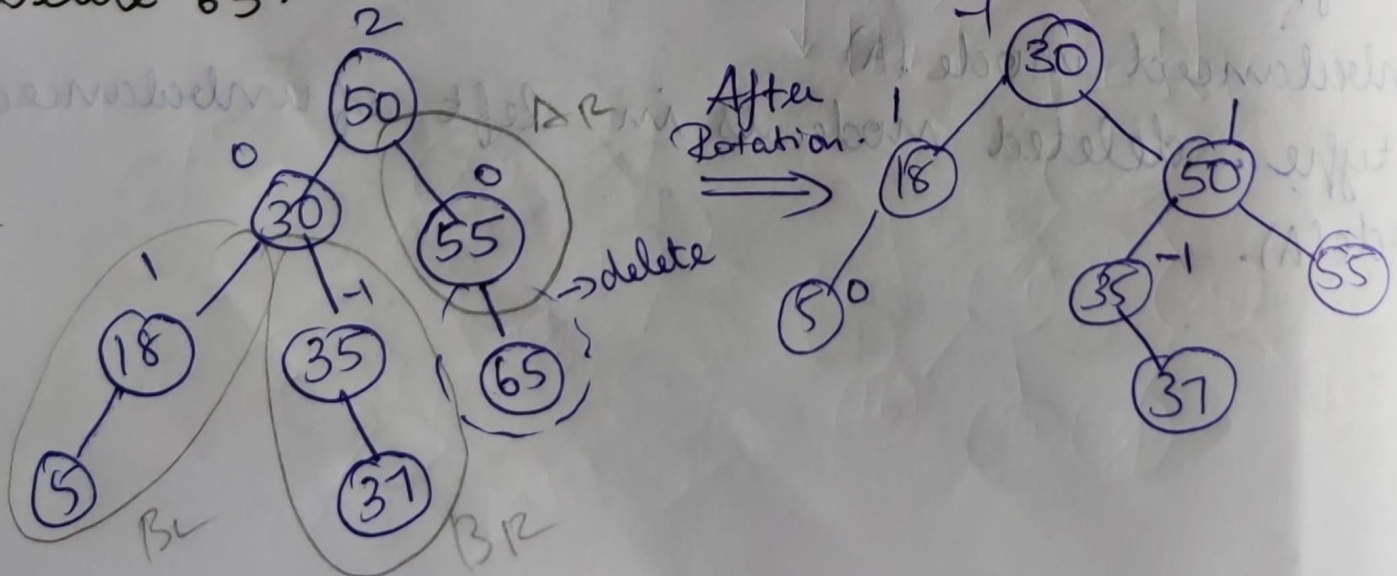


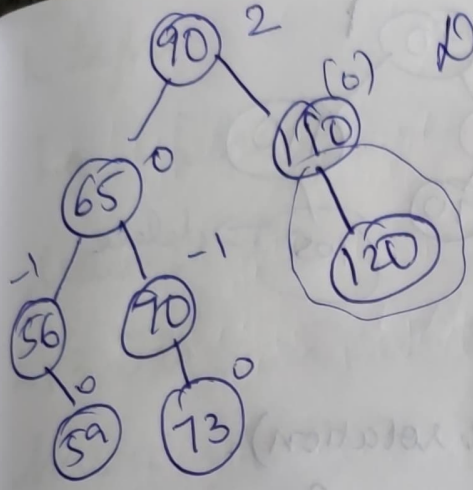
Before deletion.

After Rotation.

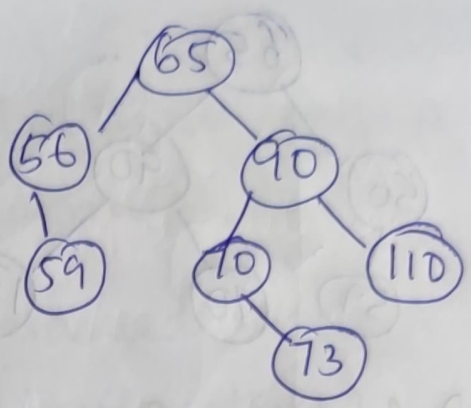


Delete 65.

