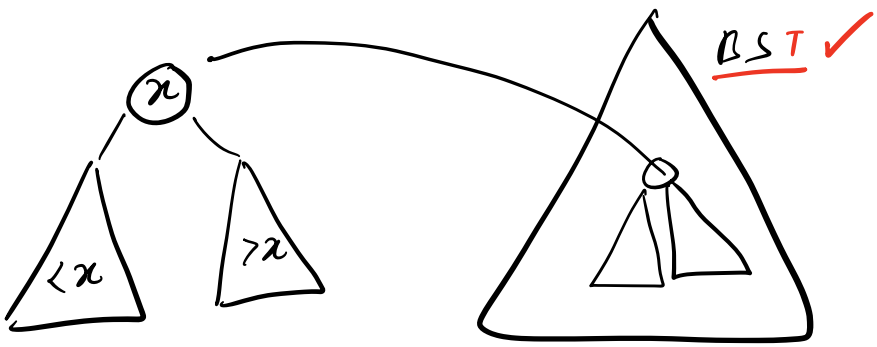
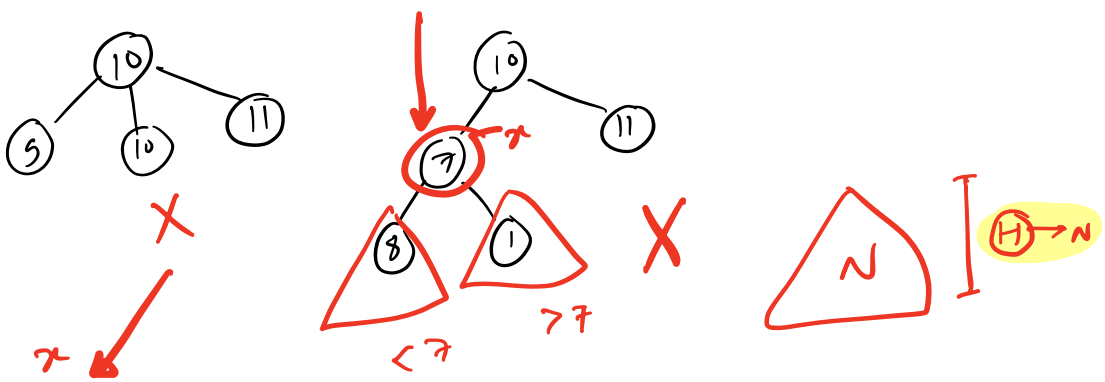
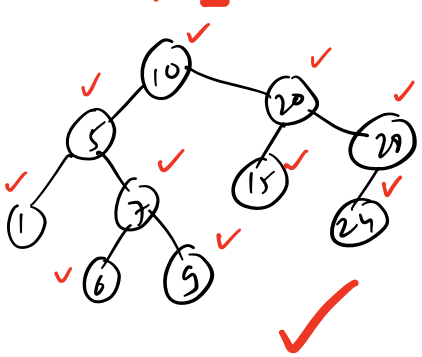
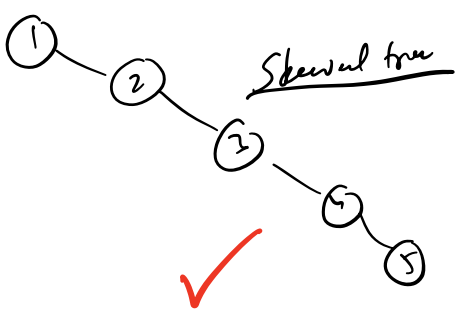
BST [Binary Search Tree]

- Binary tree
- All elements in the LST of a node < node's val
- RST >









All on left < Node.val < All on right

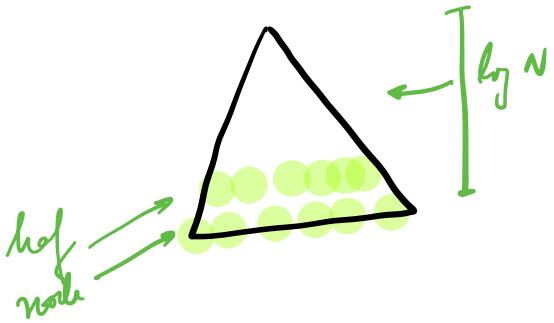
Max on left < Node.val < Min on Right

BBST → < Balanced BST >

(Abs. diff. b/w the height of LST & RST <= 1 for all nodes)

