Agenda

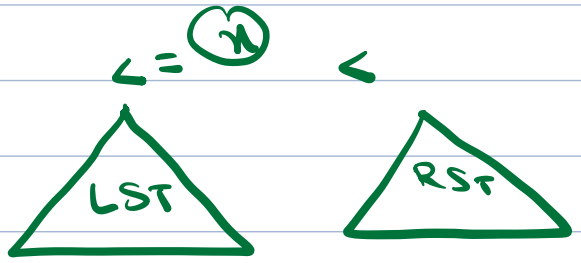
What is BST ?
Min / Max in BST
Search, Insert, Delete in BST
Construct BST from sorted array
Check valid BST

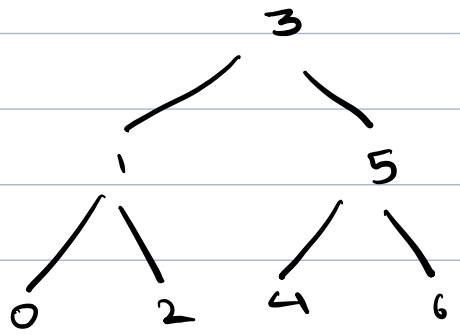
Contest 17 Jan 9 - 10:30 PM

Searching,
Linked List, Stacks and Queue, Trees

# Binary Search Tree (BST)

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




Valid BST

Inorder : L N R

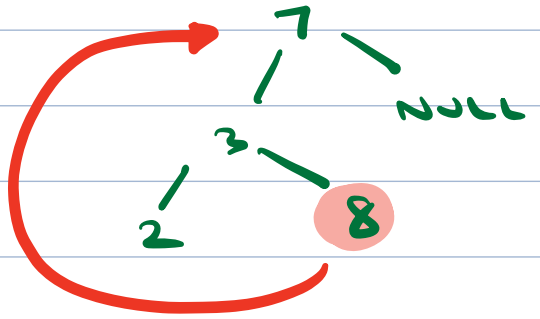
0 1 2 3 4 5 6 → Sorted

Inorder traversal of BST → sorted array

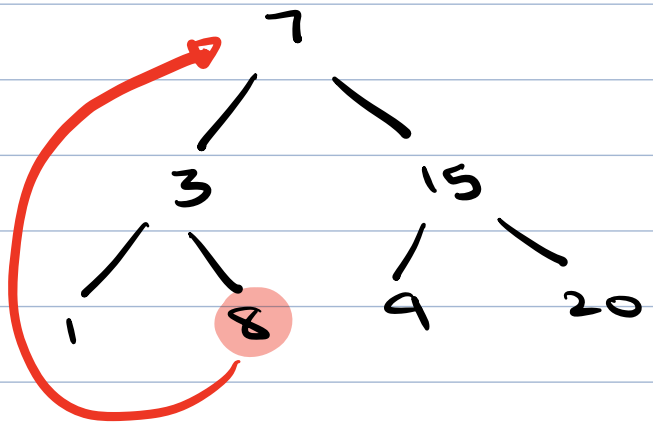
$L \leq N < R$

---

BST




Not valid BST

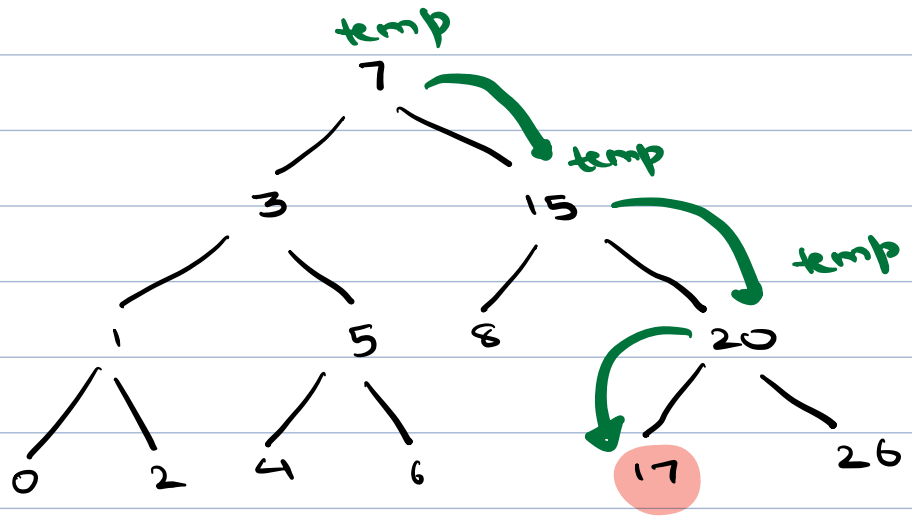



Not valid BST

---

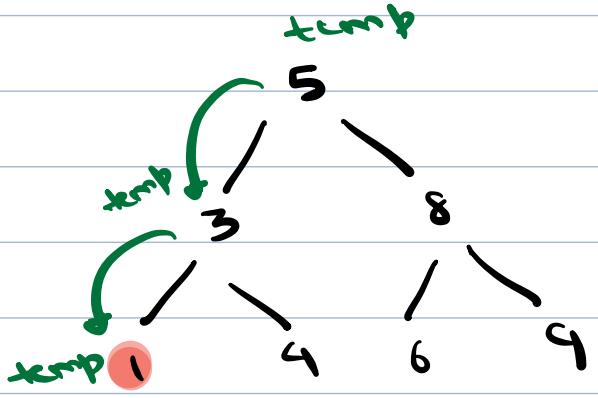
## Searching in BST

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 | F |
| | 20 > 17 |
| | Left |
| 17 | True |




