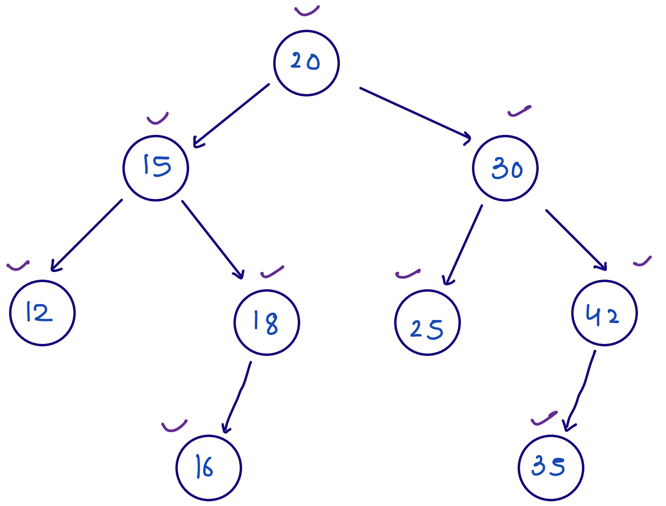
Agenda

1) BST intro and its properties
2) Search in BST (TC comparison with BT)
3) Insert in BST
4) Is BST
5) Sorted array to balanced BST

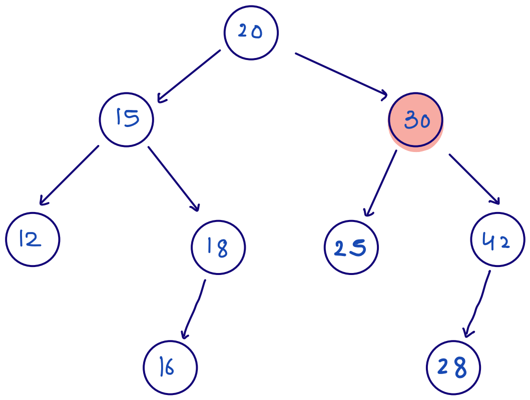
Binary Search Tree (Introduction)

A binary tree in which every node follow this property:

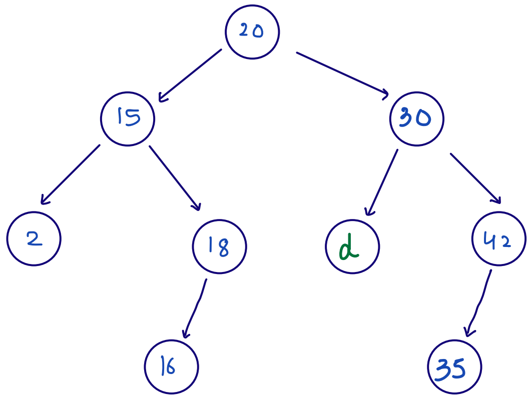
all nodes coming in left subtree < node.val < all nodes coming in right subtree



→ BST



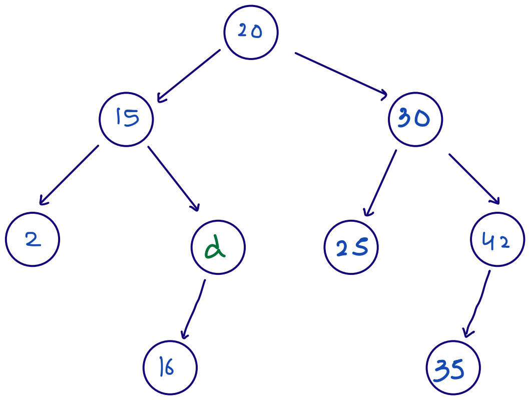
→ not a BST



range of d so that this given tree is a valid.

