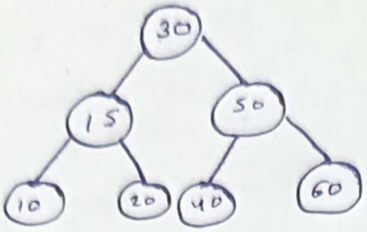
Binary Search Trees



→ In: 10, 15, 20, 30, 40, 50, 60

→ it's a Binary Tree in which for every node all the elements in its left side subtree are smaller than that node & all the elements in its right side subtree are greater than that node

→ This Binary Tree is useful for searching so, name is BST

How useful?

Suppose searching a key '40'

40 > 30 So Right side

40 < 50 so left side.

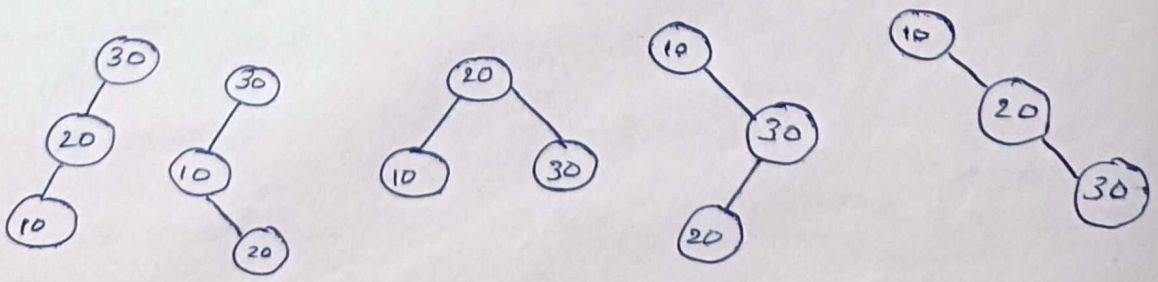
Properties

1. BST will not have duplicates
2. if we take inorder Traversal of BST then we get sorted of it
3. No. of BST for 'n' nodes.

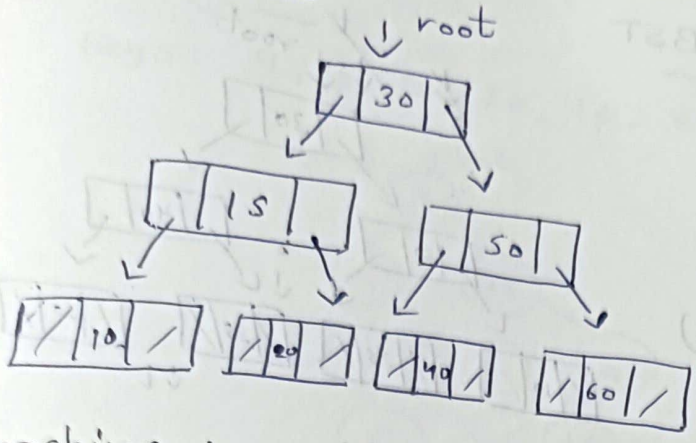
n = 3 (10) (20) (30)

Inorder :- 10, 20, 30

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



Linked Represn



Searching in a BST

The TimeTaken for searching depends on height of a Tree.

O(h)

$\log n \leq h \leq n$ (n nodes)

Best case O(1)
worst case O(n)
average case O(log n)

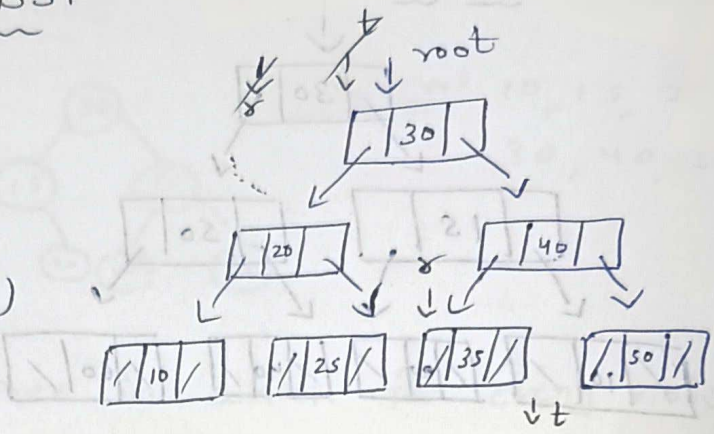
```
Node * Rsearch (Node *t, int key)
{
    if (t==NULL)
        return NULL;
    if (key==t->data)
        return t;
    else if (key<t->data)
        return Rsearch(t->lchild, key);
    else
        return Rsearch(t->rchild, key);
}
```

—> recursive

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

—> iterative.