Search for 1

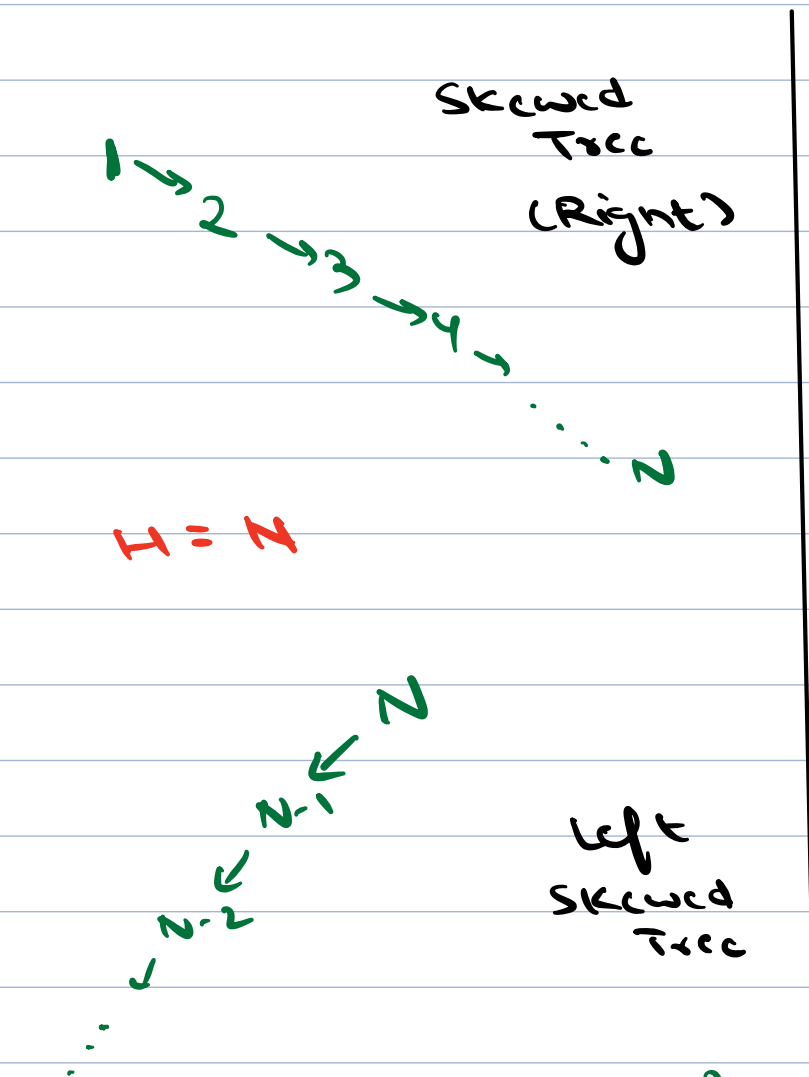
ans = true

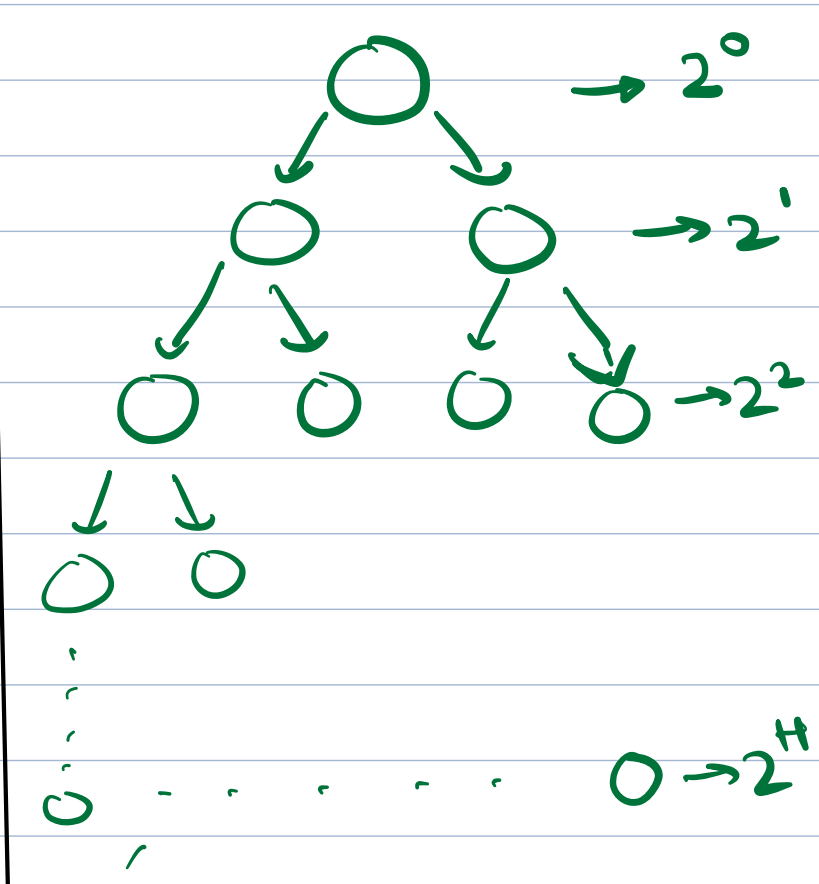
Nodes visited = 3

```
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.data > val
            temp = temp.left
        }
    }
    return false
}
```

TC: O(H)

SC: O(1)




