DSA (SKM)

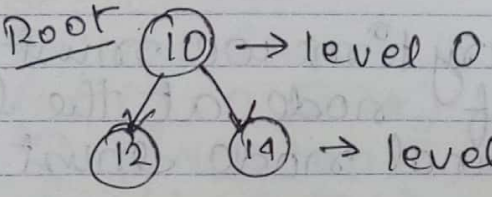
Tree

* non-linear DS.

linear	non linear
(1) sequential	(1) non-sequential.
(2) Only one path is followed.	(2) Multiple path is followed or possible.

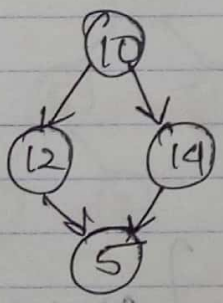
T(1) + T(2) + ---- + T(n) = Forest.

Tree has to be hierarchical.

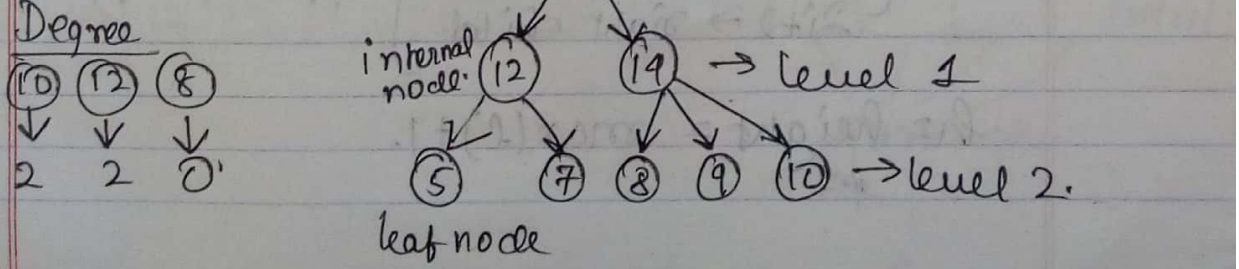


Tree → no loop allowed.

Tree is a collection of nodes, connecting with themselves, maintaining a hierarical form, & no loop



Not a tree, but a graph.



Internal node → degree ≥ 1.
Leaf node → degree = 0.

Siblings → whose parents are same.

Height of (node) 12 → Root of that node → 1.

Depth of 12 → 3 → (node to leaf).
Height of node → Root to that node.
Height of tree → (max level) + 1.

Binary tree

Total no. of nodes = $2^n - 1$ (Applicable only for complete binary tree)

FULL → Every time we must have the leaf node at the last level, & internal node must have degree 2.

Complete → Every time we must have the leaf node at the last level, & the leaf node should be filled from left to right.

BT using Array

index { i → Parent
2i+1 → left child
2i+2 → right child } Applicable iff i≥0.

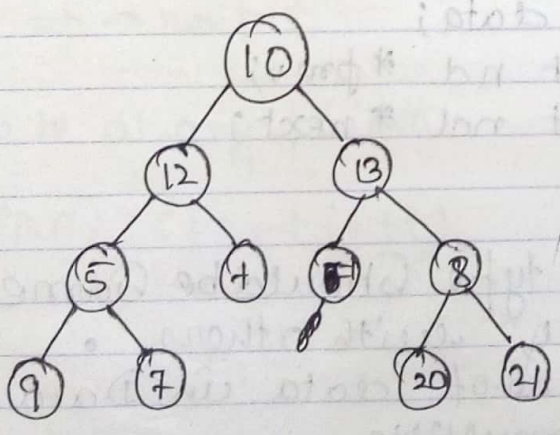
h = height = max(l) + 1.

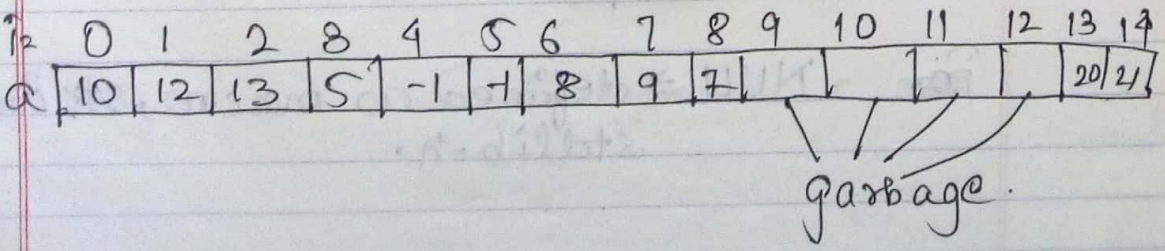
If no, child want to Insert, it should be -1.

per level we are inserting 2^k elements.