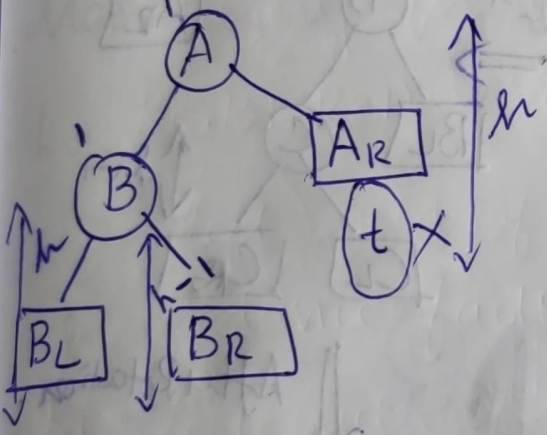




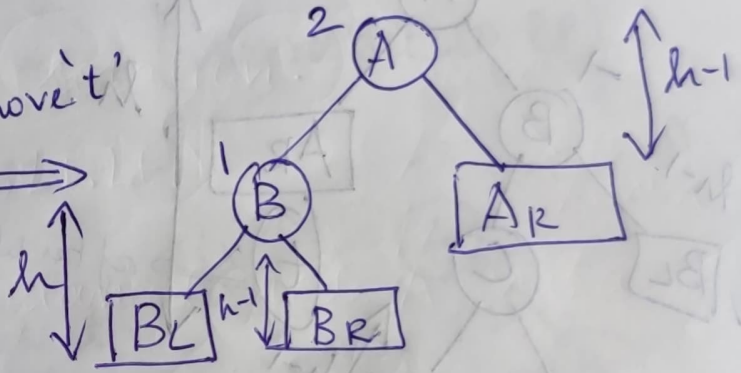
Delete 120:
R_0 rotation.



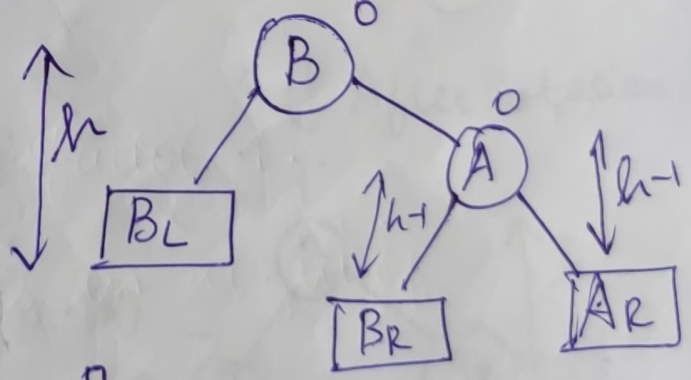
R_1 Rotation



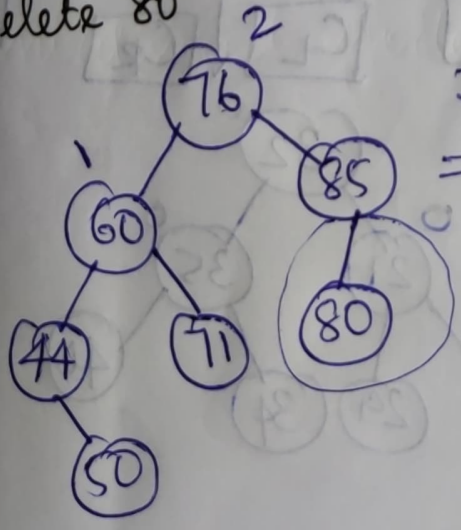
Remove 't'



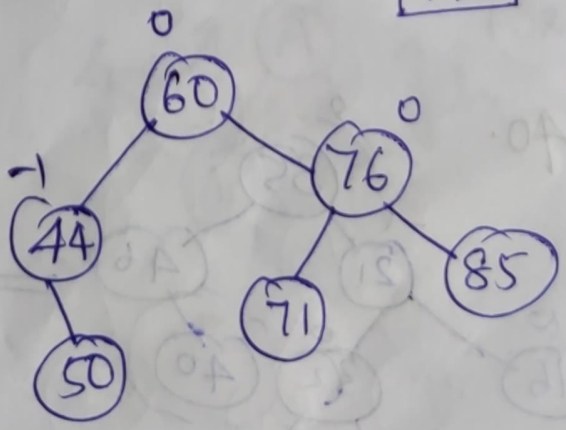
After Rotation.