$20 < d < 30$

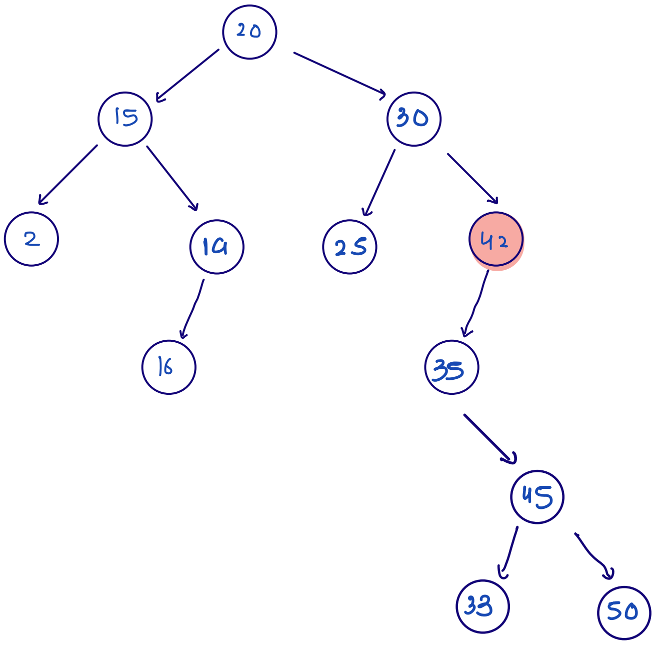
↳ 21 to 29



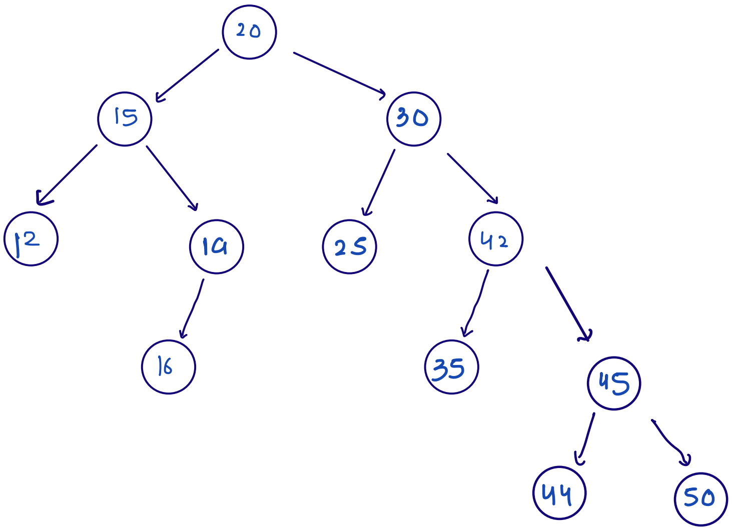
values of d such that this is a valid BST.

$15 < d < 20$

↳ ~~16~~ 17 18 19



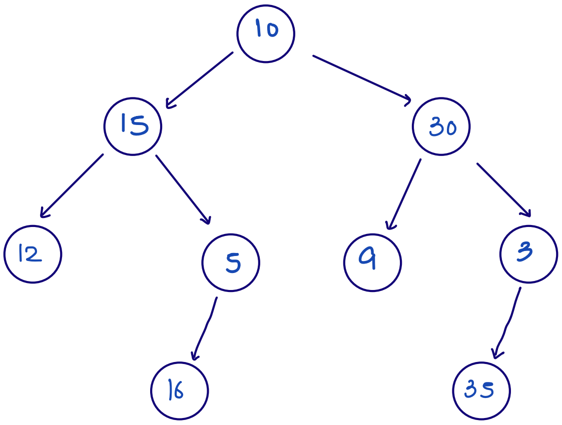
→ not a BST



→ BST

Binary tree vs Binary search tree

Binary tree



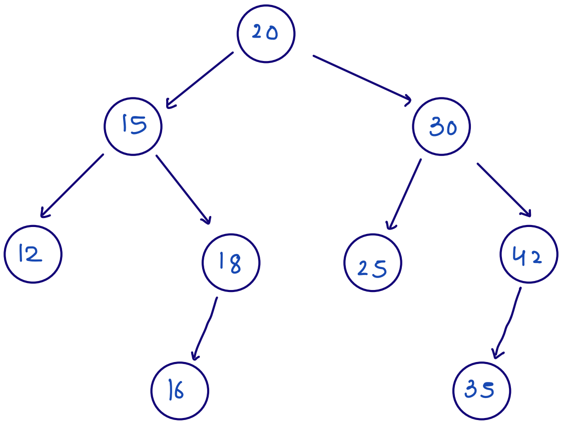
Search for K

TC : O(n)

