Agenda

What is BST ?

Min / Max in BST

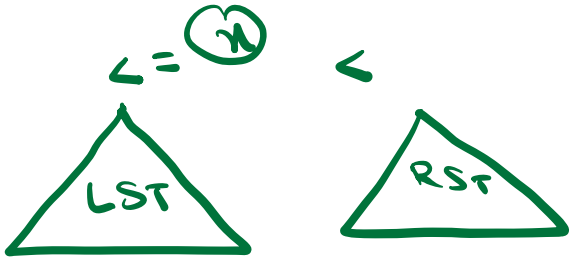
Search, Insert, Delete in BST

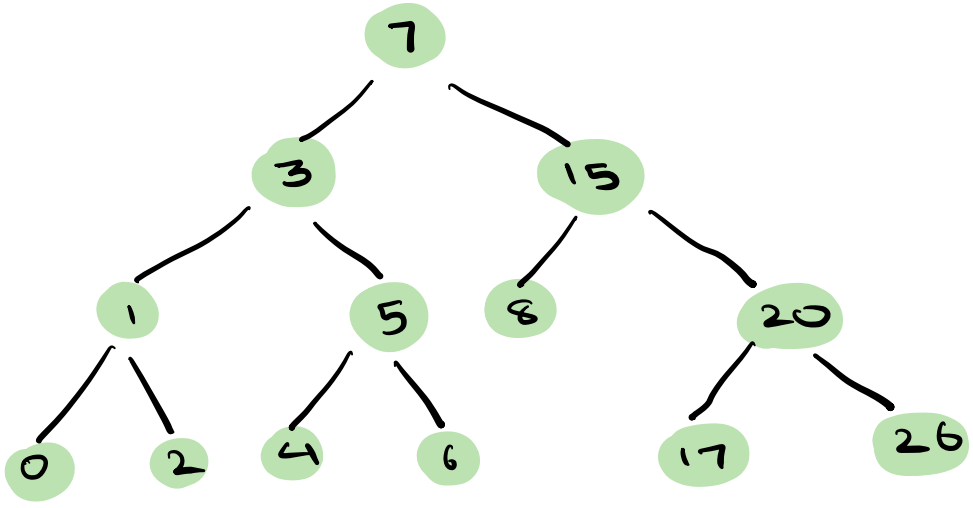
Construct BST from sorted array

Check valid BST

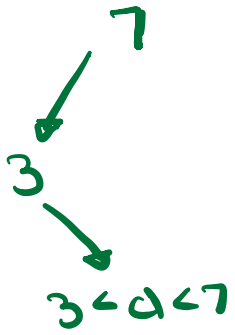
Binary Search Tree (BST)

For ∀ nodes n → all data in LST $<= n$
all data in RST $> n$





Valid BST

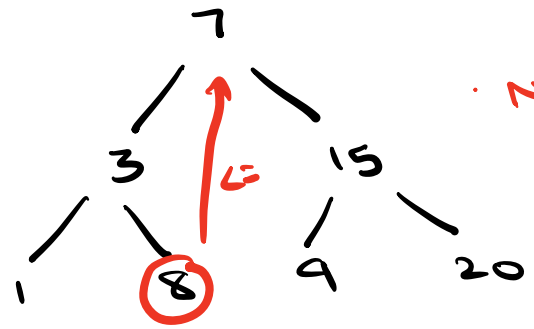


Inorder : L N R

0 1 2 3 4 5 6 7 8 15 17 20 26

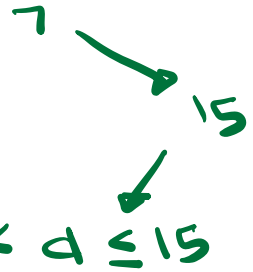
Inorder traversal of BST → sorted array

L ≤ N < R



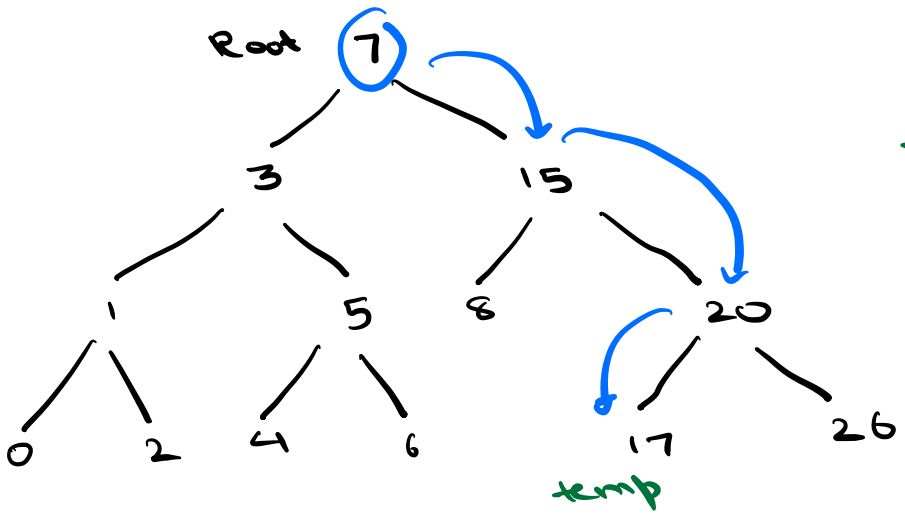
Not a valid BST

1 3 8 7



Searching in BST

We can search faster in BST using divide and conquer.



Search for 17

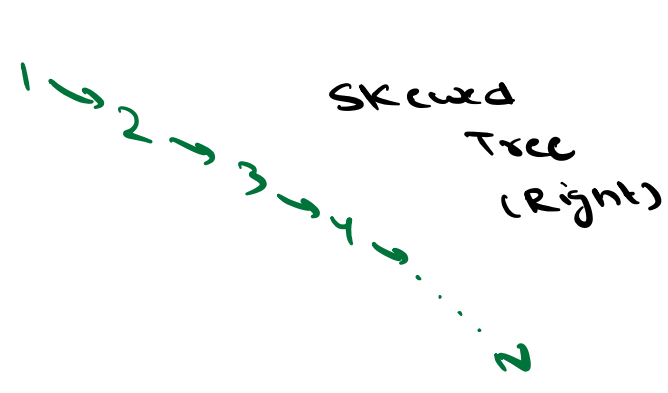
temp	temp == val
7	F 7 < 17 right
15	F 15 < 17 right
20	20 > 17 left
17	T break

```
bool search (Node root, int val) {
    Node temp = root
    while (temp != NULL) {
        if (temp.data == val)
            return true
        else if (temp.data < val)
            temp = temp.right
        else
            temp = temp.left
    }
    return false
}
```

TC : O(H)

