Trees

Trees are collection of nodes and edges

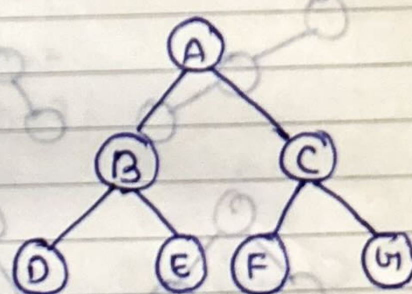
If there are 'n' nodes then there are (n-1) edges
where one of the node is Root node and the rest of the nodes are divided into disjoint subset and each subset is a tree or subtree

Terminology

1. Root
2. parent
3. Child
4. Siblings
5. Descendents
6. Ancestors
7. Degree of NODE
8. Internal node (Non-Leaf node) / External node (Leaf node) 0°
9. Levels
10. Height
11. Forest.

Binary Tree

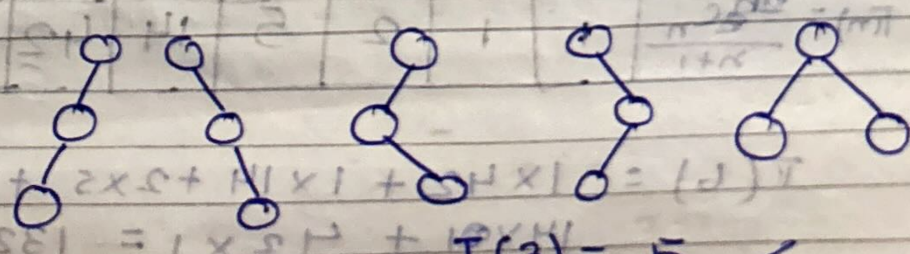
A tree having degree - '2'.
Children can be {0, 1, 2}



Number of Binary Tree.

1. Unlabelled Nodes → O O O
2. Labelled Nodes → A B C

If n = 3 then how many BT can be generate?



T(3) = 5

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

T(n=3) Max height = $4 = 2^2$

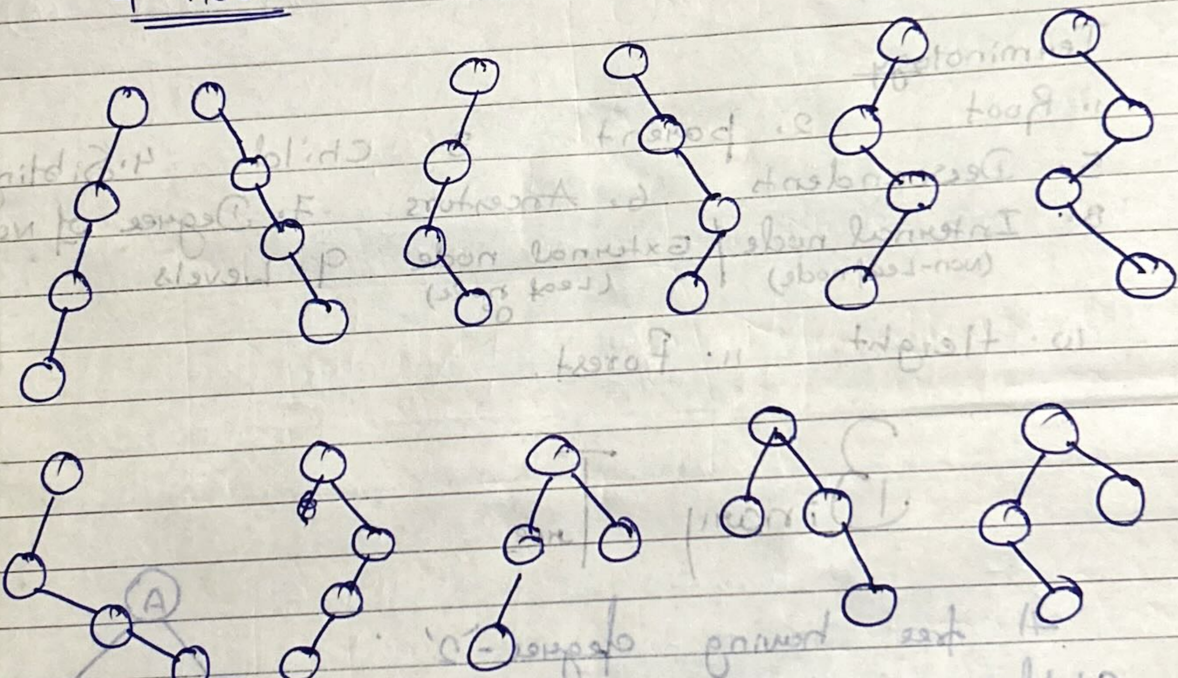
②

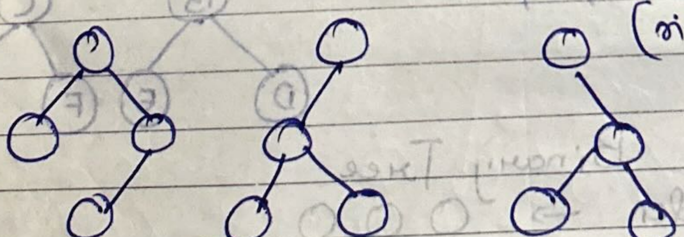
$$T(4) = \frac{{}^{2n}C_n}{n+1} = \frac{2 \times {}^{4}C_4}{5} = \frac{{}^{8}C_4}{5}$$

$$= \frac{8 \times 7 \times 6 \times 5}{4 \times 3 \times 2 \times 1} \Big/ 5 = 14$$

$$T(5) = \frac{{}^{10}C_5}{6} = 3 \times 2 \times 7 = 42$$

4 node





$(n=4)$ Max height $= 8 = 2^3$

$(n=5)$ Max height $= 16 = 2^4$

$\vdots$

(2^{n-1})

n	0	1	2	3	4	5	6
$T(n) = \frac{{}^{2n}C_n}{n+1}$	1	1	2	5	14	42	132

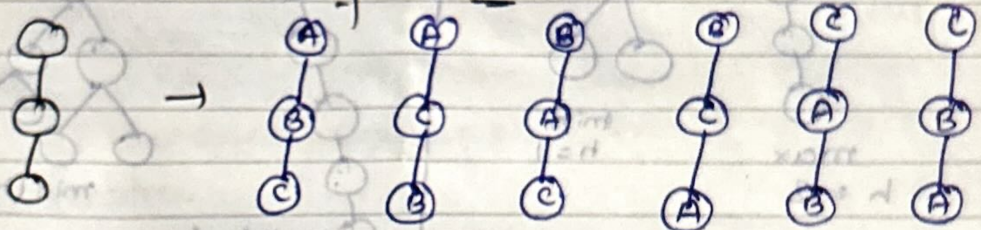
$$T(6) = 1 \times 42 + 1 \times 14 + 2 \times 5 + 5 \times 2 + 14 \times 1 + 42 \times 1 = 132$$

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

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

For Labelled Node - No. of B.T

eg.