Inserting in a BST




```
void insert(Node *t, int key)
{ Node *p;
  Node *r = NULL;
  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;
}
```

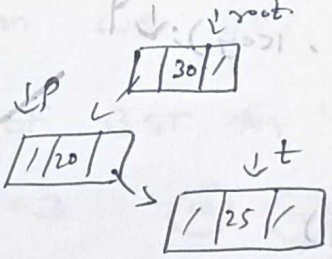
Key = 38

if this key already present Then we should not insert it.

-> we need a Tailing Pointer

↳ in, code for first node

Recursive insert in BST

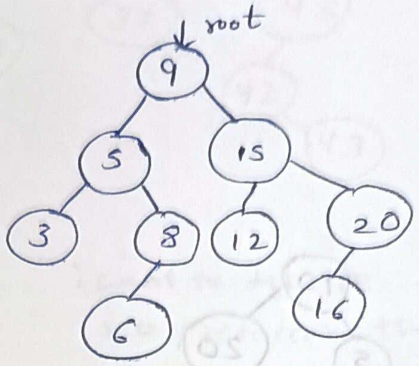



```
int main()
{
   Node *root = NULL;
   root = insert(root, 30);
          insert(root, 20);
          insert(root, 25);
}
```

```
Node * insert(Node *p, int key)
{ Node *t;
  if (p == NULL)
  { t = malloc(---);
    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;
}
```

Creating a BST

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



inserted n elements

search - log n (for where to insert).

O(n log n)

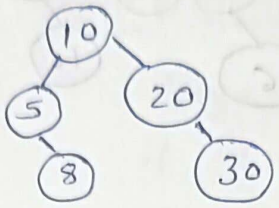
let's Code for BST

C code ✗ video = (iterative insert ✓ inorder traversal ✓ search ✓)
C++ ✓

code for iterative insert

```
#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 = (struct Node *) malloc(sizeof(struct Node));
    p->data = Key;
    p->lchild = p->rchild = NULL;
    root = p;
    return;
  }
  while (t != NULL)
  {
    r = t;
    if( Key < t-> data)
        t = t->lchild;
    else if (Key > t->data)
        t = t-> rchild;
    else if (Key == t->data)
        return;
  }
  p = (struct Node*) m  (  );
  p-> data = Key;
  p->lchild = p-> rchild = NULL;
```

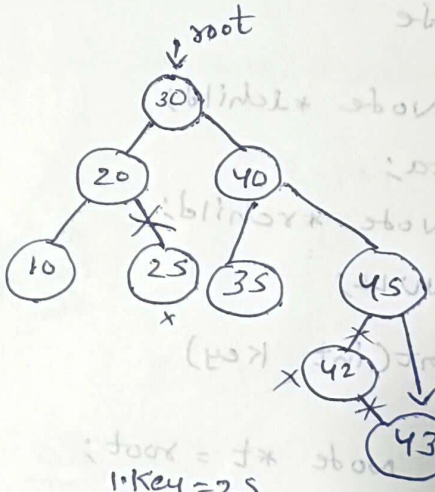
```
if (key < r->data)
    r->lchild = p;
else
    r->rchild = p;
}
int main()
{
  Insert(10);
  Insert(5);
  Insert(20);
  Insert(8);
  Insert(30);
```



——> we call inorder it gives sorted order we already know it

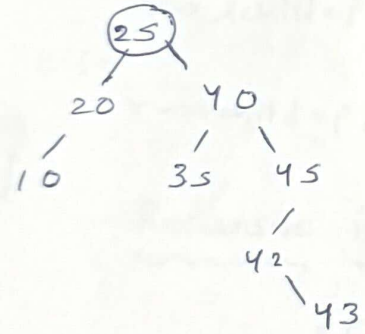
```
  return 0;
}
```

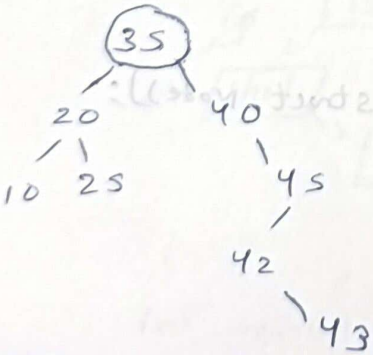
Deleting from BST



1. key = 25
2. key = 42

→ it goes to the place of deleted one.