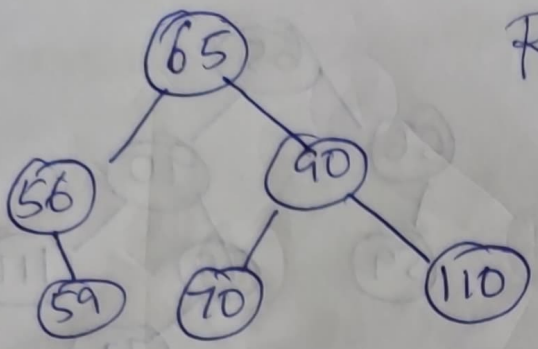


Delete 80

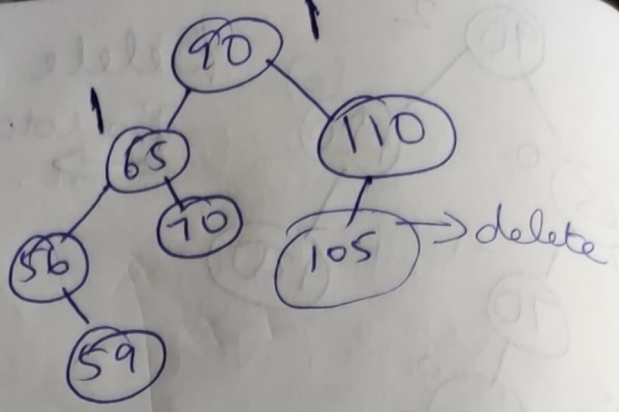


R_1

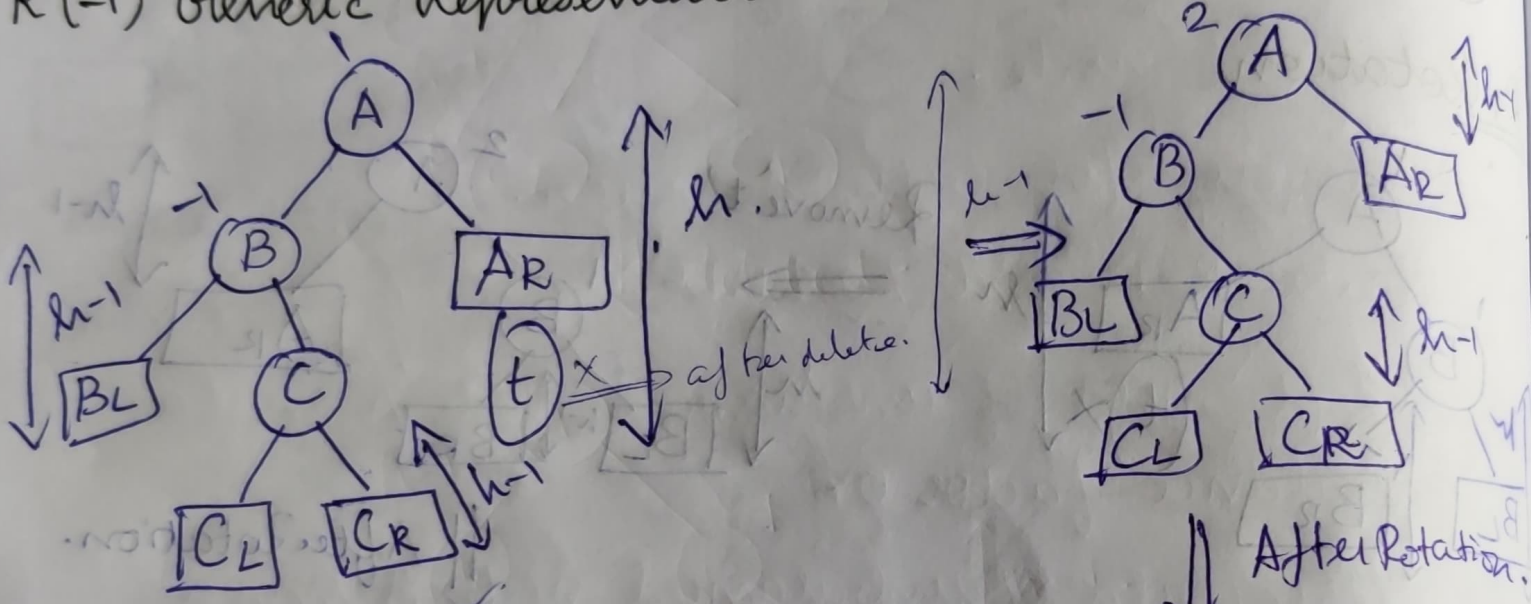