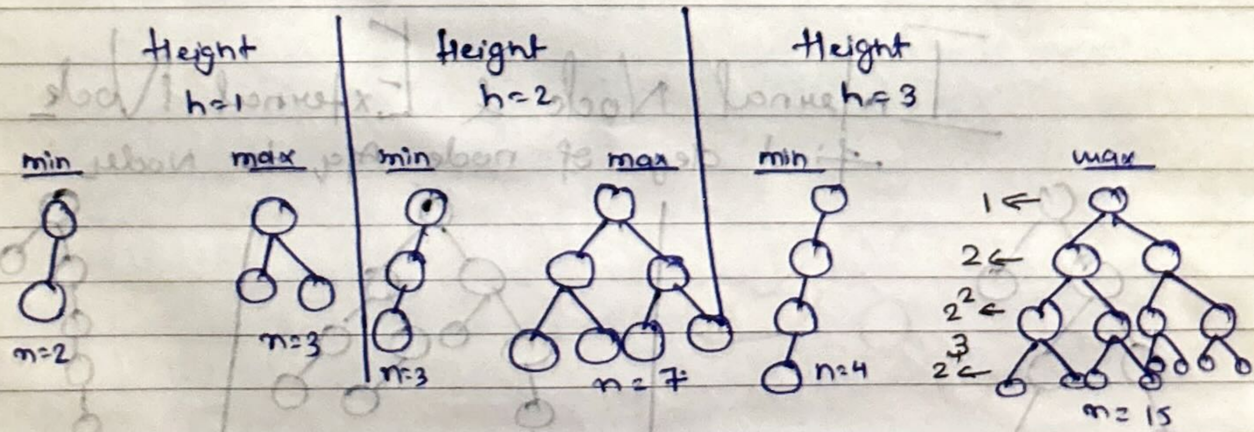


↳ $6 = 3!$

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

Height Vs Nodes

⊙ How many no. of nodes are possible ?



Min node (n) = $h+1$

$$G.P = a + ar + ar^2 + ar^3 + \dots + ar^k$$

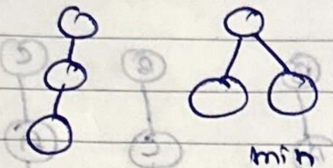
$$= \frac{a(r^{k+1}-1)}{r-1}$$

$$= \frac{1(2^{h+1}-1)}{2-1}$$

$$\boxed{\text{Max node} = 2^{h+1}-1}$$ Ans

If no. of nodes are given find what is the min^m height and Max height ?

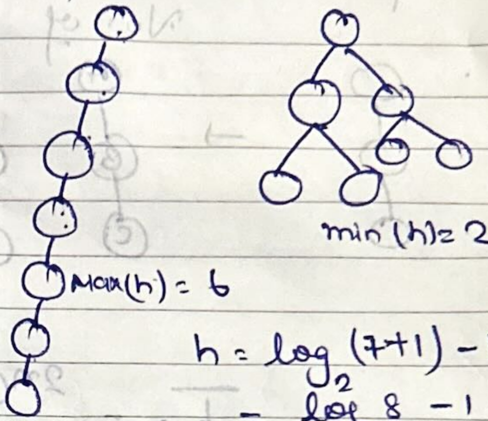
→ Node (n) = 3



max h = 2

min h = 1

(n) = 7



Max(h) = 6

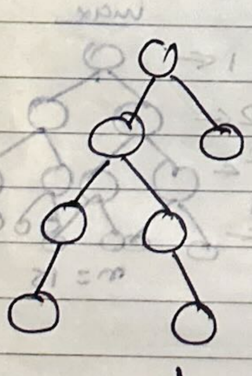
min (h) = 2

$h = \log_2(7+1) - 1$
$= \log_2 8 - 1$
$= 3 - 1 = 2$

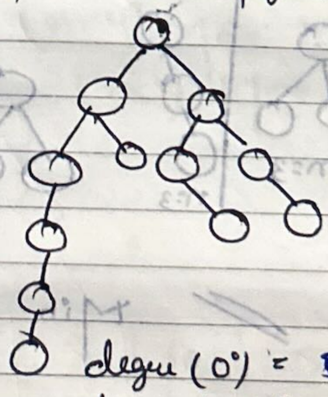
If height is given	If Nodes are given	Range
Min Node $n = h+1$	Min Height $= \log_2(n+1) - 1$	$O(\log n)$
Max Node $n = 2^{h+1} - 1$	Max Height $= N - 1$	$O(n)$

Internal Node & External Node

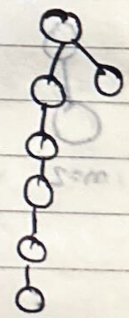
find degree of node Acc to Nodes



degree (0°) = 3
degree (1°) = 2
degree (2°) = 2



degree (0°) = ~~3~~ 4
degree (1°) = 5
degree (2°) = 3



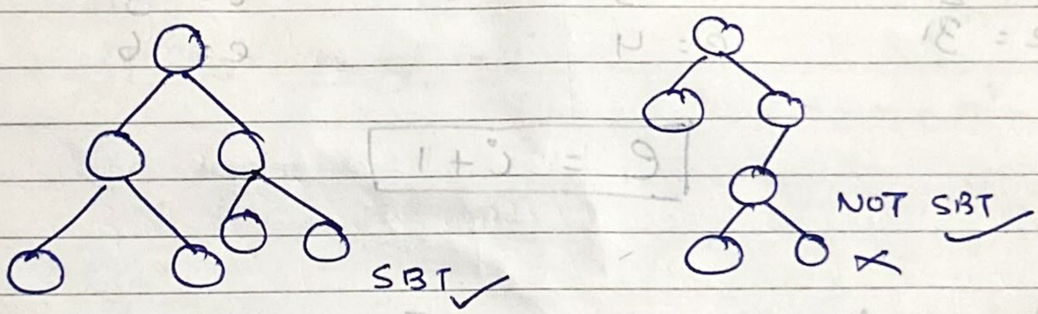
degree (0°) = 2
degree (1°) = 4
degree (2°) = 1