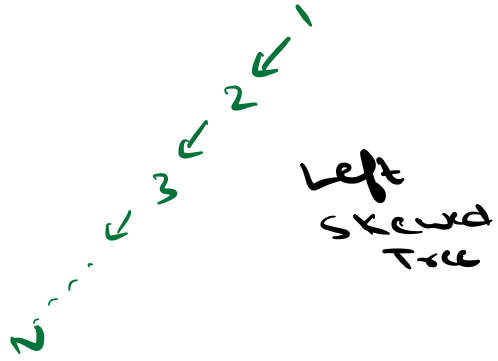
SC : O(1)

N nodes

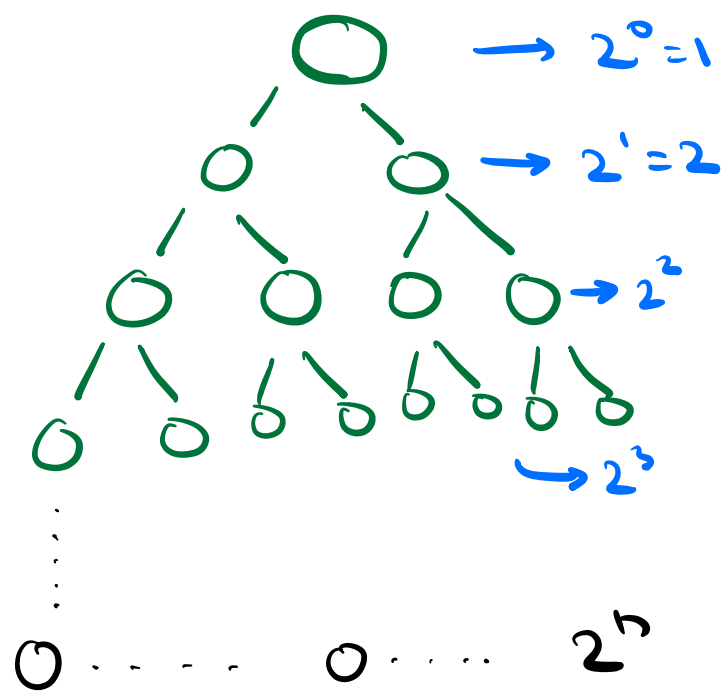


Skewed Tree (Right)

H = N



Left Skewed Tree



$2^0 + 2^1 + 2^2 + \ldots + 2^h = N$

$2^0 + 2^1 + 2^2 + \ldots + 2^h = N$

$a = 2^0 = 1 \quad r = 2$

$\frac{1(2^{h+1} - 1)}{2 - 1} = N$

$S = \frac{a(r^N - 1)}{r - 1}$

$2^{h+1} - 1 = N$

$\log_a a^x = x$

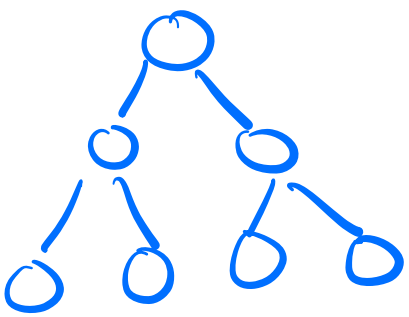
$\Rightarrow 2^{h+1} = N + 1$

Take $\log_2$ on both sides

$\Rightarrow \log_2 (2^{h+1}) = \log_2 (N+1)$

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

$h \simeq \log_2 N$



7

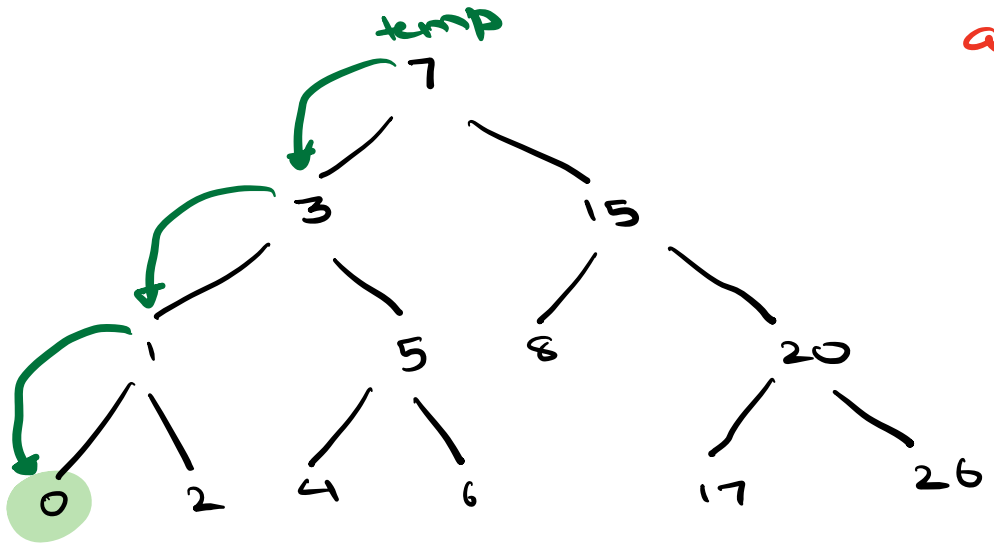
$\log_2 7 \simeq 3$

$\log N \leq h \leq N$

Balanced Tree

Skewed Tree

Find smallest node / minimum in BST



ans → 0

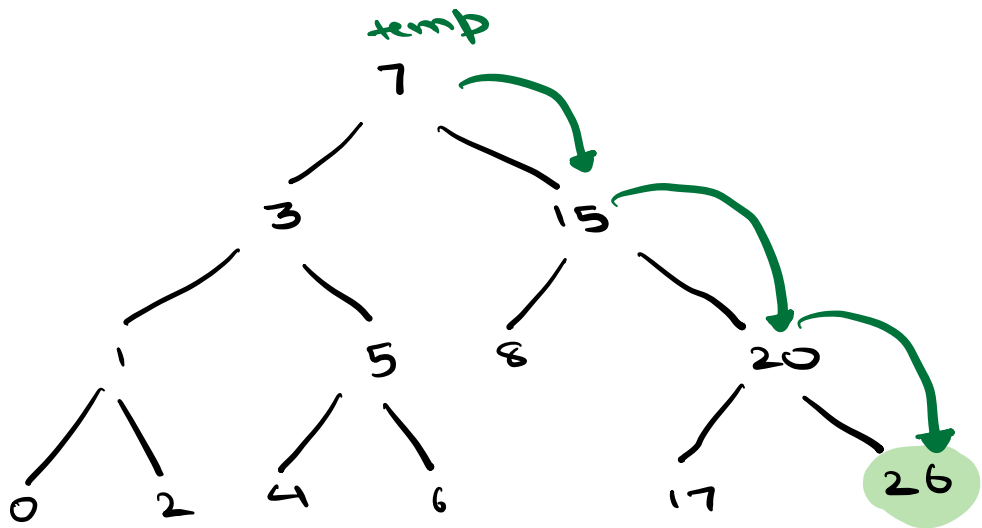
Left most node in tree is smallest

```
if (root == NULL) return NULL
Node temp = root
while (temp.left != NULL) {
    temp = temp.left
}
return temp
```

TC: O(H)

SC: O(1)

