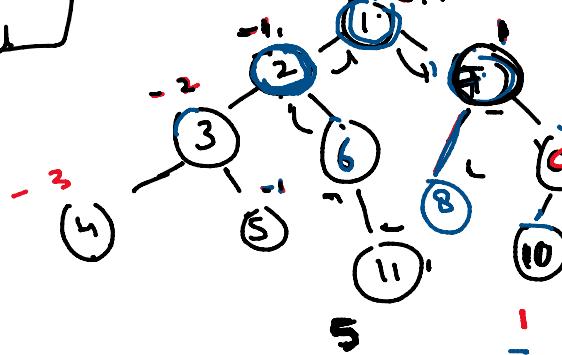
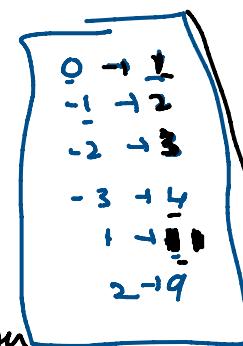


TreeTop Viewleft view / right viewalong the row / along the levelAlong the column = First Node of every column

4 3 2 1 7 9

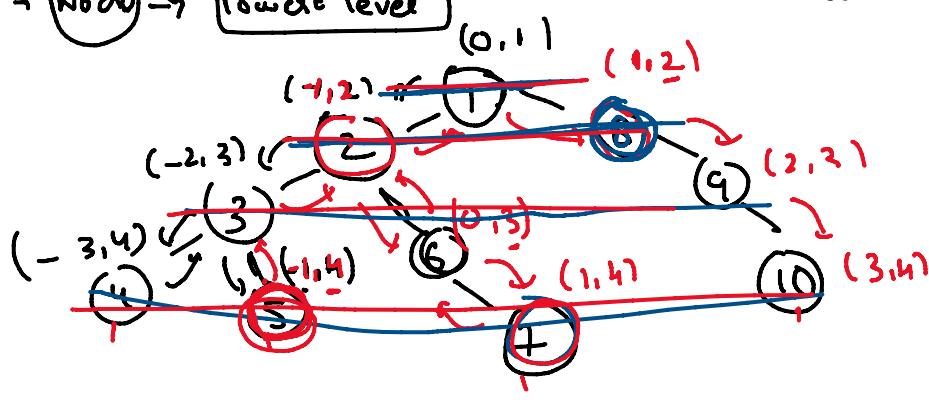
Vertical Order TraversalDFS, BFS

1 2 3 4 7 9

1 2 3 4 11 9

DFS \Rightarrow Higher depth than node at lower depthColumn \rightarrow Node \rightarrow lowest level

map < int, pair<int, int>> m



0	-2, 1, 1, 4
-1	-2, 2, 4
-2	-2, 3, 3, 4
-3	-3, 4, 4, 4
1	2, 2, 8, 4
2	2, 3, 9, 4
3	2, 4, 10, 4

void topview (Node* root, int col, int level, map<int, pair<int, int>>&m)

2 if (root == NULL)

2 return;

2

if (m[col] != m.end())

{ if (m[col].first > level)

{ m[col] = {level, root->data}; }

2

```

    |
else
{
    ma[0][l] = 2; level, root → data '2';
}

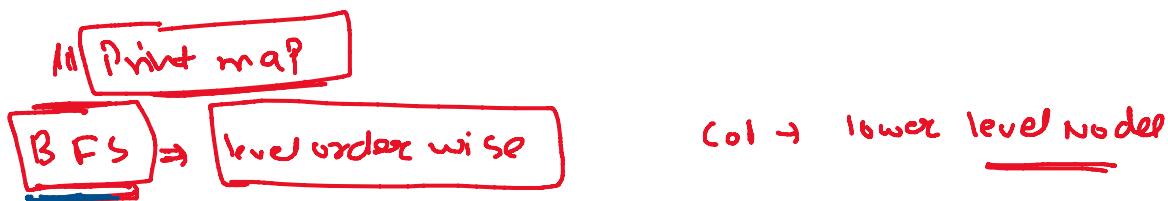
```

```

ToView ( root → left, col - 1, level + 1, ma );
"   ( root → right, col + 1, " , ma );