$1 \quad 2^0 + 2^1 + 2^2 + \ldots + 2^H = N$

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

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

$\Rightarrow 2^{H+1} - 1 = N$

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

Take $\log_2$ on both sides

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

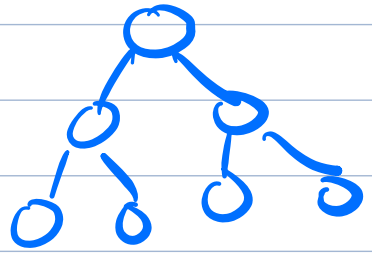
$\Rightarrow H + 1 = \log_2(N+1)$

$H \simeq \log_2 N$

$N = 7$

$H = \log_2 7$

$= 2.\_\_$

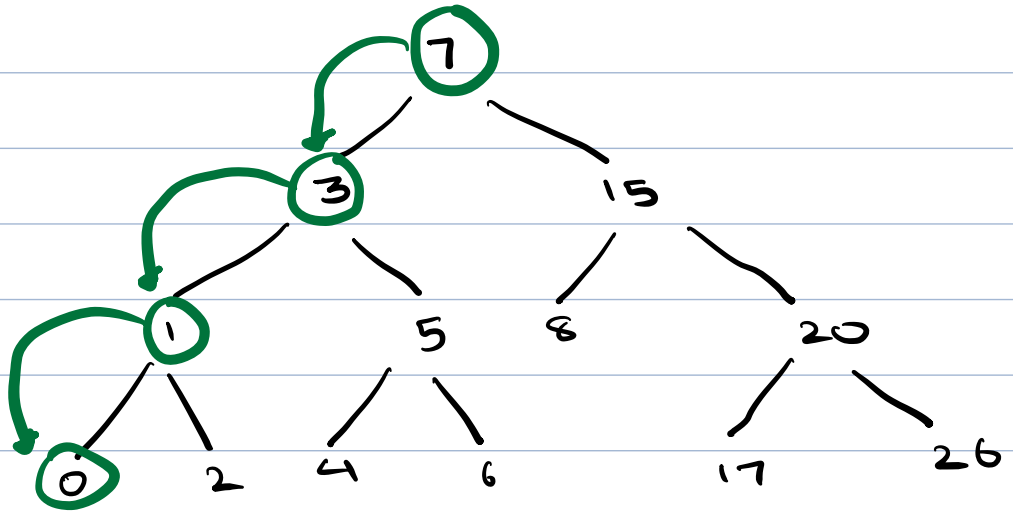


$\log_2 N \leq H \leq N$

Balanced Tree

Skewed Tree

## Find smallest node / minimum in BST




ans = 0

Left most node in tree is smallest

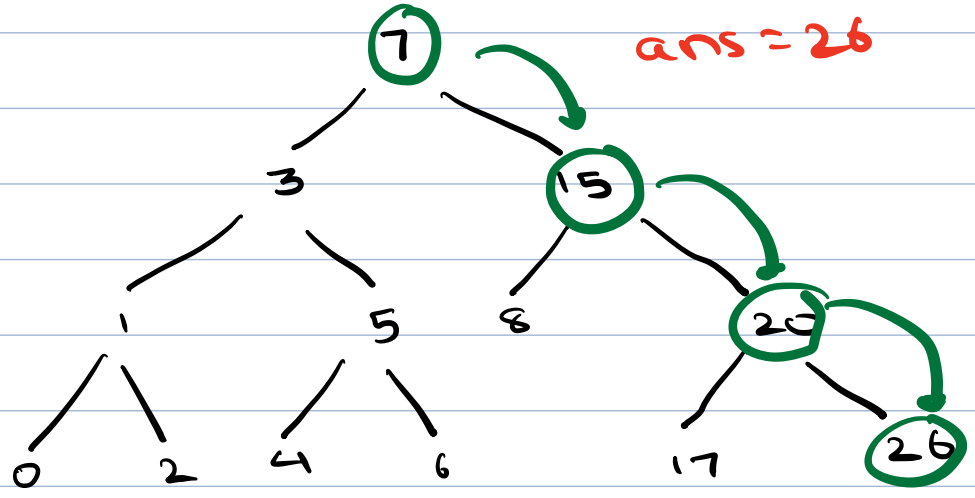
```
if (root == NULL) return NULL
Node temp = root
while (temp.left != NULL) {
    temp = temp.left
}
return temp
```

TC: O(H)

SC: O(1)

---

## Find largest node / maximum in BST




ans = 26

Rightmost node in BST → Max

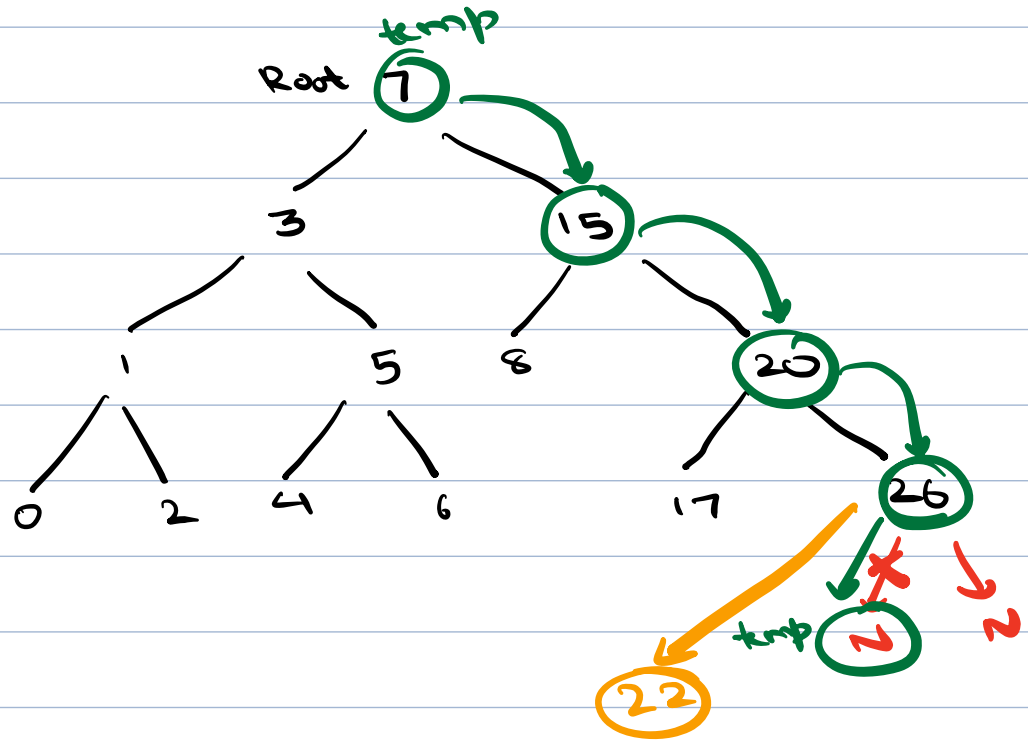
```
if (root == NULL)  return NULL
Node temp = root
while (temp.right != Null) {
    temp = temp.right
}
return temp
```

TC : O(H)

SC : O(1)

https://notability.com/n/1gMArGn7x6BuWr8L8XMd8W

# Insert a node in BST

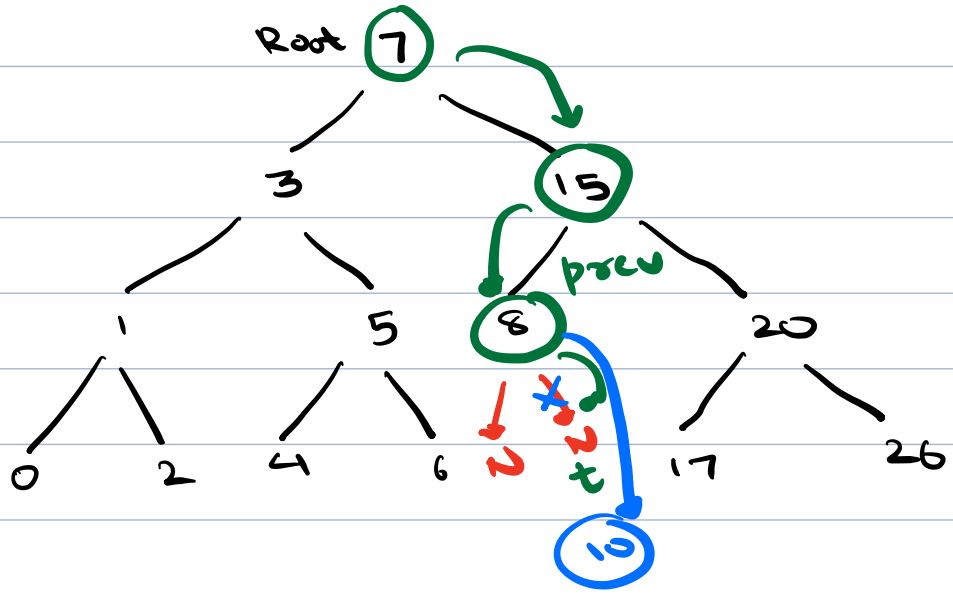



Insert 22

26.left = 22

prev.left/right = newNode

Insert 10



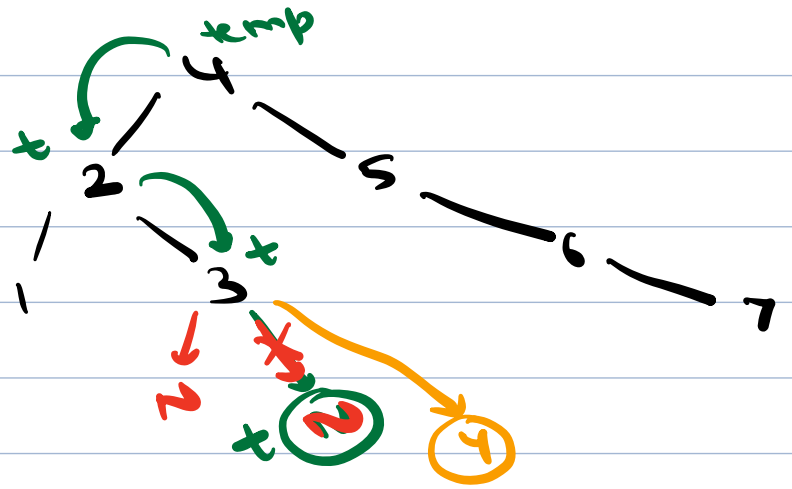

| temp | prev |
|---|---|
| 7 | NULL |
| 15 | 7 |
| 8 | 15 |
| NULL | 8 |

prev < newNode

8 < 10

prev.right = newNode

Insert 4




```
// Insert node with data x
Node nn = new Node(x)

if (root == NULL) return nn

Node temp = root, prev = NULL
while (temp != NULL) {
    prev = temp
    if (temp.data < x) {
        temp = temp.right
    }
    else { // temp.data ≥ x
        temp = temp.left
    }
}

if (x ≤ prev.data) {
    prev.left = nn
}
else {
    prev.right = nn
}

return root
```

