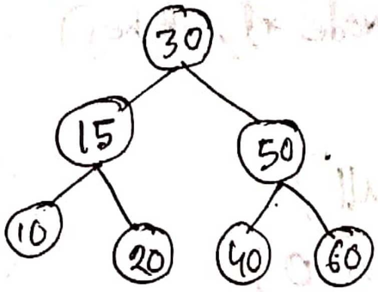
BST → Binary search tree

Lecture 1



Left side element is smaller than root

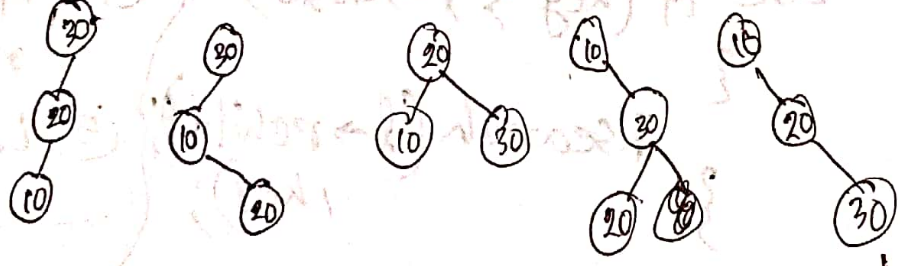
Right > root
left < root

this type Binary tree for searching

1. No Duplicates
2. Inorder gives sort order
3. No-of BST for "n" Nodes

40 chamair and search.

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



5 → shape are possiale

$$\frac{{}^{2n}C_n}{n+1}$$

Catalon numb- of tree generation

BST → represent using Link list most of the time

time taking in BST searching

Lecture 2

$\boxed{\log n \le h \le n}$ $O(h) = O(\log n)$

height

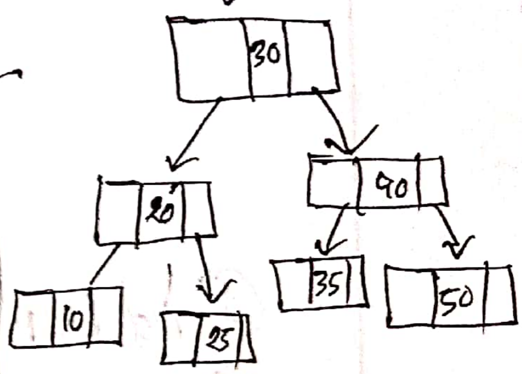
```
Node* Rsearch (Node *t, int key)
{
    if(t == Null)
        return 0;
    if (key == t->data)
    { return t;
    }
    else if (key < t->data)
    {
        Rsearch(t->lchild, key);
    }
    else if (key > t->data)
    {
        Rsearch (t->rchild, key);
    }
}
```

Recursion

Rsearch(t->lchild, key);

t

root



key = 25

1. t first root
2. t < key → lchild
   else → rchild
3. repeat 2
4. if t == key
   return t
5. Last null

⇒ iterative function using loop

```
Node * search(Node *t , int key)
{
    while(t != Null)
    {
        if(key == t->data)
        { return t;
        }
        else if (key < t->data)
        { t = t->lchild;
        }
        else if(key > t->data)
        {
            t = t->rchild;
        }
    }
    return 0;
}
```

$O(\log n)$

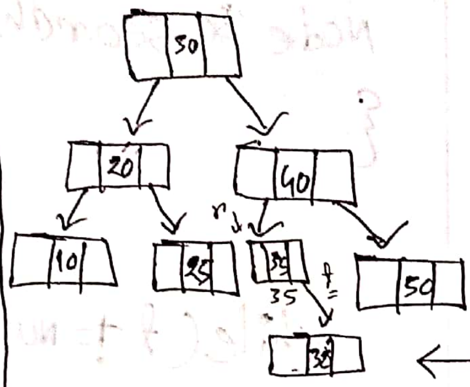
Inserting in Binary Search tree

Lecture. 3

```
void Insert(Node *t, int key)
{
    Node *r = Null, *P;
    while(t != Null)
    {
        r = t;   // follow (t)
        if(key == t->data)
        {
            return;
        }
        else if(key < t->data)
        {  t = t->lchild;
        }
        else if(key > t->data)
        {
            t = t->rchild;
        }
    }
    P = New Node;   P->data = key;
                    P->lchild = P->rchild = Null;
    if(key < r->data)
    {   r->lchild = P;
    }
    else  r->rchild = P;
}
```

Creating New Node

Compair for left or Right New Node Insert

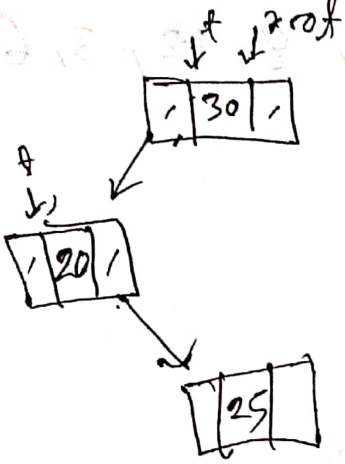


insert value key = 38

1. searching and insert the key
2. using teail pointer (r) follow the (t)