```

2



```

queue < Pair < int, Node * > > av;
av.push ( 20, root → 2 );
map < int, int > ma;
while ( ! av.empty () )
{
    auto f = av.front ();
    if ( ma[ f . first ] == ma[ f . second ] ) Remove Bottom view
        {
            ma[ f . first ] = f . second → data ;
        }
    av.pop ();
    if ( f . second → left )
        {
            av.push ( { f . first - 1, f . second → left } );
        }
    if ( f . second → right )
        {
            av.push ( { f . first + 1, f . second → right } );
        }
}

```

"bind map"

Bottom view

→ Deepest Node of each column

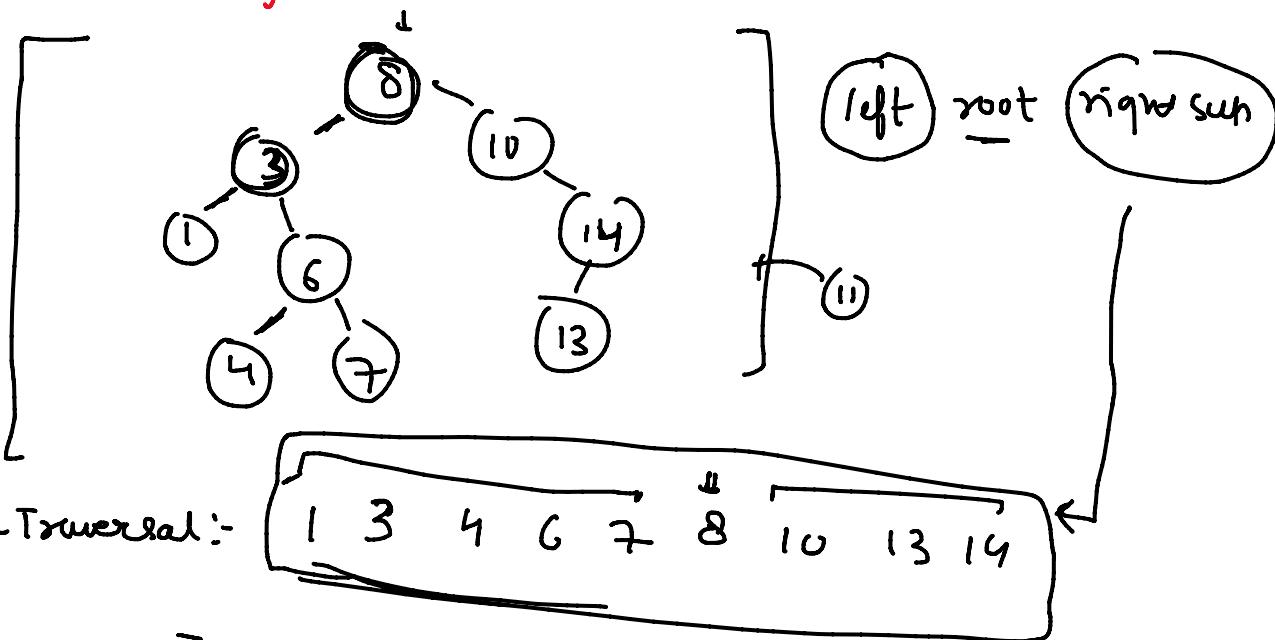
Difference?

Binary Tree

Binary Search Tree (BST)

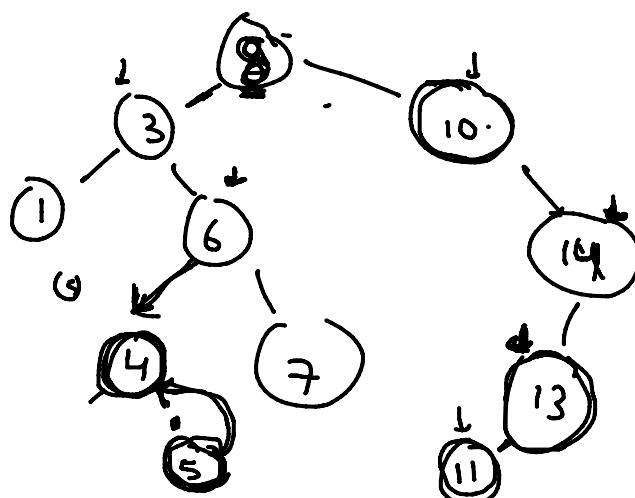
Special Type of Binary Tree

- Each node in left subtree is less than root Node value
- " " " Right " " more " "



[Insertion in BST]

5



Insertion in BST
always at leaf Node



Node *insert BST (Node *root, int d)

if (root == null)

