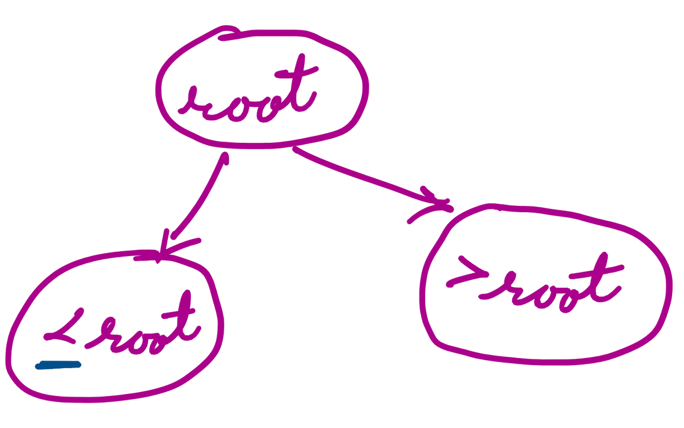
BST

Friday, 14 January 2022 9:06 PM

Binary Search Tree

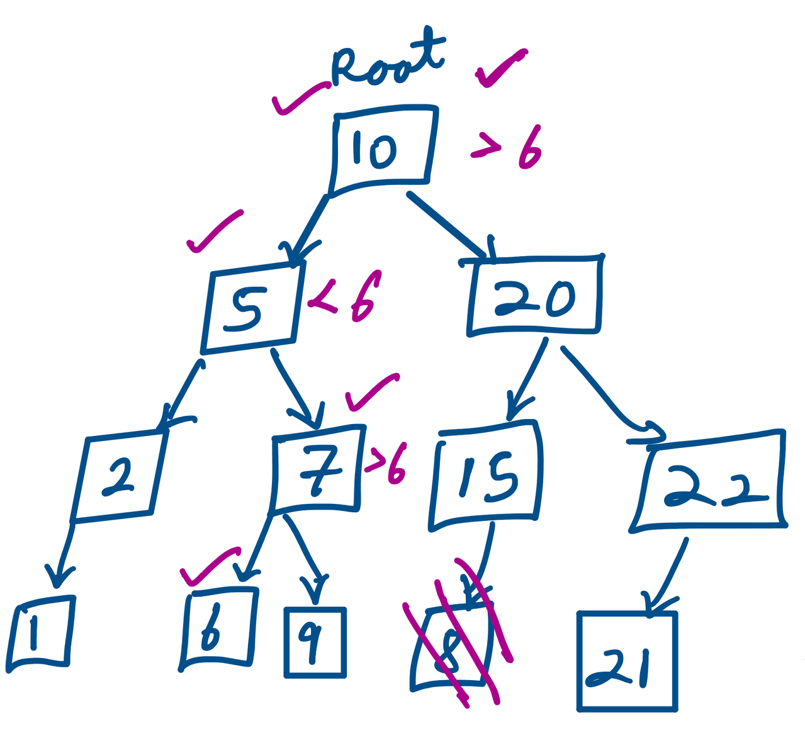
↳ Binary Tree optimized for searching.



1) only applicable on data which can be compared. ✓

2) Equality can go on any side. It's prefered to keep all equal elements on one side. ✓

∀nodes → all elements on left subtree are smaller or equal.
↳ all elements on right subtree are larger. ✓