Algorithm:- Create Binary tree.

Step 1: Start.
Step 2: Read h as height of Binary Tree.
Step 3: Print "Enter Root data", set $i \leftarrow 0$.
Step 4: Read a[i] as Root data.
Step 5: if ($a[i] \neq -1$).
Step 6: Print "Enter the left Child of", a[i].
Step 7: Read a[2i+1] as left child.
Step 8: Print "Enter the Right Child of", a[i].
Step 9: Read a[2i+2] as Right Child.
Step 10: set $i \leftarrow i+1$.
Step 11: Repeat Step 5 to 11.
for ($j=0$ to 2^k).
Step 12: Repeat Step 5 to 12.
for ($k=0$ to h).
Step 13: Return a.
Step 14: Stop

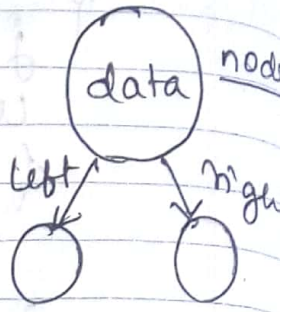




SKM

Binary Tree Construction using LinkedList

```
typedef struct Bnode
  {
    int data;
    struct Bnode *left;
    struct Bnode *right;
  } bnode;
bnode * CreateBinaryTree();
void main()
{
  bnode *root;
  root = NULL;
  root = CreateBinaryTree();
}

bnode * CreateBinaryTree()
{
  int x; Bnode *nw;
  printf("\n Enter node data");
  scanf("%d", &x);
  if(x == -1)
  return NULL;
  nw = (bnode *) malloc(sizeof(bnode));
  nw->data = x;
  printf("Enter the left child of %d", x);
  nw->left = CreateBinaryTree();
  printf("Enter the right child of %d", x);
  nw->right = CreateBinaryTree();
  return nw;
}
```



Page :
Date : 11/10/17

SKM

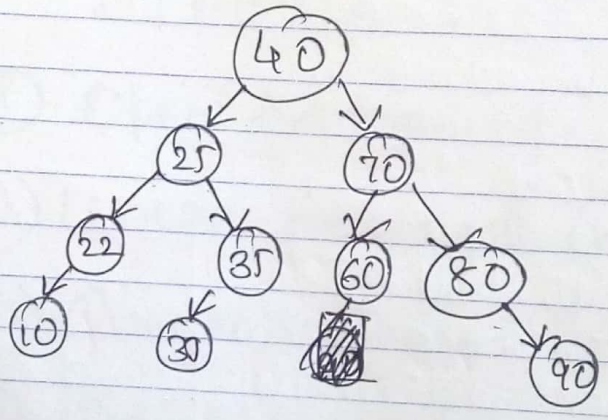
Topic :-

1. Create a BST.
2. Delete a node from BST.

Binary Search Tree :- (Order of log n).

(For Insertion) Rule :- (1) value of left subtree is less than the value of Parent node.
(2) Value of Right subtree is greater than the value of Parent node.

BST using 40, 25, 70, 22, 35, 60, 80, 90, 10, 30.

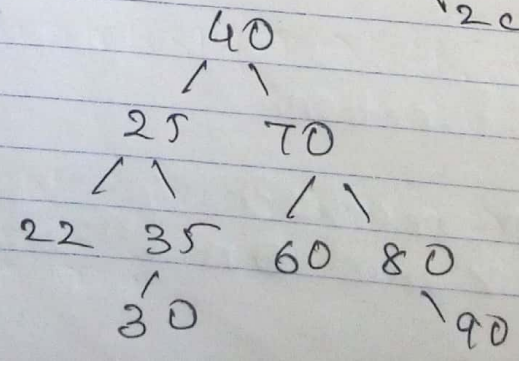


Main BST.

For Deletion :-

Rule :- (1) leaf node → Remove that.
(2) Non-leaf node ← 1 child
2 child.

Delete (10)

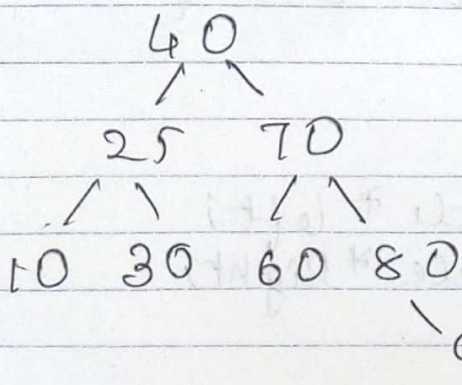


If delete 90, then simply remove 90, as like 10.