Find largest node / maximum in BST

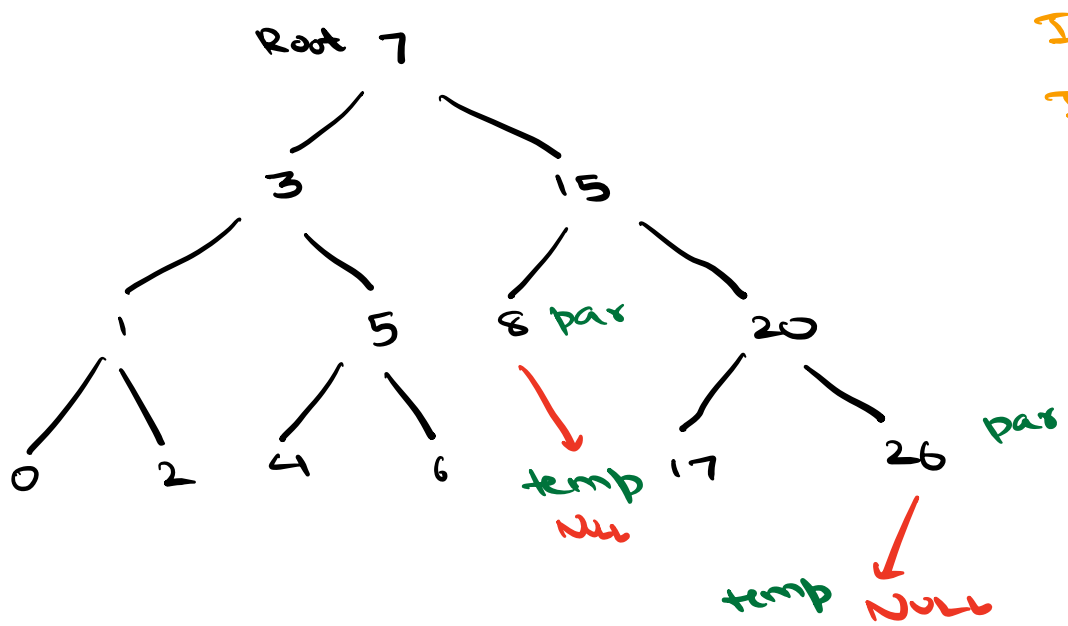


Rightmost node in BST ↓ Max

TC: O(H)

SC: O(1)

Insert a node in BST



Insert 22

Insert 10

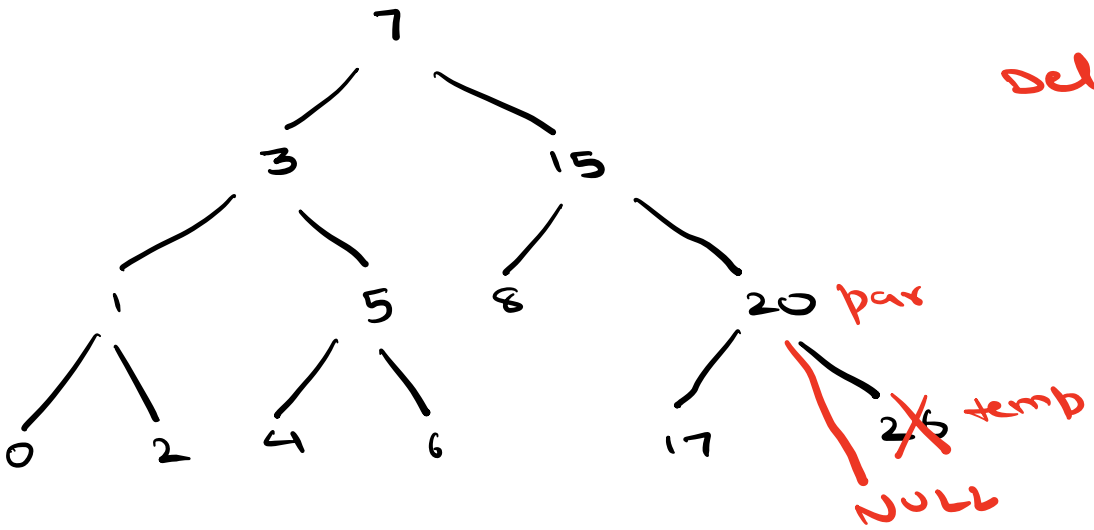
```
// Insert node with data x
Node nn = new Node(x)
if (root == NULL) return nn

Node temp = root
Node par = NULL
while (temp != NULL) {
    par = temp
    if (temp.data < x)
        temp = temp.right
    else
        temp = temp.left
}
if (x <= par.data)
    par.left = nn
else
    par.right = nn
```

TC: O(H)

SC: O(1)

Delete a node in BST

Case 1 : Node with no children (Leaf Node)