* Node * new Node= new Node (d);

return new Node j

if(d > root->data)

$\text{root} \rightarrow \text{right} = \text{insert BST}(\text{root} \rightarrow \text{right}, d);$

7

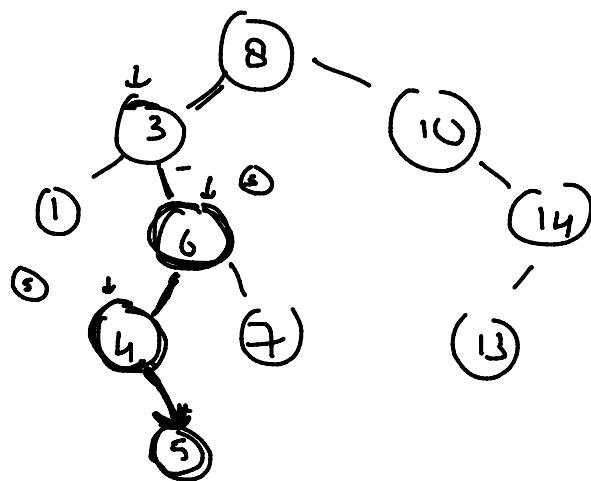
~~else~~ $\leftarrow \text{root} \rightarrow \text{left } t = \text{insert BST } (\text{root} \rightarrow \text{left } t, d);$

1

return route;

三

5



三

A hand-drawn diagram of a tree structure. The root node is labeled 'X'. Node 1 is a child of the root, with a label '1' above it. Node 2 is a child of node 1, with a label '2' above it. Node 3 is a child of node 2, with a label '3' above it. Node 4 is a child of node 3, with a label '4' above it. Node 5 is a child of node 4, with a label '5' above it. Node 6 is a child of node 5, with a label '6' above it. Node 7 is a child of node 6, with a label '7' above it. Node 8 is a child of node 7, with a label '8' above it. Node 9 is a child of node 8, with a label '9' above it. Node 10 is a child of node 9, with a label '10' above it. Node 11 is a child of node 10, with a label '11' above it. Node 12 is a child of node 11, with a label '12' above it. Node 13 is a child of node 12, with a label '13' above it. Node 14 is a child of node 13, with a label '14' above it. Node 15 is a child of node 14, with a label '15' above it. A horizontal line with arrows at both ends is positioned below the tree, spanning from node 1 to node 15. Above the tree, there is a sequence of numbers: 8, 7, 6, 10, 22, 11, 15. The numbers 8, 9, 10, 11, and 15 are circled in red.

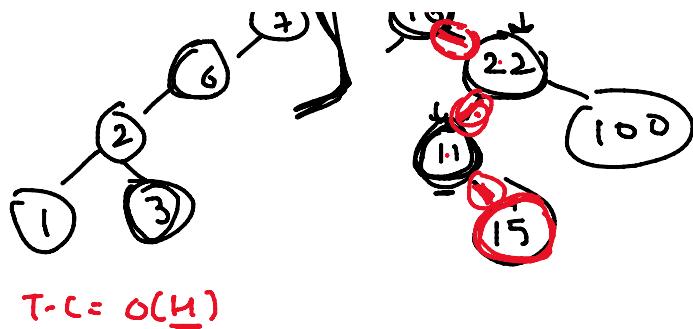
2 - 3 = ?

O(log n)

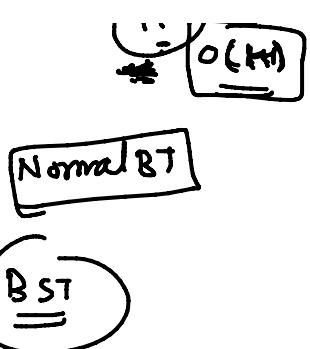
O(1)

!!