Rule → Non leaf → 1 child → Attach the child to the parent of the to be deleted node.

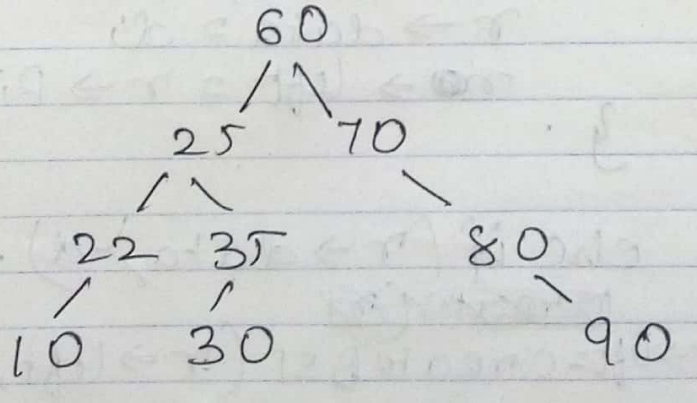
→ 2 child → Find the Inorder succesor to be deleted node. Replace with that.

Delete 22.



Inorder → 10 22 25 30 35 40 60 70 80 90.

Delete → 40.

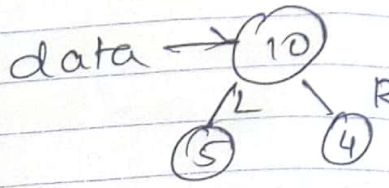


When removing parent node, then the node that will be parent is the smallest number in the right path.

Construct a BST :-

→ using recursive
→ using non-recursive.

Recursive :-



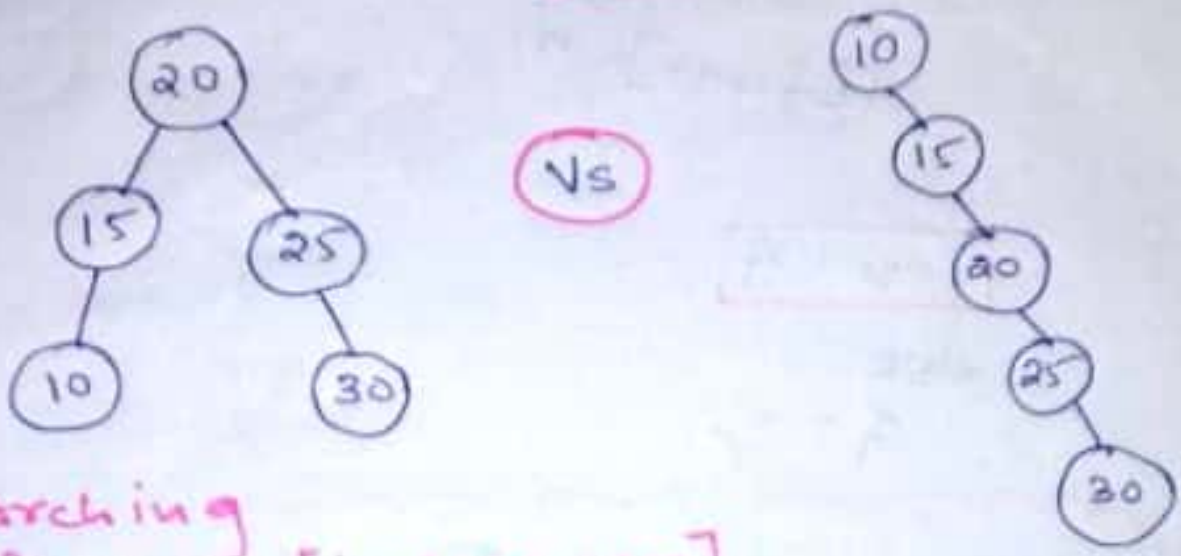

```
typedef struct Bnode{ *
{
  int data;
  struct Bnode * left;
  struct Bnode *Right;
} bnode;

bnode* CreateBST (bnode *r, int d).
{
  if (r == NULL).
  {
    r = (bnode *) malloc (sizeof (bnode));
    r -> data = d;
    r -> left = r -> Right = NULL;
  }.

  else if (r -> data > d).
r -> left = CreateBST (r -> left, d);
  else
  r -> Right = CreateBST (r -> Right, d);
  return r;
}.
```

```
void main()
{
    bnode *Root;
    Root = NULL;
    int n;
do{ scanf ("%d", &n);
  Root = CreateBST(Root, n).

    printf ("Do u want to insert?");
    ch = getch();

} while (ch != "Y" || ch != 'N')
```

Possible Question:

→ construct a height balance tree.