- Red Black tree
- AVL tree

$H = O(\log N)$

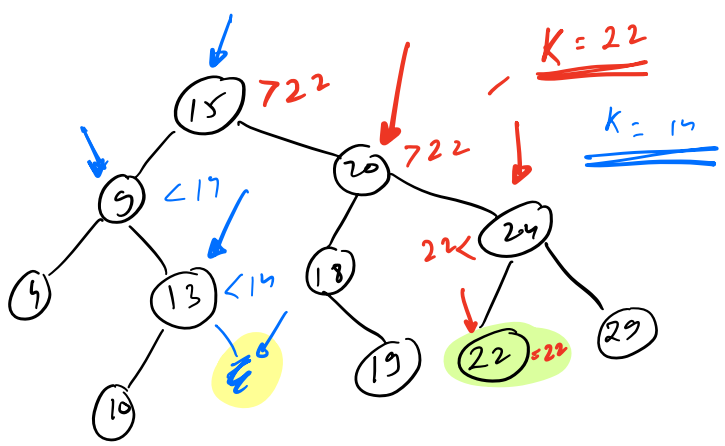


treeSet
treeMap
: Keys sorted !

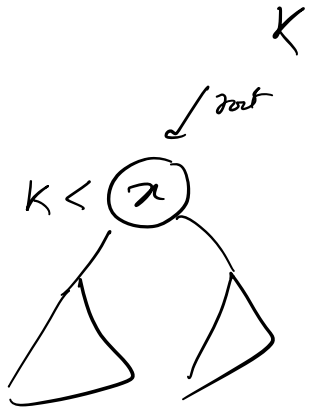
TC: O(log(N))

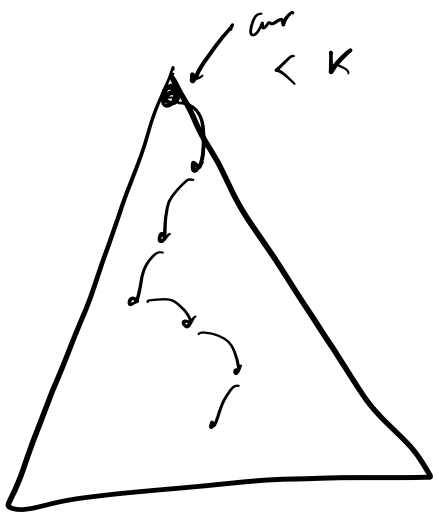
Normal BST →

① Searching



```
bool search(Node root, int K) {
    if( root == NULL) ret false;
    if( root.data == K) ret true;
    else if( root.data < K) {
        ret search(root.right, K);
    }
    else {
        ret search(root.left, K);
    }
}
```





TC = O(H)

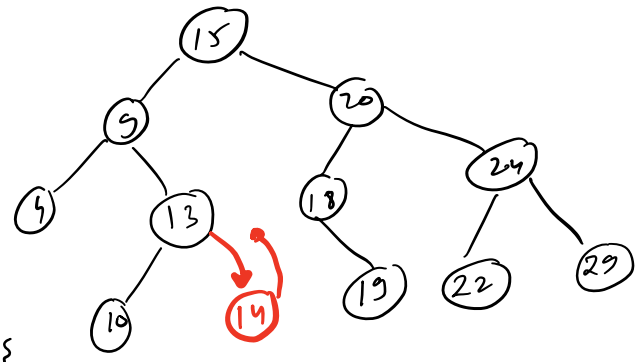
SC = O(H)

O(1) : iterative code!