** → How to find inorder predecessor (or) successor?

for predecessor, go to its left subtree, Then right, right ... so rightmost child

for successor leftmost child

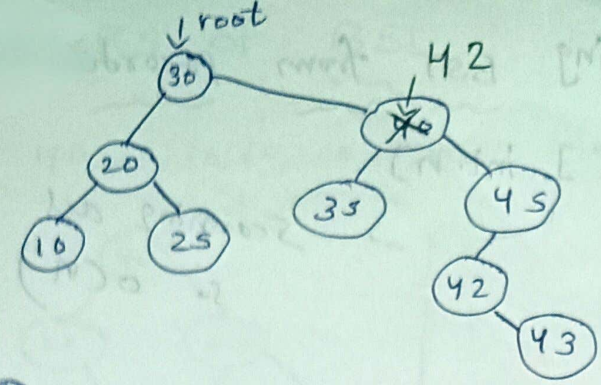
* if we remove 30 Then it place 20 Takes it Then who will Take 20's place 25 (or) 10 / 30's place 40 Then 40's 45 45's 42 Then 42's 43 so, so many changes have to do so →

for 30 find inorder predecessor (before 30) (or) successor.

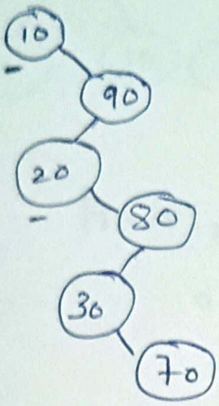
for 30

25 inorder prede

35 inorder Succ



if we delete 40
& i want to
replace it with
successor (42)?
Then 42's place
Takes up by (33 +10 43)



i want to delete '10'
no predeccor there
so successor ✓ (20)
20's successor 30
Takes 20's place Then 30 place
Takes up by 70.

(deln o(logn)
modfcn o(logn))

code video

Generating BST from Preorder

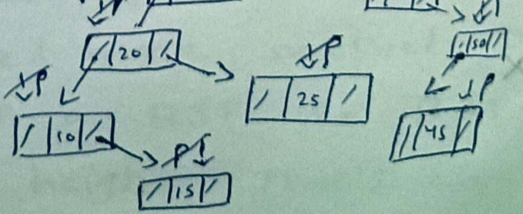
if we have pre+in (or) Post+in Then we can create (Gene) B T

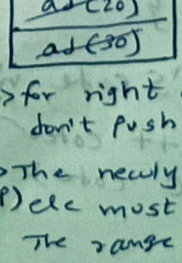
if (Pre (or) Post only given) in(sorted) so, no problem (inorder)

pre	30	20	10	15	25	40	50	45

* without inorder also we can generate BST

i) Scan from preorder





→ for right child don't push address

→ The newly coming (if>P) ele must be in The range of P & stack Top add(data). if not pop

→ if stack empty Then let it be ∞

Program for generating BST from preorder

```
Void createPre(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);
            }
        }
    }
```

Scanning all so O(n) TC

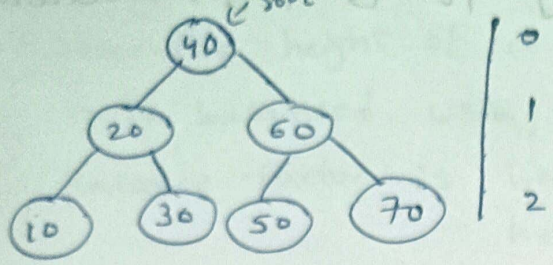
(Here one condn not written if the stack is empty it should be infinity.

—> i don't more.

(let's code video X.
C++ code✓
C code X)

Drawbacks of BST

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

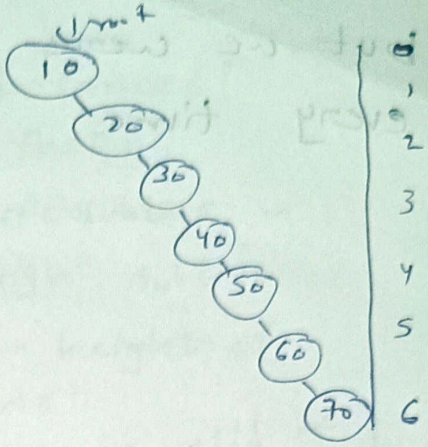


$h = 2$

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

$O(\log n)$

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



$h = 6$

$h = n - 1$

$O(n)$

-> The height of a Binary Search Tree can be as minimum as 'logn' as maximum as 'n'. it depends how u are inserting the keys, So there is no control over the height of a BST it all depends on order of insertion. Can u control order of insertion? NO, why? I am writing a progm for BST & That applicn, that progm, u are going to use it. Now how i can control u to give the values properly so that a min height is formed. u are using my applicn you'll be giving the keys as u have (or) as u are getting. Then my program will be generating the tree. So, we cannot control the order of insertn. So, we need some method to control the height of a BST so BST, itself should control its height. That's what we call them as AVL TREES.

AVL are 'h' balanced BST's.

Conclusion

'h' of a BST's not logn always.

but we were assuming it is logn always every time