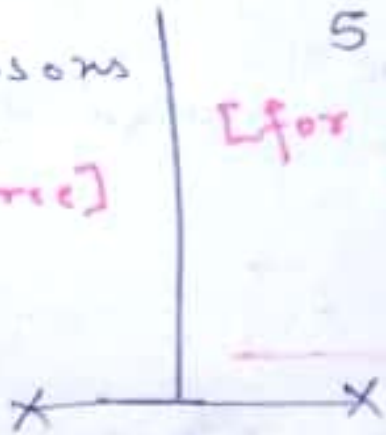
Height Balanced Tree



1) searching efficiency. [in some case]
2) less time required.

3 comparisons [for shorter height tree]

5 comparisons [for longer height tree.]



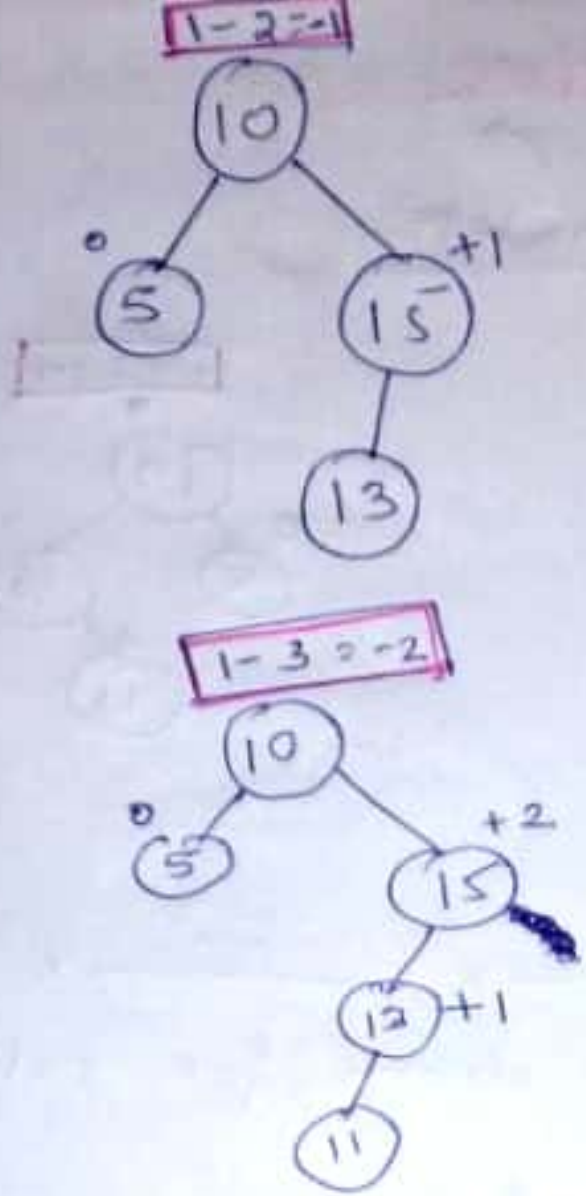
Using: Balance Fector.

Balance Factor of a node.

Height of left sub-tree

Height of right sub-tree.

permissible balance of AVL Tree = (+1, 0, -1) = left - right



Rotation.

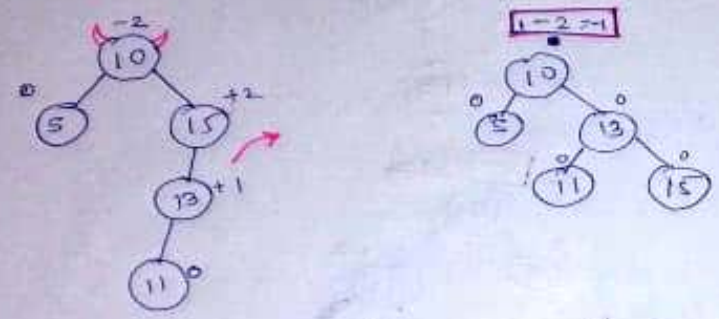
if not permissible balanced factor = then affected

Position of Rotation is the nearest effected node with respect to the last inserted node. due to which the tree has become unbalanced Tree.

Direction of the inserted node with respect to → the affected node.

Rotation.

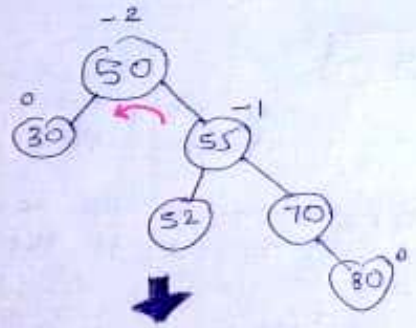
LL ↷
from left.