Q Given a BST & a val K.
Insert K in the BST

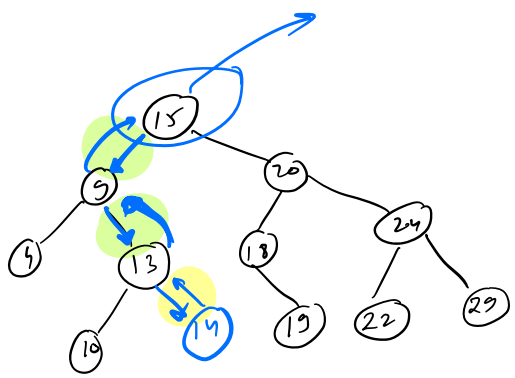
K = 13

K = 14




```
Node insert(Node root, int k){
    if(root == NULL){
        Node temp = new Node(k);
        ret temp;
    }
    if(root.data < k){
        root.right = insert(root.right, k);
    }
    else if(root.data > k){
        root.left = insert(root.left, k);
    }
    ret root;
}
```

K = 14



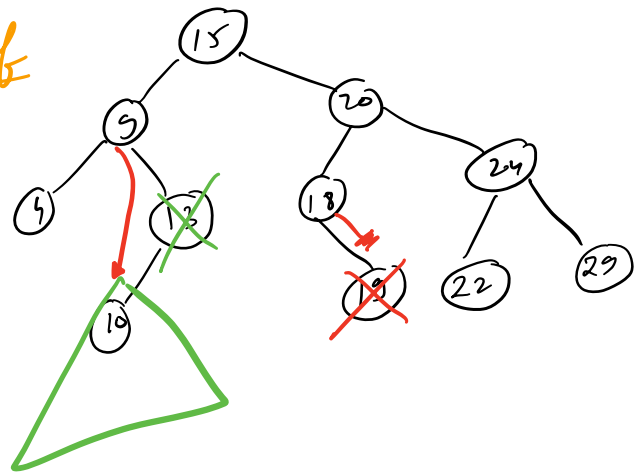
TC = O(H)

SC = O(H)

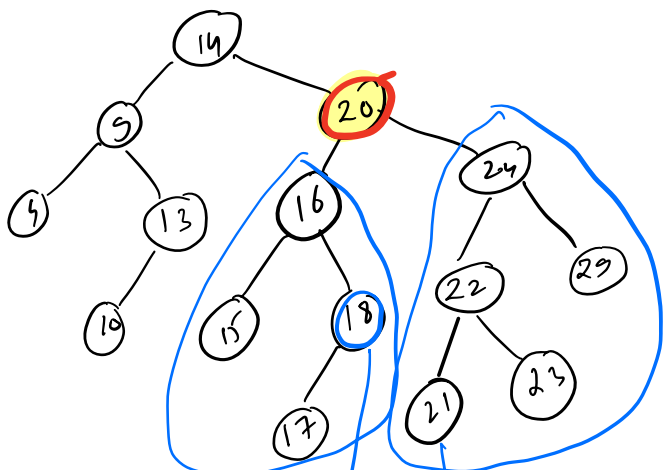
O(1)
iterative

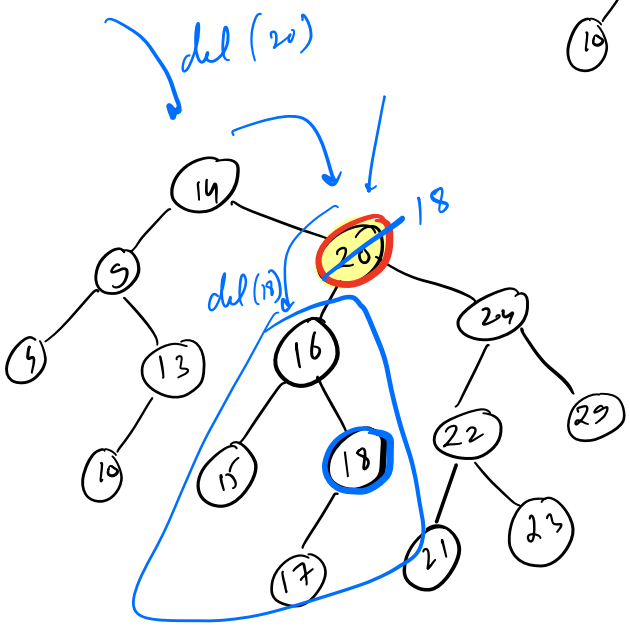
Q Given a BST & K. Del the node with val k