$$\boxed{\deg(0) = \deg(2) + 1}$$

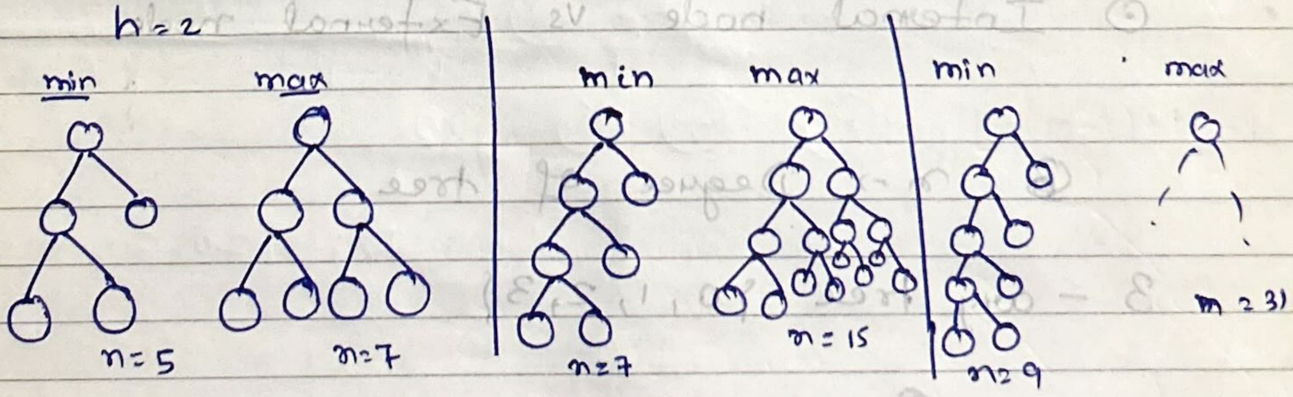
Strict Binary Tree

A general Binary Tree can have {0, 1, 2} children, but S.B.T It should not and any Node with degree '1'.

Only had {0, 2}



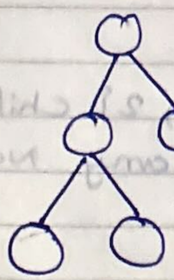
Height vs Node of S.B.T



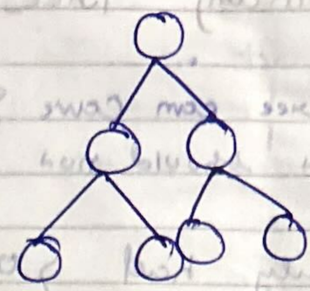
If h is given:
- Max^m no. of node are same as Binary tree formula
- Max^m no. of node $(n) = 2^{h+1} - 1$
- Min^m node $(n) = 2 \times h + 1$

If nodes are given:
- Max^m ~~no. of~~ height $(h) = \frac{n-1}{2}$
- Min^m height $(h) = \log_2(n+1) - 1$

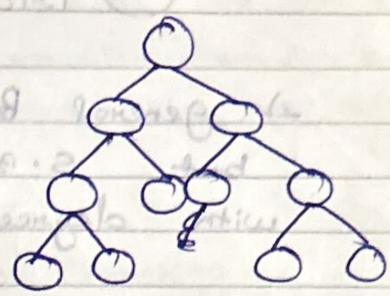
Internal node vs External Node



$i = 2$
$e = 3$



$i = 3$
$e = 4$



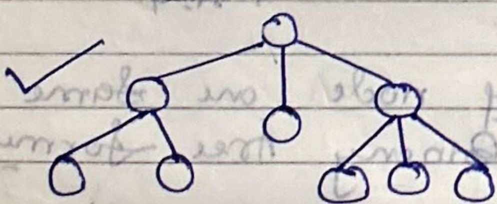
$i = 5$
$e = 6$

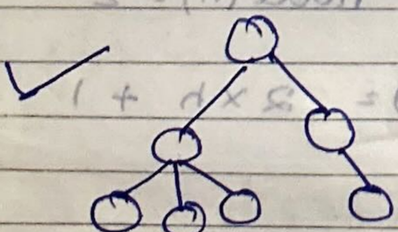
$$\boxed{e = i + 1}$$

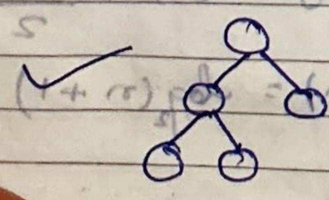
- What are n-ary Tree
- Strict n-ary tree
- Height vs Node
- Internal node vs External node

$n \to$ Degree of tree

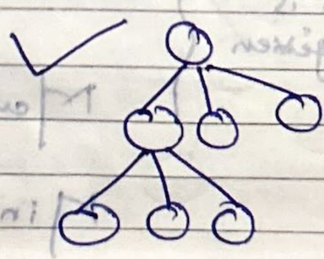
3 - ary tree $\{0, 1, 2, 3\}$

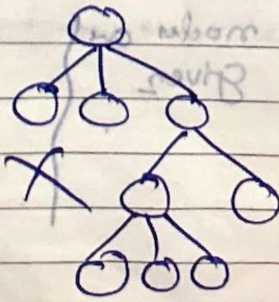




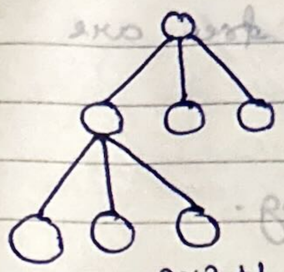


Strict 3-ary tree
$\{0, 3\}$

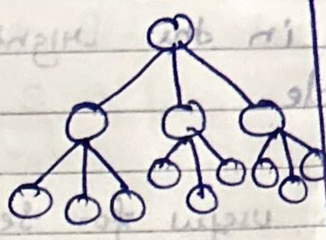


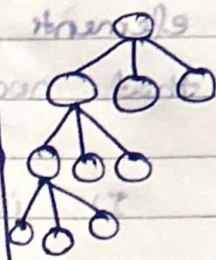


Strict 3-ary Tree

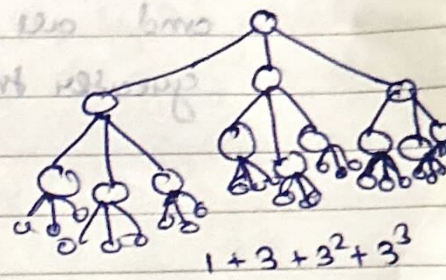
min 

$n = 2\times3+1$

max 

min 

$n = 3\times3+1$

max 

$1+3+3^2+3^3$

If Height is given

$= 1 + m + m^2 + m^3 + \dots + m^k$

Min nodes $n = 3\times h+1$

$n = mh+1$

$= a\frac{(r^{k+1}-1)}{r-1}$

Max node $= \frac{(m^{h+1}-1)}{m-1}$

If 'n' nodes are given

Min Height $h = \log_m[n(m-1)+1]-1$

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

No. of Internal / External Nodes

Acq to above pic.