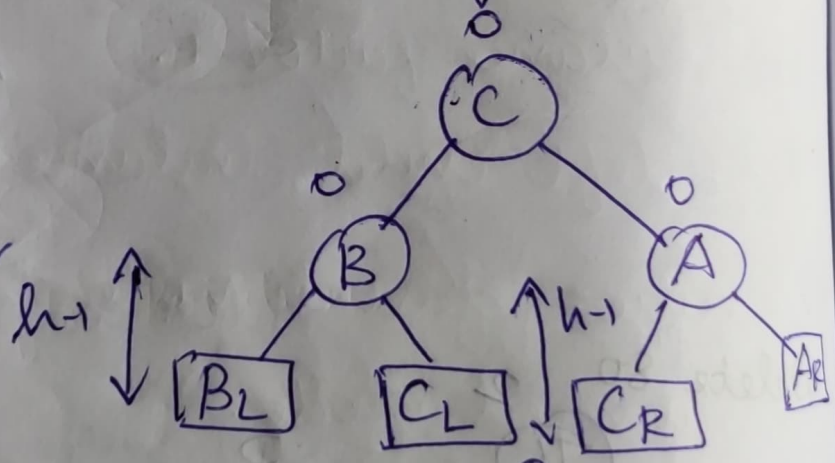
R_1 Rotation.



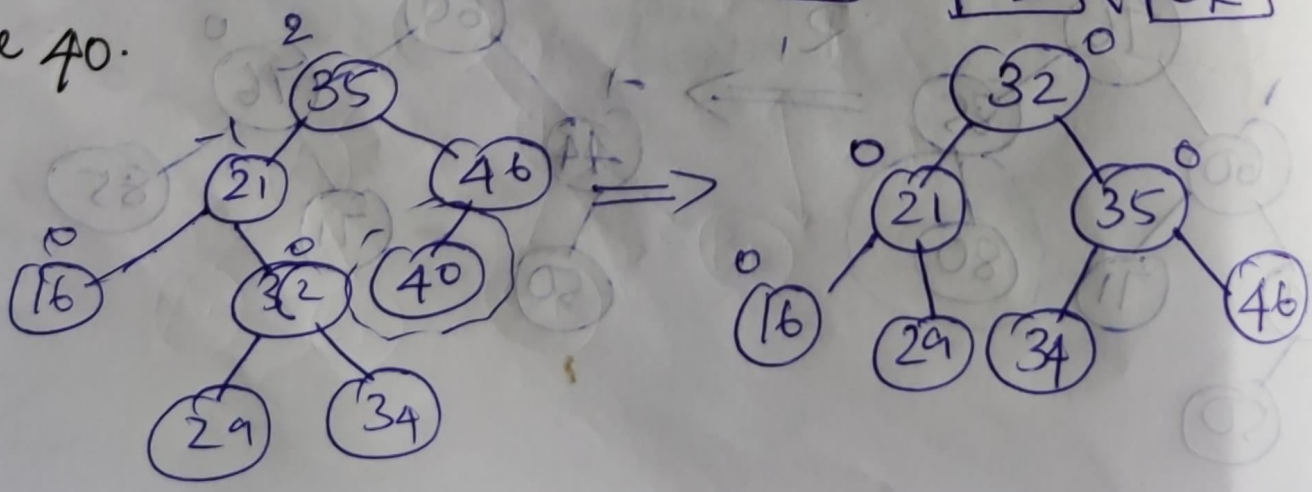
R(-1) Generic Representation (like LR rotation).