SC : O(h) {recursive space}

↳ height of tree

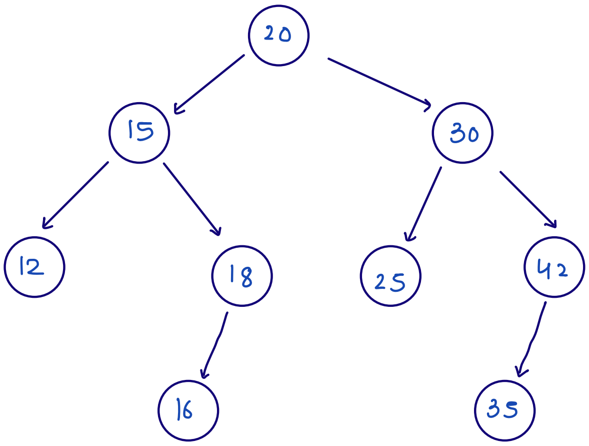
Binary search tree



Search for K

TC : O(h) {travelling a single branch}

SC : O(h)

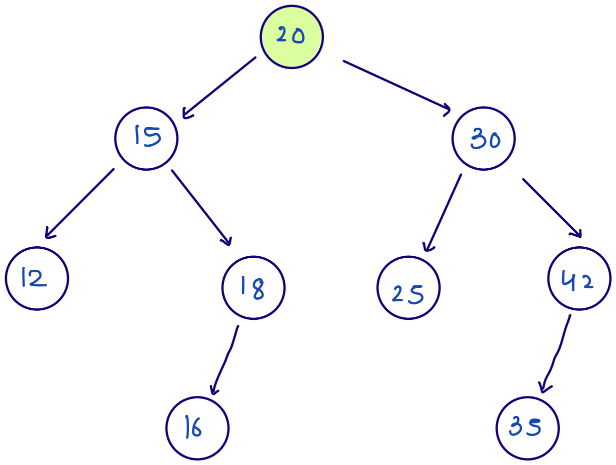


L N R

Inorder: 12 15 16 18 20 25 30 35 42

Inorder of a BST is sorted

Q.1 Given root node of a BST, search if k exists or not.

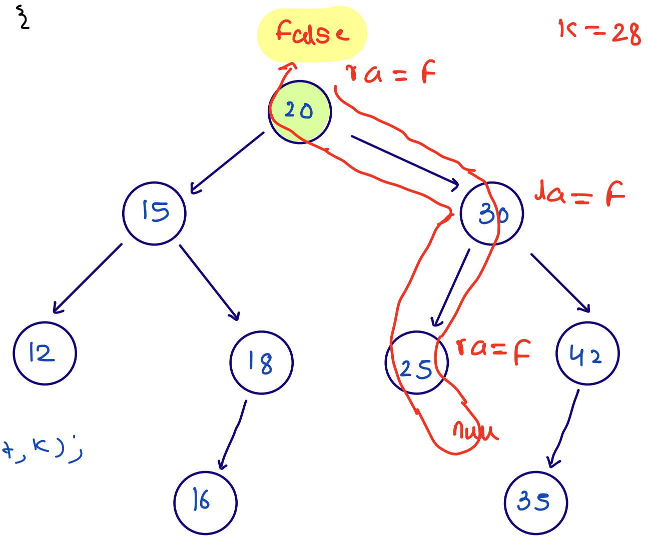


k = 28 (false)

