Trees – 3

Madhan Kumar M S
Abhishek Sharma
Akansh Nirmal
amit khandelwal
Balaji S K
Bhaveshkumar
Burhan
Gagan Kumar S
Gowtham
Ishan
Khushi Raj
Murali Mudigonda
Naval Oli
Nikhil Pandey
Pankaj Bhanu
Prajwal Khobragade
Purusharth A
Rajat Sharma
Rajendra
Sanket Giri
Saurabh Ruikar
Shani Jaiswal
sharath r
Shrikanth
Sneha L
Subhashini
Subhranil Kundu
Sumit Adwani
Sushant Patil
Suyash Gupta
Vasanth
Vetrivel H M
Yugesh v

GREAT JOB!

AGENDA:

- BST Introduction
- Searching in BST.
- Insertion in BST.
- Deletion in BST.
- Construct balanced BST from sorted array
- Check if the given binary tree is a BST.

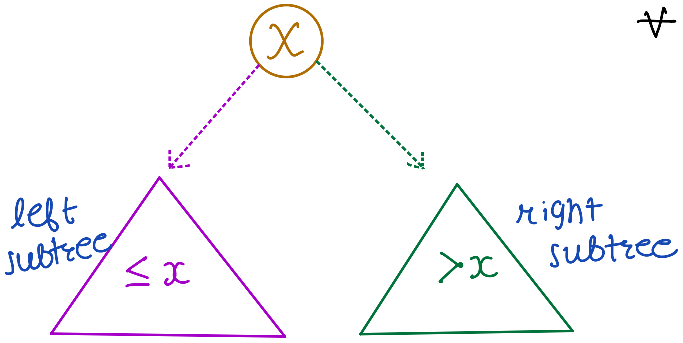