$i = 2$	$i = 4$	$i = 3$	$i = 13$
$e = 5$	$e = 9$	$e = 7$	$e = 27$

Check pattern

$2\times2+1=5$ $\quad 2\times4+1=9$ $\quad 2\times3+1=7$ $\quad 2\times13+1=27$

$$\boxed{e = 2\times i+1}$$

$$\boxed{e = (m-1)\,i+1}$$

m → no. of ary tree

8

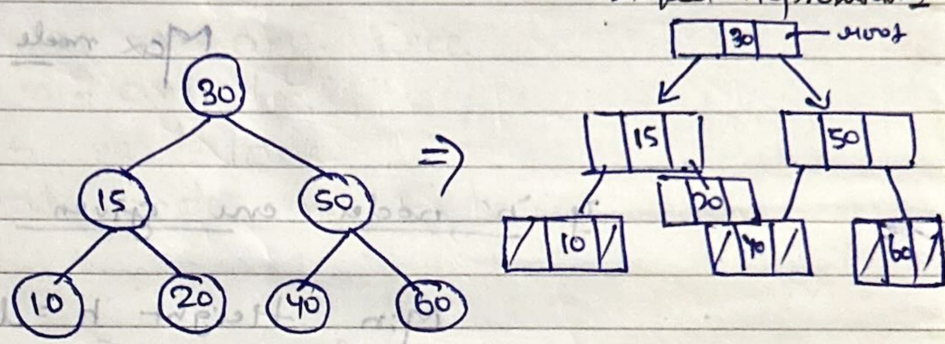
Binary Search Tree

Binary tree in which for every node all the nodes in left of tree are smaller than the nodes and all the elements in the rights of tree are greater than that node.

It is useful for searching.

properties

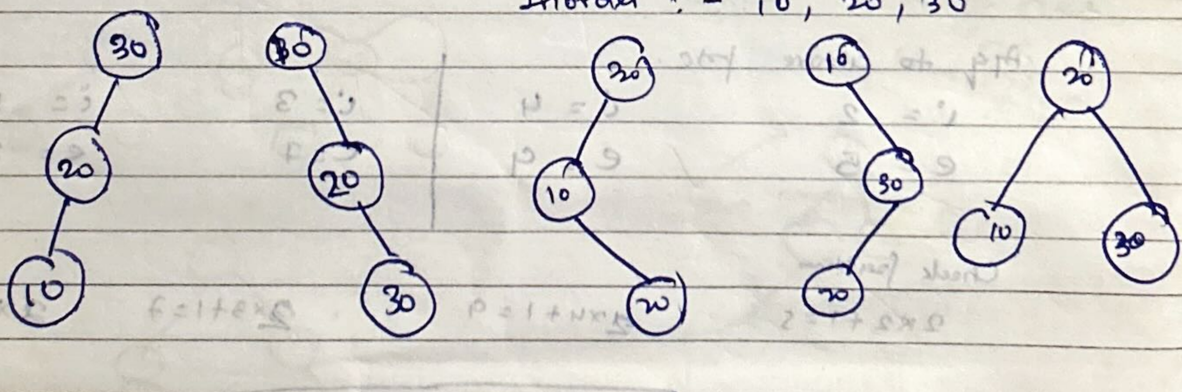
- No Duplicates
- In order gives Sorted Order
- No. of BST for 'n' nodes



Inorder :- 10, 15, 20, 30, 40, 50, 60

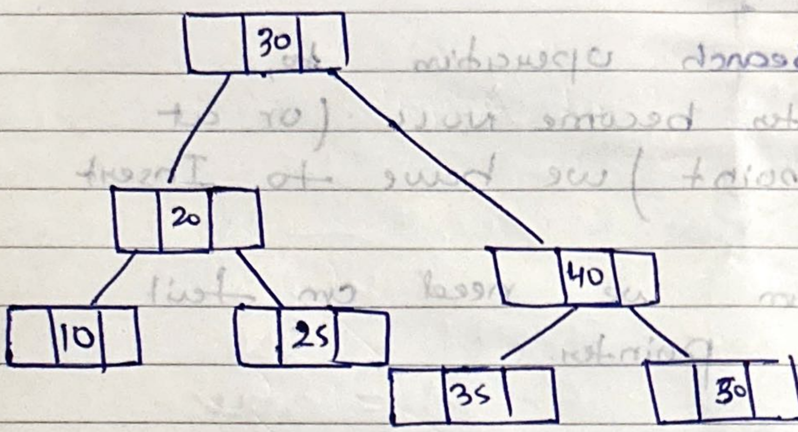
$n = 3$ 10 20 30

Inorder :- 10, 20, 30



BST is represented using Linked representation

Searching In BST.



~~void insert (Node *t, int key)~~
~~{~~
~~Node *r = NULL;~~
~~while (t != NULL)~~
~~{~~
~~r = t;~~
~~if (key = = t→data)~~

Search (key = 25)

Take a pointer on the root of a node then check for the data and compare with key if it is smaller (key) then move to left child then repeate same thing check and compare if it is greater then that then move to Right child . . . found

Maxm comparing of a data depend on height of tree $O(h)$ time

$$\log n \le h \le n$$

If key is NOT found (NOT present)

when our pointer become NULL mean's NOT FOUND.