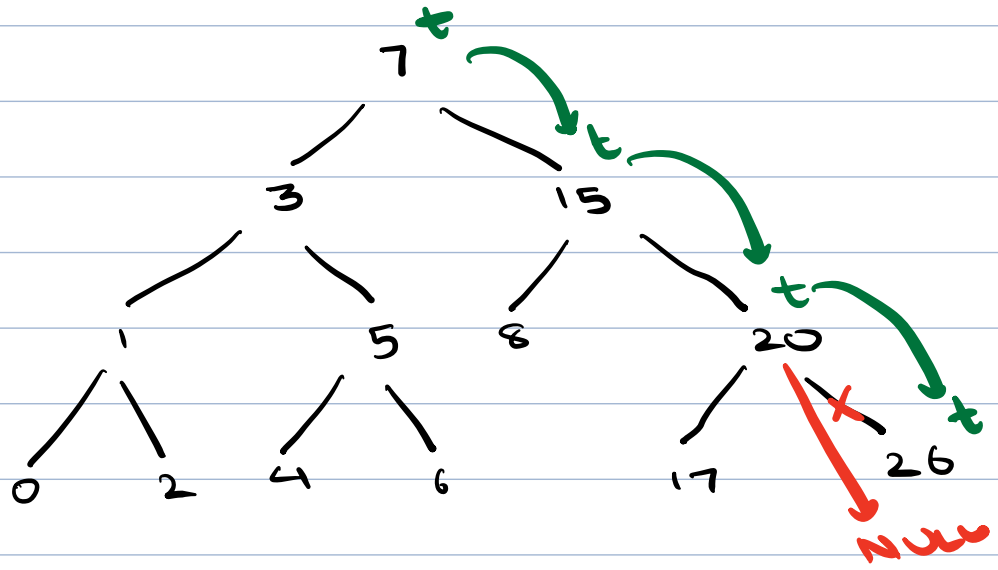
TC : O(H)

SC : O(1)