Contest 4 5th April

Current PSP
↓
62 %

Binary Search Tree

Searching data in organised dataset using divide and conquer

Convention for today's class keep equal elements on left

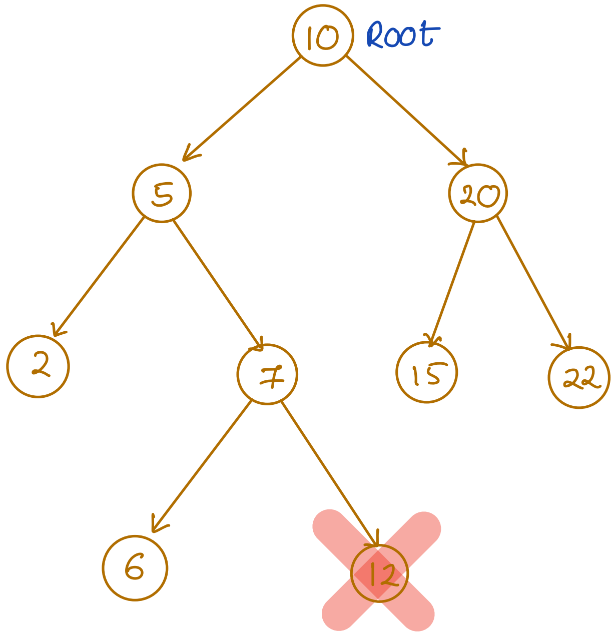


∀ nodes x

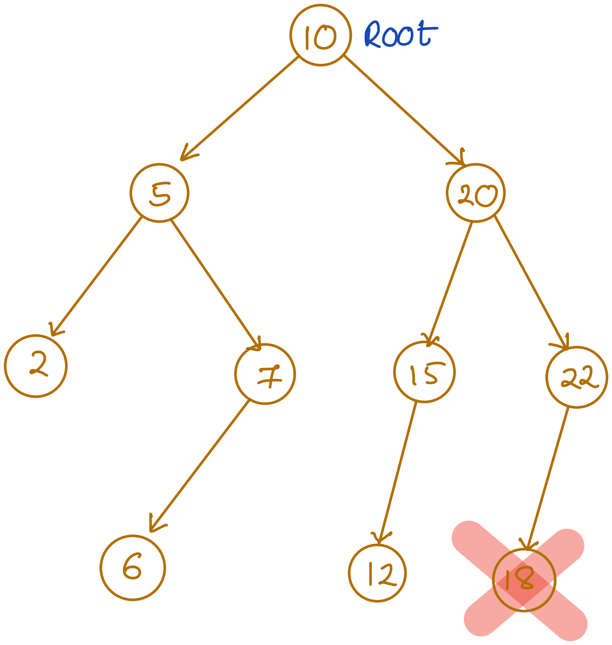
all the data in the left subtree <= x

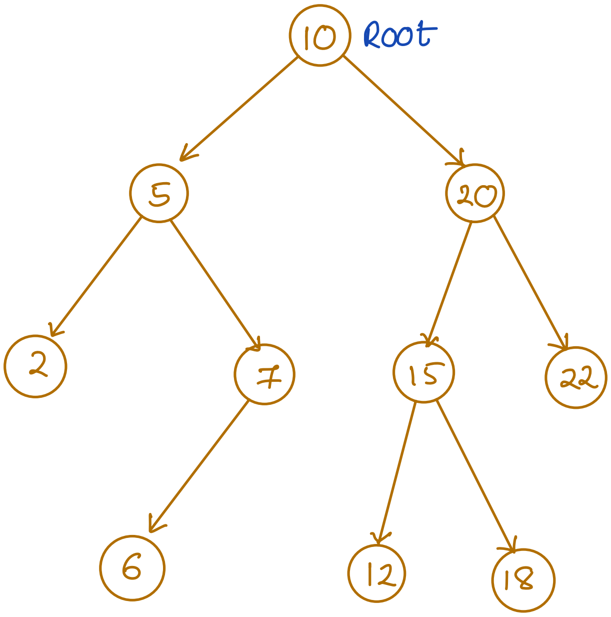
all the data in the right subtree > x

equality on the left side for this lecture

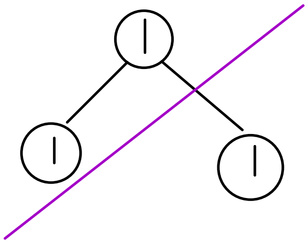


Not BST



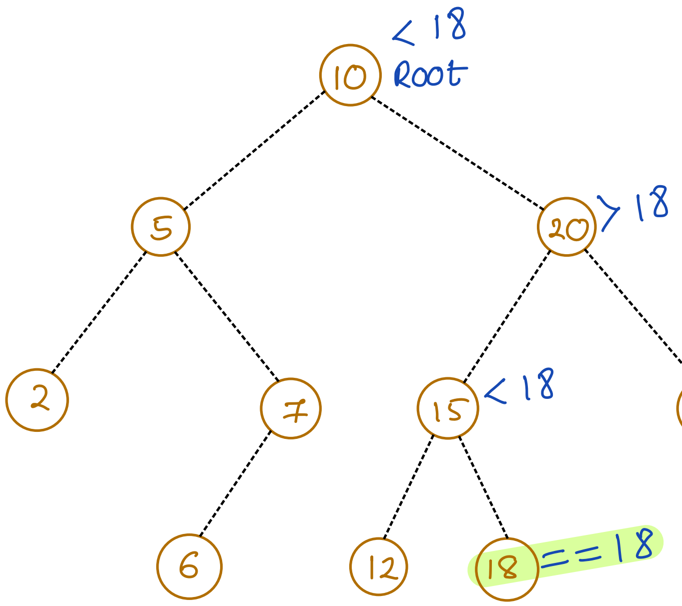


Valid BST



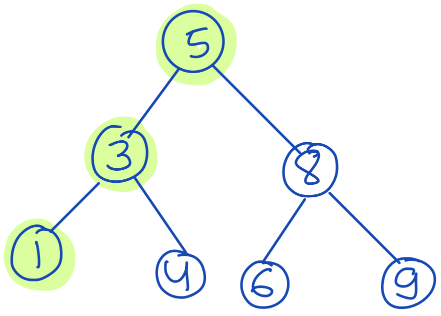
Not valid
as per our def''