After Rotation.



Delete 40.



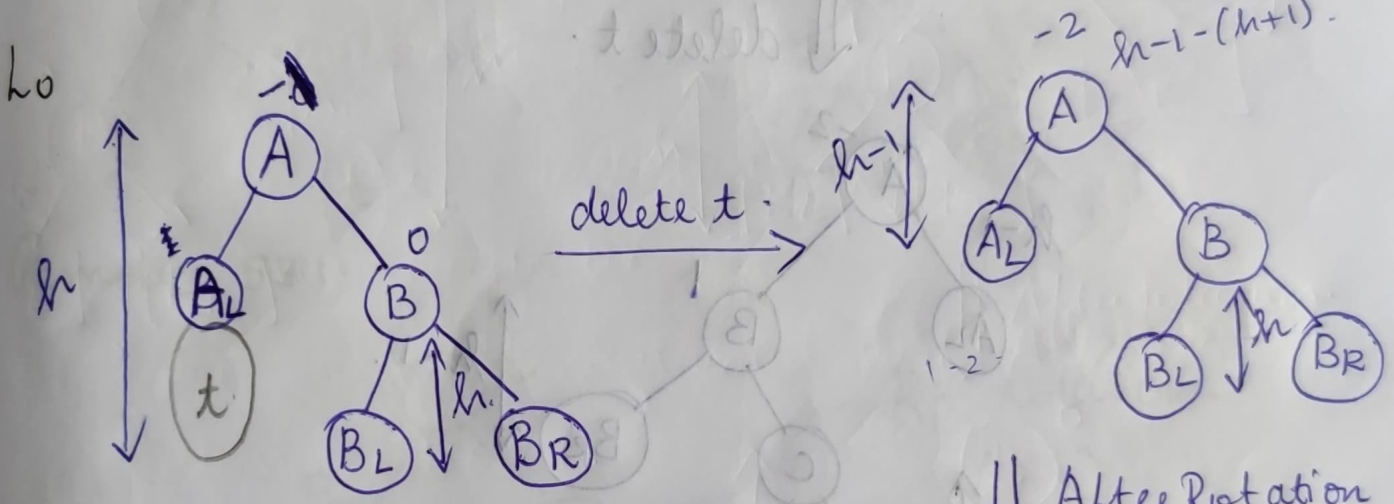
L type.

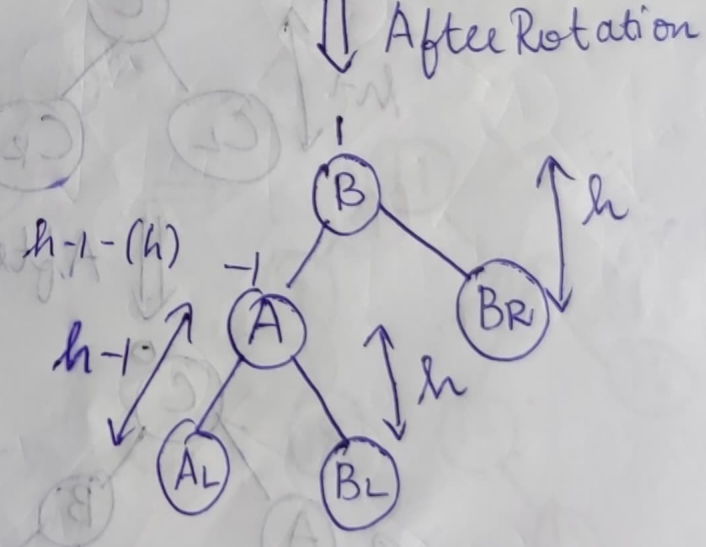
L0: The right child of unbalanced node A has a balance factor of 0. (similar to RR).

