Trees

Root → The top-most node.

Parent → The node which is predecessor of a node.

Child → The node which is immediate successor.

Sibling → children of a same parent node.

Descendent → any successor node on the path from leaf node to that node.

Ancestor → any predecessor node on the path of the root to that node.

Degree of node → The total count of subtree attached to the node

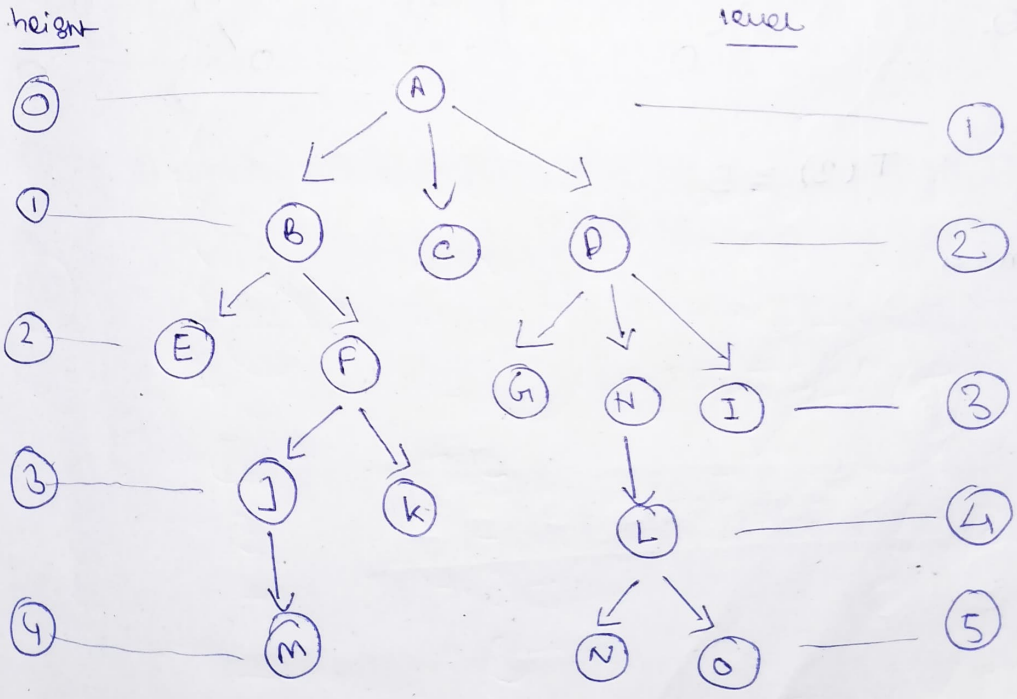
Internal / external node → leaf node.
(the node with atleast one child)

level → no of nodes

height → no. of edges.

forest → collection of tree is forest

no. of edges → no. of nodes - 1



count edges

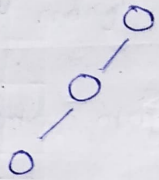
count node

Binary tree

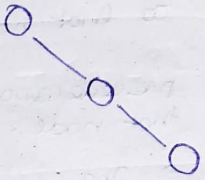
It cannot have more then two children and can have less then 2.

$deg(T) = 2$

Children = $\{0, 1, 2\}$

① 

left skewed

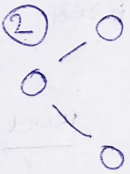
② 

right skewed

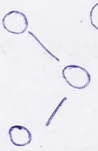
Number of Binary Tree for unlabelled

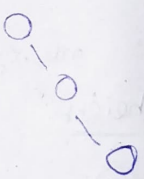
$n = 3$

① 

② 

③ 

④ 

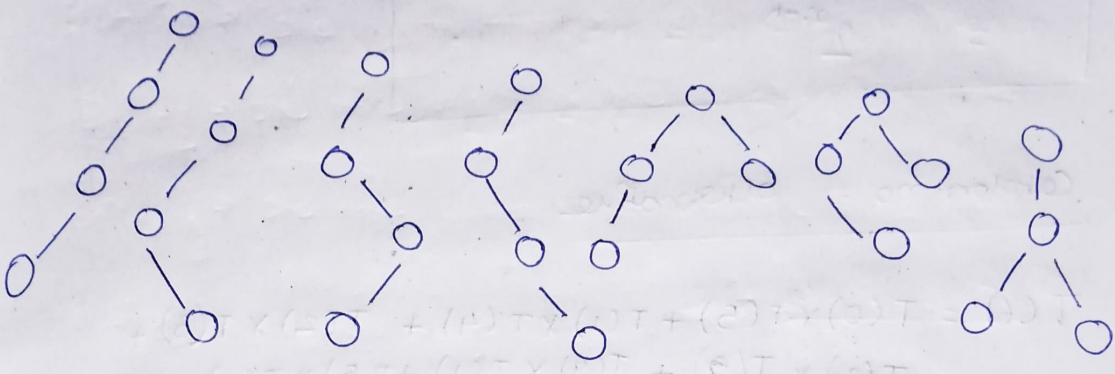
⑤ 

$T(3) = 5$

$n = 4$

$n = 4$

0000



2 more mirror image

$T(4) = 14$

$$T(n) = \frac{2n C_n}{n+1} = \text{catalan number}$$

$$nC_r = \frac{n!_0}{r!_0 (n-r)!_0}$$

$$nP_r = \frac{n!_0}{(n-r)!_0}$$

$\frac{2nC_n}{n+1}$

for no. of node = 5

$$\frac{10C_5}{6} = \frac{10!_0}{5!_0\,5!_0} \times \frac{1}{6}$$

$6 \times 7 \times 8 \times 9 \times 10$

$1 \times 2 \times 3 \times 4 \times 5$

$14 \times 3 = 42$

42 no. of binary tree

no. of nodes with max height

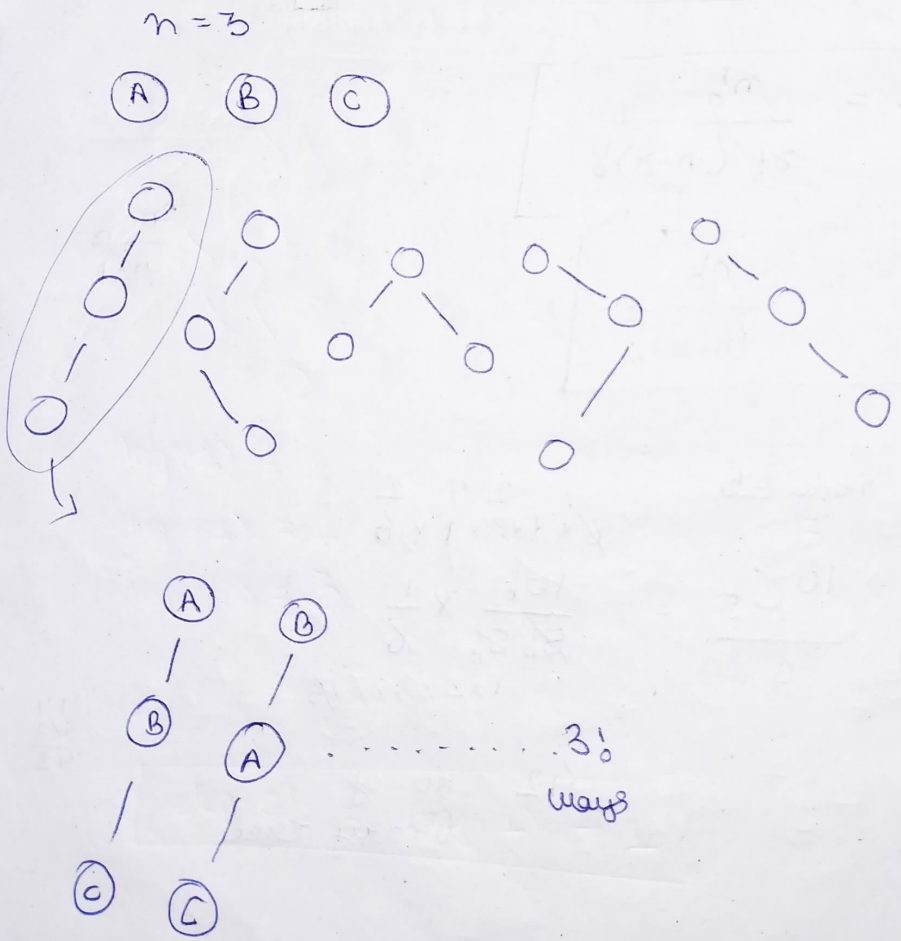
$= 2^{n-1}$

Catalan no. alternative

$T(6) = T(0) \times T(5) + T(1) \times T(4) + T(2) \times T(3) + T(3) \times T(2) + T(4) \times T(1) + T(5) \times T(0)$.