Searching in BST



Find 18

Find 8



Find 1

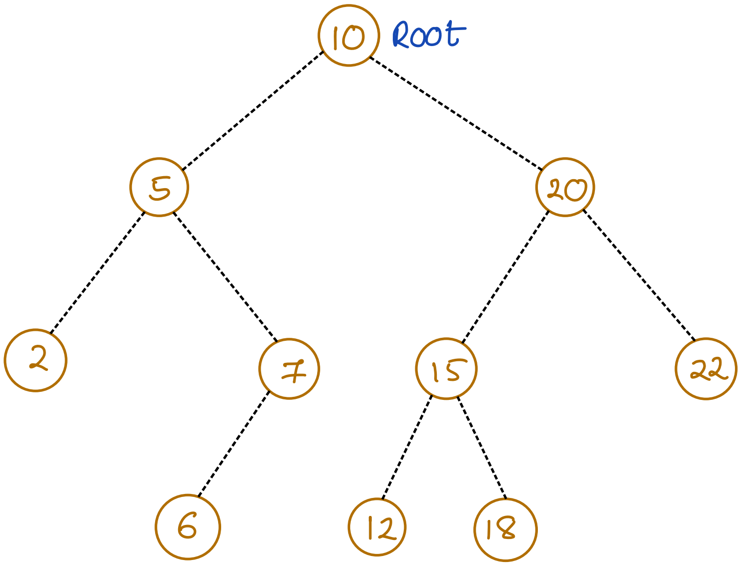
TC: O(H)

SC : O(H) / O(1) → iterative

Pseudocode

```
found = false

void    findTargetInBST ( root , target ) {
    if ( root == null )   return
    if ( root.data == target )   found = true
    if ( root.data < target )
            findTargetInBST ( root.right , target )
    else
            findTargetInBST ( root.left , target )
}
```

Insertion in a binary search tree



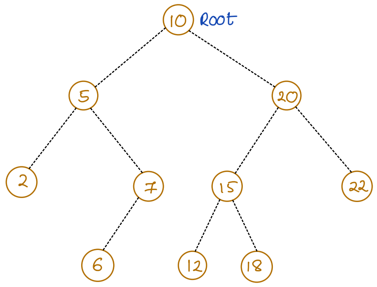
Insert (8)

① Search for the right place node will be inserted at .

Pseudocode

TC : O(H)
SC : O(1)

```
TreeNode  insertBST ( root , x ) {
        xn   =   new TreeNode(x)
        if ( root == null )  return xn
        temp = root
        while ( True ) {
            if ( temp .data < x ) {
                if ( temp. right == null ) {
                    temp.reight = xn
                    return root
                }
            temp = temp.right
        }
        else {
            if(temp.left == null) {
                temp.left = xn
                return root
            }

            temp = temp.left
        }
    }
}
```



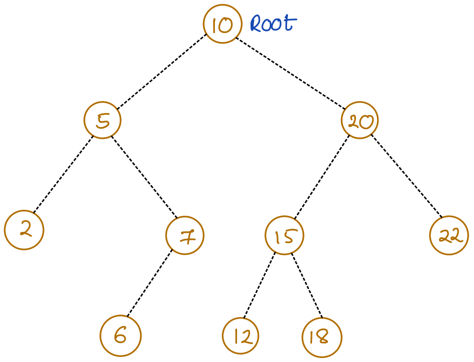
TC: O(H)   SC: O(H)

```
TreeNode  insertBST ( root ,  x  ) {
        if (root == null) return new TreeNode (x)

        if ( root.val < x ) {
                root.right = insertBST (root.right, x)
        }
        else {
                root.left = insertBST (root.left, x)
        }
        return root

        HW : Understand recursive tree codes
}
```

Smallest & Largest in a BST

Q) Find the smallest element in a BST.

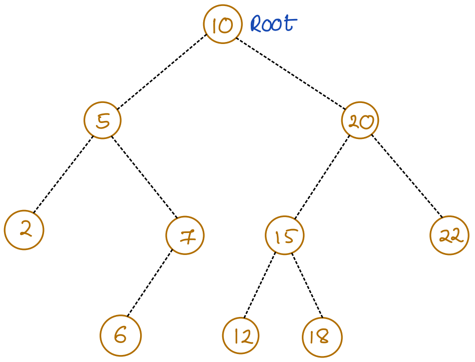
```
if (root == null) return -1
temp = root
while (temp.left != null) {
        temp = temp.left
}
return temp.data
```



TC : O(H)   SC : O(1)