search(6) ⇒ 10 → 5 → 7 → 6

search smallest value ⇒

while(root.left != null) root = root.left;
return root; ✓

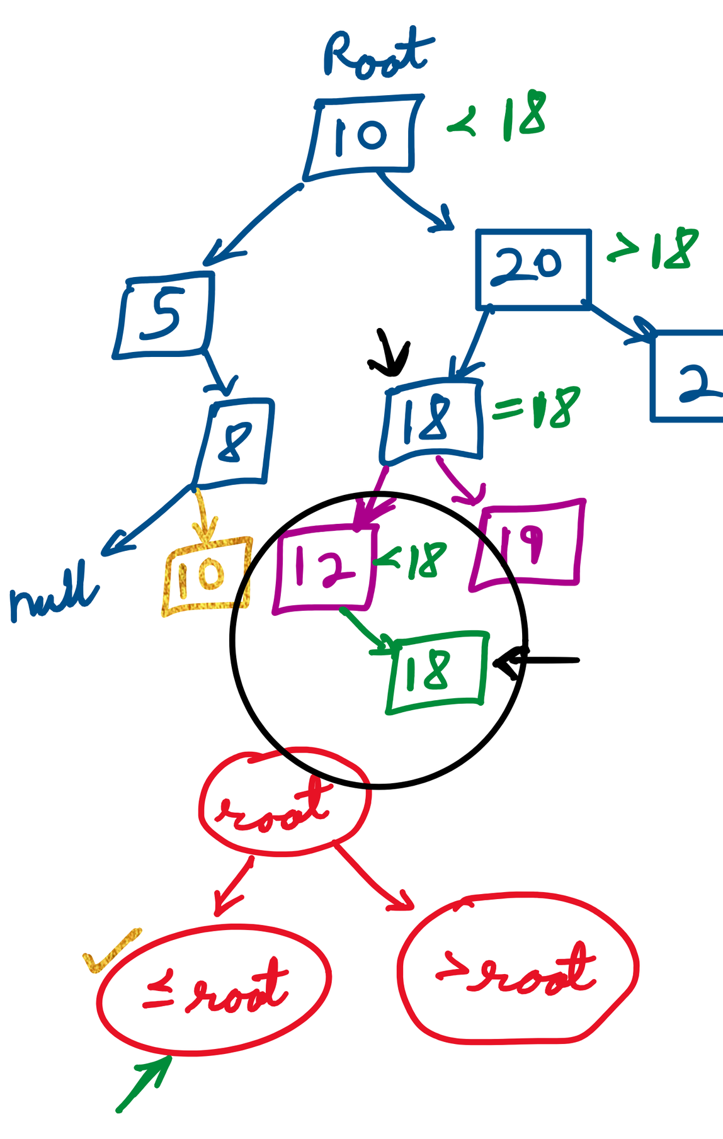
search largest value →

while (root.right != null) root = root.right;
return root; ✓

$TC = O(H)$

$SC = O(1)$

```
while(root != null) {
  if (root.data == val) return root; ✓
  if (root.data < val) root = root.right; ✓
  else root = root.left; ✓
}
return null;
```



Insert in BST

insert (12) → a) Search for 12
b) Insert it like a leaf node.

insert (19) insert (18)

insert (10)