$$T(n) = \sum_{i=1}^{n} T(i-1) * T(n-i)$$

For labelled



formula =

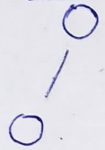
$$T(n) = \frac{{}^{2n}C_n}{(n+1)} * n!_0$$

Height vs node

height

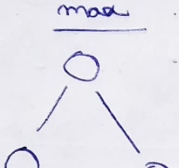
h = 1

min



n = 2

max



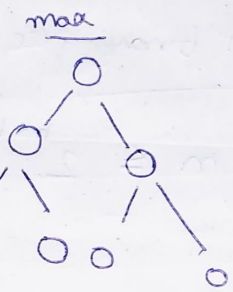
n = 3

h = 2

min

n = 3

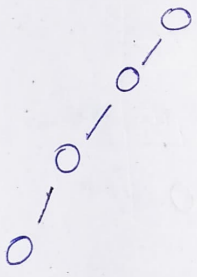
max



n = 7

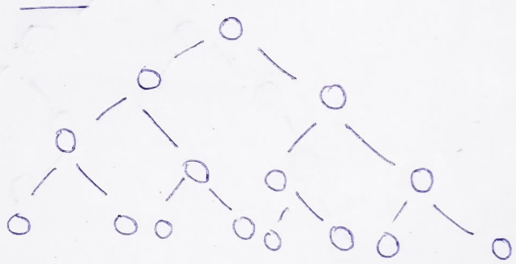
h = 3

min



n = 4

max



n = 15

min nodes n = h + 1

max nodes = $2^{h+1} - 1$

minimum no. of nodes = h + 1

maximum no. of nodes = $2^{h+1} - 1$

When nodes are given,

max height $\Rightarrow h = n-1$

min height $\Rightarrow$ $n+1 = \log_2(n+1)$

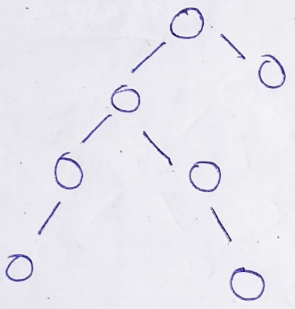
Height of binary tree

$$\log_2(n+1) - 1 \leq h \leq n-1$$

No. of nodes in binary tree

$$h+1 \leq n \leq 2^{h+1} - 1$$

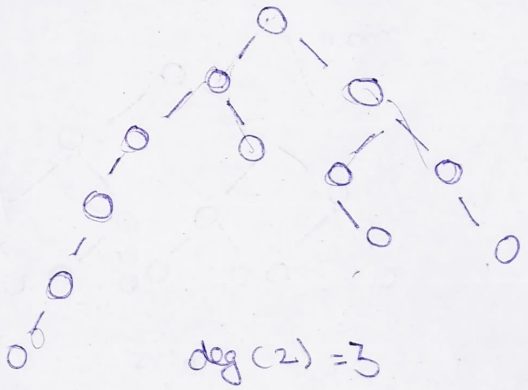
Height vs Node



deg(2) = 2

deg(1) = 2

deg(0) = 3



deg(2) = 3

deg(1) = 5

deg(0) = 4

$$\deg(0) = \deg(2) + 1$$

Trees in binary tree

Strict binary tree

1. Strict / Proper / complete
2. Height vs Nodes
3. external vs internal

Strict

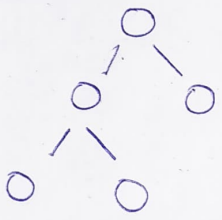
{0, 1, 2}

0 or 2

Height vs Nodes

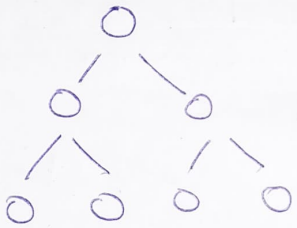
h = 2

min



n = 5

max



n = 7

for strict

If height 'h' is given

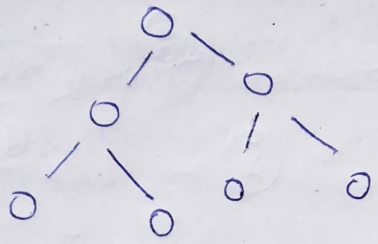
min nodes	= n	=	$2h + 1$
max "	= n	=	$2^{h+1} - 1$

If 'n' nodes are given

min	Height h	=	$\log_2(n+1) - 1$
max	Height h	=	$\frac{n-1}{2}$

Internal & external node

For strict



$i = 3$

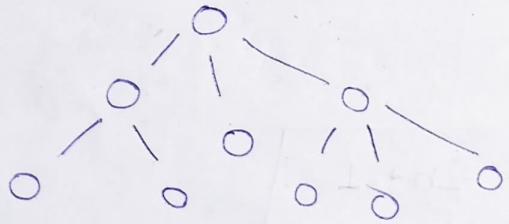
$e = 4$

$e = i + 1$

no. of external = no. of internal + 1

n-ary trees

3-ary tree {0,1



full binary tree.