Q) Find the largest element in a BST

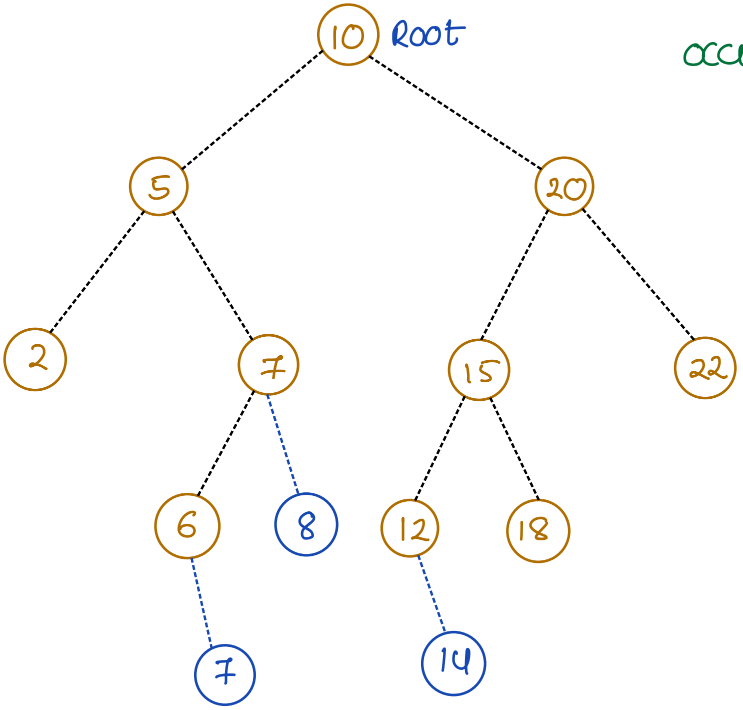
```
if (root == null) return -1
temp = root
while (temp.right != null) {
        temp = temp.right
}
return temp.data
```



TC : O(H)   SC : O(1)