$TC = O(H)$
$SC = O(1)$

Delete a node from BST

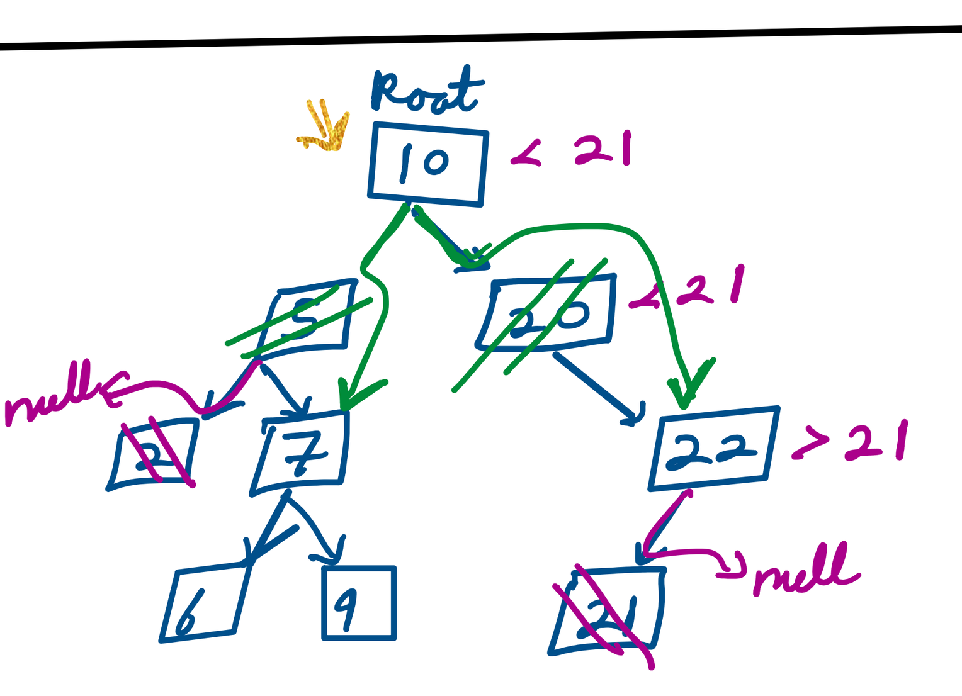
1) Search for the node to delete. ✓

2) Delete a leaf node

delete (2), delete (21)

3) Delete a node with one child → Directly point the parent to that child.

delete (20)
delete (5)

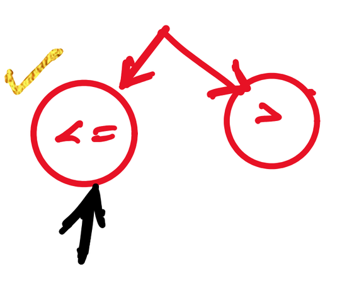


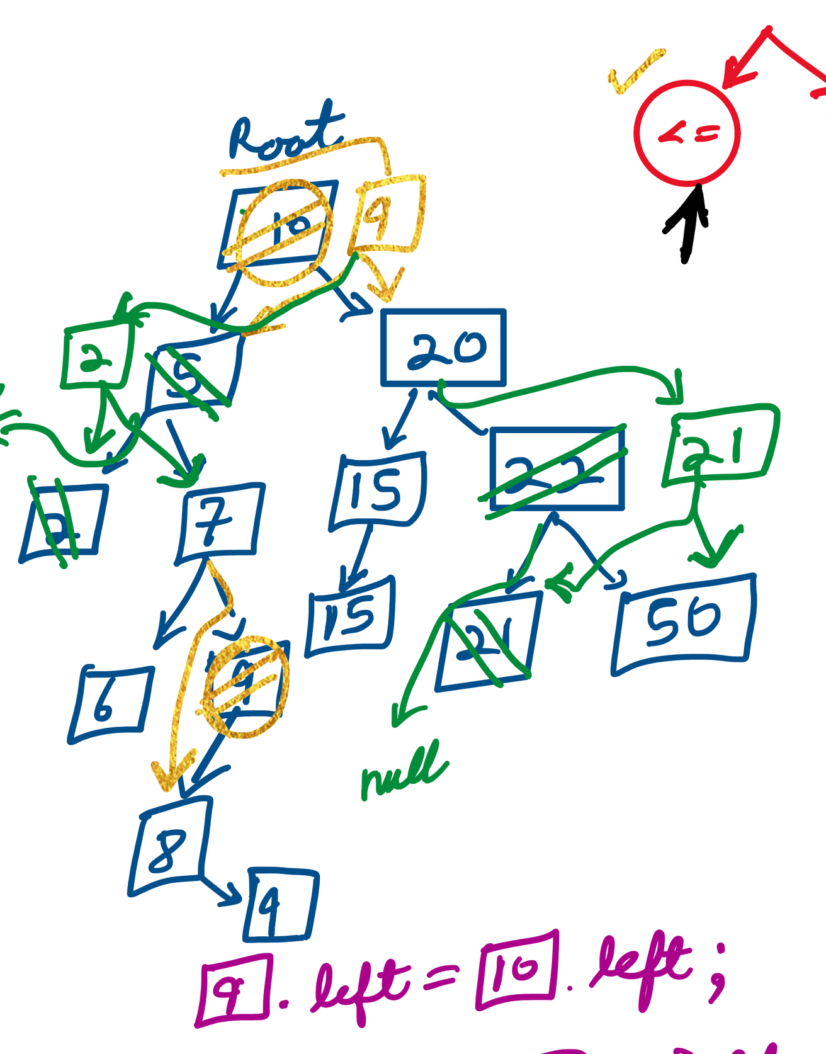

```
if (22.left.data == 21) {
    22.left = null;
}
```