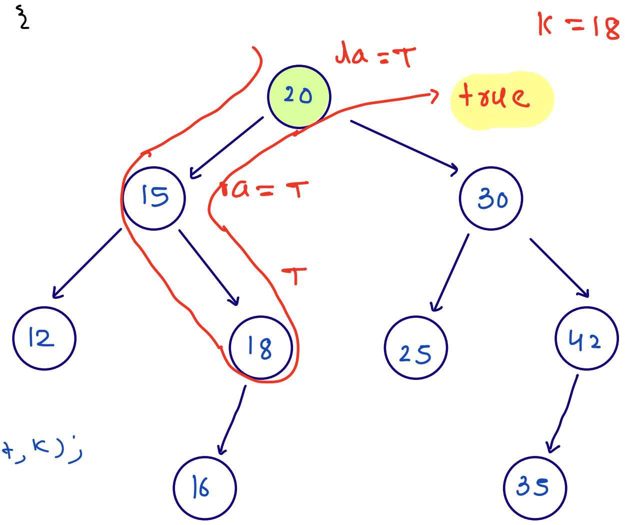
k = 18 (true)

```
boolean searchInBST ( Node node , int k) {
    if( node == null) {
        return false;
    }

    if (node.val == k) {
        return true;
    }
    else if ( k < node.val ) {
        boolean la= searchInBST (node.left, k);
        return la;
    }
    else {
        boolean ra= searchInBST (node.right, k);
        return ra;
    }
}
```

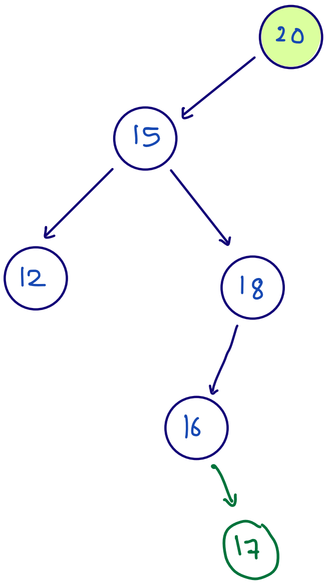
```
boolean searchInBST ( Node node , int k) {
    if( node == null) {
        return false;
    }

    if (node.val == k) {
        return true;
    }
    else if ( k < node.val ) {
        boolean la= searchInBST (node.left, k);
        return la;
    }
    else {
        boolean ra= searchInBST (node.right, k);
        return ra;
    }
}
```



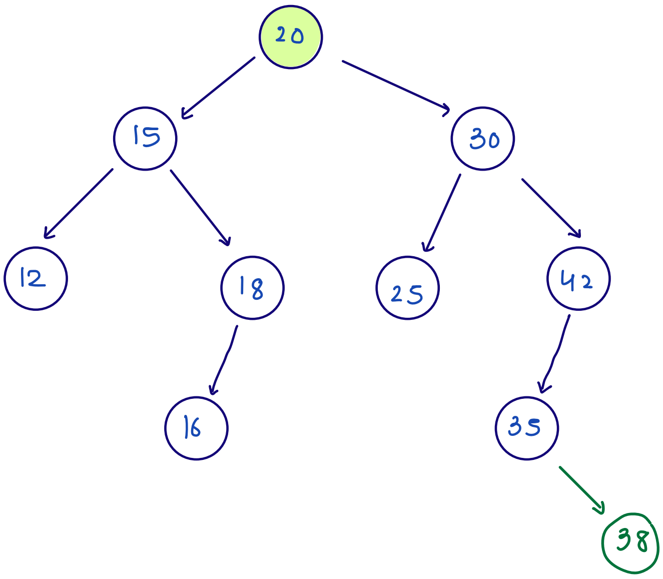
```
boolean searchInBST ( Node node , int k) {
    if( node == null) {
        return false;
    }

    if (node.val == k) {
        return true;
    }
    else if ( k < node.val ) {
        boolean la= searchInBST (node.left, k);
        return la;
    }
    else {
        boolean ra= searchInBST (node.right, k);
        return ra;
    }
}
```



Q.2 Given root of a BST, insert node with data K in this BST.
(Insertion should be done without shuffling the existing node)