Deletion in BST

In case of multiple nodes == target delete first occurrence of target

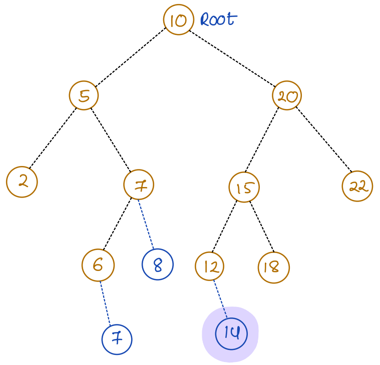


DN = Node to be deleted

Case A

DN is a leaf node

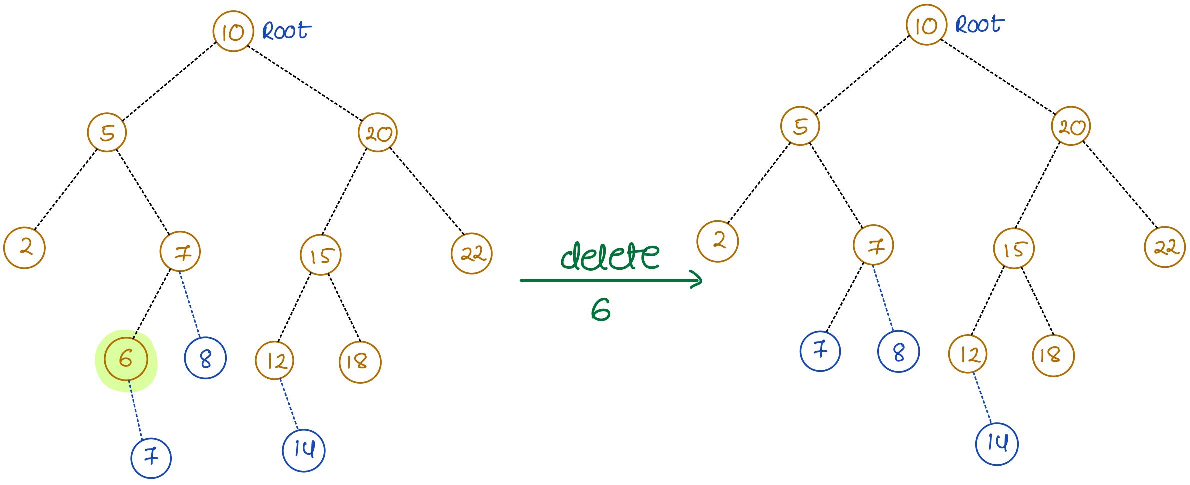
DN.left
and == null
DN.right



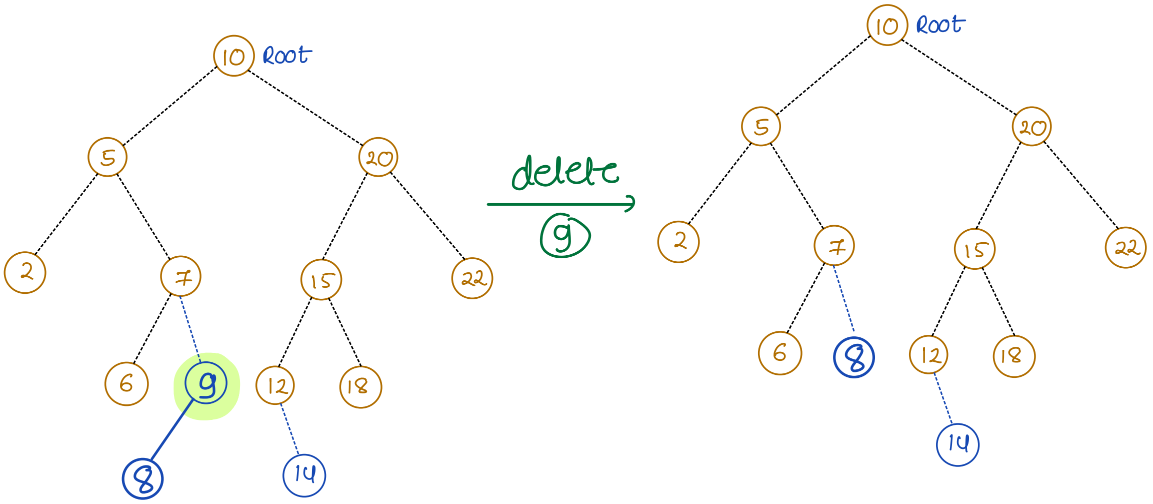
12.right = null

Case B

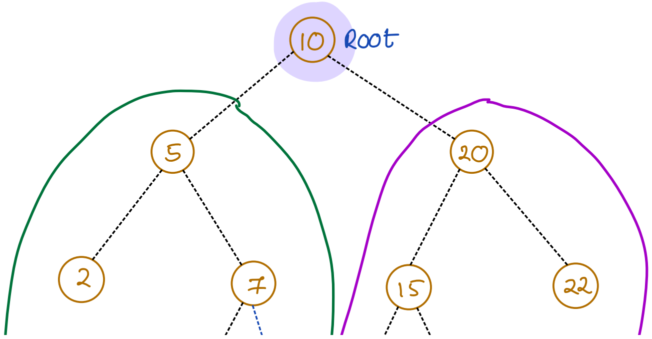
DN's left == null

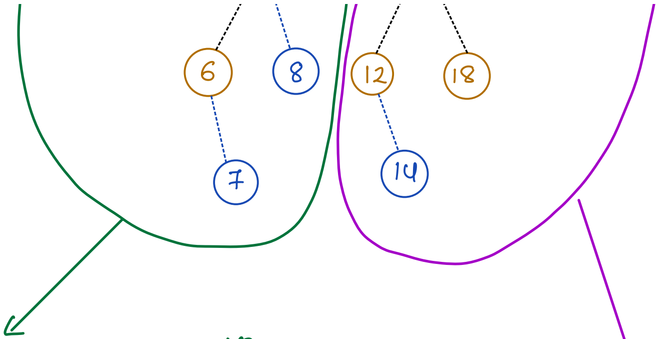


Case C DN's right == null



case D DN has both children left & right



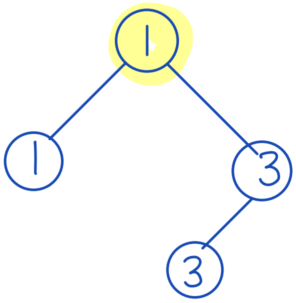


Replace 10 with max on left

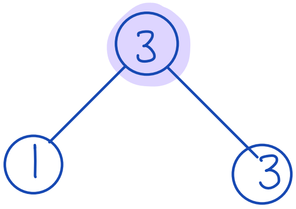
1

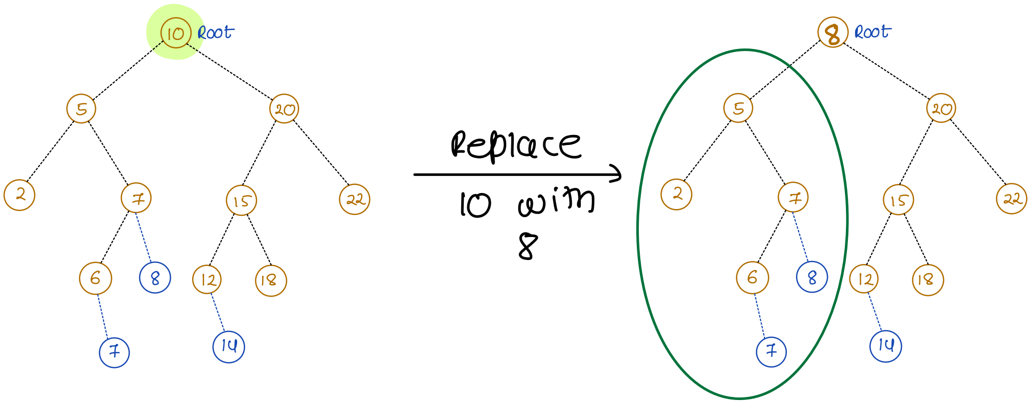