L1: The right child of unbalanced node A has a balance factor of 1 (similar to RL)

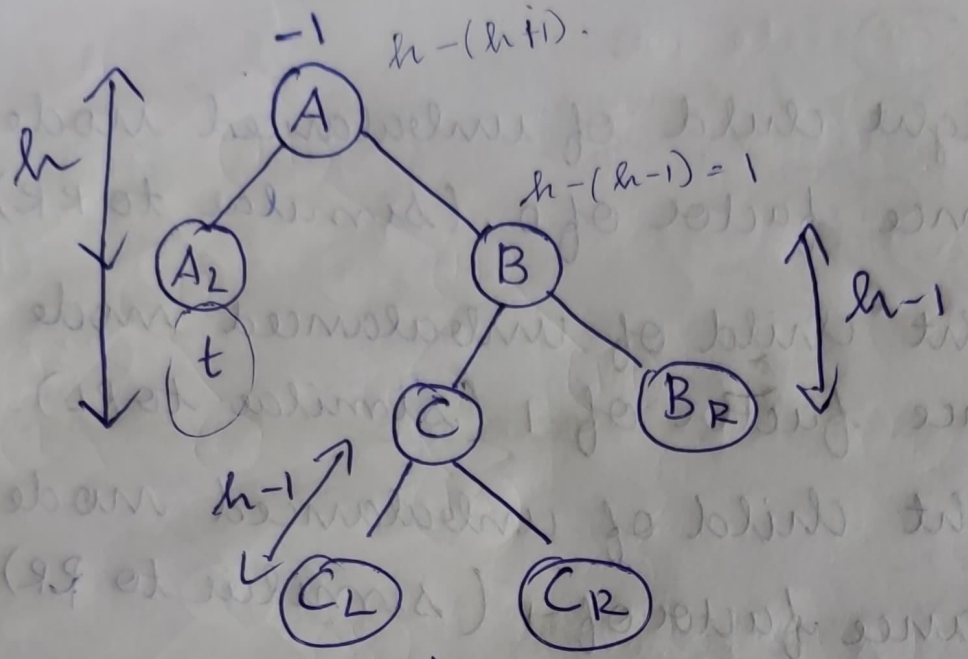
L-1: The right child of unbalanced node A has a balance factor of -1 (similar to RR)

L0

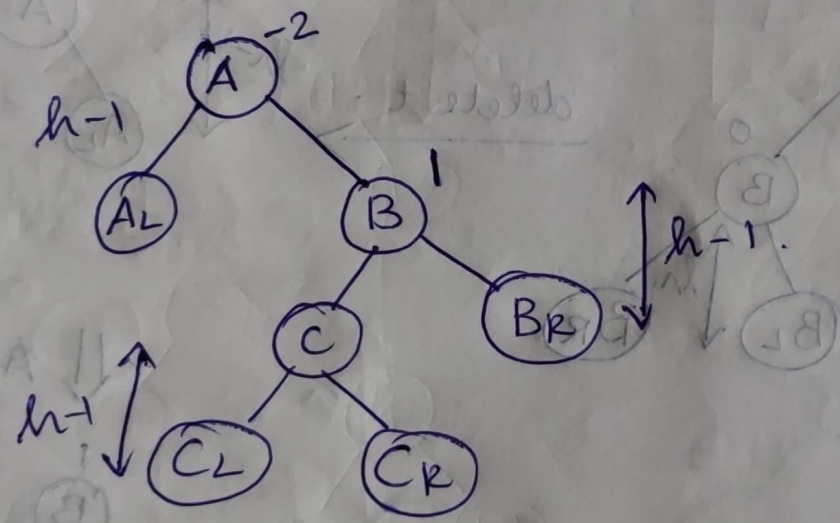




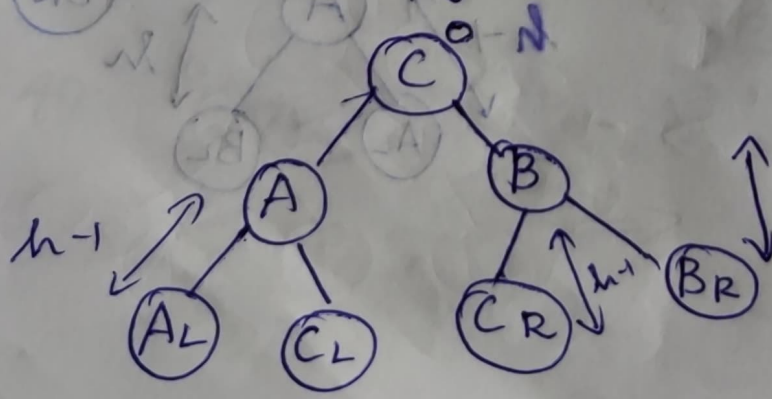
h_1:



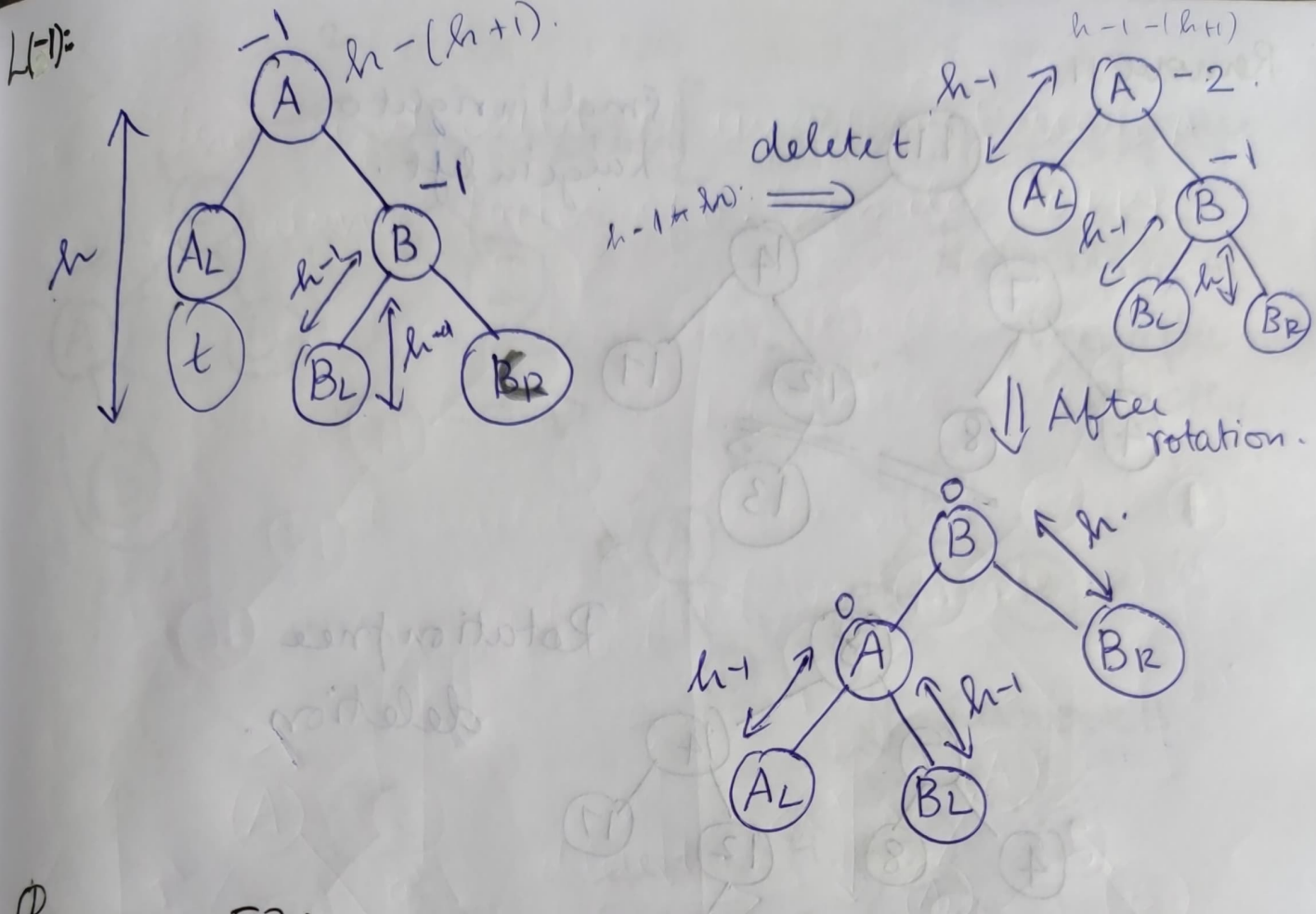
⇓ delete t.