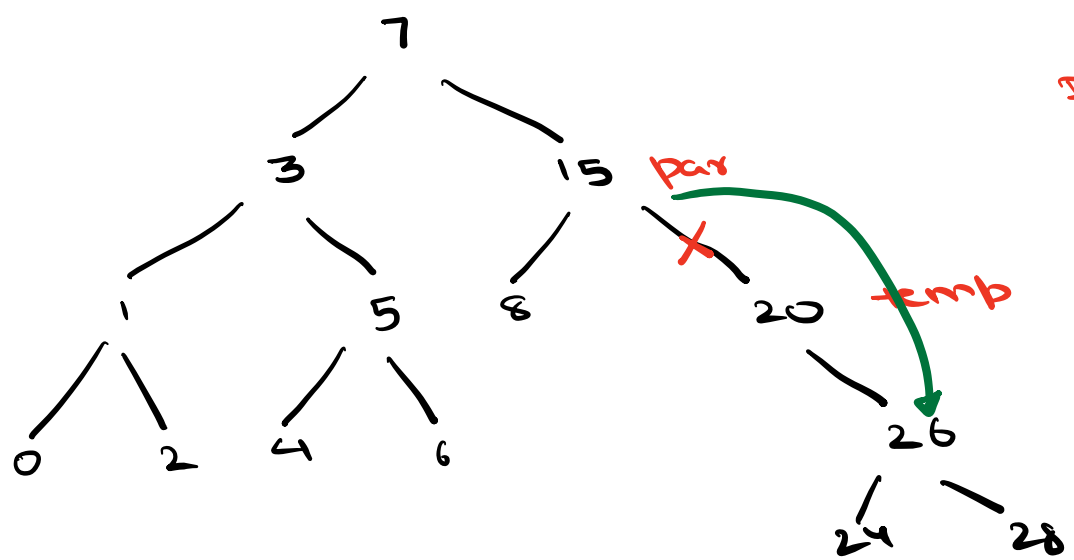


Delete 26

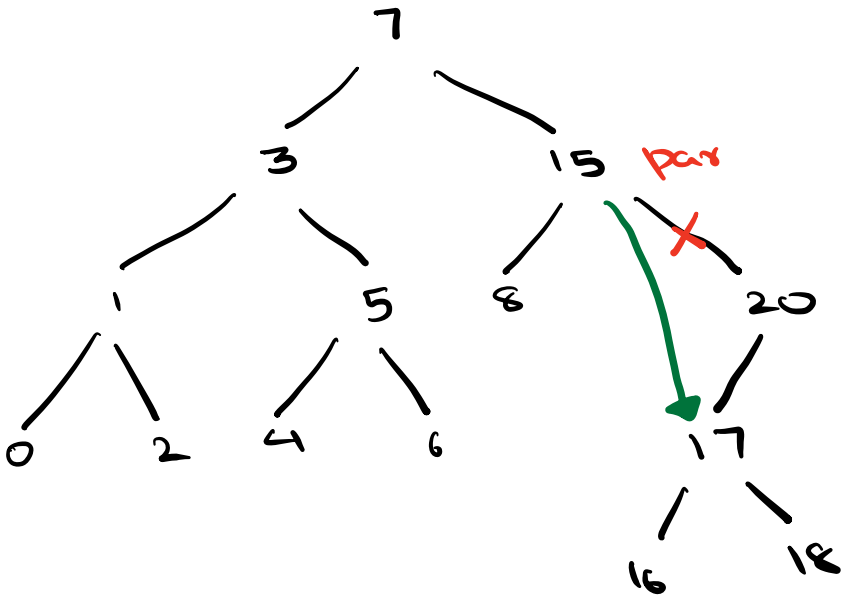
Search for 26, while searching maintain par

```
if (par.left == temp)
    par.left = NULL
else if (par.right == temp)
    par.right = NULL
```