Insertion In BST

First perform Search operation so, when our pointer become NULL (or at terminating point) we have to Insert

for Insertion we need an tail pointer.

```
void Insert (NODE *t, int key)
{
    NODE * r = NULL;
    NODE * p ;
    while ( t! = NULL)
    {
        r = t;
        if (key = = t -> data)
            return ;
        else
            if (key < t -> data)
            {
                t = t -> lchild ;
            }
            else
            {
                t = t -> rchild;
            }
    }
    p = malloc (— );
    p -> data = key ;
    p -> lchild = p -> rchild = NULL;
    if (p -> data < r -> data)
        r -> lchild = p;
    else
        r -> rchild = p;
}
```

O(logn)

Time as same Height of tree

Recursive Search on BST

```
Rsearch (Node *t, int key)
{
        if (t == NULL)
            return NULL;

        if (key == t -> data)
            return t;

        else if (key < t -> data)
            return Rsearch (t -> lchild, key);

        else
            return Rsearch (t -> rchild, key);
}
```

Iterative function Search on BST

```
Node * search (Node *t, int key)
{
        while (t != NULL)
        {
            if (key == t -> data)
                return t;        (FOUND)
            else if (key < t -> data)
                t = t -> lchild;
            else
                t = t -> rchild;
        }
        return NULL;   (NOT FOUND)
}
```

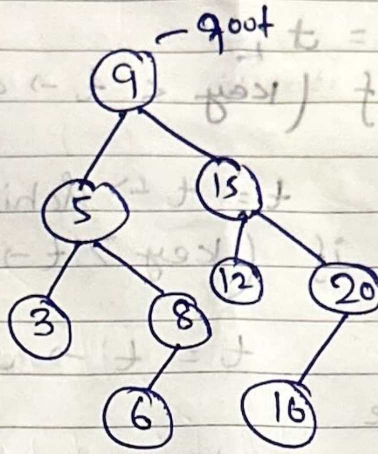
(2)

Recursive Insertion in B.ST.

```
Node * insert (NODE *p , int key)
{
    Node *t;

    if (p == NULL)
    {
        t= malloc ( — );
        t -> data = key;
        t -> lchild = NULL;
        t -> Rchild = NULL;
        return t;
    }
    if ( key < p -> data)
    {
        p -> lchild = insert
                (p->lchild, key);
    }
    else
    if ( key > p -> data) {
        p -> Rchild = insert
                (p-> Rchild, key);
    }
    return p;
}

int main ()
{
    NODE * root = NULL;
    root = insert (root, 30);
           insert (root ,20);
           insert (root ,25);
```

Create BST.

Keys :- 9, 15, 5, 20, 16, 8, 12, 3, 6.



Insert → n element
for each element we have to search that take log n
so,
$= (n \times \log n)$
$= O(n \log n)$

BST in C.

```
# Include <stdio.h>

Struct NODE
{
    Struct NODE * lchild;
    int data;
    Struct NODE * Rchild;
} *Root = NULL;


Void Insert (int key)
{
        struct NODE *t = Root;
        Struct NODE *r, *p;

        if (Root == NULL)
        {
            p = malloc (size of (struct node));
            p -> data = key;
            p -> lchild = p -> Rchild = NULL;
        } root = p;
          return;
```

(14)

```
while (t != NULL)
{
        r = t;
        if (key < t -> data)
                t = t -> lchild;
        else if (key > t -> data)
                t = t -> rchild;
        else
                return;
}

p = (struct NODE *) malloc (sizeof(sn));
p -> data = key;
p -> lchild = p -> Rchild = NULL;

if (key < r -> data)
{
        r -> lchild = p;
}
else
{
        r -> Rchild = p;
}
}


Void Inorder (struct NODE *p)
{
        if (p)
        {
                Inorder (p -> lchild);
                printf ("%d ", p -> data);
                Inorder (p -> Rchild);
        }
}
```

```
int main ( )
{
    Insert (10);
    Insert (5);
    Insert (20);
    Insert (8);
    Insert (30);

    Inorder (root);

    printf ("\n");

    Struct NODE * temp;
~~Struct NODE~~
    temp = Search (20);
    if (temp != NULL)
    {
        printf ("Element %d is
        found, temp -> data);
    else
    {
        printf ("NOT FOUND\n");
    }
    return 0;
}
```

Done

```
Struct NODE * Search (int key)
{
    Struct NODE *t = Root;
    while (t != NULL)
    {
        if (key == t -> data)
            return t;
        else if (key < t -> data)
            t = t -> lchild;
        else
            t = t -> Rchild;
    }
    return NULL;
}
```

Deletion in B.S.T.

Search for the key If found then, delete it.

If you delete a NODE then, you have to make it's parent node as NULL;

If the node is getting deleted and it's having a one child then the child will take it's place.

If you want del root of tree then you have to find it's Inorder predecessor or Inorder Sucessor and replace any of them with root...

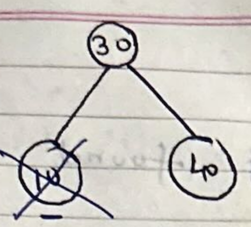
Find Inorder predecessor
Go to left's of tree and then go to at most right of it's.

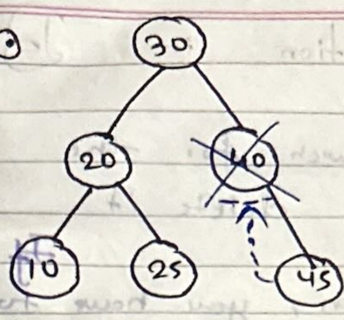
Find Inorder Successor
Go to Right's then, go to left at the last node.

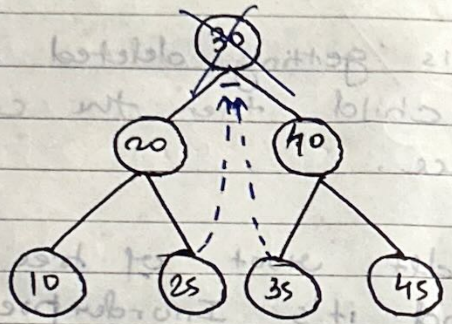
We were taking Inorder predecessor/sucessor bcz we wanted to avoid multiple modification in the tree
but some time if Inorder sucessor/predecessor is not a leaf node then you may have to make multiple modification

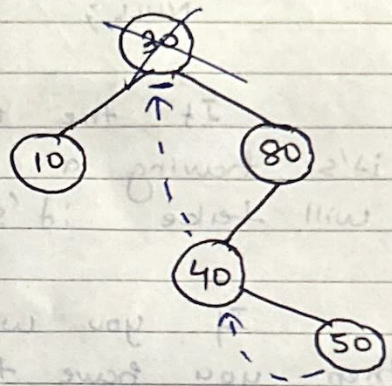
Time - O(logn)
No. of modification - O(logn)

(18)

- 

- 

- 

- 

Recursive function of B.ST. Insertion

Add In previous program of Insertion in BST

```
struct NODE * RInsert ( struct NODE *p,
{   struct NODE *t = NULL;    int key)
    if (p == NULL)
    {
        t = (struct NODE *) malloc
        (sizeof (struct NODE));

        t -> data = key;
        t -> lchild = t -> rchild = NULL;
        return t;
    }
    if (key < p -> data)
        p -> lchild = RInsert (
                     p -> lchild, key);
```

```
else if (key > p -> data)
{
    p -> rchild = RInsert (p -> Rchild, key);
}
    return p;
}

int main ()
{

    Root = RInsert (root, 10);
    RInsert (root, 30)
    RInsert (root, 15)
    RInsert (root, 20)
    RInsert (root, 8)

}
```