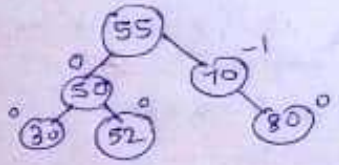
—— X —— X —— X ——

‖Select longest chain for calc‖

insert (85)

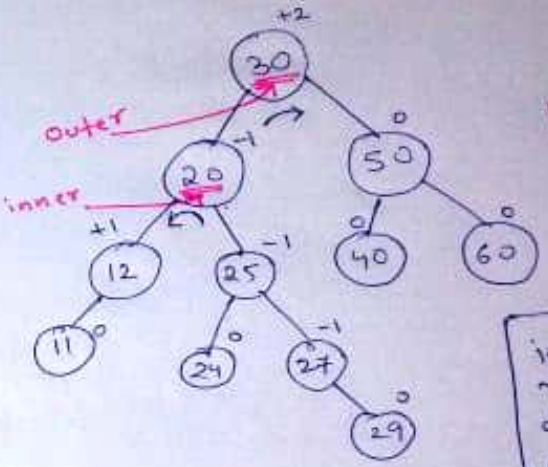


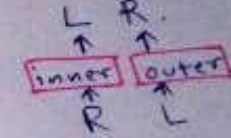
RR ↶



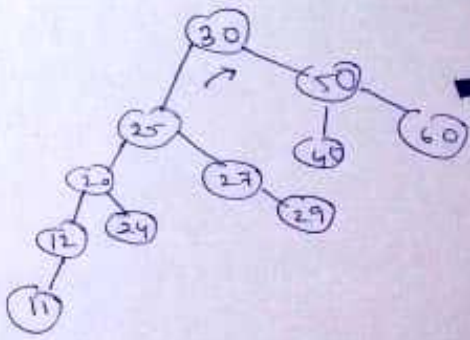
→ Double Rotation.

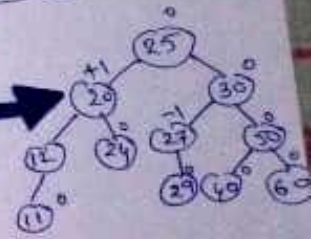
LR ⟨ ⟩ RL





innermost node in the direct [during direction change]





Lecture: 6 Binary Search Tree 11/10/2017

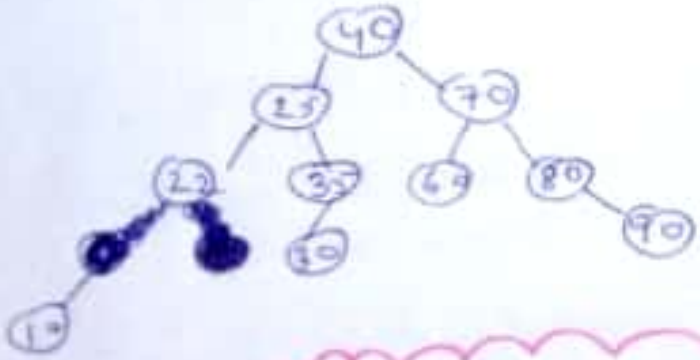
SKM.

① Create a Binary Search Tree.
② Delete a node from Binary Search Tree.

Rule :-

1. Value of left subtree is less than the value of parent node.
2. Value of Right subtree is greater than value of parent node.

40, 25, 70, 22, 35, 60, 80, 90, 10, 30



Two types of node

1) leaf node → remove that.
2) non leaf node → 1 child.
   → 2 children.

1 child → Attach the child to the parent of the to be deleted node.

2 child → Find the inorder (LR/Right) successor of to be deleted node and replace with that.

Construct BST: → Recursive
→ Non-Recursive

```
typedef struct Bnode{
        int data;
    struct Bnod* left;
    struct Bnode* right;
        }bnode;
```

Rewrsive method:

```
bnode *   create BST (bnode *r) int d
{ if (r==NULL)
  {
  r=(bnode*)malloc (sizeof(bno
      r->data=d;
      r->left=r->right=NU
  }
```

```
        else
            if (r->data > d)
    r->left = CreateBST(r->left, d);
        else
    r->right = CreateBST(r->right, d);
        return r;
}

void main()
{
    bnode *root;
    root = NULL;
    int x;
do { scanf("%d", &x);
    root = CreateBST(root, x);
    printf("Do you want to
            insert");
        ch = getch();
    } while(ch!='n' || ch!='N');

}
```