10:24

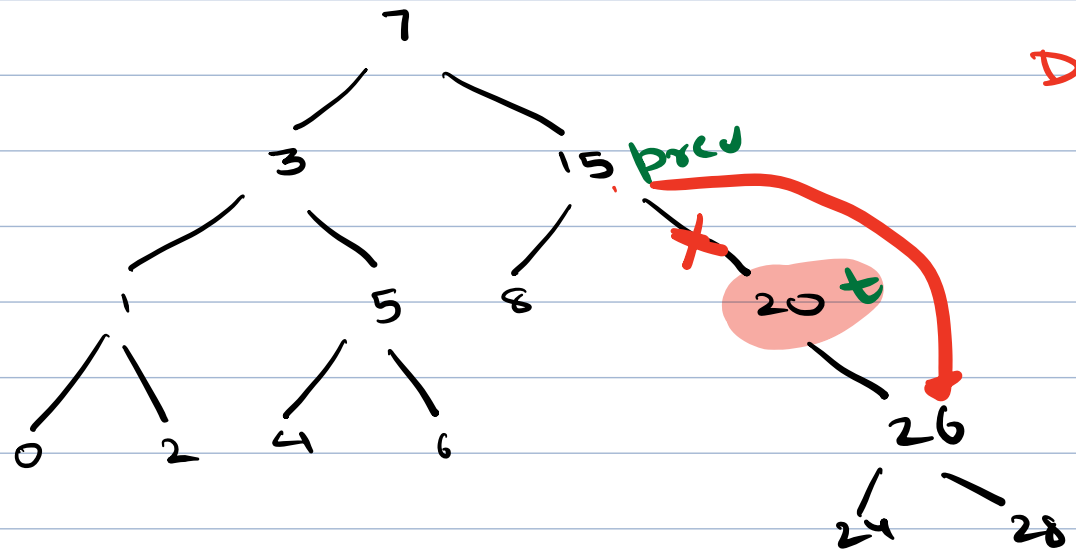
# Delete a node in BST

## Case 1 : Node with no children (Leaf Node)

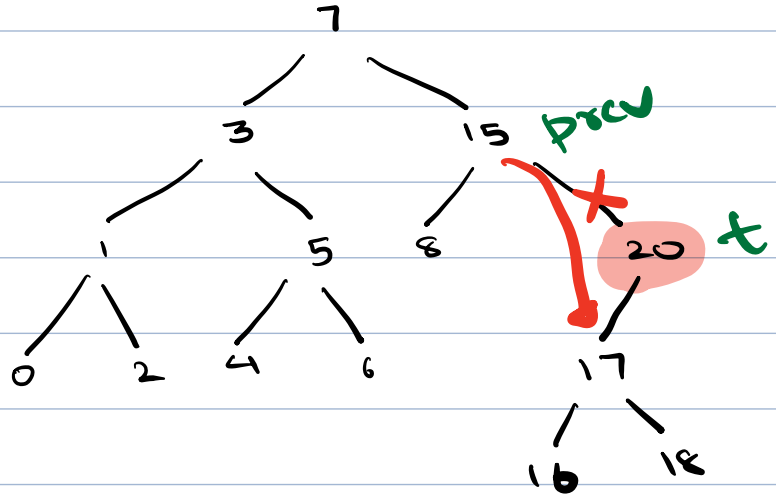



Delete 26

## Case 2 : Node with 1 child

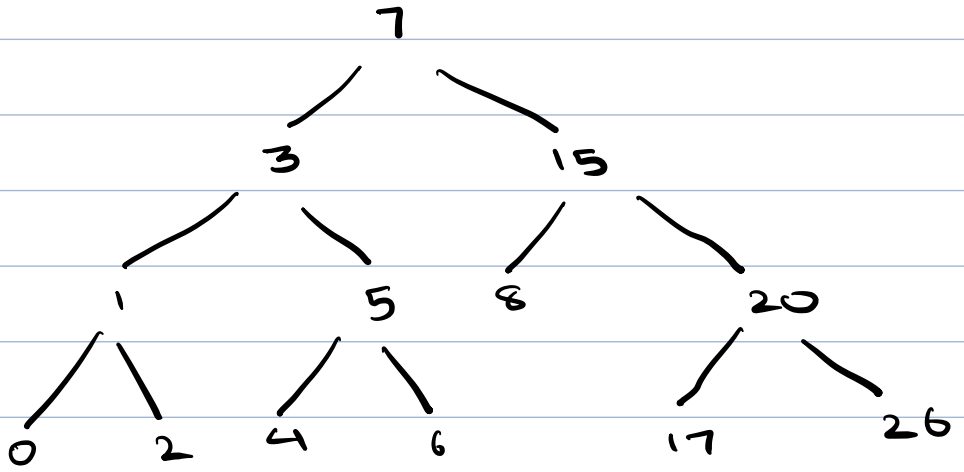



Delete 20

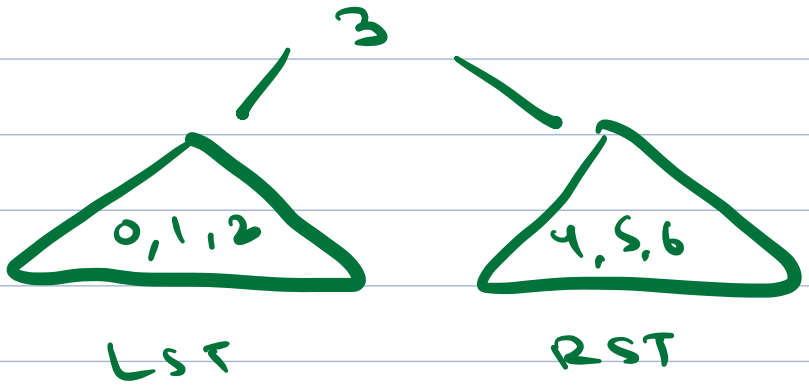


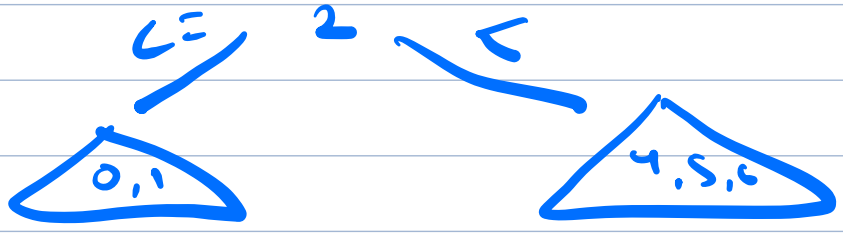

Delete 20

Case 3 : Node with 2 children

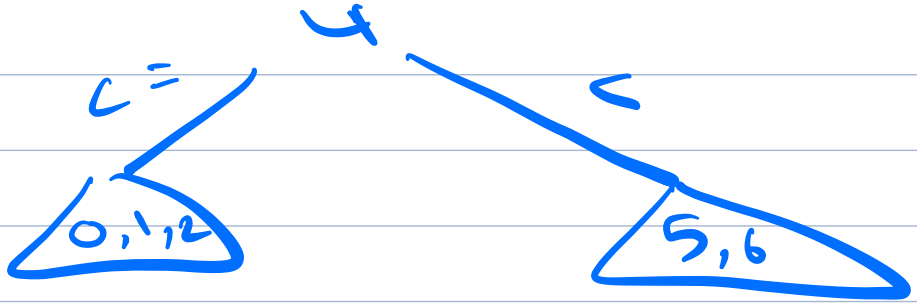



Delete 3

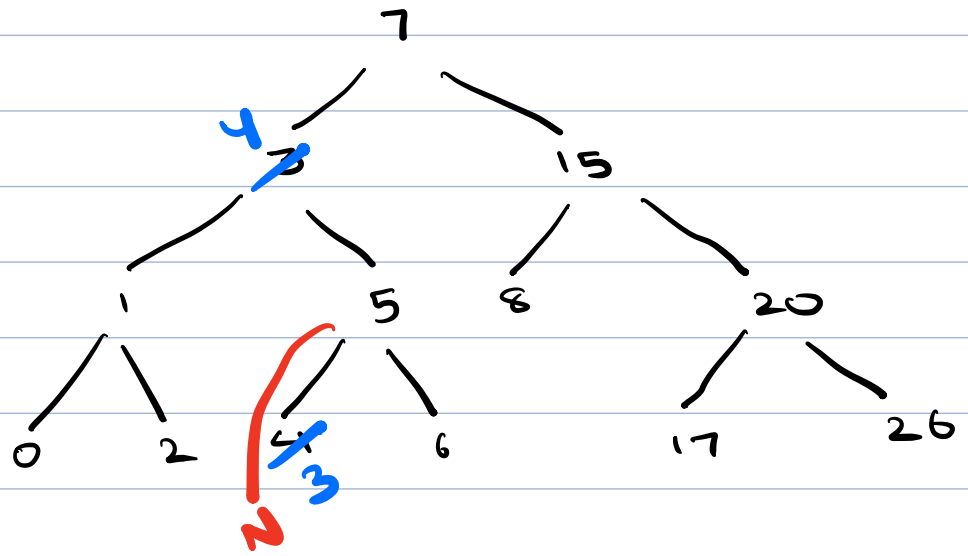


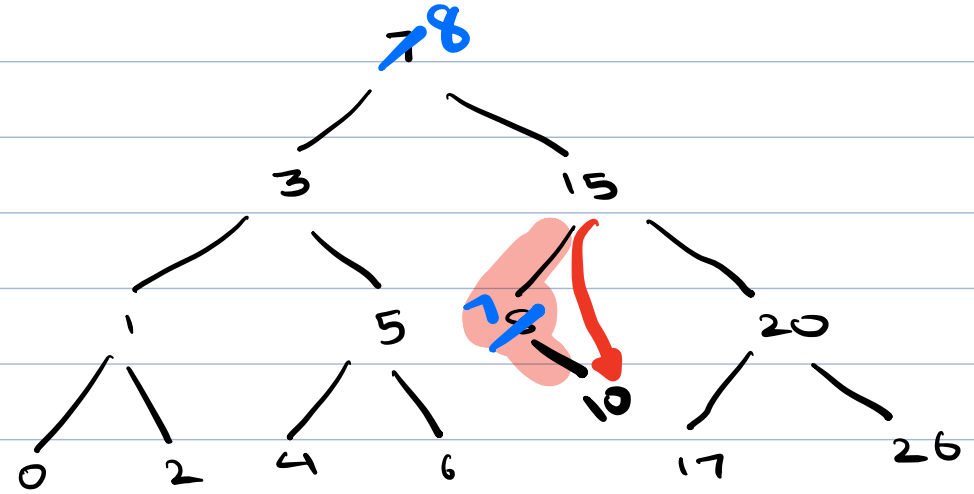




① Pick max from LST

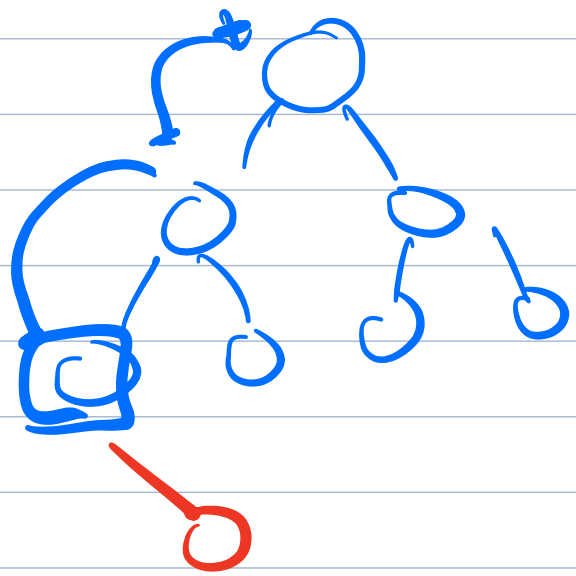


② Pick min from Rt



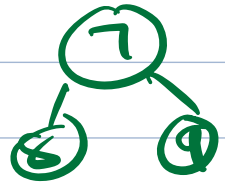




Delete 7

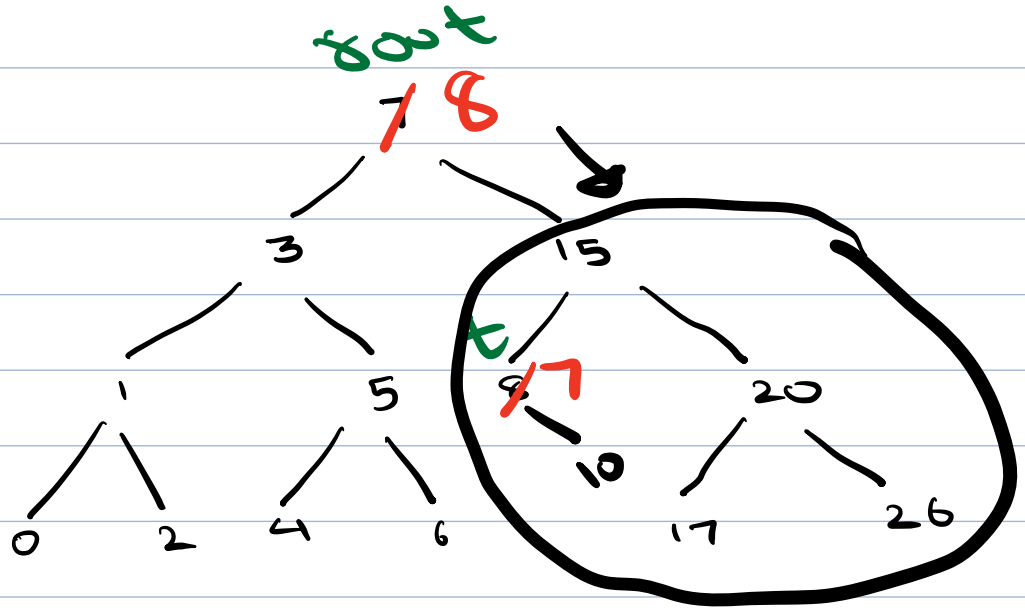


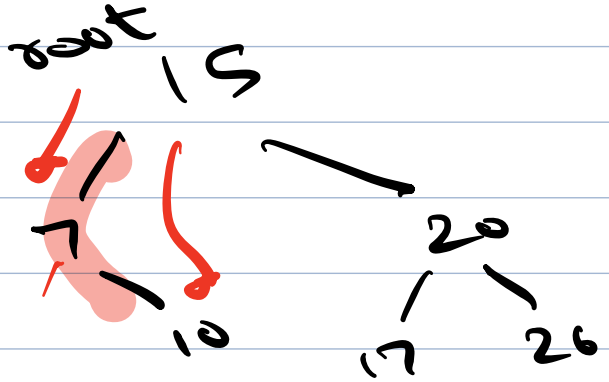
// Delete node with data k from tree and return new root

```
Node delete (Node root, int k) {