Recursive Insert

Lecture 4




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

    if(p == Null)
    {   t = new Node;
        t->data = key;
        t->lchild = 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;
}
```

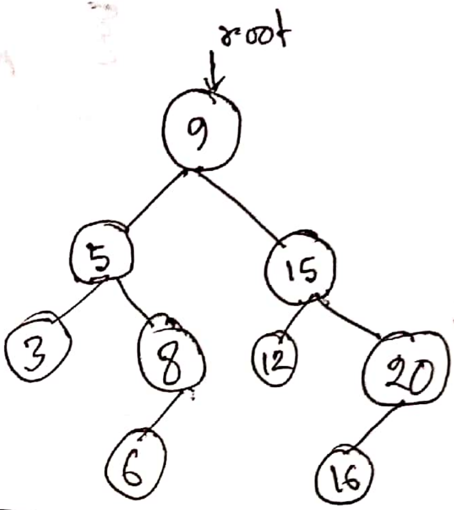
When return that time Link up

```
int main()
{  Node *root = Null;
      root = insert(root, 30);
           insert(root, 20);
           insert(root, 25);
}
```

Lecture 5

Creating Binary search tree. and Delete

key: 9, 15, 5, 20, 16, 8, 12, 3, 6



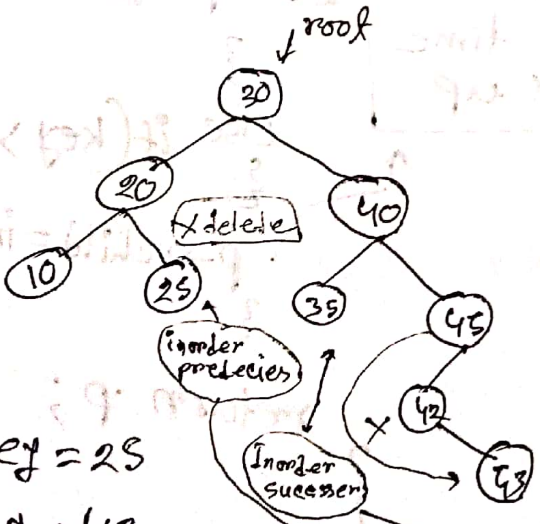
Lecture 6 ⇒ code in pdf

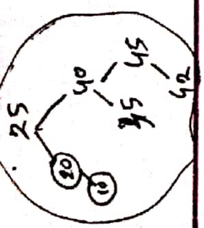
Lecture 7

Deleting Binary search tree

key = 25

1. search key
2. Deleting the Node. If





key = 25

key = 43

key = 42 → here is a problem sub tree are there we have to modife the link

key = 30

→ Find out Inorder predecies // who are before 30 root

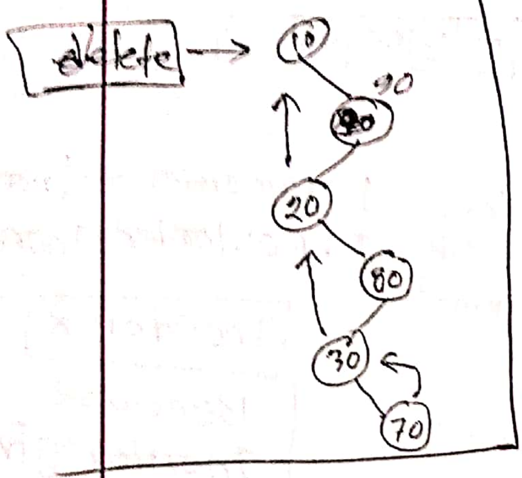
before and after root

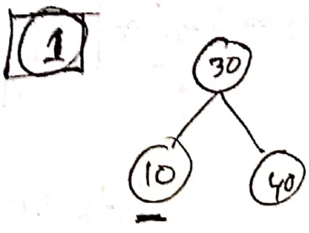
like

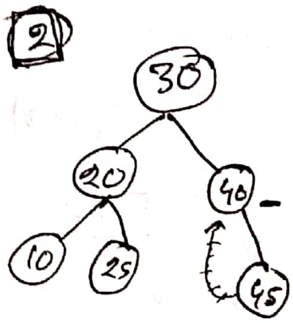
In: 10, 20, 25, 30, 35, 40 - - -

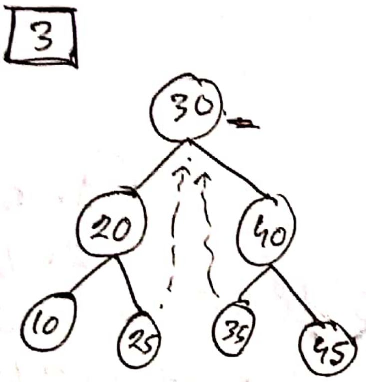
Number of modification $O(\log n)$

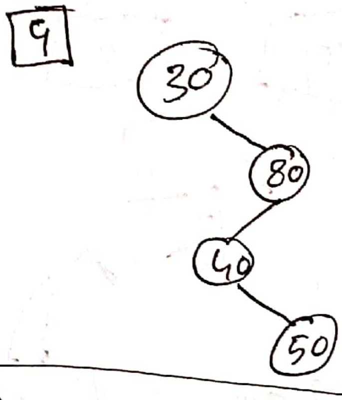
deleting $O(\log(n))$











Between (25, 35) one of them, predicies or succsur