Strict m ary tree

If height is given

min nodes $\Rightarrow n = mh + 1$

max nodes $\Rightarrow \frac{m^{h+1} - 1}{m-1}$

If height is given

n_{min}(nodes) $= mh + 1$

$n_{max} = \frac{m^{h+1} - 1}{m-1}$

$n(m-1) = m^{h+1} - 1$

$n(m-1) + 1 = m^{h+1}$

$\log\, n(m-1)+1 = h+1 \log m$

$\frac{n(m-1)+1}{\log m}$

$\log_m n(m-1)+1$

If nodes are given

min height $\Rightarrow h = \log_m [n(m-1)+1] - 1$

max height $\Rightarrow \frac{n-1}{m} = h$

For strict m-ary tree

$e = 2i + 1$

external $= 2 \times$ internal $+ 1$

form

$e = (m-1)i + 1$

min height

$mh + 1$

max

$n(m-1) = m^{h+1} - 1$

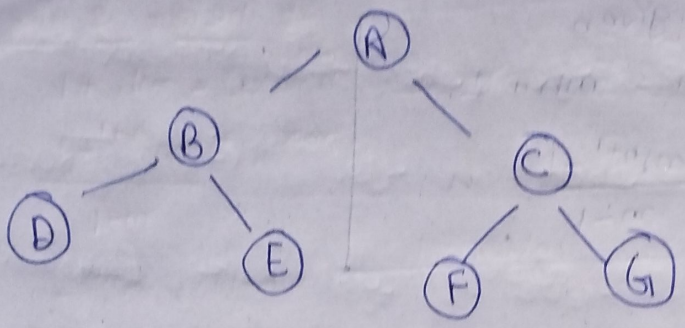
$\frac{\log\{n(m-1)+1\}}{\log m} = h+1$

$\frac{n-1}{m} = h$

$n = \frac{m^{h+1} - 1}{m-1}$

$\frac{m^{h+1} - 1}{m+1}$

Representation

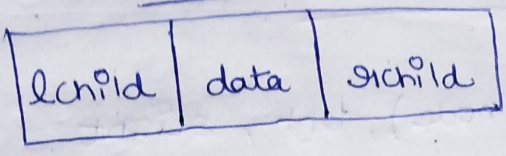


A	B	C	D	E	F	G	
0	1	2	3	4	5	6	
1	2	3	4	5	6	7	

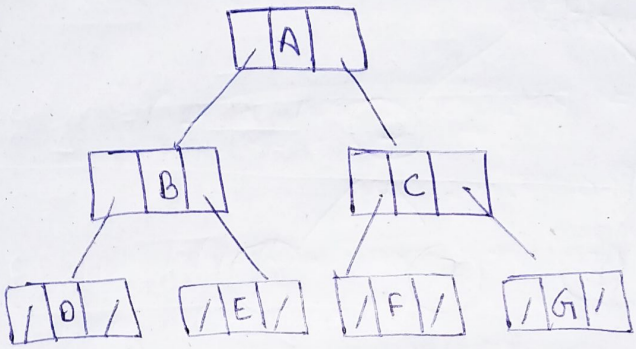
element	index	L child	R child
A	1	2	3
e	i	2*i	2*i + 1

Parent = i/2

Node




```
struct Node
{
    struct Node * lchild;
    int data;
    struct Node * rchild;
};
```



full is always complete binary tree

$n = 7$ $\quad e = i+1$

null pointers = $n+1$ = NULL

Full vs complete binary tree

↳ for a given height maximum no. of nodes must be present.

Complete → Full binary tree uptill $(h-1)$ height.
and h^{th} height element must be filled from left to right without skipping.

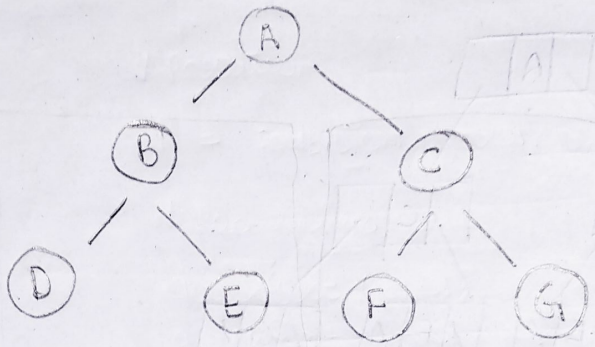
Tree Traversal

Preorder → visit(node), left, right

Inorder → left, node, right

Postorder → left, right, node

level order → level by level.



Preorder → A B D E C F G

In order → D B E A F C G

Post order → D E B F G C A

level order → A B C D E F G

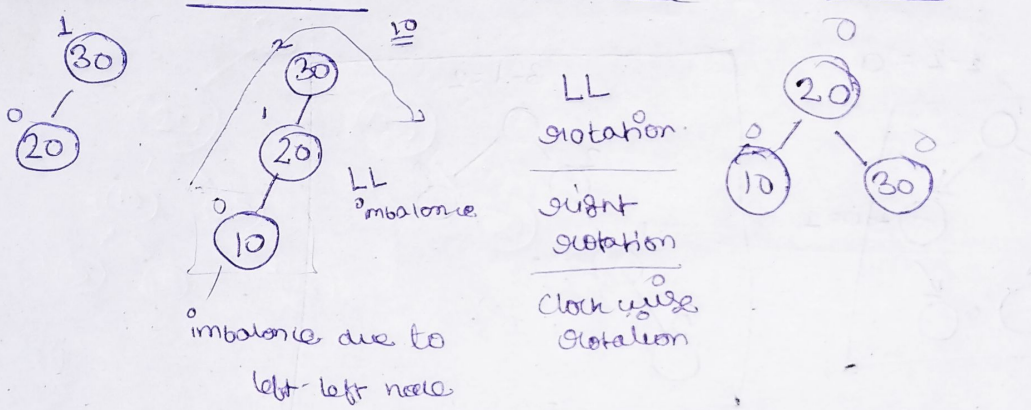
Rotation for insertion

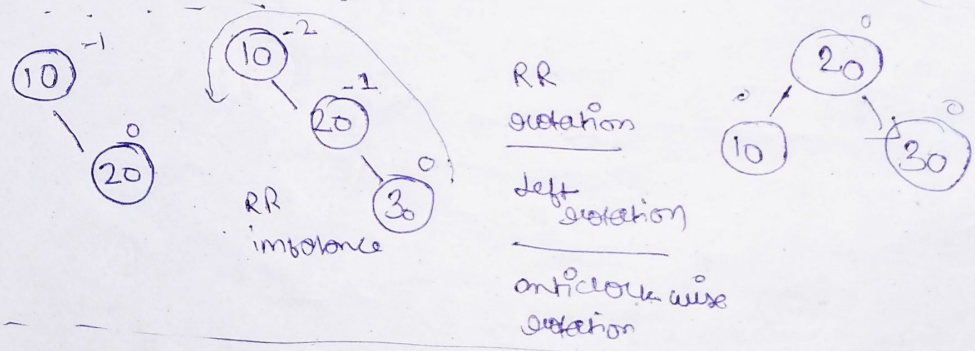
Four rotation

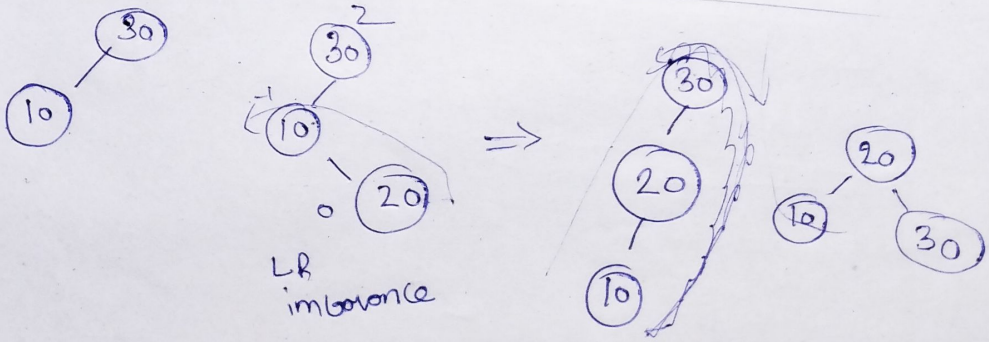
1. LL rotation
2. RR rotation
3. LR rotation
4. RL rotation