Case 2 : Node with 1 child

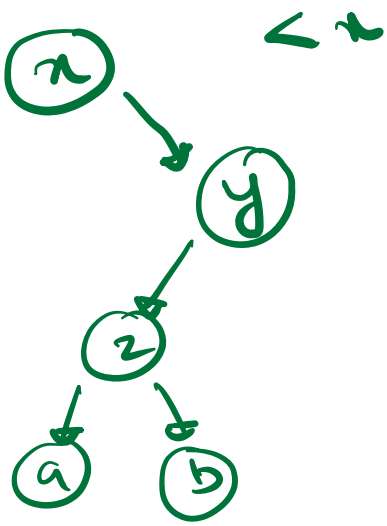


Delete 20



// Node del → temp

if (temp.left != null)
child = temp.left
else
child = temp.right



x < z

if (par.left == temp)
par.left = child

else
par.right = child

cnt = 0 temp

if temp.left != NULL

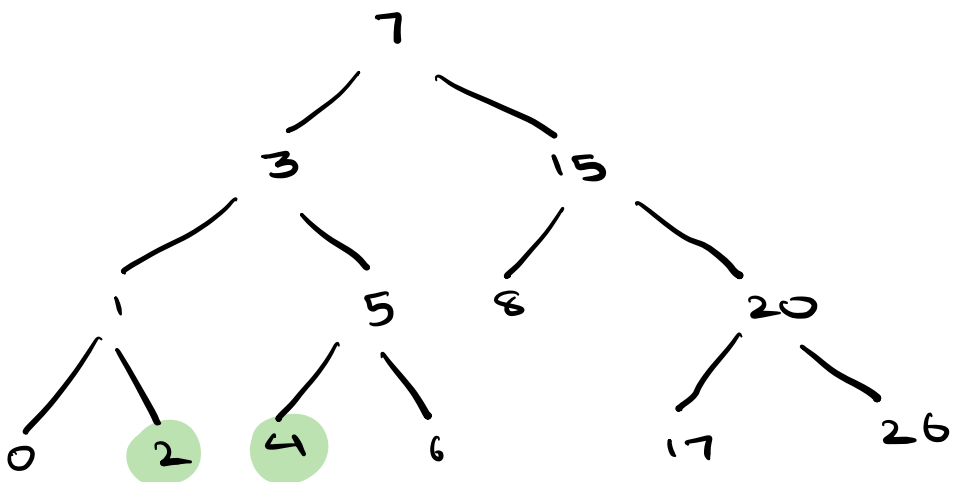
cnt ++

if temp.right != NULL

cnt ++

cnt=0 cnt=1 cnt = 2

Case 3 : Node with 2 children



Delete 3

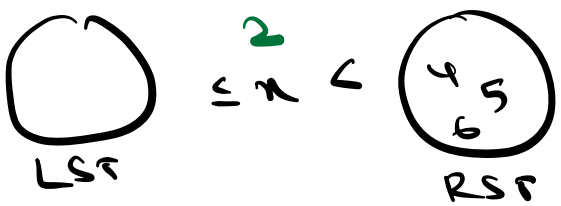
↓

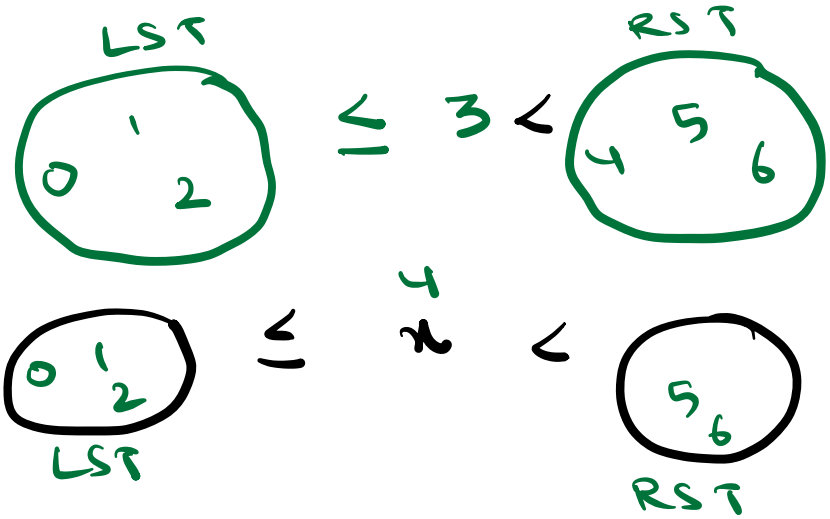
Replaced

a) with 4

or

b) with 2



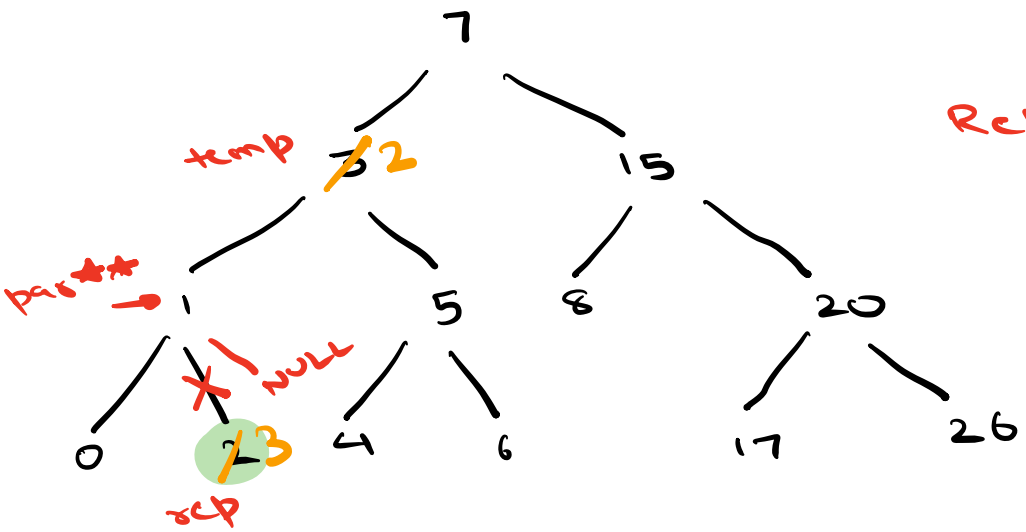


1. Replace x with min node in RST

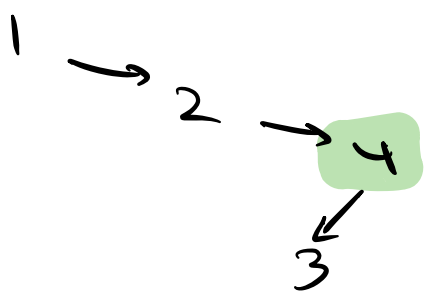
leftmost node in RST

2. Replace x with max node in LST

rightmost node in LST



Replace 3 with 2



Max in BST → Right most node

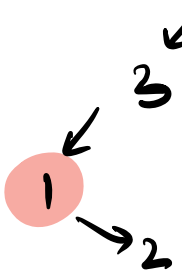
↓

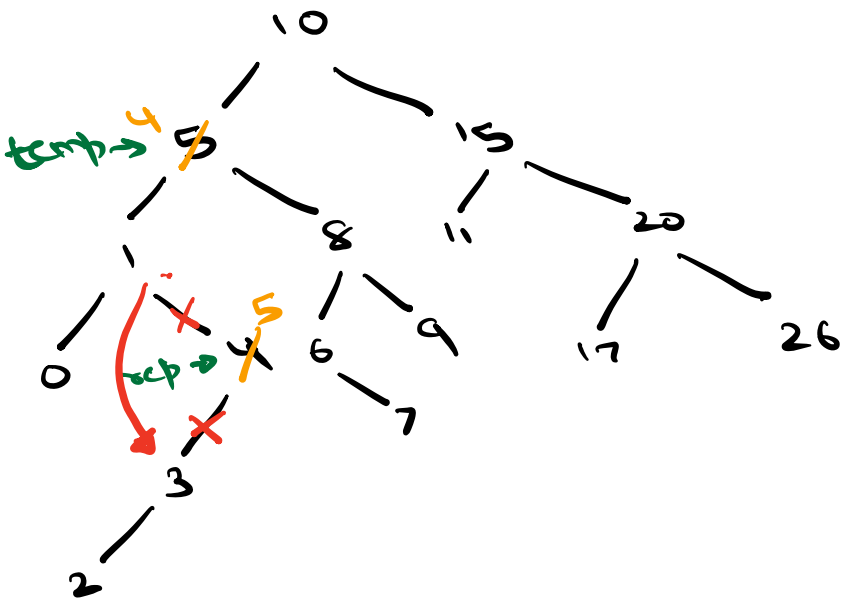
can have left child

Min in BST → Left most node

↓

4 can have right child





Delete 5

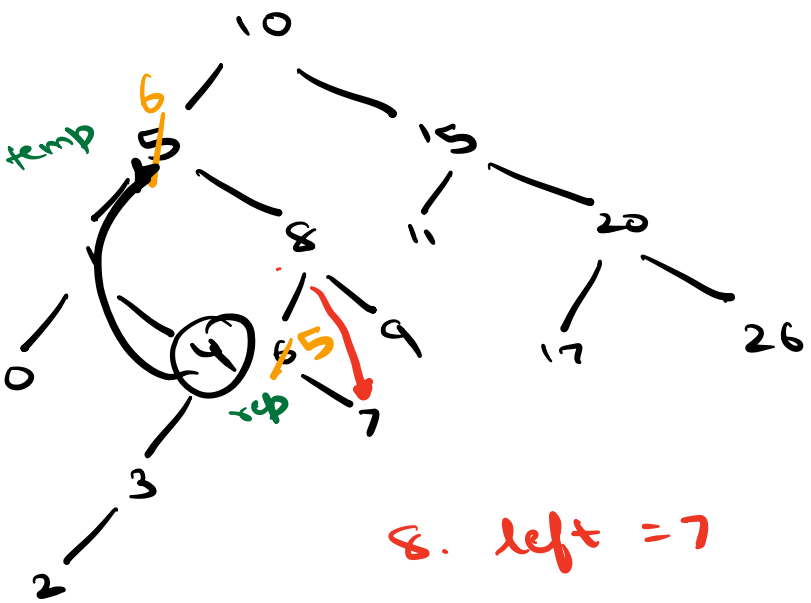
a) LST → Max (Rightmost)

4

b) RST → Min (Leftmost)

6

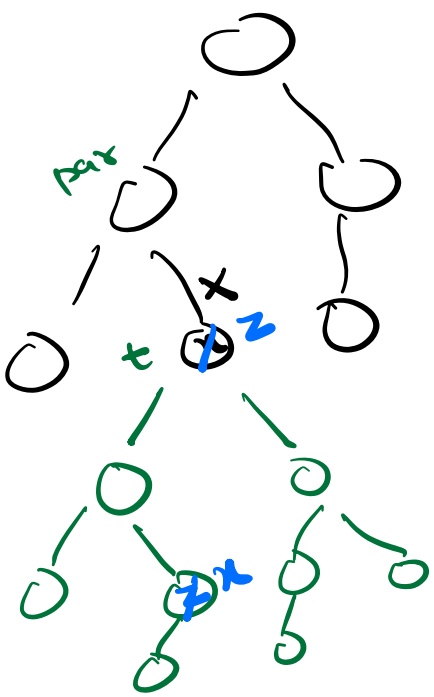
1. right = 3



8. left = 7

Replace temp with LST max

```
// temp, par
Node rep = temp.left
par = temp

while (rep.right != NULL) {
    par = rep
    rep = rep.right
}

int x = rep.data
swap (rep.data, temp.data)
rep.data = temp.data
temp.data = x

// delete rep Node
if (rep.left)
    par.right = rep.left
else
    par.right = NULL
```