- del (15)
  18.right = NULL | leaf
- del (13) | 1 child



- del (20):





largest of LST or smallest of RST

<= 1 child

```
Node del ( Node root, int K) {
    if ( root == NULL) {
        ret NULL;
    }
    if ( root.data < K) {
        root.right = del( root.right, K);
    }
    eln if ( root.data > K) {
        root.left = del (root.left, K);
    }
    eln {
        if ( root.left == NULL && root.right == NULL) {
            ret NULL
        }
        eln if ( root.left == NULL){
            ret root.right;
        }
        eln if ( root.right == NULL) {
            ret root.left;
        }
        eln {
            G = find Greatest (root.left);
            root.data = G;
            root.left = del ( root.left, G);
        }
    }
```

int root;

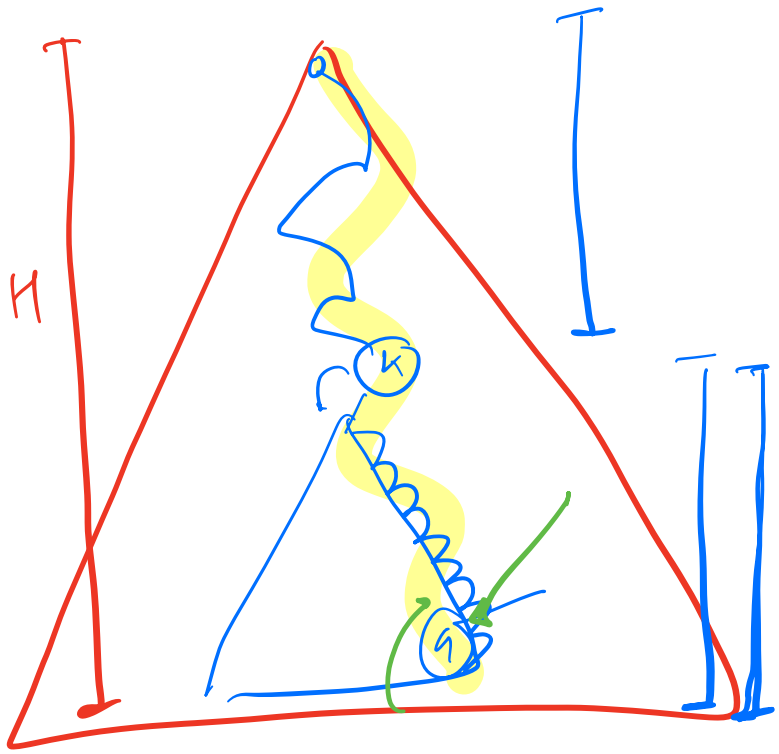
}

TC = O(H)

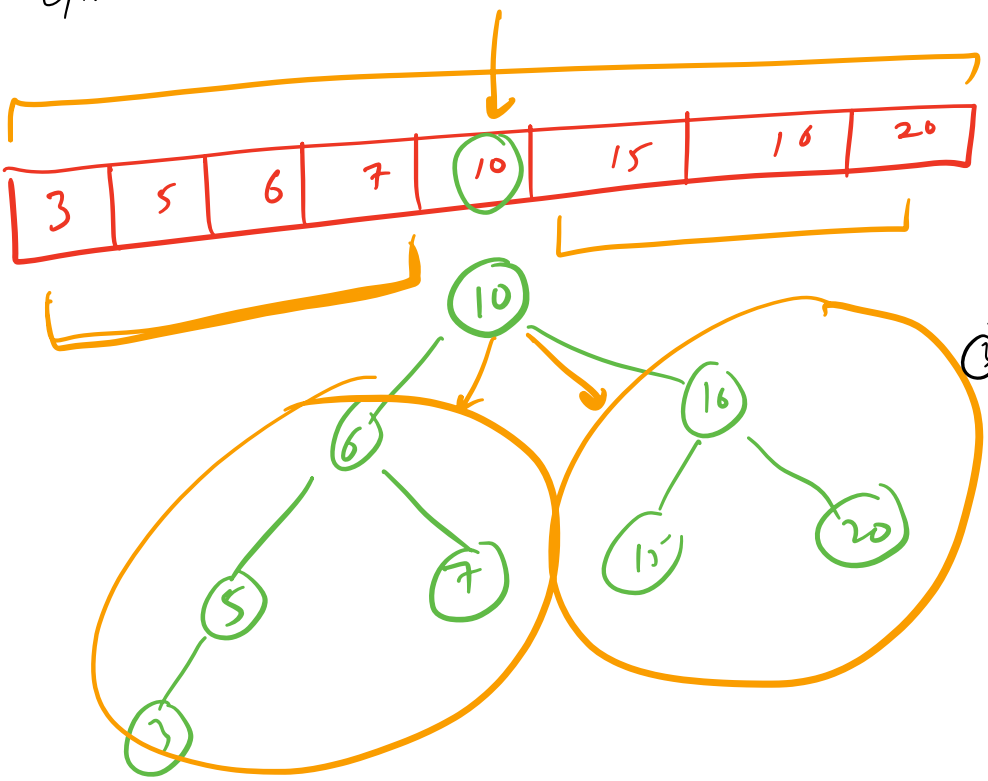
SC = O(H)