LL Rotation

Initial | After insert | Perform rotation | After rotation

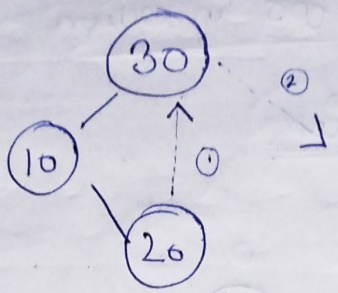


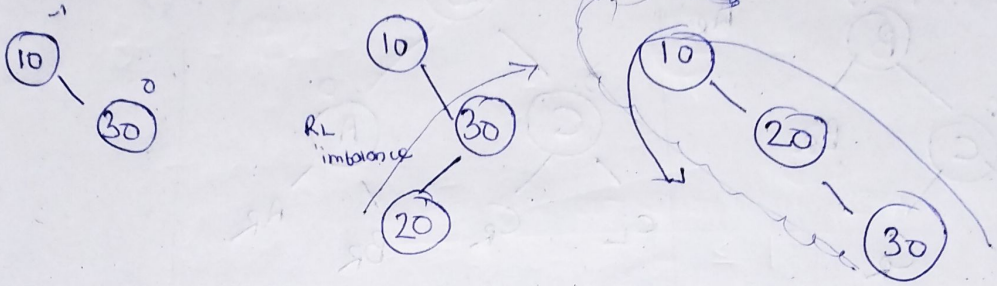


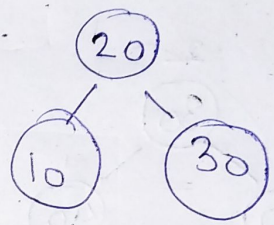


double rotation

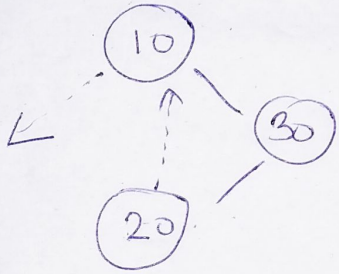
Shortcut







Shortcut



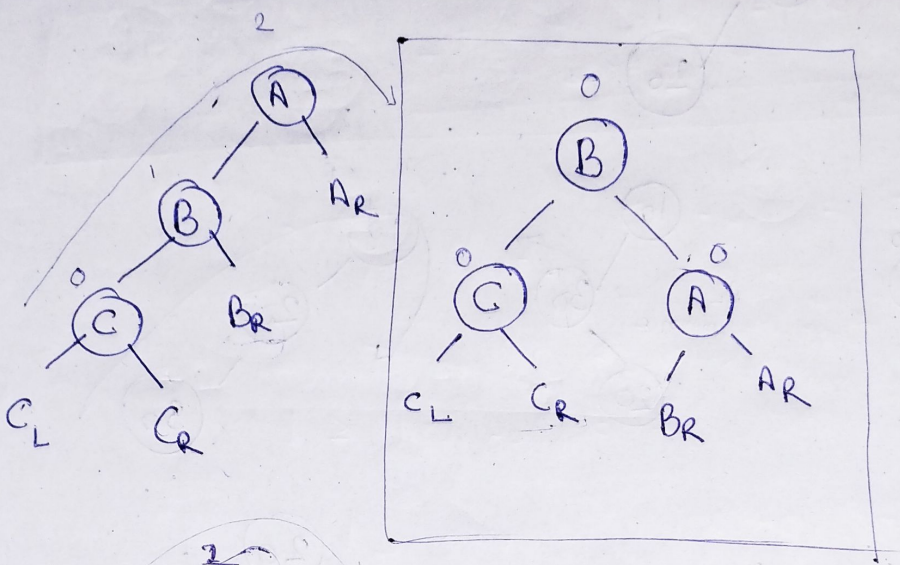
double rotation

LL Rotation }
RR " } single rotation.

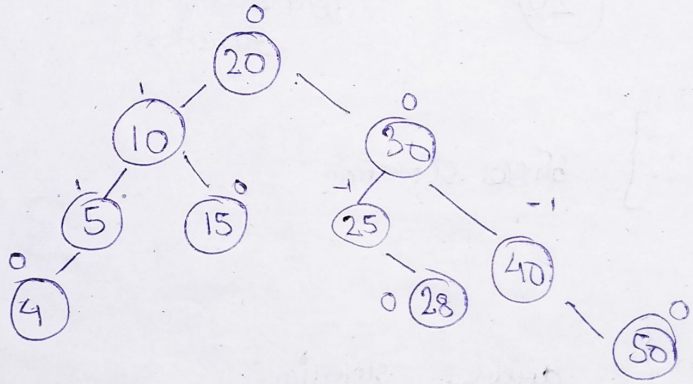
LR Rotation }
RL " } double rotation

Formula of rotation for insertion

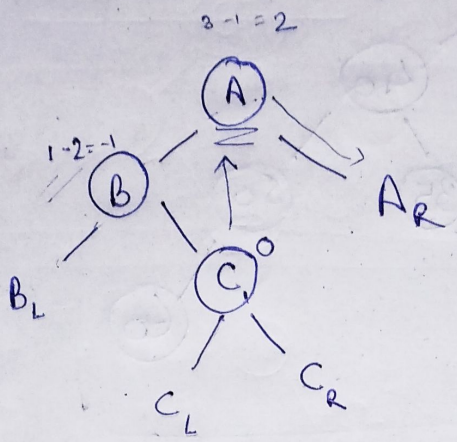
LL Rotation

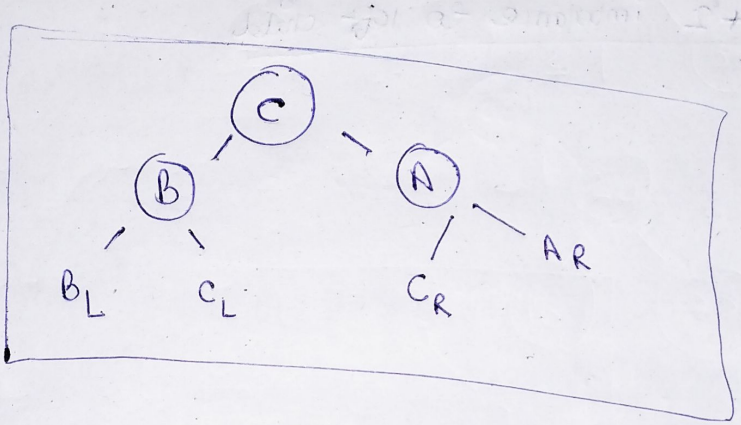


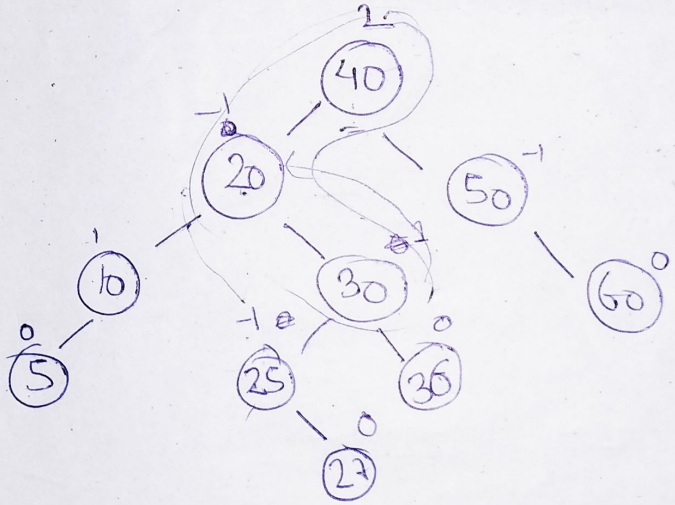
consider only 2 ~~steps~~ nodes.