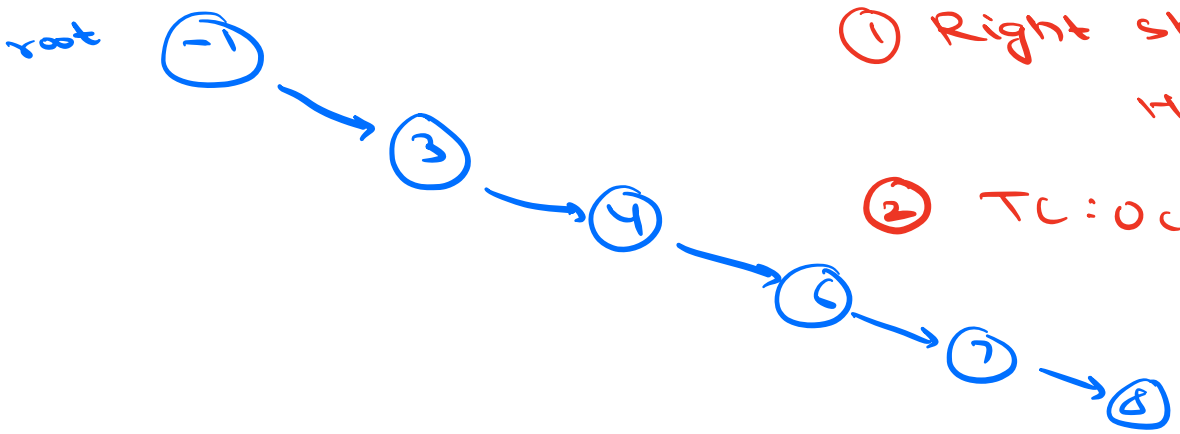
TC : O(H) → $\log_2 n \to n$

SC : O(1)

Construct BST from sorted array of unique elements

A[]: -1 3 4 6 7 8 10 13 14

Approach 1: Take element one by one, insert into BST



① Right skewed tree

H = N

② TC: O(N²)

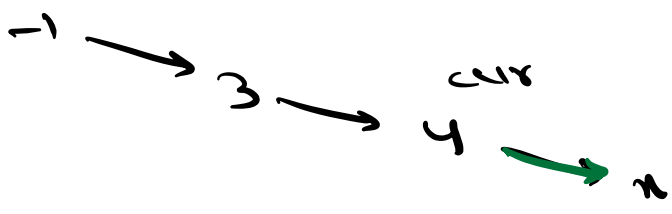
TC of 1 insertion → O(H)

N insertions → O(N*H)

N^2

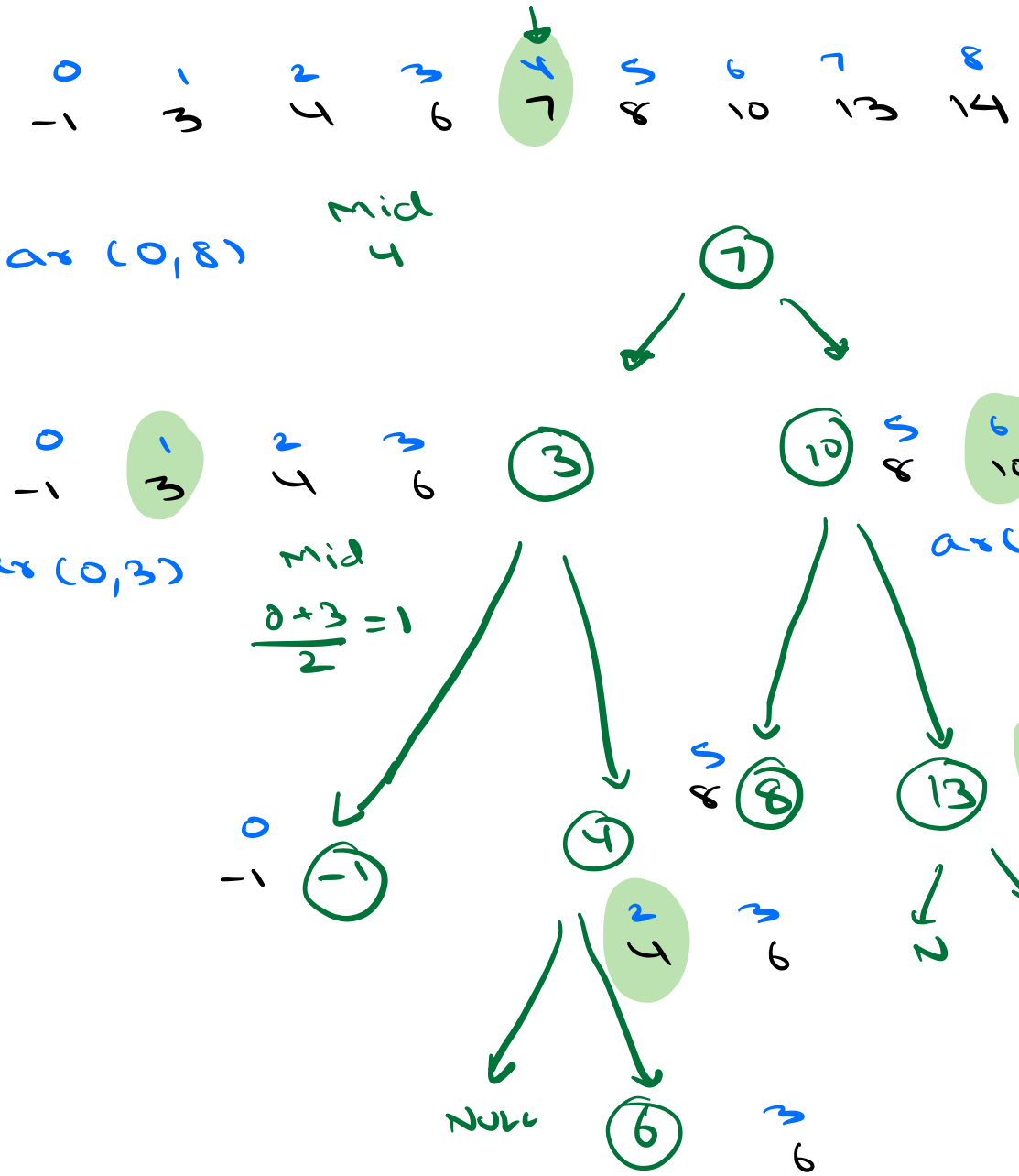
Approach 2: Maintain rightmost node

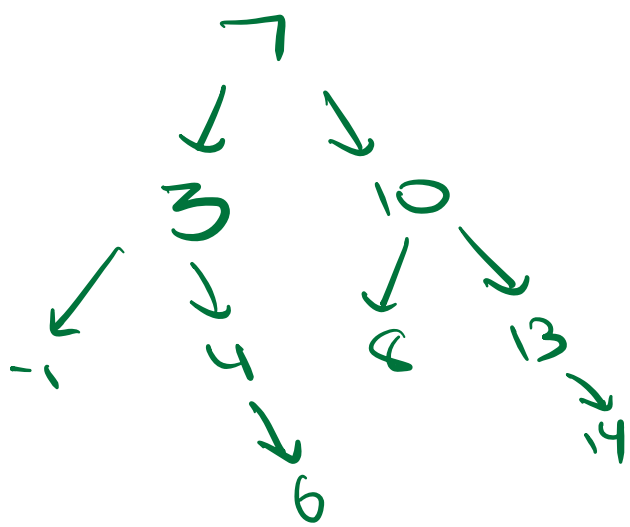
Insert x → rightmost node's right = x



TC: O(N)