Delete function in BST. Recursively

```
struct NODE * Delete (struct NODE *p,
                      int key)
{          —— Next page.
    if (key < p -> data)
        p -> lchild = Delete (p -> lchild, key);
    else if (key > p -> data)
        p -> rchild = Delete (p -> Rchild, key);

    else
    {   if (Height (p -> lchild) >
            Height (p -> Rchild))
        {
            q = Inpre (p -> lchild);
```

```
            p -> data = q -> data;
            p -> lchild = Delete (p -> lchild, q -> data
        }
    else
    {
        q = Insucc (p -> Rchild);

            p -> data = q -> data;
        p -> rchild = Delete (p -> Rchild, q -> data);
    }
    return p;
  }
}
```

```
    Struct NODE* q;
    If (p == NULL)
            return NULL;
    if (p -> lchild == NULL && p -> rchild == NULL)
    {
        if (p == root)
                root = NULL;
            free (P);
                return NULL;
    }
```

```
int Height (struct NODE* p)
{
    int x, y;
    if (p == NULL)
            return 0;
    x = Height (p -> lchild);
    y = Height (p -> Rchild);

    return x > y ? x + 1 : y + 1;
}
```

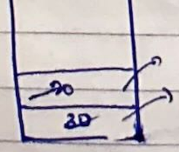
```
struct NODE *Inpre (struct NODE *p)
{
    while (p && p->Rchild != NULL)
        p = p->Rchild;
    return p;
}

struct NODE *Insucc (struct NODE *p)
{
    while (p && p->lchild != NULL)
        p = p->lchild;
    return p;
}

int main ()
{
    // Insert
    Delete (root, 20);
}
```

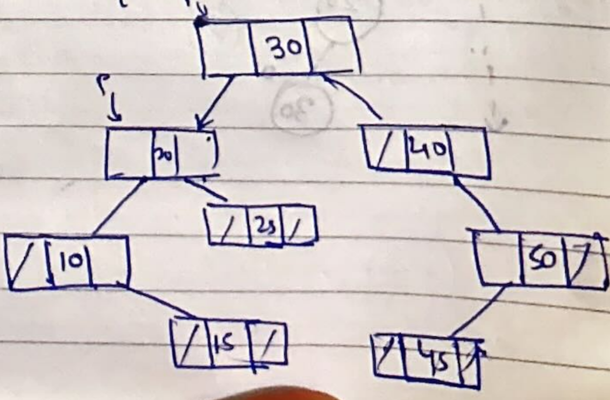
Generating BST from preorder.

1. preOrder + Inorder
2. postOrder + Inorder (or)

Pre

30	20	10	15	25	40	50	45
0	1	2	3	4	5	6	7





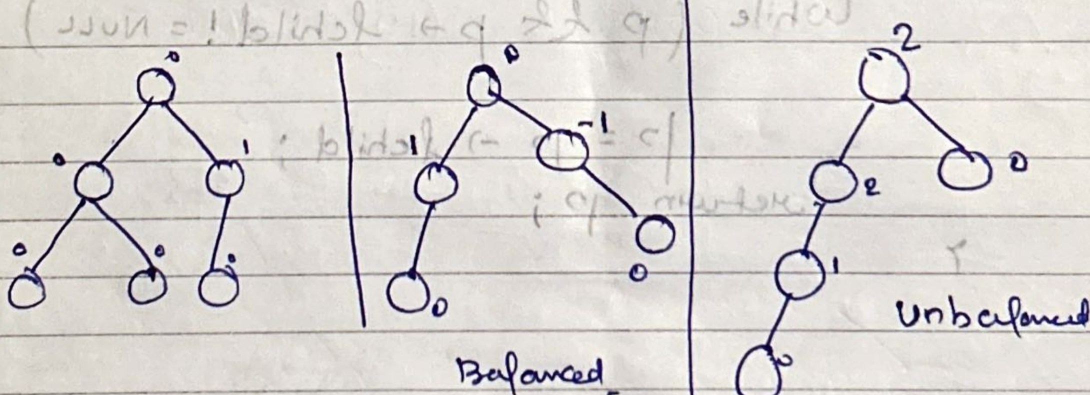
AVL Tree

Balance factor = Height of left subtree − Height of Right subtree.