    if (root == NULL)
        return NULL

    if (root.data < k) {
        root.right = delete (root.right, k)
    }
    else (root.data > k) {
        root.left = delete (root.left, k)
    }
    else {  // root.data == k
        if (root.left == NULL && root.right == NULL)
            return NULL
        else if (root.left == NULL || root.right == NULL) {
            if (root.left == NULL)
                return root.right
            else
                return root.left
        }
        else {
            // swap root with some node
```



```
// Min from RST
Node temp = root.right
while (temp.left != null){
    temp = temp.left
}
swap (root.data, temp.data)
root.right = delete (root.right, k)
return root
```



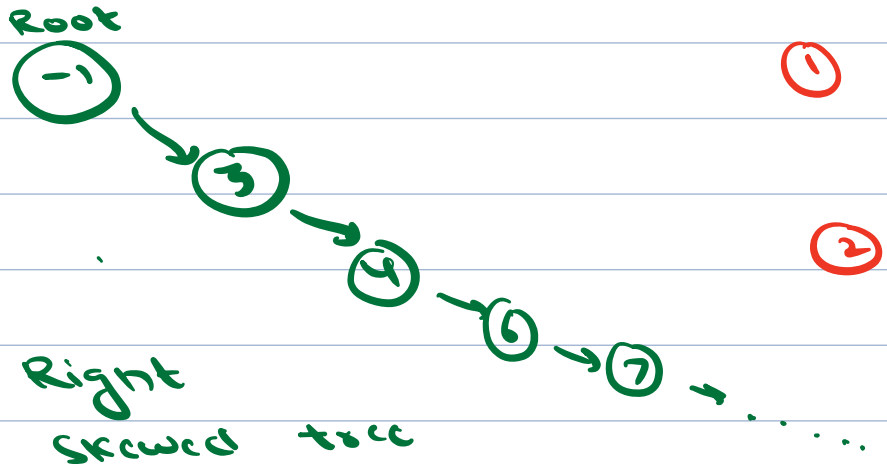

TC: O(H)

SC: O(H)

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 = O(N)