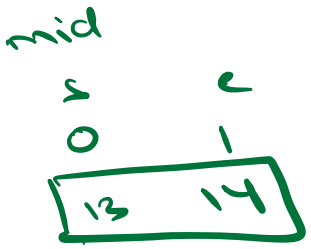
Approach 3 : Balanced Tree





// Given ar from s->e, return root

```
Node constructTree(ar, 0, N-1)        TC: O(N)
    if (s > e) return NULL            SC: O(log2 N)

    int mid = (s+e)/2   //   s + (e-s)/2

    Node root = new Node(ar[mid])
    root.left = constructTree(ar, s, mid-1)
    root.right = constructTree(ar, mid+1, e)
    return root
```



mid

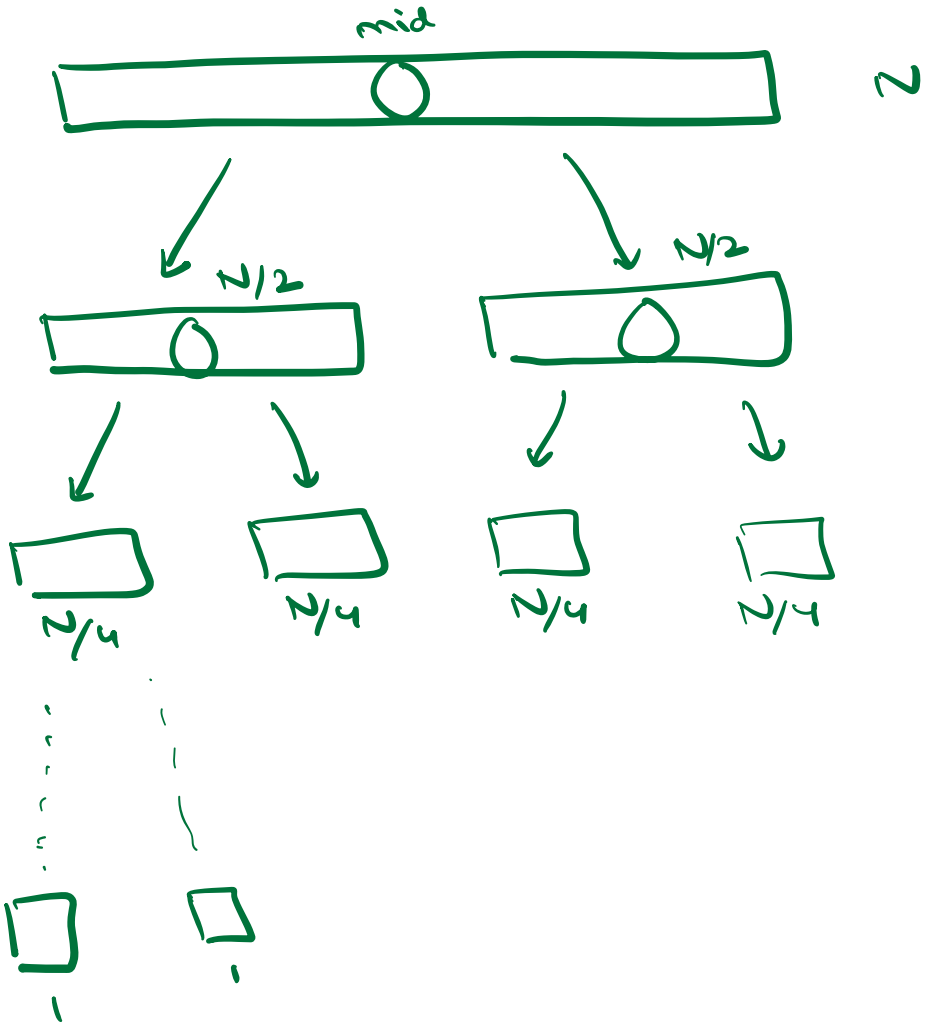
$\frac{0+1}{2} = 1$

(13)