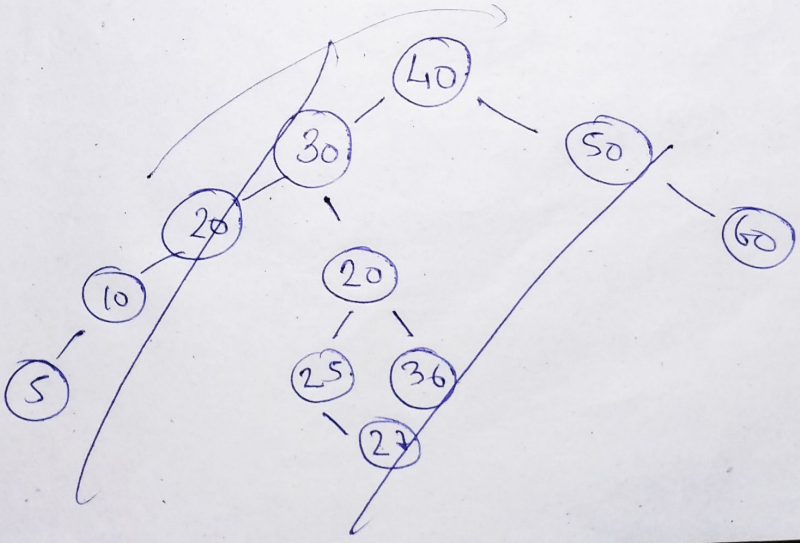


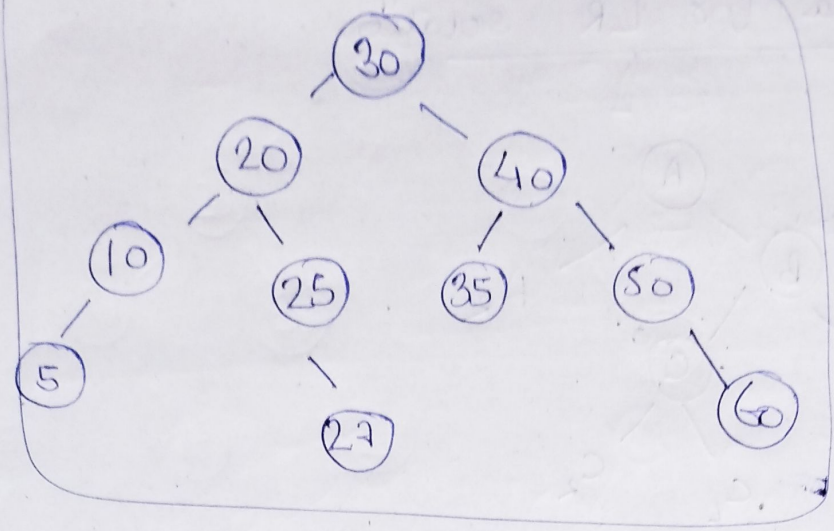
formula for LR rotation











+2 imbalance to left child

AVL

Balance factor = height of left subtree − height of right subtree

$bf = hl - hr = \{-1, 0, 1\}$

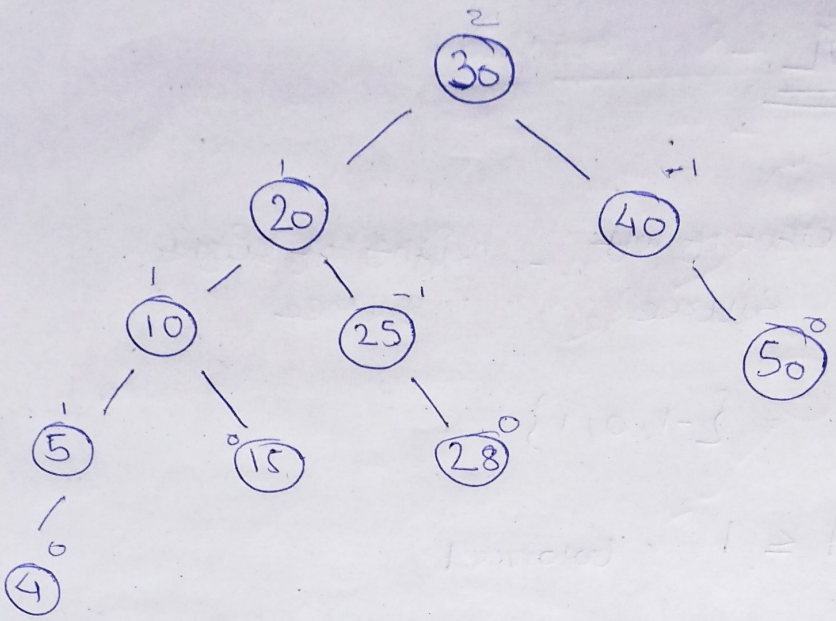
$bf = |hl - hr| \leq 1$ balanced

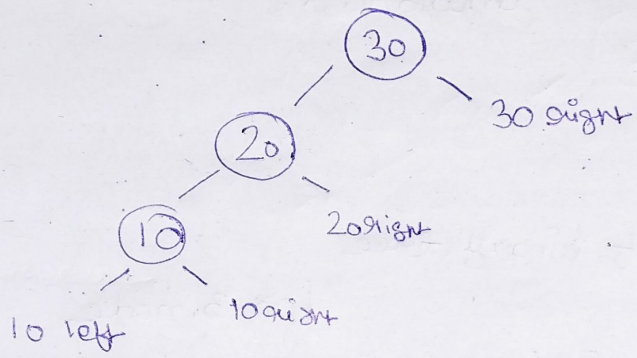
If $bf = |hl - hr| > 1$ imbalanced

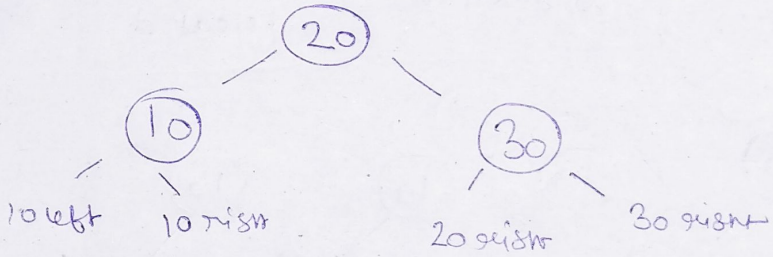
$\left(\frac{^{2n}C_n}{n+1}\right)$ no. of binary tree

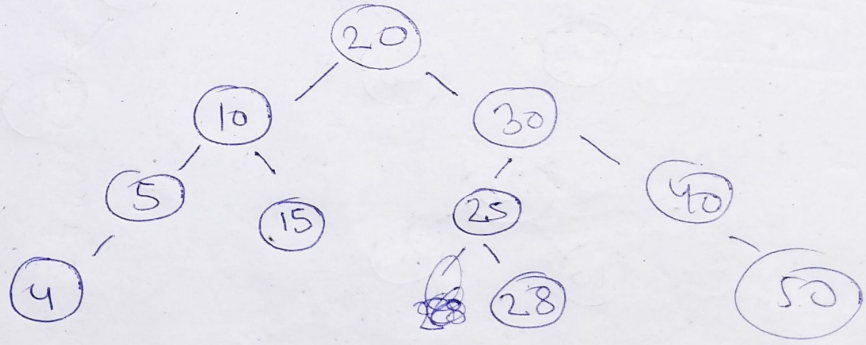
for 3 node

$n = 3$ $\quad \frac{^{6}C_3}{4} = \frac{6 \cdot 5 \times 6 \times 4}{3! \cdot 3 \times 4} = 5$ trees can be created