4) Delete a node with both children ✓

delete (10)

a) Find the largest node in left subtree = X ✓
b) Delete X → can be solved with point 2 or 3. ✓
c) Replace current node with X. ✓





9.left = 10.left;
9.right = 10.right;

$TC = O(H)$
$SC = O(1)$

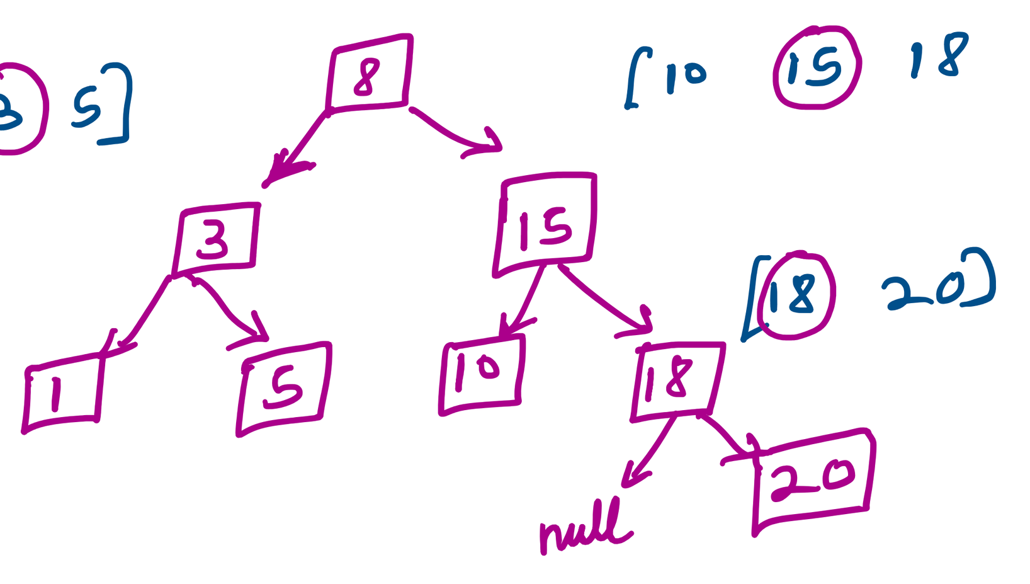
Q→ construct balanced BST from sorted array with unique elements.

balanced ↓

∀nodes $\left|\text{height of left child} - \text{height of right child}\right| \le 1$

Eg → [1 3 5 (8) 10 15 18 20]

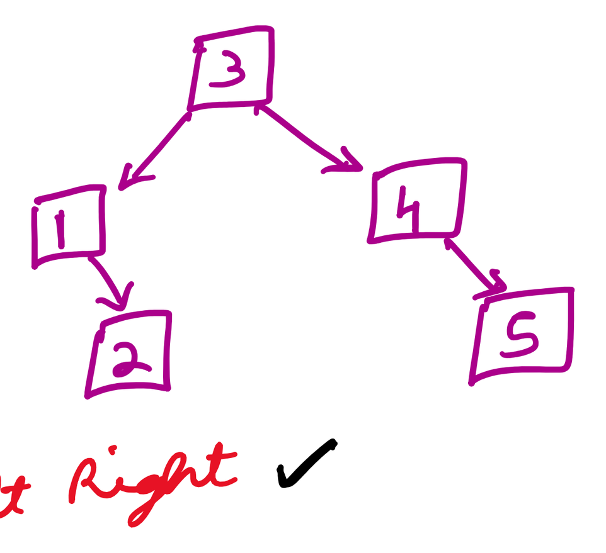
[1 (3) 5] [10 (15) 18 20] [(18) 20]