~~Replace 10 with min on right~~

2



For option 2 1 will be replaced by 3 .





Replace 10 with 8

delete 8 in the left subtree

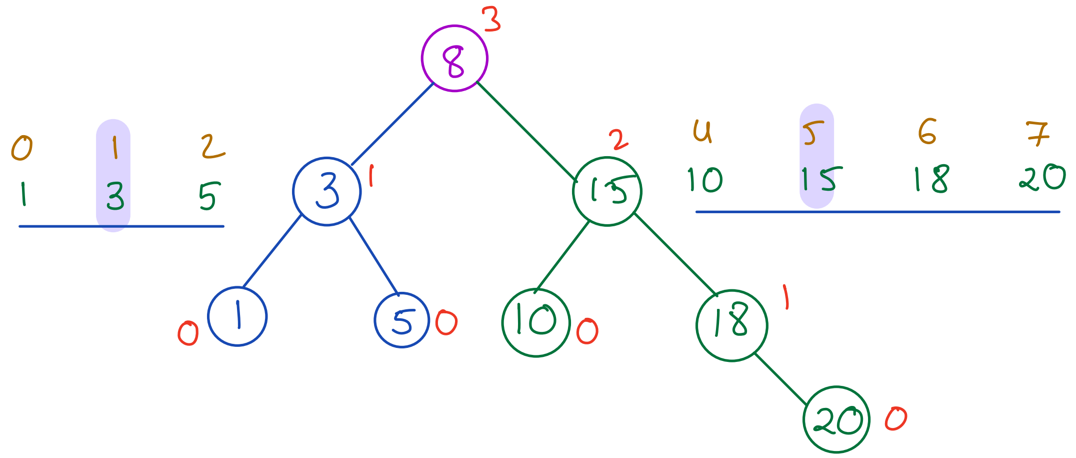
22 : 43

Contruct Balanced BST from sorted array

all unique

Height Balanced

	0	1	2	3	4	5	6	7
A =	1	3	5	8	10	15	18	20



The above tree is height balanced

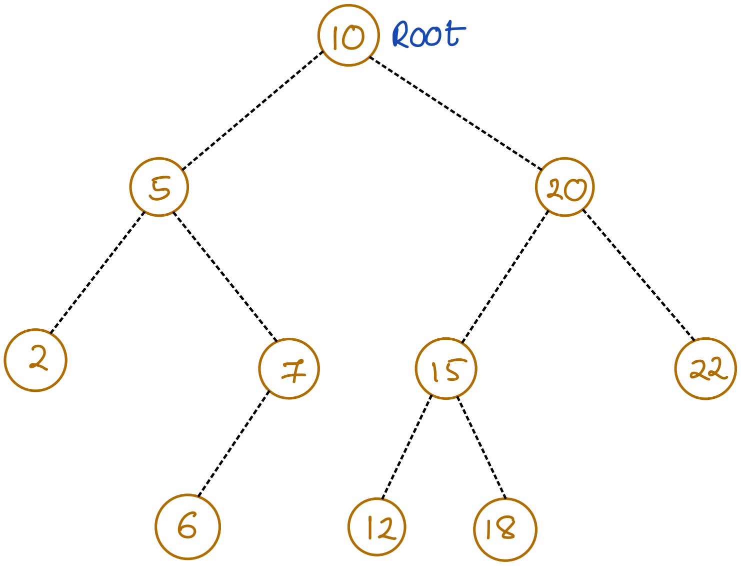
Pseudocode

```
                                  0       N-1
TreeNode  build ( A , l , r ) {
        if( l > r )  return  null
        mid  =  (l+r)/2
        root  =  new  TreeNode ( A[mid] )
        root.left  =  build ( A , l , mid-1 )
        root.right  =  build ( A , mid+1 , r )
        return  root
}
```