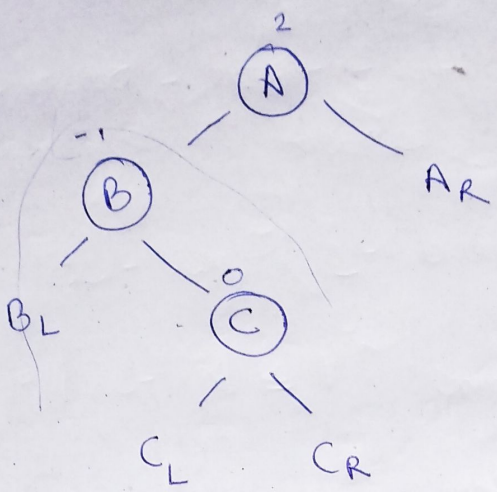

30
20
40
10
25
50
5
15
28
4

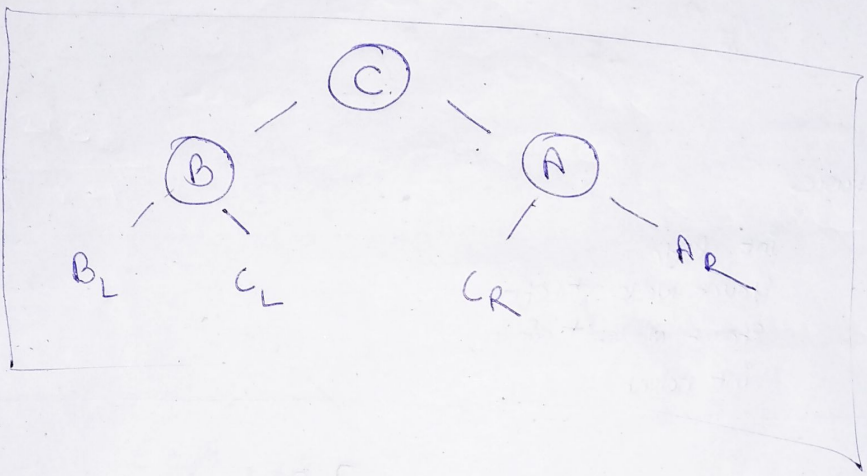

30
30 right
20
20 right
10
10 left
10 right


20
10
30
10 left
10 right
20 right
30 right

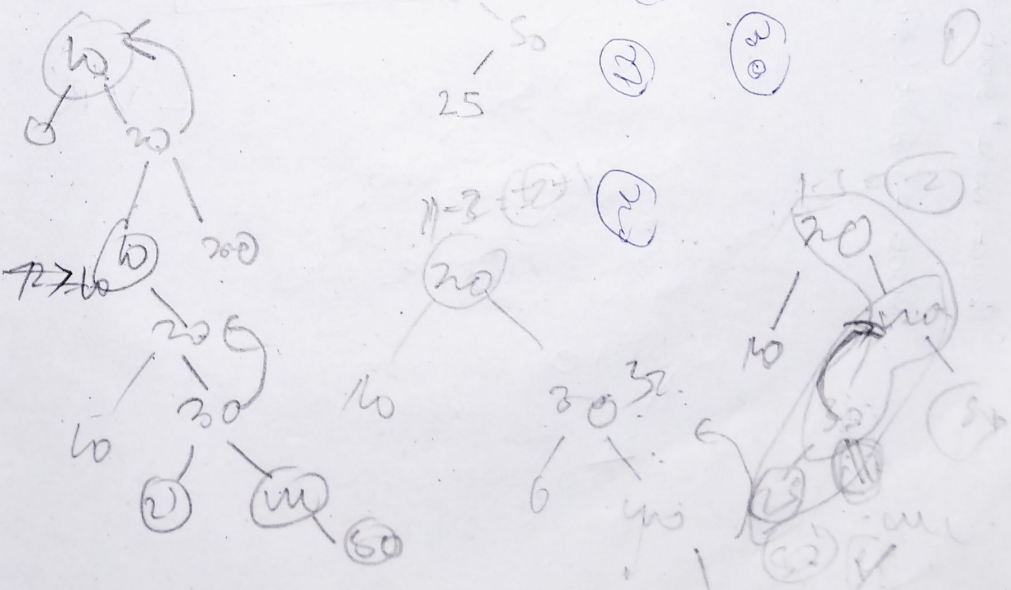

20
10
30
5
15
25
40
4
28
50

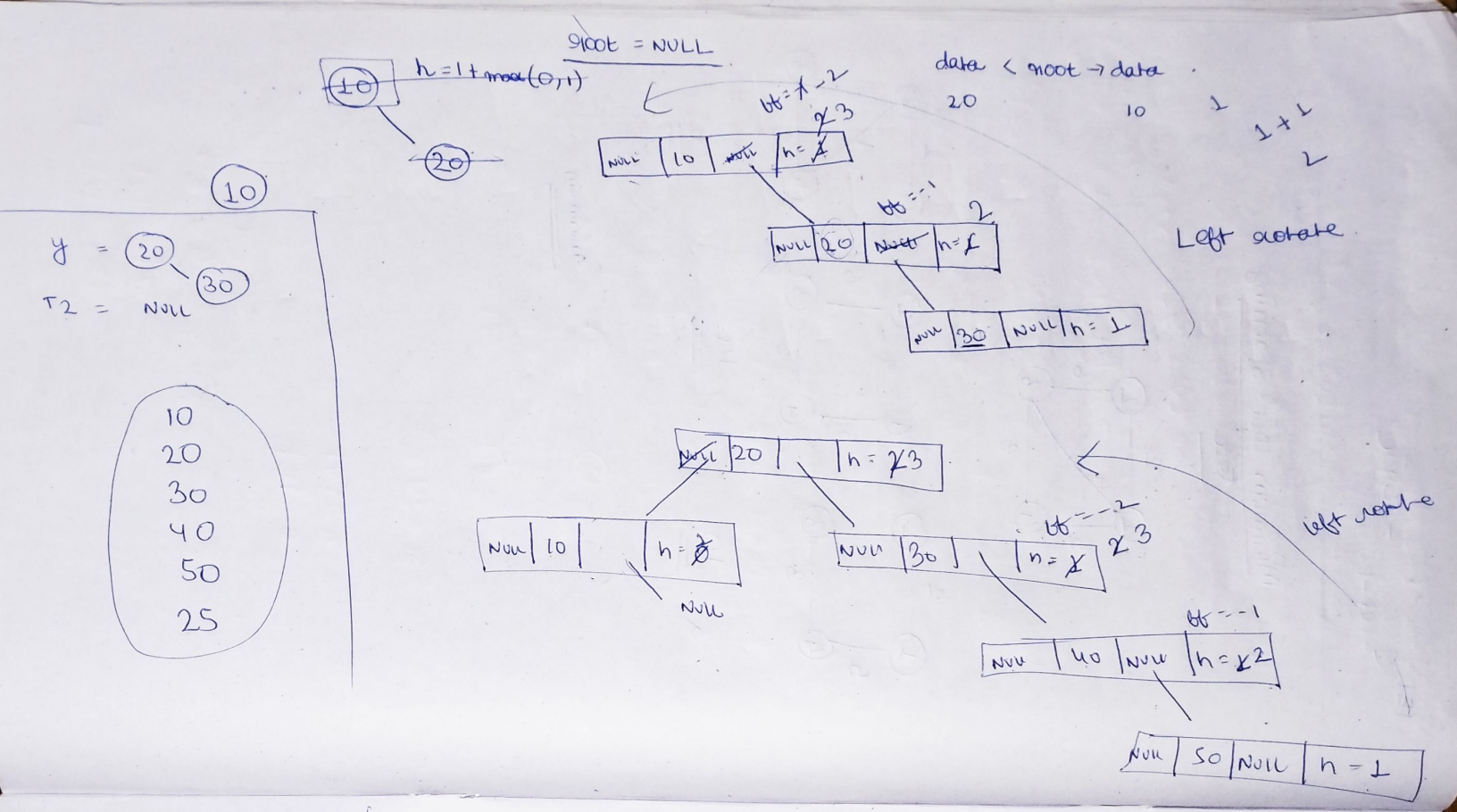
LR - Rotation

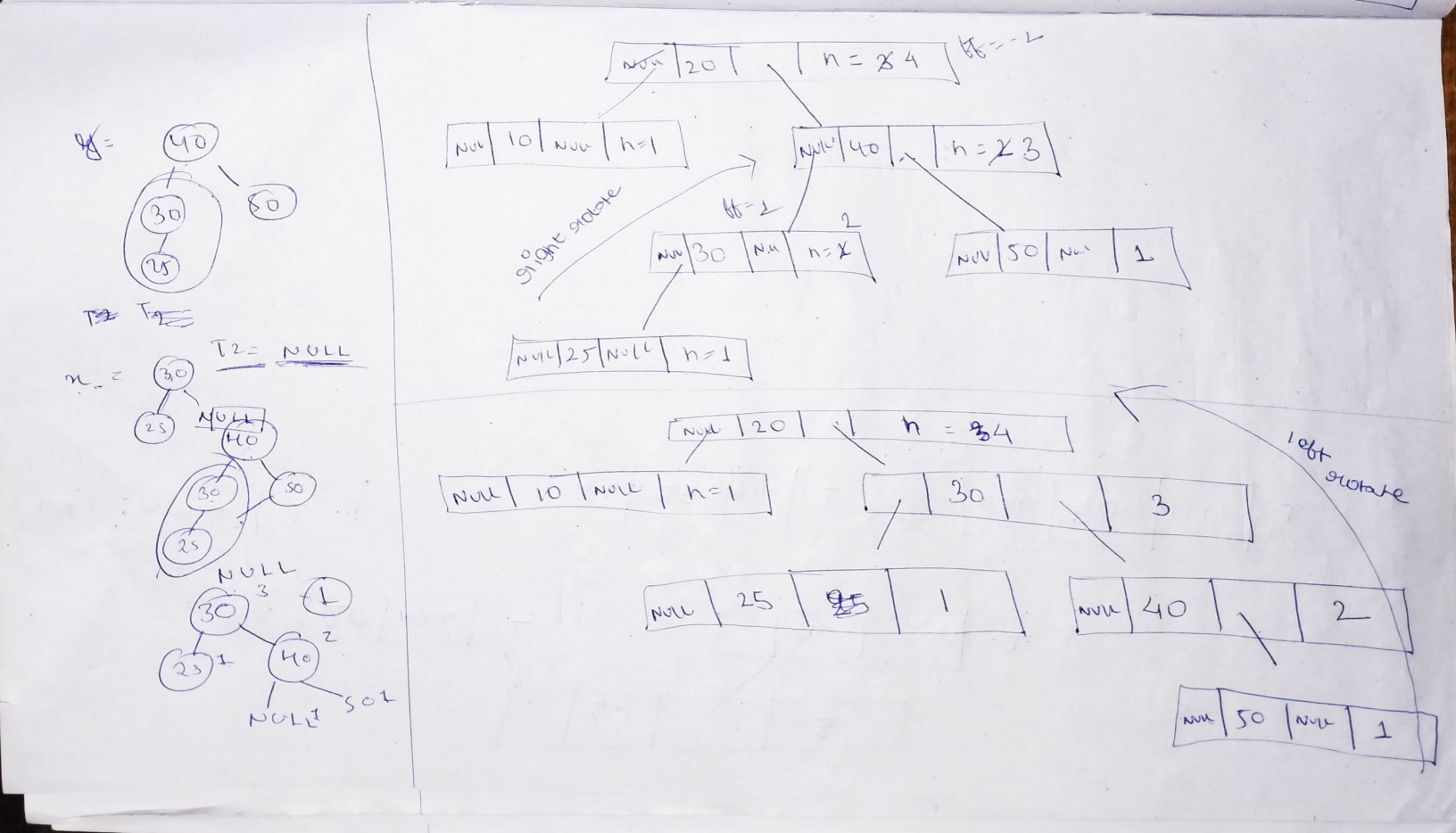


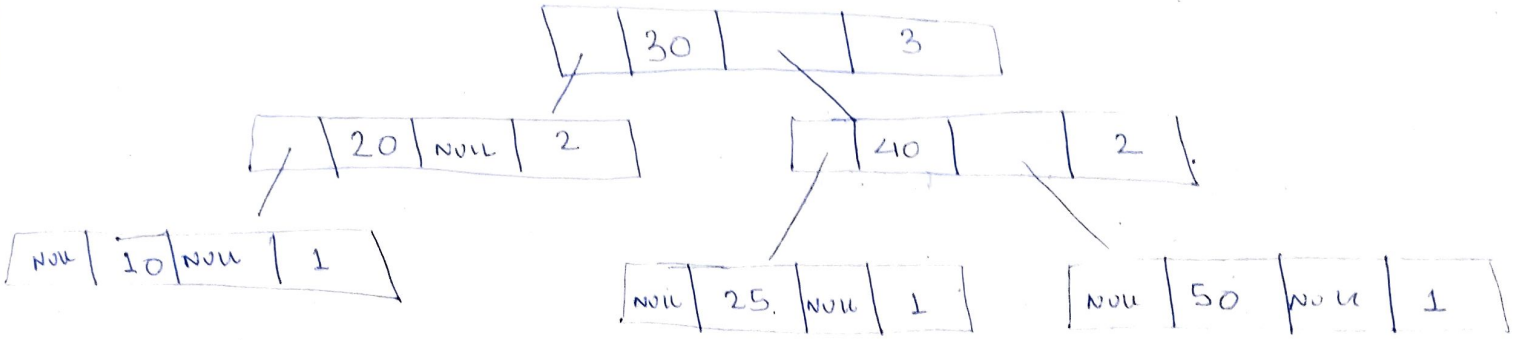


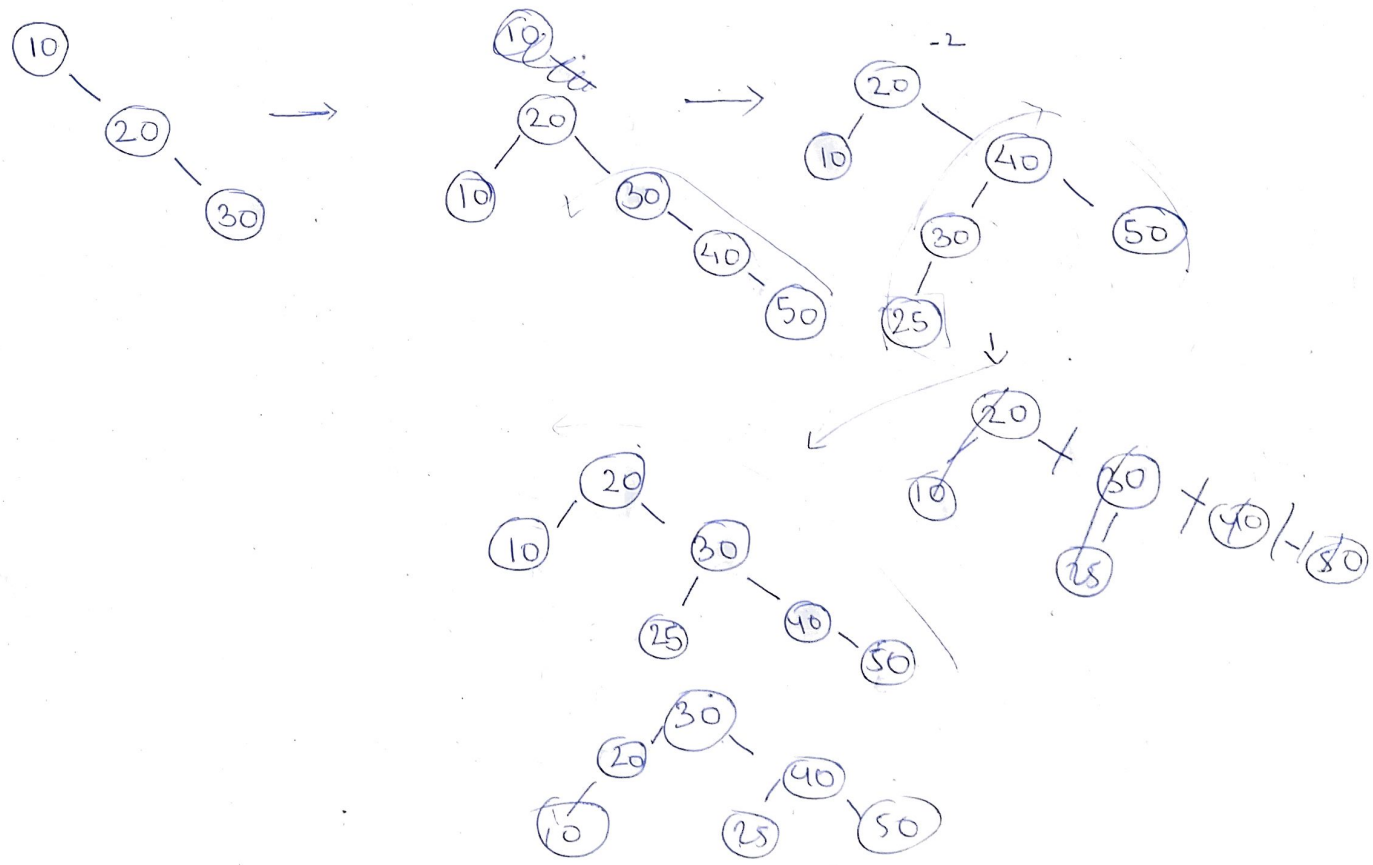
AVL




root = NULL
h = 1 + max(0,1)
10
20
bf = 1 - 2
x3
NULL | 10 | NULL | h = 1
bf = -1
2
NULL | 20 | NULL | h = 1
NULL | 30 | NULL | h = 1
data < root → data
20
10
1
1 + 1
2
Left rotate
y = 20
30
T2 = NULL
10
20
30
40
50
25
NULL | 20 | h = x3
NULL | 10 | h = 0
NULL
NULL | 30 | h = 1
bf = -2
x3
left rotate
bf = -1
NULL | 40 | NULL | h = x2
NULL | 50 | NULL | h = 1

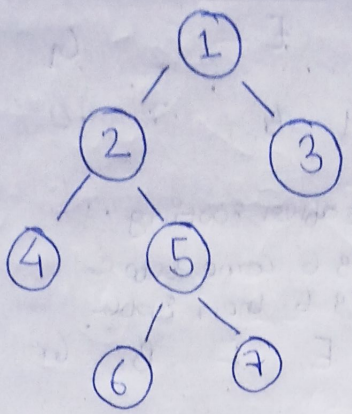

40
30
50
25
T2 = NULL
30
25
NULL
40
30
50
25
NULL
30
3
1
25
1
40
2
NULL
NULL
NULL | 20 | h = 4
bf = -2
NULL | 10 | NULL | h = 1
NULL | 40 | h = x3
right rotate
bf = 2
NULL | 30 | NULL | h = 1
NULL | 50 | NULL | 1
NULL | 25 | NULL | h = 1
NULL | 20 | h = 4
NULL | 10 | NULL | h = 1
30
3
NULL | 25 | 1
NULL | 40 | 2
left rotate
NULL | 50 | NULL | 1


30
3
20
NULL
2
40
2
NULL
10
NULL
1
NULL
25
NULL
1
NULL
50
NULL
1


10
20
30
-2
40
50
25
+40
-80

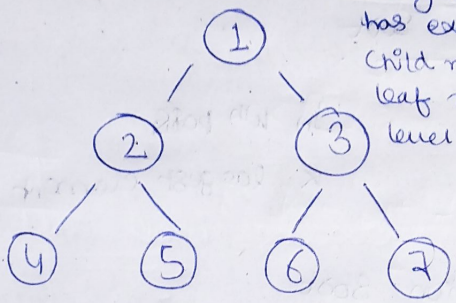