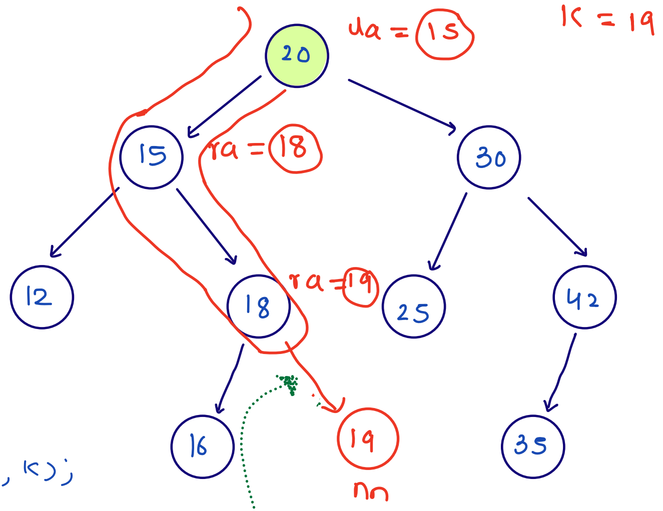
K = 17



K = 38

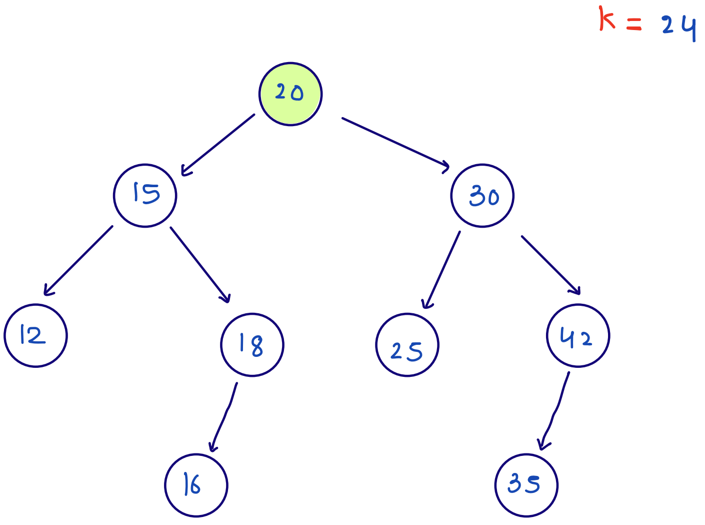
```
Node    insertInBST (Node node, int k) {
      if( node == null ) {
            Node nn= new Node(k);
            return nn;
      }

      if(node.val==k) {
            return node;
      }
      else if ( k < node.val) {
            Node la= insertInBST (node.left, k);
            node.left=la;
            return node;
       }
       else {
            Node ra= insertInBST (node.right, k);
            node.right=ra;
            return node;
       }

}
```

```
Node insertInBST (Node node, int k) {
      if( node == null) {
            Node nn= new Node(k);
            return nn;
      }
      if(node.val==k) {
            return node;
      }
      else if (k < node.val) {
            Node la= insertInBST (node.left, k);
            node.left= la;
            return node;
      }
      else {
            Node ra= insertInBST (node.right, k);
            node.right=ra;
            return node;
      }
}
```