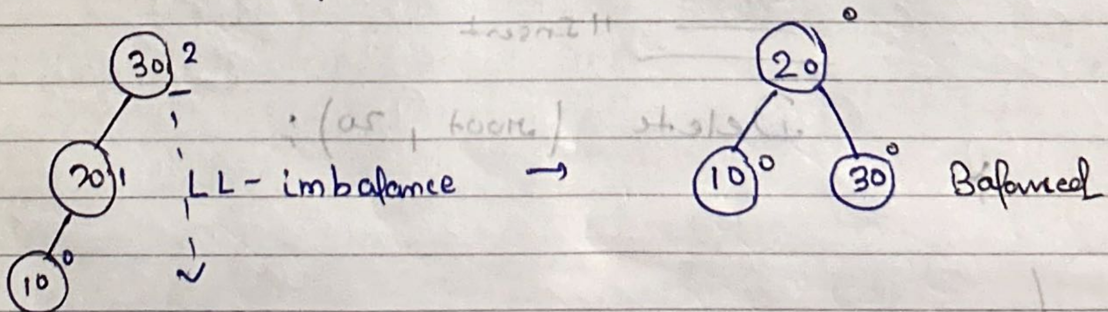
$BF = h_l - h_r = \{-1, 0, 1\}$

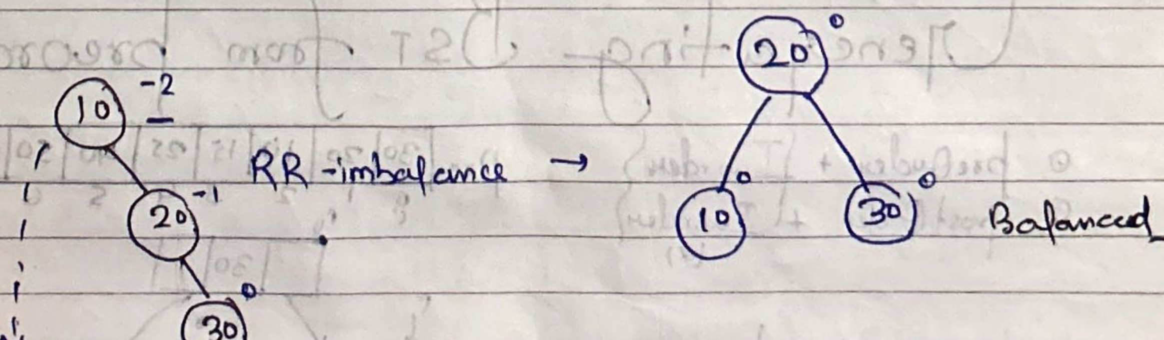
If $|BF| = |h_l - h_r| \leq 1$ balanced

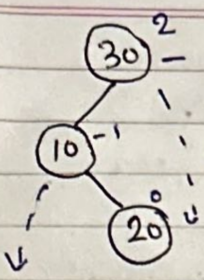
if $|BF| = |h_l - h_r| > 1$ imbalanced



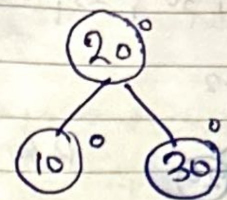
Rotation for Insertion

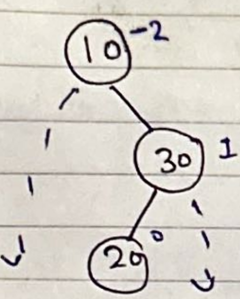




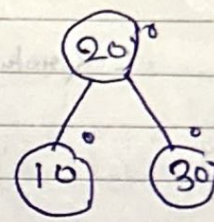


LR - imbalance →



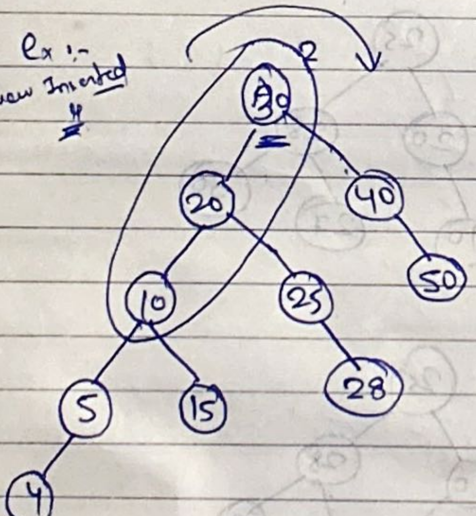


RL-imbalance

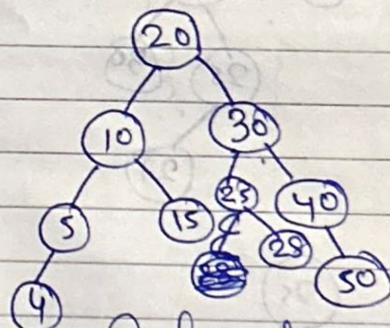


~~Formula of Rotation for Insertion~~

Ex :-
New Inserted
4

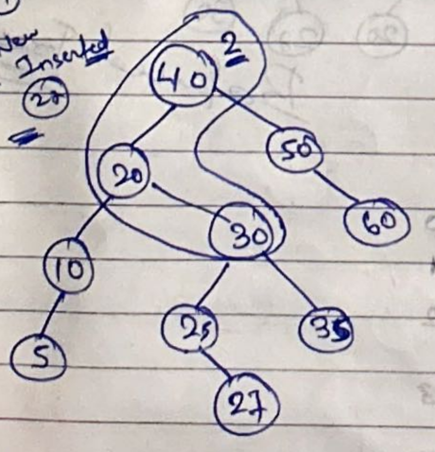


L-L Rotation

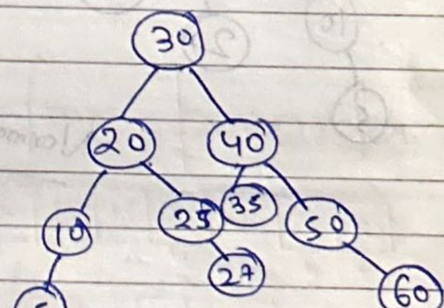


Balanced AVL Tree

Ex:- New Inserted 27



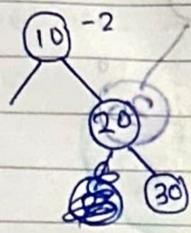
L-R rotation



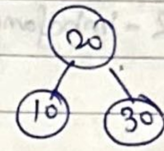
Balanced AVL Tree

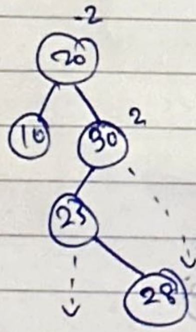
Create an AVL Tree

key - 10, 20, 30, 25, 28, 27, 5.

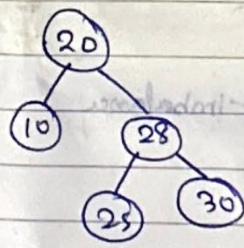


RR rotation.