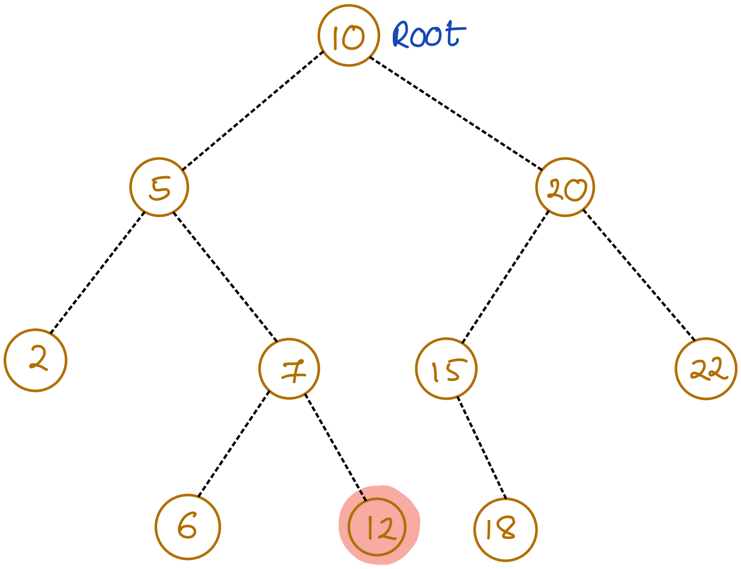
TC : O(N)

SC : O(log(N))