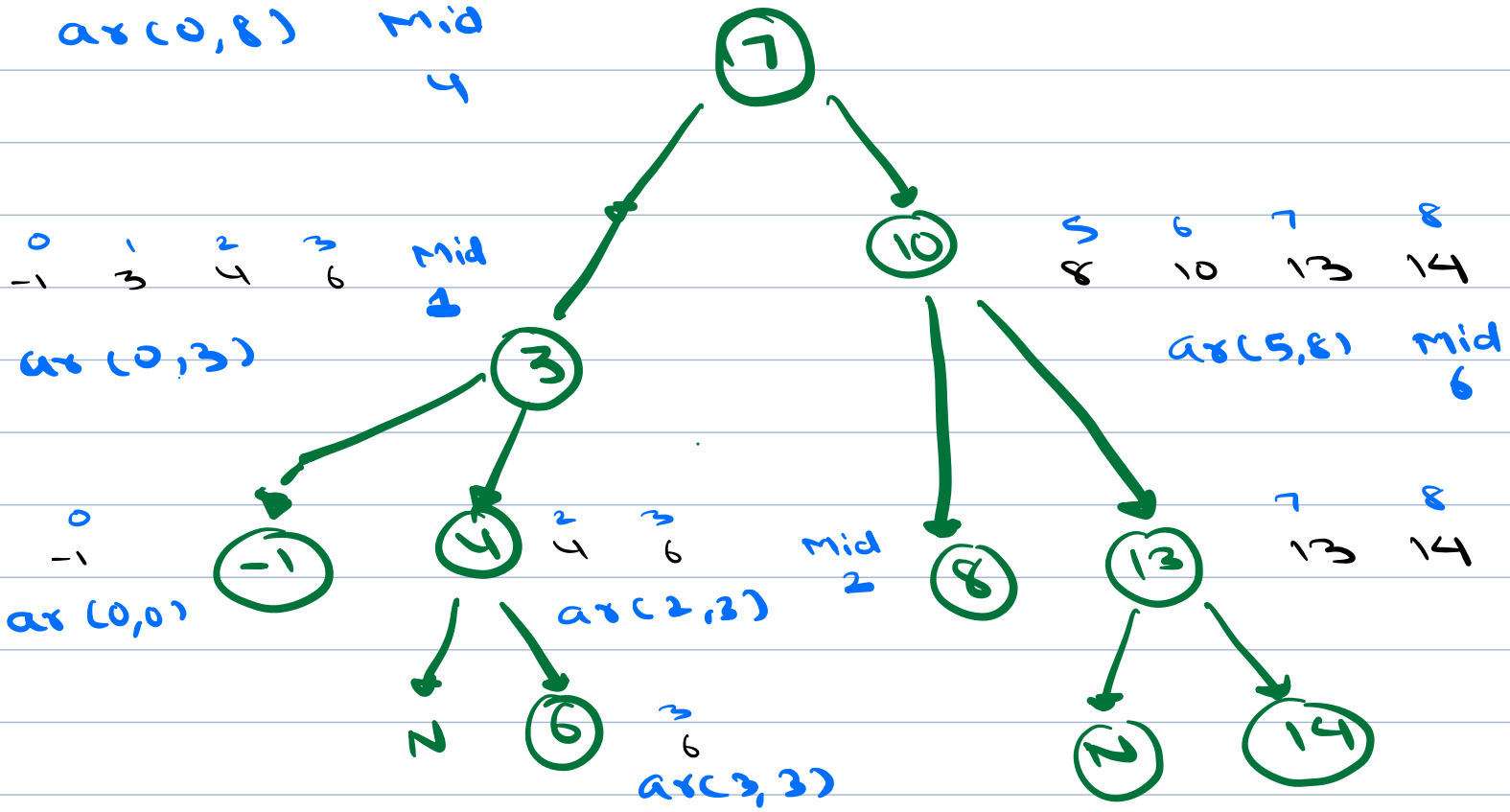
② TC: $O(N^2)$

TC of 1 insertion → O(H)

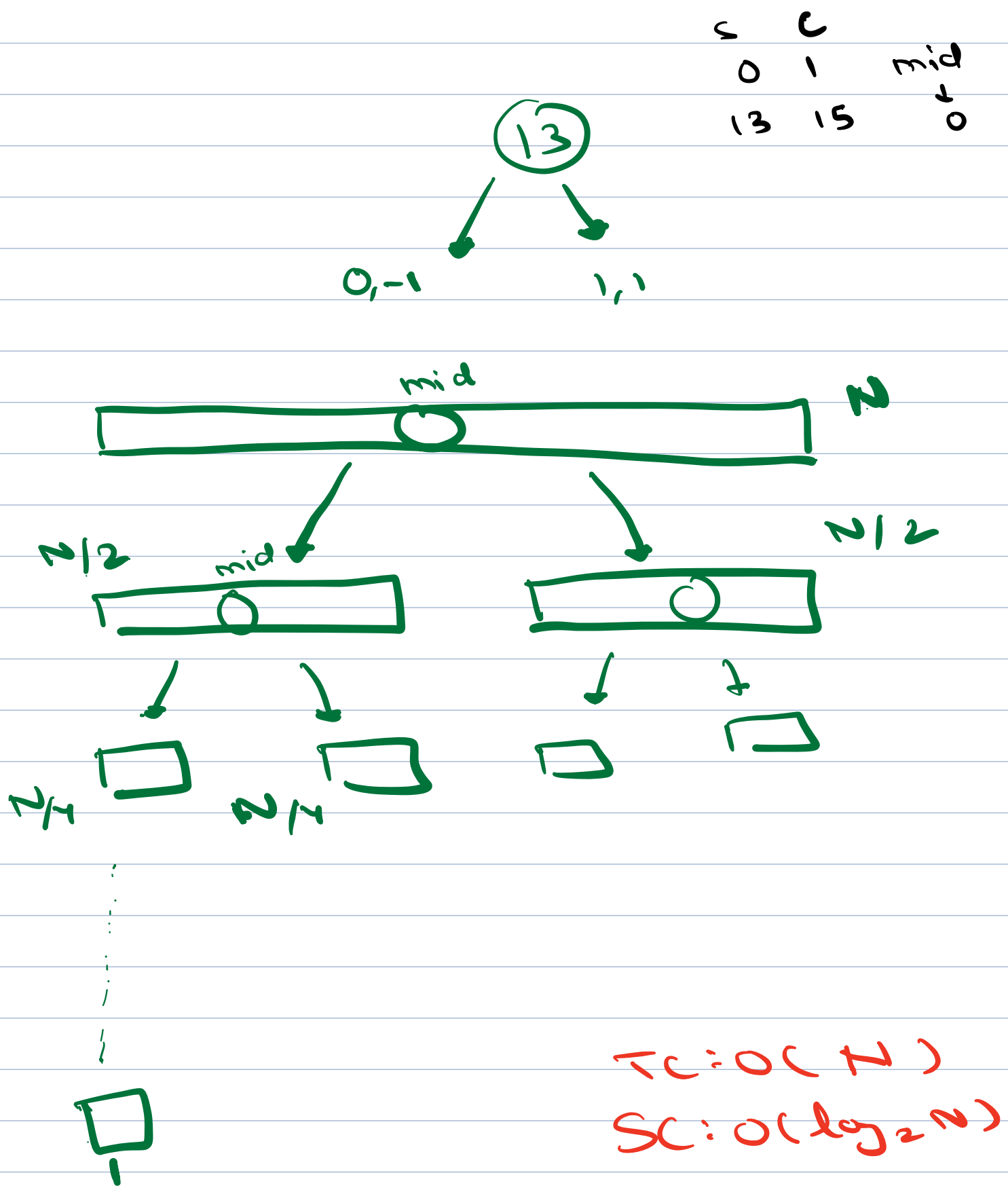
TC of N insertion → O(N*H)