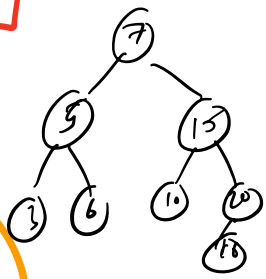
O(1)

iterative



Q Given a sorted array. Construct a BBST from it!

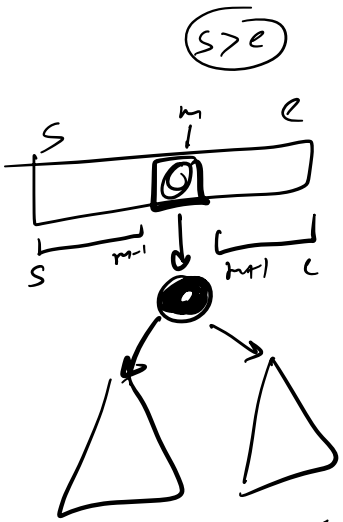


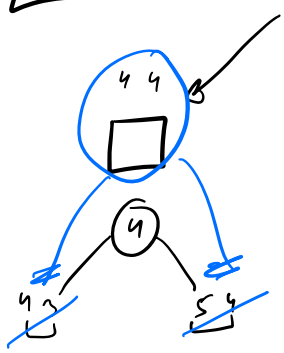


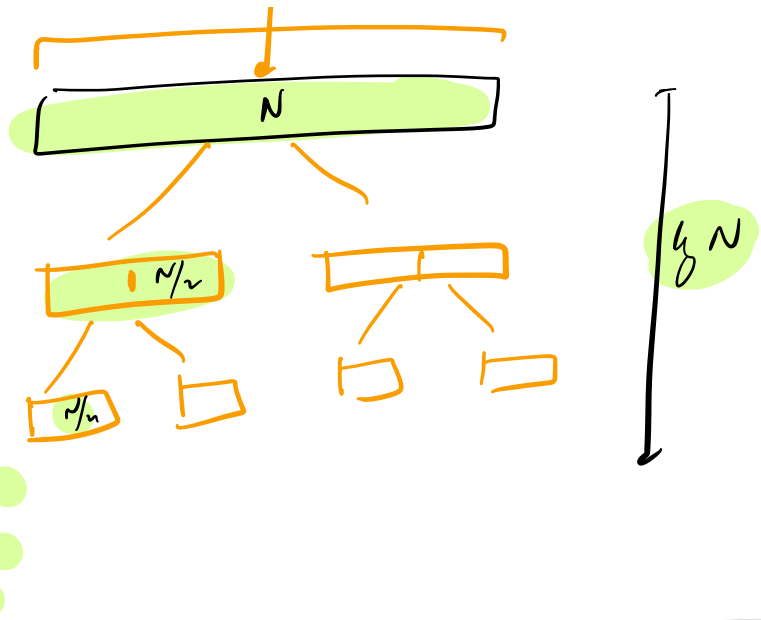
```
Node build(A[], s, e) {
    if (s>e) ret NULL;
    m = (s+e)/2;
    Node root = new Node(A[m]);
    root.left = build(A[], s, m-1);
    root.right = build(A[], m+1, e);
    ret root;
}
```