Full binary tree

→ either two or zero child.



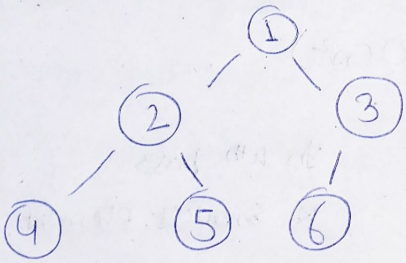
Perfect binary tree

→ Every internal node has exactly two child node and all the leaf node are at same level.



Complete binary tree

→ same as perfect binary tree but lean towards the left.



Deleting binary Search Tree

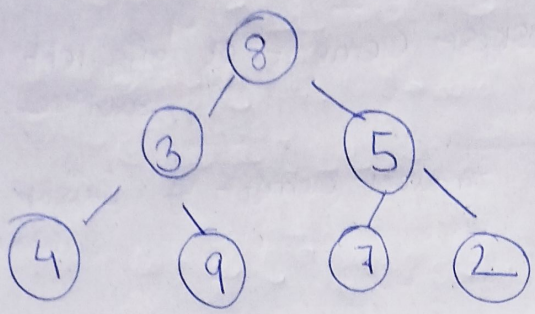
either go to largest element of left subtree

or go to smallest element of right subtree

1. If the node is the leaf node
   → simply remove from the tree
2. If the node has only one child.
   → copy the child to the node and delete the child.
3. If the node has two child
   → find the inOrder successor of the node copy the content and delete it.

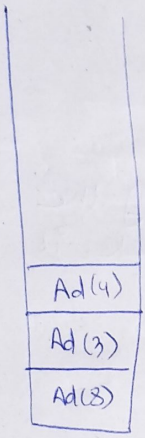
```
If (P == NULL)
    return NULL;

If (P->lchild == NULL && P->rchild == NULL)
{
    if (P == root)
        root = NULL;
    free (P);
    return NULL;
```

Iterative



Print left right

8	3	4	9	5	7	2



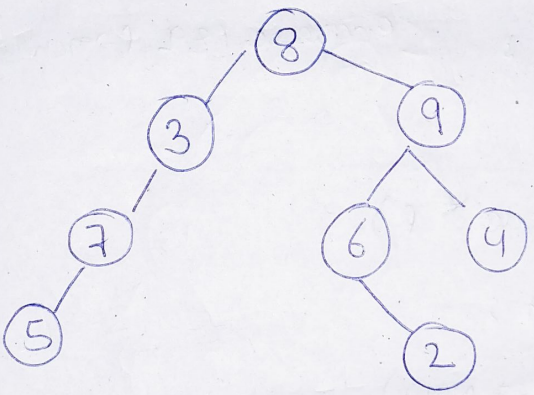
8 3

```
void preorder (Node *t)
{
    struct stack st;

    while (t != NULL || !IsEmpty(st))
    {
        if (t != NULL)
        {
            Printf (t->data);
            Push (&st, t);
            t = t->lchild;
        }
        else
        {
            t = pop (&st);
            t = t->rchild;
        }
    }
```

```
inorder
{