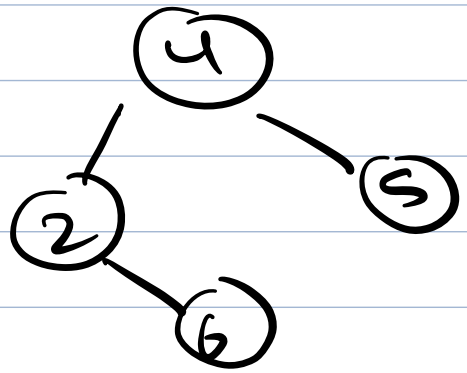
## Approach 2:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| -1 | 3 | 4 | 6 | 7 | 8 | 10 | 13 | 14 |




```
// Given an ar from s->e, return root
Node constructTree(ar, s, e) {
    if (s > e) return NULL
    int mid = 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
}
```


s
c
0
1
mid
13
15
0
13
0,-1
1,1
mid
N
N/2
mid
N/2
N/2
N/2
1
1
TC: O(N)
SC: O(log2N)



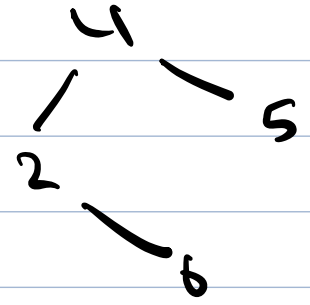

(unique values)

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 correct check

we are NOT checking subtree

∀ nodes x

1. All data in LST ≤ 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)
↓ stack space ↓ Array

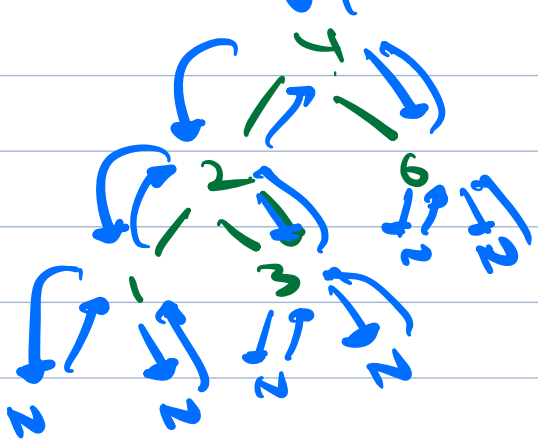
Approach 2: Maintain prev data during inorder traversal

int prev = INT_MIN, bool ans = true

```
void inorder(Node root){
    if(root == NULL)
        return

    inorder(root.left)
    if(root.data < prev)
        ans = false

    prev = root.data

    inorder(root.right)
}
```

prev = -∞ 1 2 3 4 5 6

ans = T



TC : O(N)

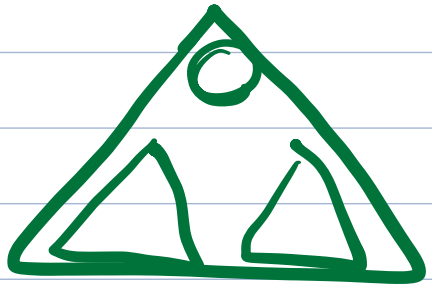
SC : O(H)

---

Post → L R N

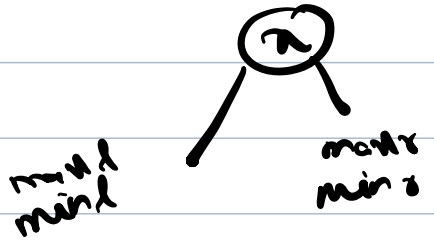
isValid (LST)

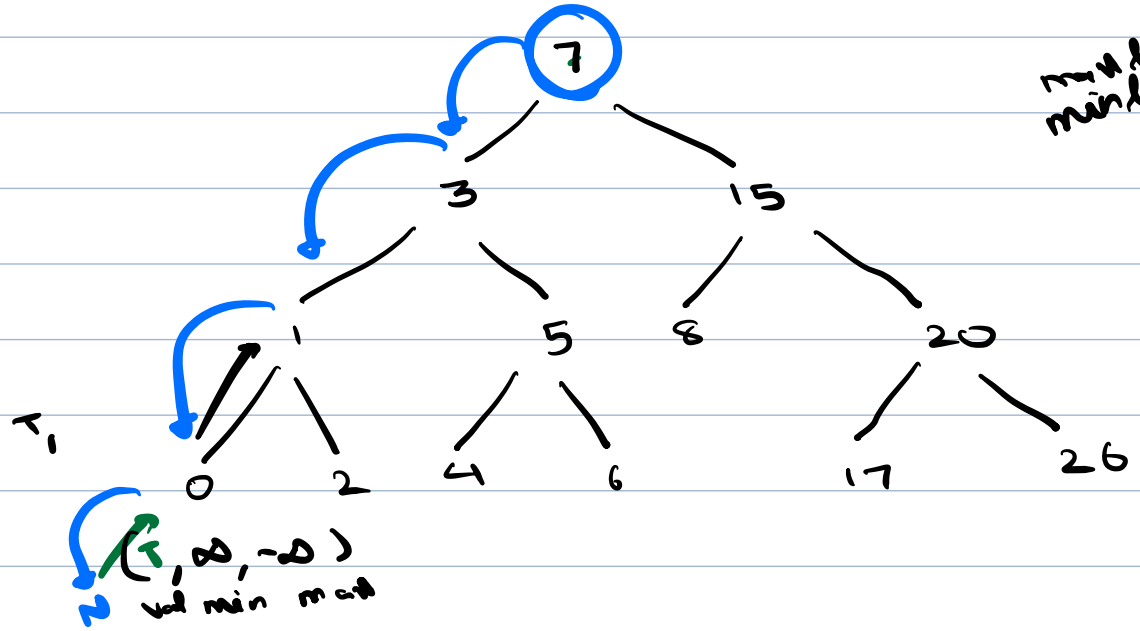
isValid (RST)

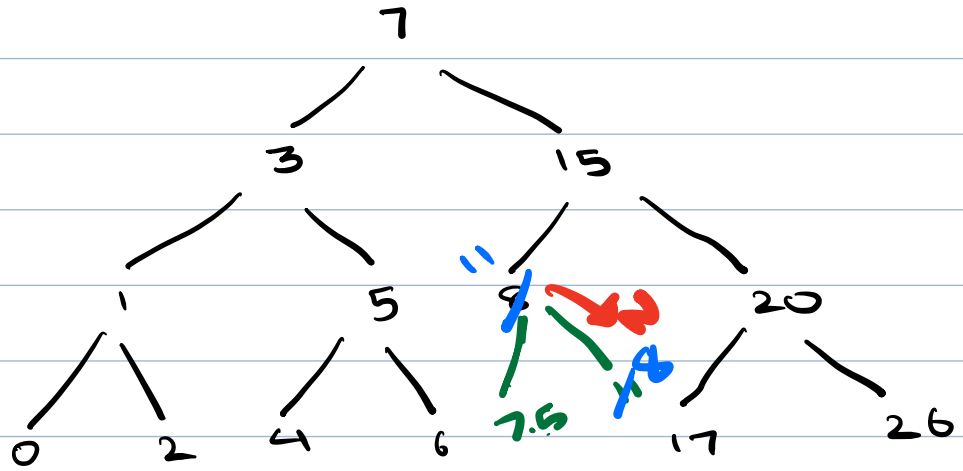


(root, min, max) isValid (root)

min-l, max-1




7
3
15
1
5
9
20
0
2
4
6
7.5
17
26