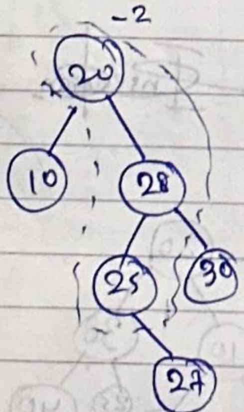


L-R rotation

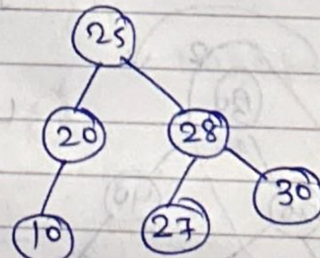


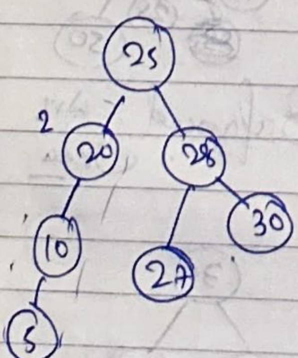
27

Insert



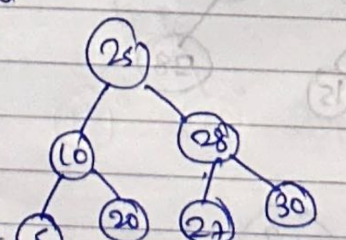
R-L rotation





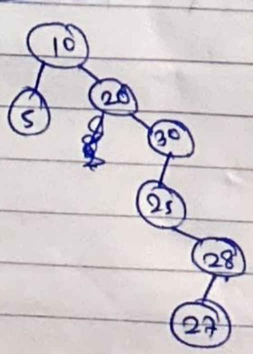
LL Rotation

Height
0
1
2



Balanced Tree

Normal BST ↑

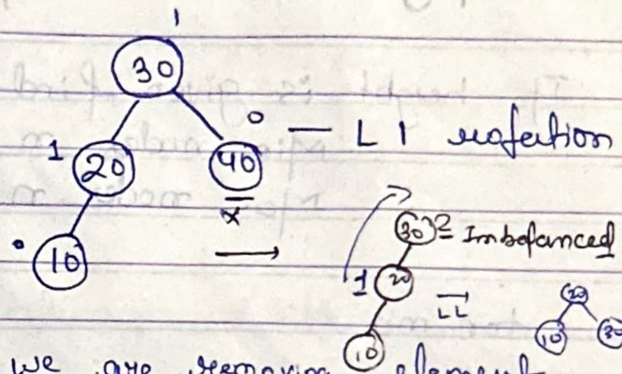


0
1
2
3
4
5
Height

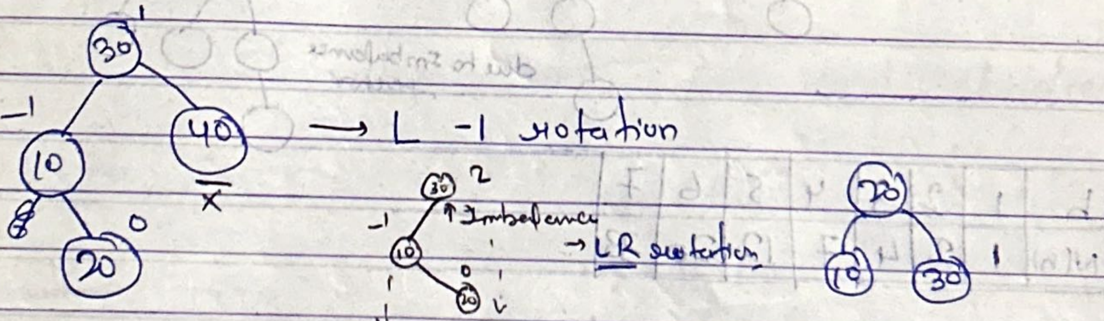
Deletion in AVL Tree

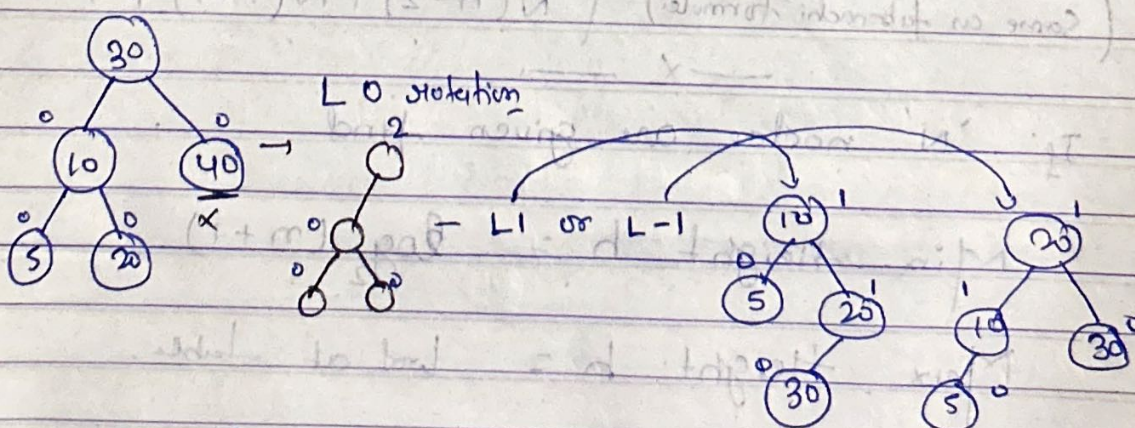
L Same as from BST

L 1 — Rotation
L -1 — Rotation
L 0 — Rotation



we are removing element from right hand side therefore it is getting Imbalanced from Left side.





- R1 R-1 R0 are same just the Mirror Img of these del occur from left side and Imbalance to Right Side.

Height Vs Nodes of AVL Tree

If height is given find.

Min nodes $n = N(h-2) + N(h-1) + 1$

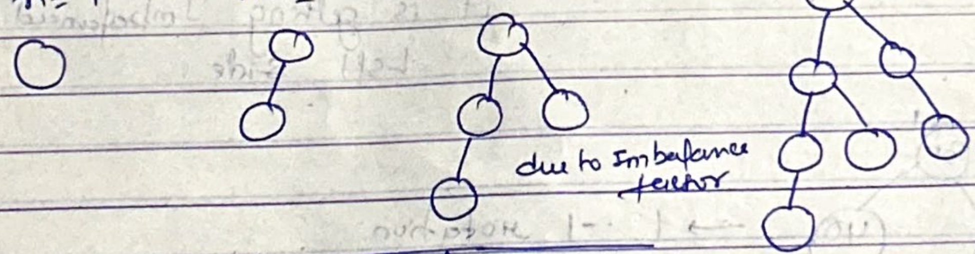
Max node $n = 2^{h} - 1$

bcz height is starting from 1

for min

ex:-

$h=1$, $n=1$ | $h=2$, $n=2$ | $h=3$, $n=4$ | $h=4$, $n=7$



h	1	2	3	4	5	6	7	
N(h)	1	2	4	7	12	20	33	. . .

$$N(h) = \begin{cases} 0 & 0 \\ N(h-2) + N(h-1) + 1 & \text{otherwise} \end{cases}$$

(Same as fibonachi formula)

— x —

If 'N' nodes are given find

Min Height $h = \log_2(n+1)$

Max Height $h =$ look at table.

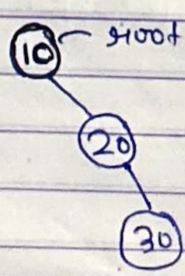
Red-Black Tree

- It is Height Balanced Binary Search Tree.
- Every Node is either Red or black
- Root of a Tree is always Black.
- Null are also taken as Black (leaf node)
- No. of blacks on paths from root to lef are same.
- No 2 consecutive Red, parent and children of Red are Black.
- New Inserted node is Red.
- The Height of a red black tree
  ↳ $\log n \leq h \leq 2 \log n$.

Create Red black tree

Key - 10, 20, 30, 50, 40, 60, 70, 80, 4, 8

↳ Insertion takeplace just like BST



root i.e. ~~red~~ black

Always Inserted node is red

Now his parent or child both are red

So, we had to adjust
↳ re colouring
↳ rotation.