    struct stack st;
    while ( t != NULL || !isEmpty(st) )
    {
        if ( t != NULL)
        {   push(&st, t);
            t = t -> lchild
        }
        else
        {   t = t = pop(&st);
            printf ( t -> data);
            t = t -> rright;
        }
    }
```

---

level order traversal



8	3	9	7	6	4	5	2

```
Void levelOrder(Node *p)
{
    Queue q;
    printf("%d", p->data);
    enqueue(&q, p);
    while(!isEmpty(q))
    {
        p = dequeue(&q);
        if(p->lchild)
        {
            printf
        }   enqueue(&q, p->lchild);
        if(p->rchild)
        {
            Printf
        }   enqueue(&q, p->rchild)
}
```

```
int count(Node * p)
{
    int x, y;
    if(p != NULL)
    {
        x = count(p->lchild);
        y = count(p->rchild);
        if(p->lchild && p->rchild)
            return x+y+1;
        else
            return x+y;
        return 0;
}
```

Counting no. of leaf node

```
int count ( struct Node * P)
{
    int x, y;
    if (P != NULL)
    {
        x = count (P -> lchild);
        y = count (P -> rchild);
        if (P -> lchild == NULL && P -> rchild == NULL)
            return x+y+1;
        else
            return x+y;
    }
    return 0;
}
```

```
int count ( struct Node * P)
{
    if (P == NULL)
        return 0;
    return ( count(P -> lchild) + count(P -> rchild) );
}
```