TC = O(N)

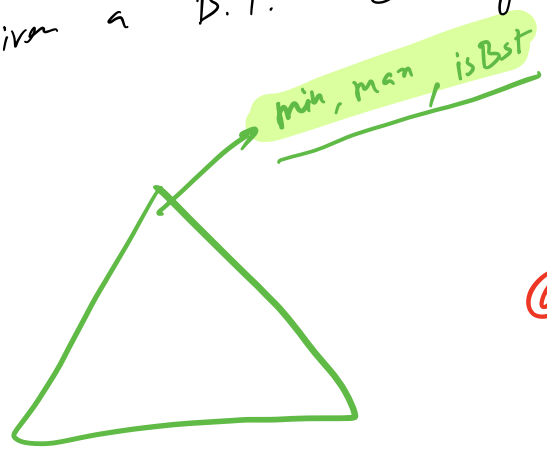
SC = O(4N)

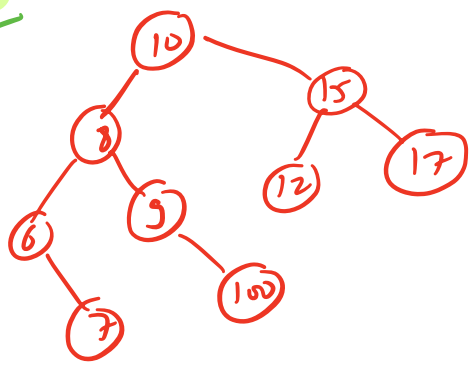






Q Given a B.T. Check if it is a BST!



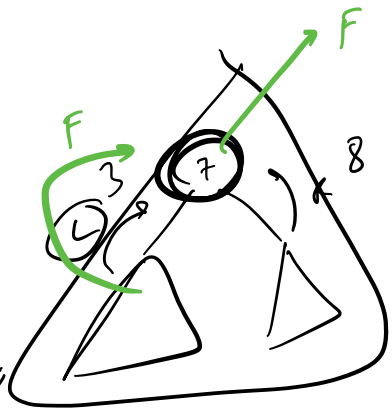


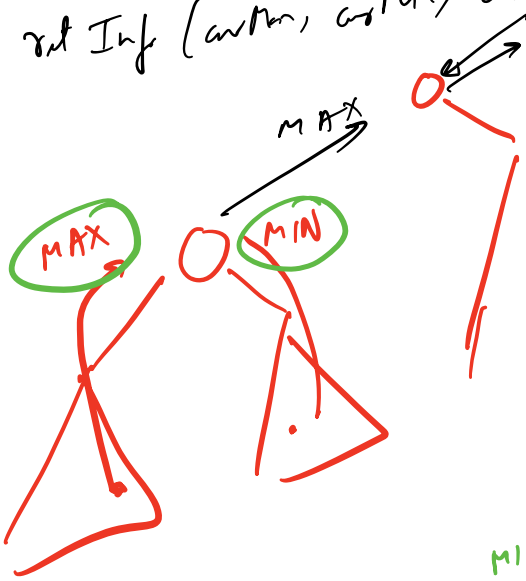

```
In Info {
  int maxV;
  int minV;
  bool isBst;
}
```

```
Info check ( Node root) {
    if ( root == NULL) {
        ret Info ( -∞, +∞, true);
    }
    Info L = check ( root.left);
    Info R = check ( root.right);
    curMax = max ( L.maxV, R.maxV, root.data);
    curMin = min ( L.minV, R.minV, root.data);
    curIsBst = ( L.isBst == true  && 
                 R.isBst == true  &&
                 L.maxV < root.data && root.data < R.minV );
    ret Info (curMax, curMin, curIsBst);
}
```

ret check(root).isBst

ret early!





TC = O(N)

SC = O(11)