30

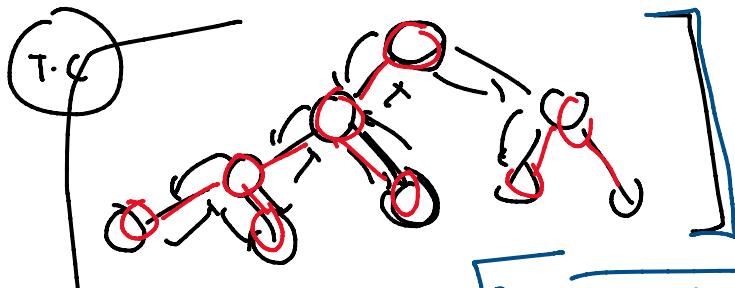


Worst case
longest Path



Searching in a BST

only comparison



B.T

$T.C = O(1)$

$$O(n) = O(k \cdot H) \times$$

Balanced BST

$$(k_H - k_1) \leq 1$$

$$H = \log_2(n)$$

$$T.C = O(\log_2 n)$$

$BST \rightarrow O(H)$

Balanced Binary Tree

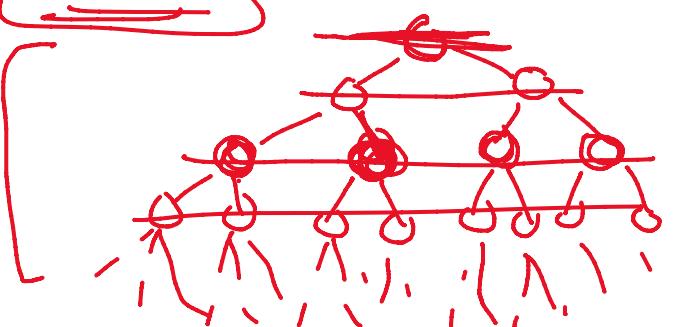
$$T.C \propto \text{depth}$$

{
0
1
2
3
4
5}

$$H = 5$$

Complete BT

Node - 2 child



$$\left[2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^h \right] = n$$

$$\log_2 (2^h - 1) = n$$

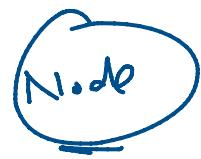
$$\frac{1}{2} 2^h = n+1 \Rightarrow$$

$$\text{height of a BT} = \log_2(n+1) \approx \log_2(n)$$

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

Balanced

Searching in BST



```

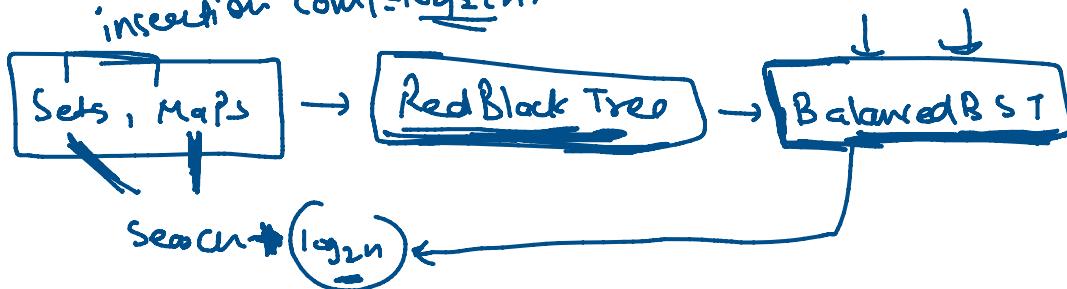
Node* search( Node* root, int d )
{
    if (root == NULL)
        return NULL;

    if (root->data == d)
        return root;

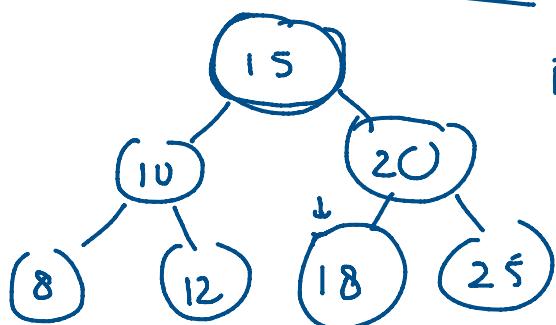
    if (root->data > d)
        return search( root->left, d );
    else
        return search( root->right, d );
}