```
Node *Delete(Node *p, int key)
{
    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
        {
            if (
            with rchild
        }
```

~~Local code~~ code pd

Lecture 9 : Generating BST from Preorder

pre

30	20	10	15	25	40	50	45	

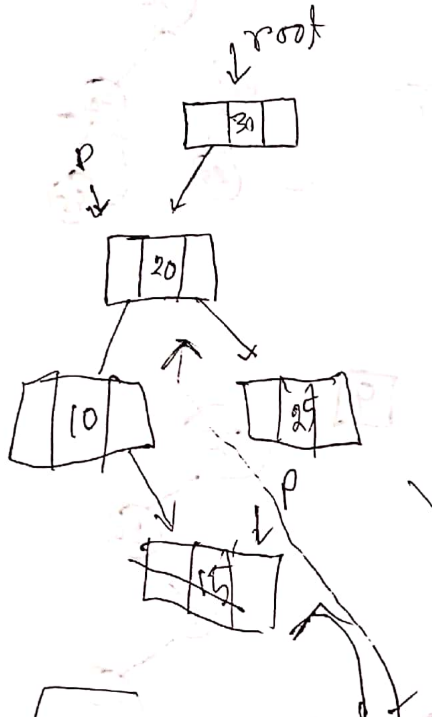
i ---- ↑

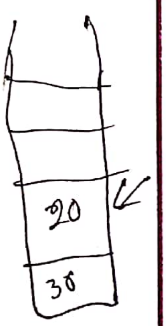
ith is smaller than root add left and Right

1. preorder + Inorder
2. postorder + Inorder

Inorder X

Because Inorder gives sorted order BST





stack

25 time

when righ child and root not lying value pop in ~~the~~ stack

1. Need stack
2. left ~~element~~ ~~ad~~ push in stack
3. ith check for lef and Right
4. when left child then push addss in stack

if Right X push

simply think that creating tree by human being

what condition follow check root value and Left and Right this type follow and code

Lecture 10

Create code.

BST → preorder

```
void create(int pre[], int n)
{  stack stk;
   Node *t;
   int i=0;
   root = new Node;
   root->data = pre[i++];
   root->lchild = root->rchild = Null;
   p = root;
   while (i<n)
   {  if (pre[i] < p->data)
      {   t = New Node;
          t->data = pre[i++];
          t->lchild = t->rchild = Null;
          p->lchild = t;
          push(&stk, p);
          p = t;
      }
      else
      {  if (pre[i] > p->data && pre[i] < stacktop(stk)->data)
         {  t = New Node; t->data = pre[i++];
            t->lchild = t->rchild = null;
            p->rchild = t; p = t;
         }
         else{
            p = pop(&stk);
   }     }
```

if data is pre[] getter from root value each then it is Range[root, last value] then one case or another case

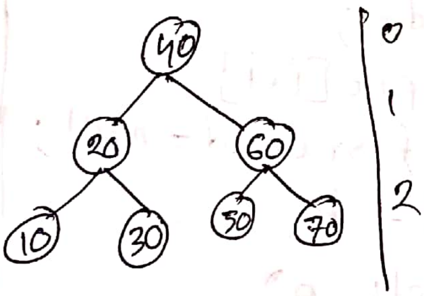
pop from stack

Lecture 11

Drawback of Binary search tree

* we can not control height of Binary search tree :
* we are expecting that height $O(\log n)$ But not
* we need another method to control of height BST.
* we can not control user Insertion :

key : 40, 20, 30, 60, 50, 10, 70

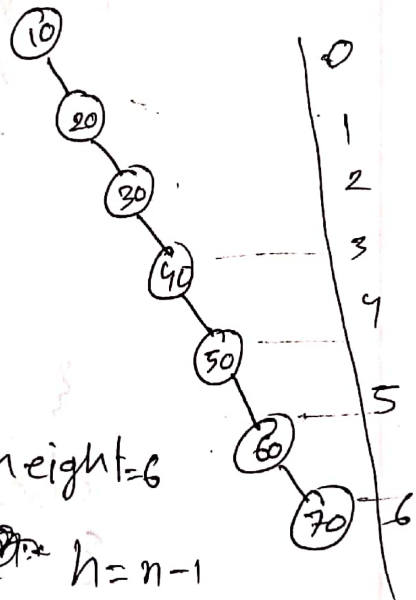


height = 2

$\log_2(n+1) - 1$

$\Rightarrow O(\log n)$

key : 10, 20, 30, 40, 50, 60, 70



height = 6

$h = n - 1$

$O(n)$

$\boxed{\text{We need AVL tree}}$

tree itself control height