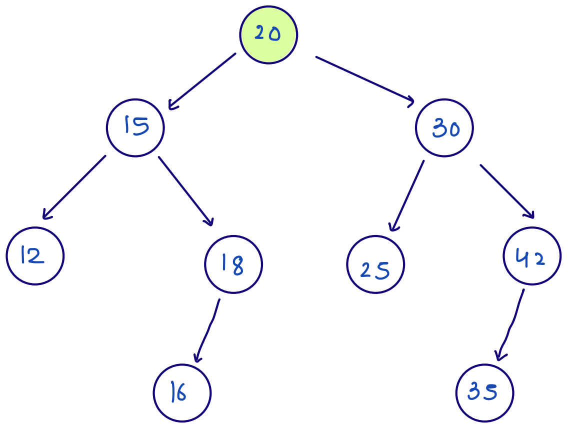


todo: do it with void return type in insert function

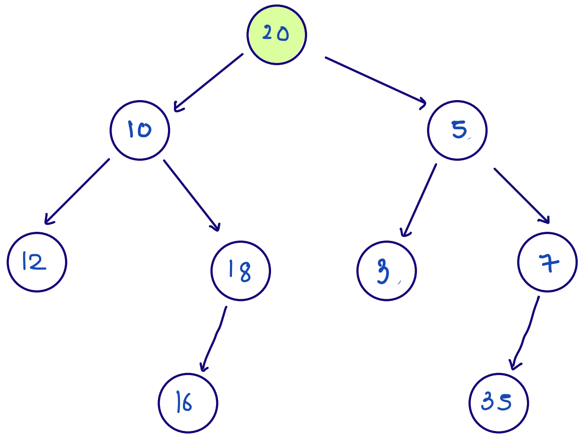
proactive



Q.3 Given root of a binary tree, check if it is BST or not.



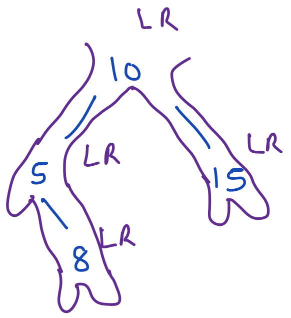
→ BST (true)



→ Not BST (false)

i) fill tree's inorder in arraylist and check if this list is sorted or not.

```
AL <Integer> list = new AL<>();

void travel (Node node) {

     if(node == null) {
          return;
     }

     travel(node.left);

     list.add(node.val);

     travel(node.right);
}
```

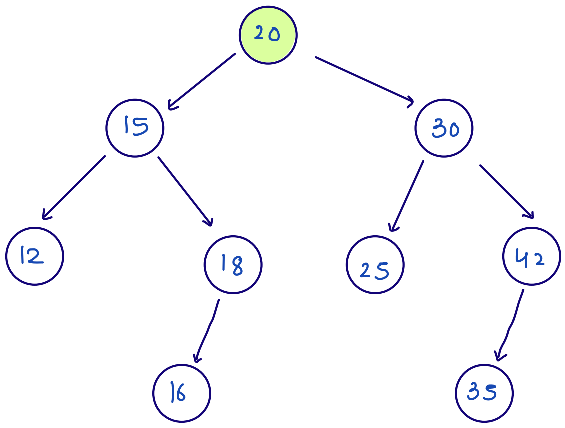


list : 5 8 10 15

TC : $O(n)$

SC : $\underset{\text{travel}}{\underset{\downarrow}{O(h)}} + \underset{\text{AL}}{\underset{\downarrow}{O(n)}} \approx O(n)$

ii) Can you do it only by using recursive space ?



Inorder : 12 15 16 18 20 25 30 35 42

prev

whenever I am at at node

prev is storing this node

inorder predecessor {the node

coming before curr node in Inorder}

```
boolean ans= true;
int prev= -∞;
void helper(Node node) {
    if(node== null) {
        return;
    }

    helper (node.left);

    if(prev >= node.val) {
        ans = false;
        return;
    }
    prev= node.val ;

    helper (node.right);
}
```

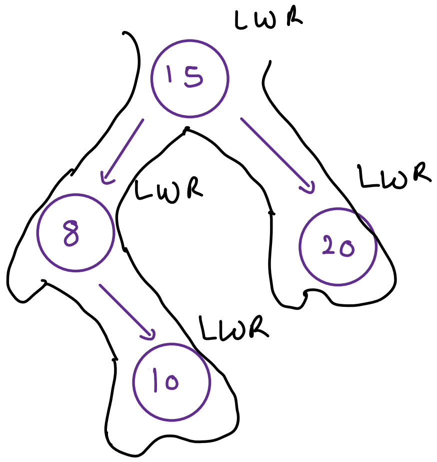
```
boolean isBST ( Node node) {
    ans= true;
    prev= -∞;
    helper (node);
    return ans;
}
```

TC : O(n)

SC: O(h)

→ work

```
boolean ans= true;
int prev= -∞;
void helper(Node node) {
    if(node == null) {
        return;
    }
    helper(node.left);

    if(prev >= node.val) {
        ans = false;
        return;
    }
    prev= node.val;

    helper(node.right);
}
```