Check if the given binary tree is a BST

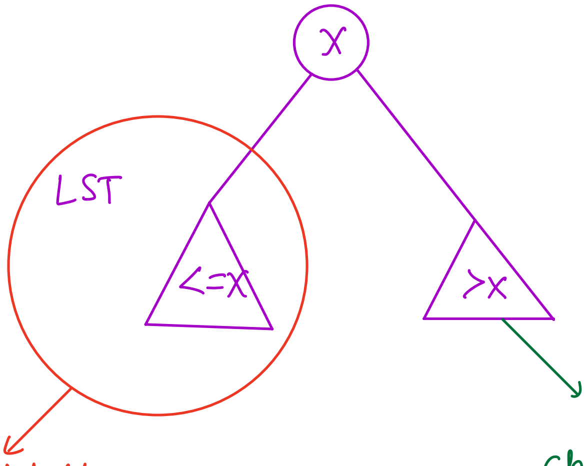


ans = true



ans = false

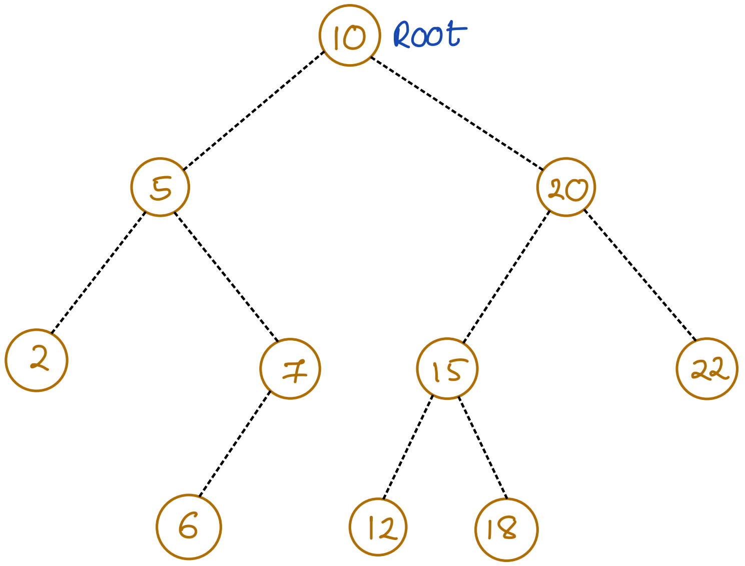
Def^n of BST



check left
check all nodes in subtree <= X

checkRight
check all nodes in subtree > X

TC : $O(N^2)$



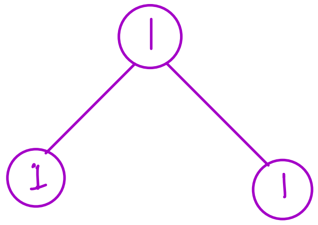
Inorder ⟶ 2 5 6 7 10 12 15 18 20 22

NOTE : Inorder of BST is always sorted

Inorder is sorted ⇒ BST

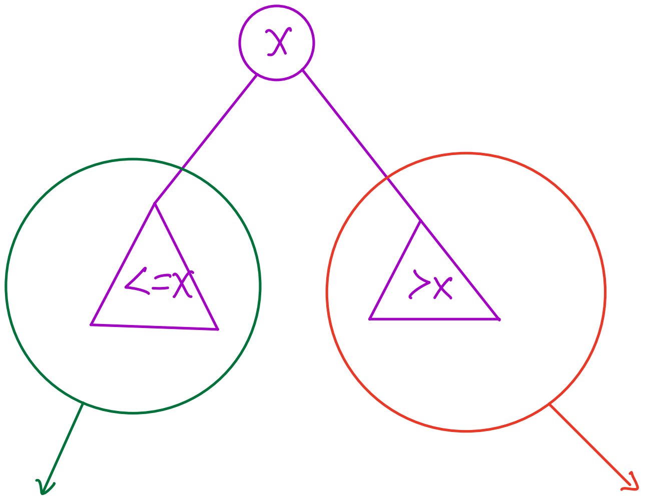
→ All distinct yes

Duplicates No



Inorder ⇒ 1 1 1

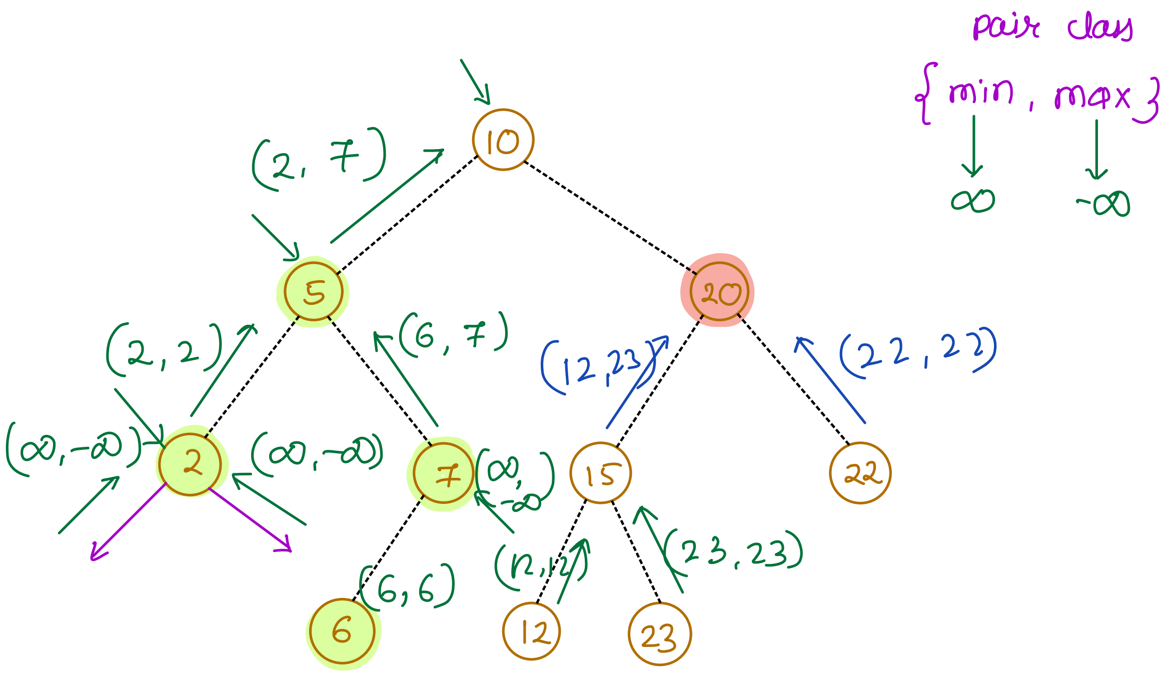
Def^n of BST



max on left <= x

min on right > x

Postorder → L R N



max on left <= x

min on right > x

Pseudocode

isBst = true　　　　TC: O(N)　　SC: O(H)

```
checkBST ( root) {
|    if (root == null)   {∞, -∞}
|    L = checkBst (root. left)
|    R = checkBst (root. right)
|
|     if ( ! ( L .max <= root.data < R.min))
|            isBST = false
```

```
    minX = min(L.min, root.data, R.min)
    maxX = max(L.max, root.data, R.max)
    return {minX, maxX}
}
```