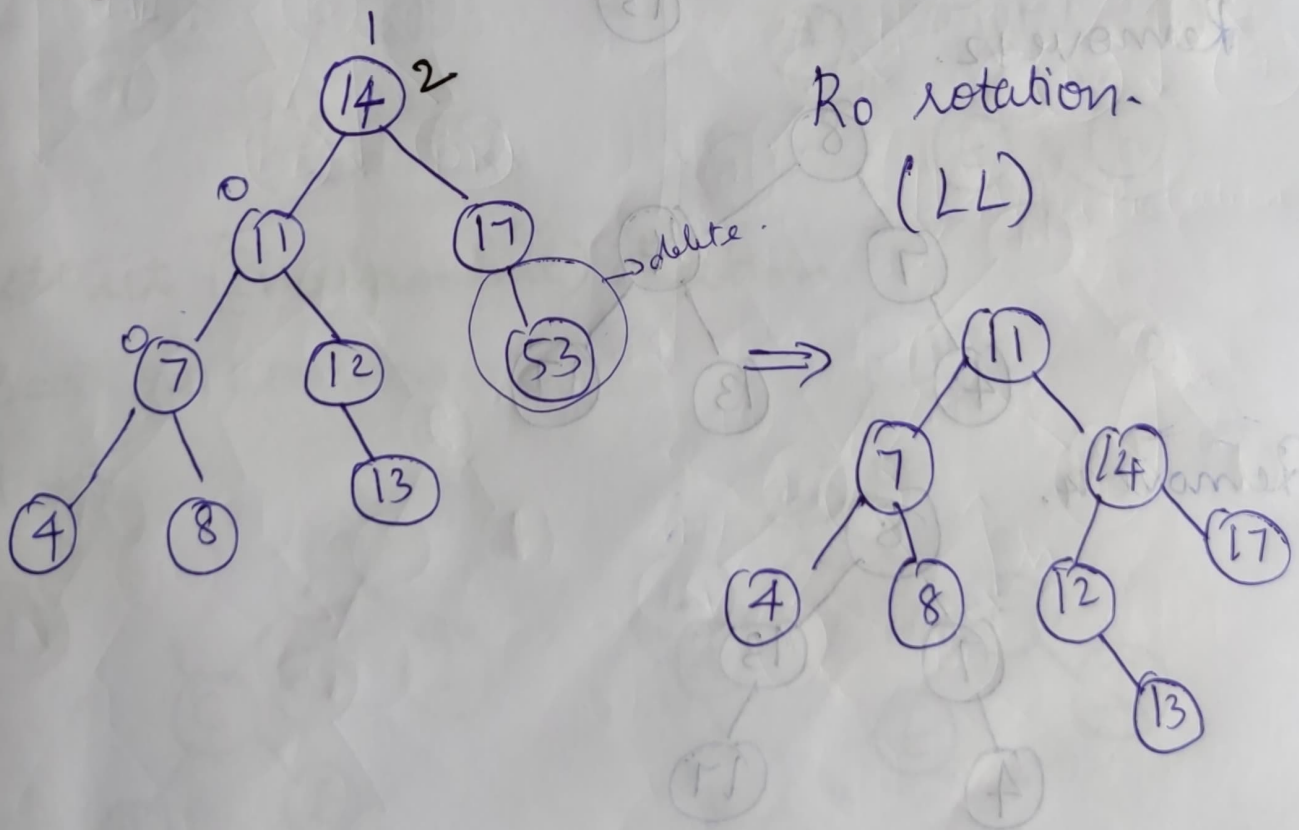
⇓ After Rotation.



L(-1):


A
AL
t
B
BL
BR
delete t
After rotation.
A
AL
B
BL
BR

Remove 53:


14
11
17
7
12
4
8
13
53
delete.
Ro rotation.
(LL)
11
7
14
4
8
12
17
13