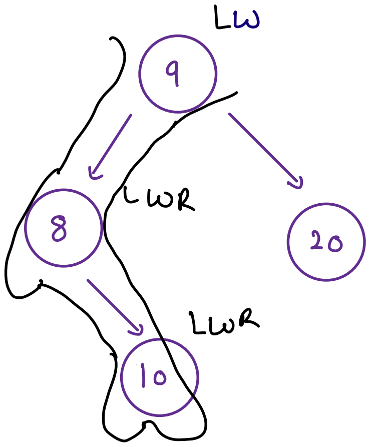


Inorder: 8 10 15 20

prev = ~~-∞~~ ~~8~~ ~~10~~ ~~15~~ 20

ans = true

```
boolean ans= true;
int prev= -∞;
void helper(Node node) {
    if(node == null) {
        return;
    }
    helper(node.left);

    if(prev >= node.val) {
        ans = false;
        return;
    }
    prev= node.val;

    helper(node.right);
}
```



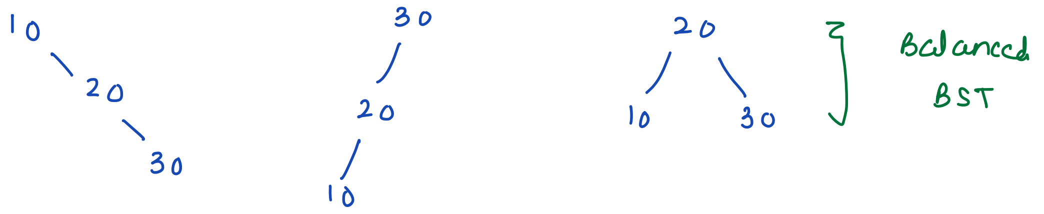
Inorder: 8 10 9 20

prev = ~~-∞~~ ~~8~~ 10

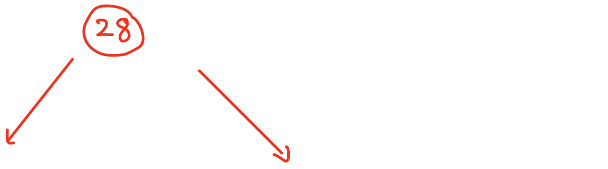
ans = ~~true~~ false

Q.4 Given a sorted array, construct balanced BST using this array and return its root node.

	0	1	2
A =	10	20	30



	lo			m				hi
	0	1	2	3	4	5	6	7
A =	10	15	20	28	32	35	42	45



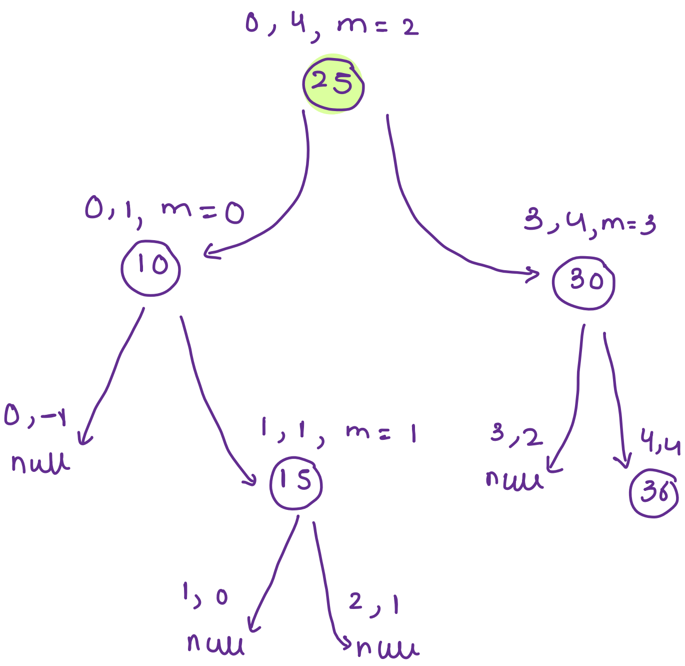
construct (A, lo, m-1)   construct(A, m+1, h)