(0,-1) NULL/LST

s → mid-1

0 → -1

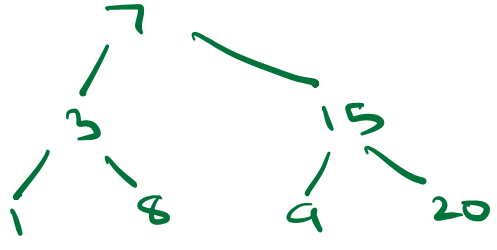


Q. Check if binary tree is BST

1. $\forall$ node x, if $x.data \geq x.left.data$

&&

$x.data < x.right.data$



Not a valid BST

NOT correct check we are not check subtree

$\forall$ nodes x

① All data in LST $\leq x$

② All data in RST $> x$

Approach 1: Do inorder traversal → array

↓

check if it is sorted

TC: $O(N+N)$

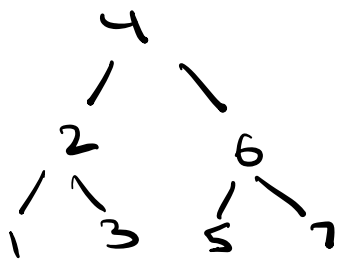
SC: $O(H+N)$

↓ fn call stack

→ storing array

inorder traversal

Binary Tree



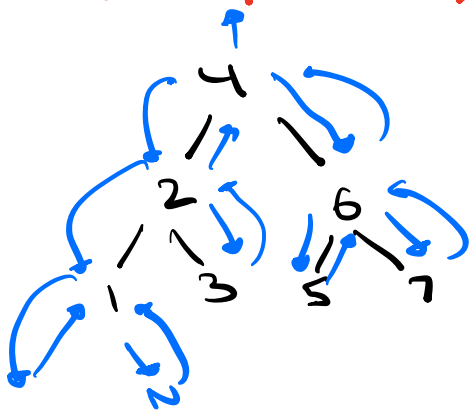
A: 1 2 3 4 5 6 7

prev $\leq$ cur

Approach 2: Maintain prev node in inorder traversal

prev = -∞ 1 2 3 4 5 6 7

isBST = true



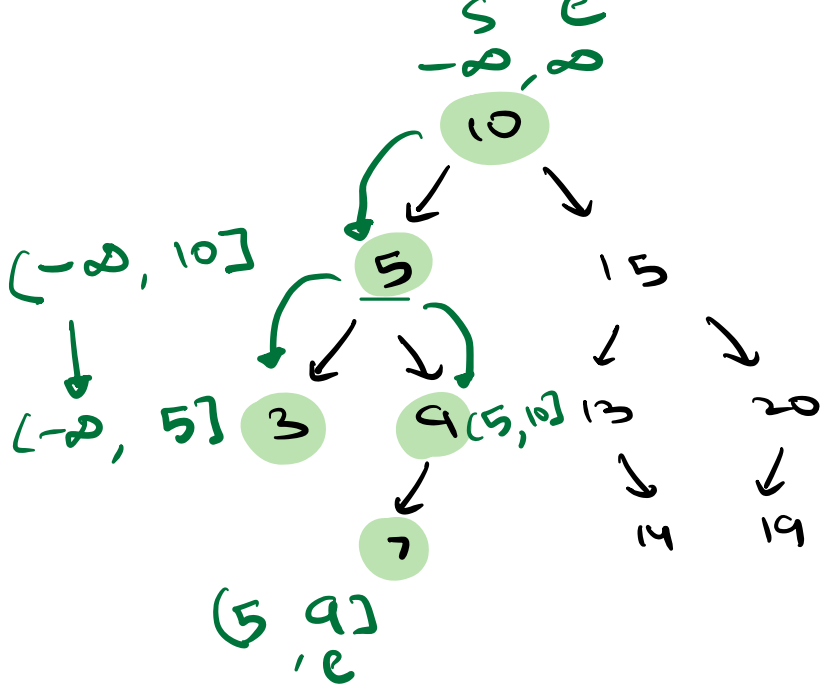

```
int prev = INT_MIN
isBST = true

void inorder (Node root) {
    if (root == NULL)
        return
    inorder (root.left)
    if (prev > root.data) {
        isBST = false
        return
    }
    prev = root.data
    inorder (root.right)
}
```

TC: O(N)

SC: O(H)

↓

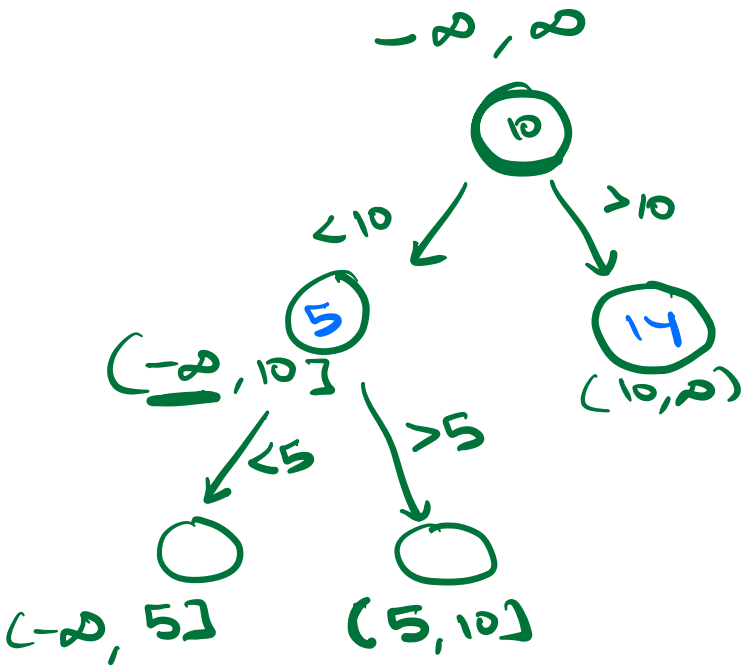
in N → N

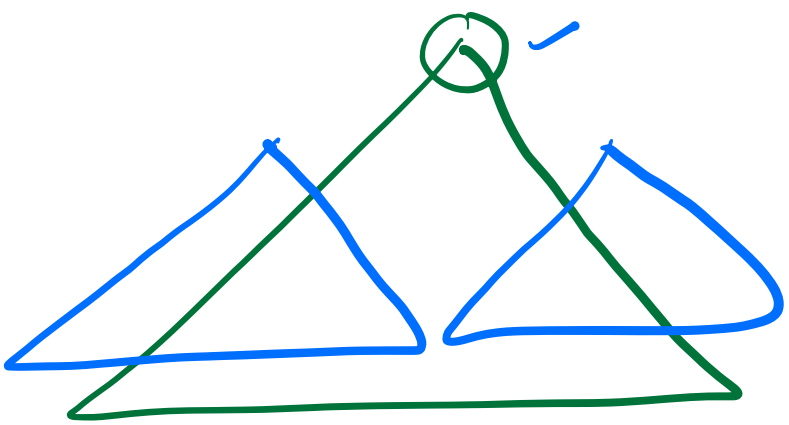


Preorder

NLR

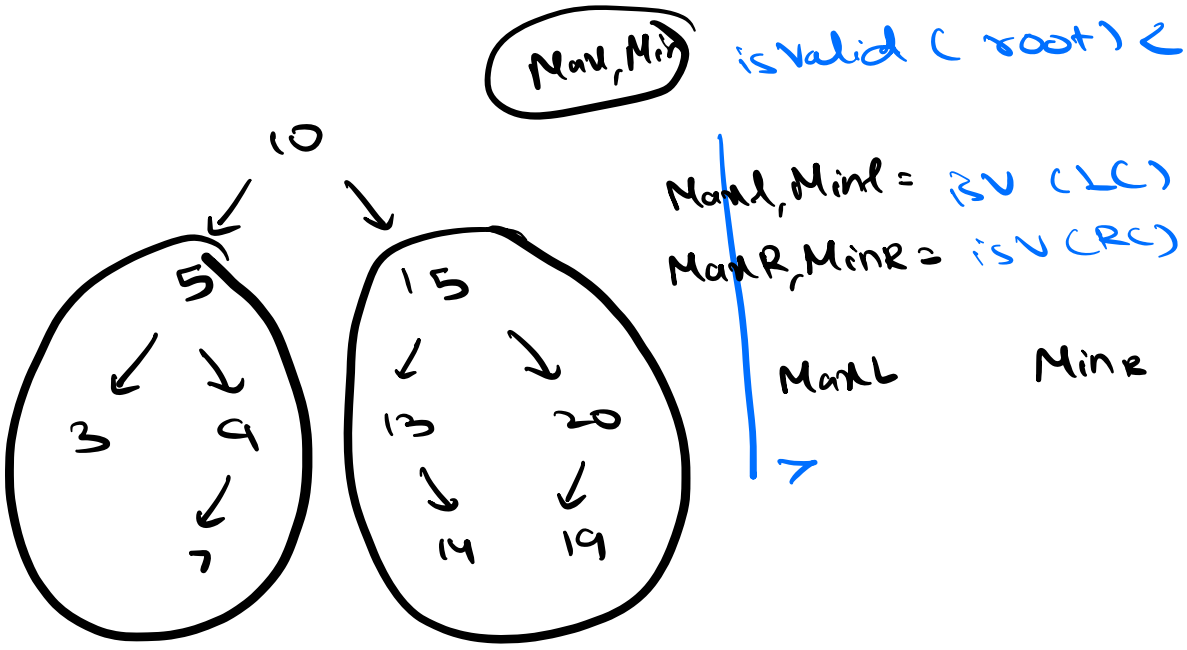
Left child





LRN

LST $\leq$ root $<$ RST



Max in LST

$\downarrow$

$9 \leq 10$ ✓

Min in RST $\rightarrow 13$

$10 < 13$