$TC = O(N)$
$SC = O(H) = O(\log(N))$

I/P 0 n-1

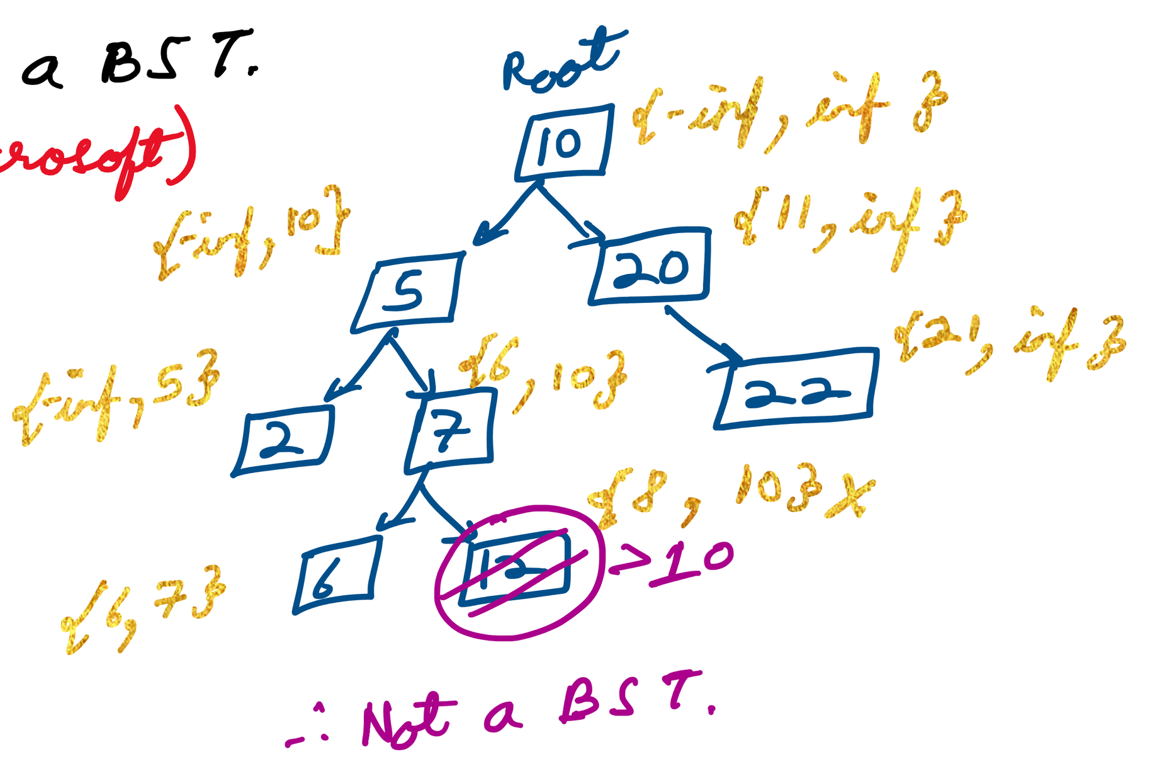
Eg → A = [1 2 (3) 4 5]



Pre → Node Left Right ✓
In → Left Node Right ✗

```
node build(a[], st, end) {
    if(st > end) return null; ✓
    Node { mid = (st + end) / 2; ✓
           root = new node (a[mid]); ✓
    Left { root.left = build (a, st, mid-1); ✓
    Right { root.right = build (a, mid+1, end); ✓
    return root; ✓
}
```

Q→ Check if the given binary tree is a BST. (Microsoft)



Sol 1 → Inorder traversal of BST is always sorted.

(Left) (Node) (Right)

2 5 6 7 10 (10) 20 22 ✓

Sol 2 → start from root define a range going left or right, & ∀elements should be in that range. (preorder)