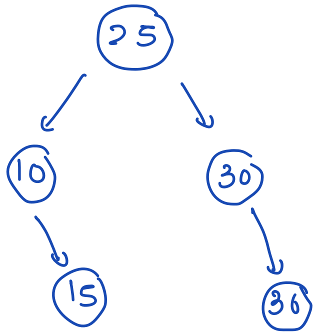
```
Node solve (int [] A) {
    n → A.length;
    return construct(A,0,n-1);

}
```

```
Node construct(int[] A, int lo, int hi) {
    if (lo > hi) {
        return null;
    }
    int m= (lo+hi)/2;
    Node nn= new Node(A[m]);

    nn.left = construct(A,lo,m-1);
    nn.right = construct(A,m+1,hi);

    return node;
}
```

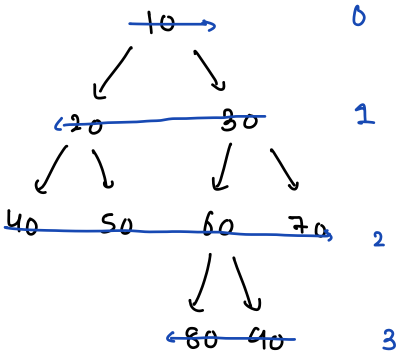
```
Node construct (int[ ]A, int lo, int hi) {
    if (lo > hi) {
        return null;
    }
    int m = (lo+hi)/2;
    Node nn = new Node(A[m]);
    nn.left = construct(A, lo, m-1);
    nn.right = construct(A, m+1, hi);
    return node;
}
```

$$A = \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ 10 & 15 & 25 & 30 & 36 \end{array}$$





Doubts



10

30 20

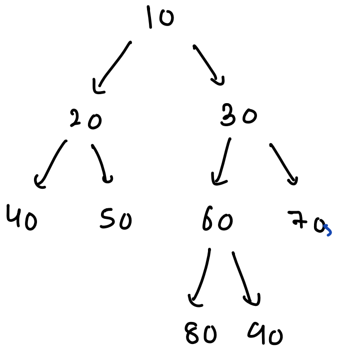
40 50 60 70

90 80

even levels → L to R

odd levels → R to L

```
q.add(root);
int lev=0;
while (q.size() >0) {
    int cs= q.size();
    AL<Integer>al = new AL<>();
    for(int i=1; i<=cs; i++) {
        Node temp= q.remove();
        al.add(temp.val);
        // add child
        if(temp.left !=null) {
              q.add(temp.left);
        }
        if(temp.right != null) {
              q.add(temp.right);
        }
    }
    [ if lev is even L to R
      else lev is odd R to L
    lev++;
}
```



~~10~~	~~20~~	~~30~~	~~40~~	~~50~~	~~60~~	~~70~~	~~80~~	~~90~~

cs = 2

al = 80 90

lev = ~~0~~ ~~1~~ ~~2~~ 3

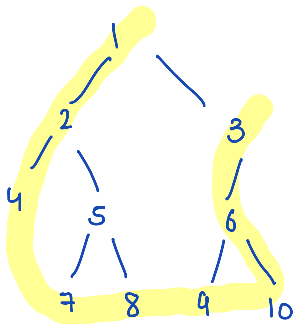
10

30 20

40 50 60 70

90 80

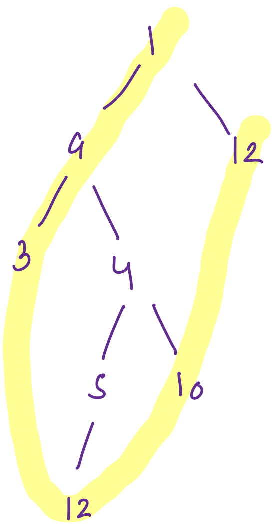
boundary traversal



left boundary : 1 2

leaf nodes : 4 7 8 9 10

right boundary : 3 6 (rev)
↓
6 3



left boundary : 1 9

leaf nodes : 3 12 10 12

right boundary :

```
void lb ( Node node ) {
    if (node == null ) {
        return;
    }
    if ( node is non-leaf) then use

    if (node.left != null) {
        lb (node.left);
    }
    else {
        lb (node.right);
    }
}
```

left_boundary