```

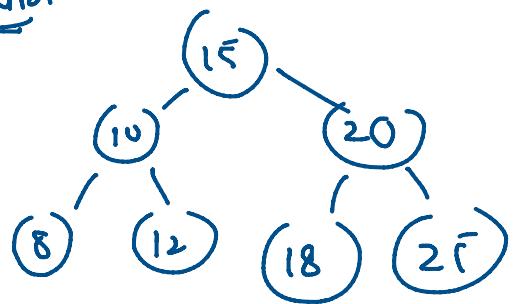
insertion comp = $\log_2(n)$

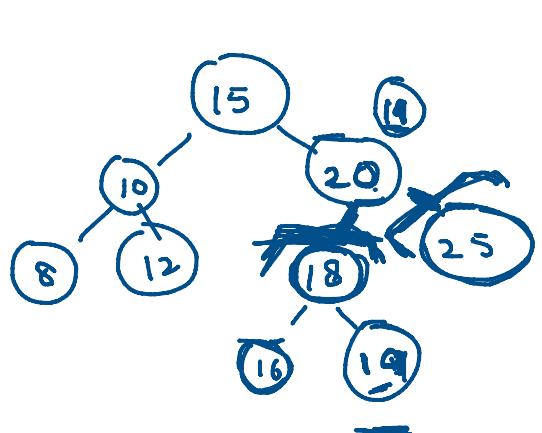
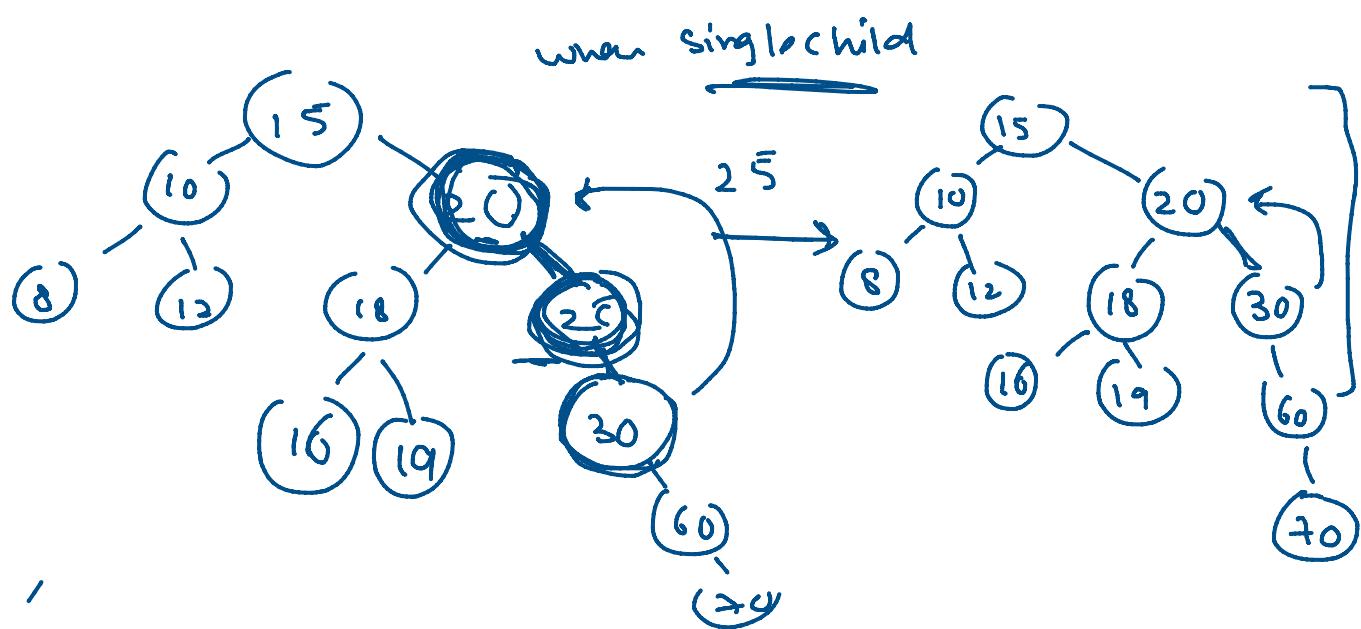


Deletion in BST on child

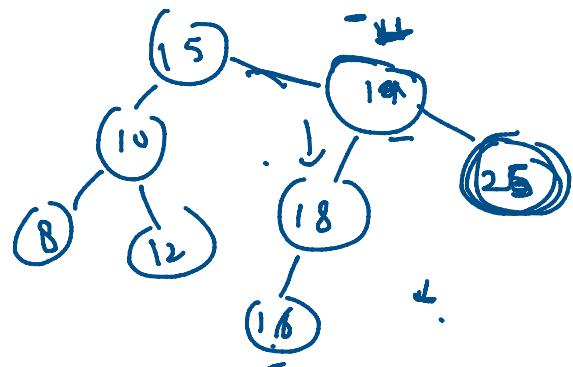


Delete 18





Node 20



Single child