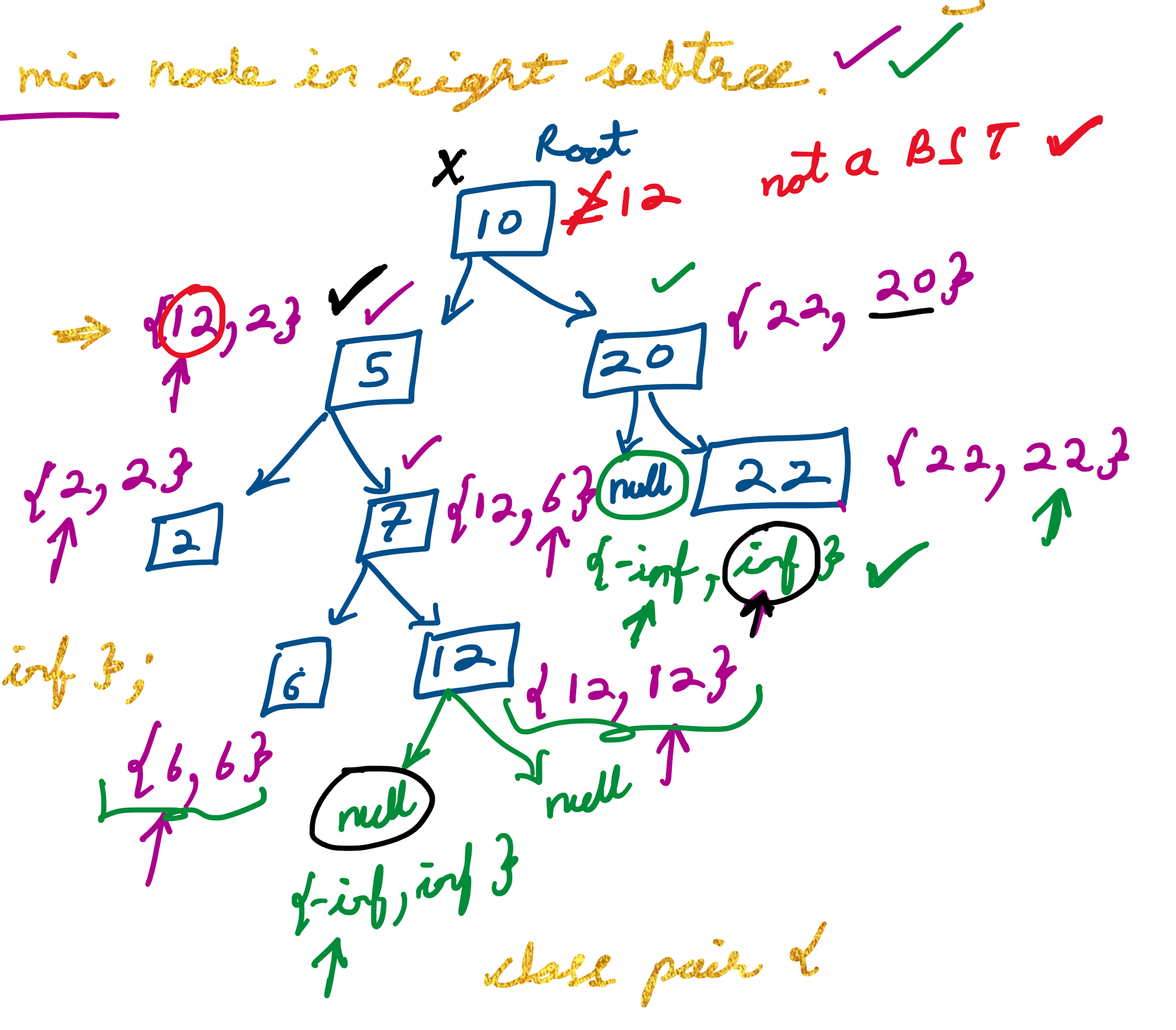
$TC = O(N)$
$SC = O(H)$

Sol 3 → ∀nodes → node.data ≥ ∀node data in left subtree
↳ node.data < ∀node data in right subtree

⇒ ∀nodes → node.data ≥ max node in left subtree ✓✓
↳ node.data < min node in right subtree ✓✓ } ✓

✓(postorder)

{max, min} ✓




```
isBST = true;
pair check (root) {
    if(root == null) return {-inf, inf};
    l = check (root.left);
    r = check (root.right);
    if(root.data < l.max ||
       root.data >= r.min) isBST = false;
    return { max(r.max, root.data),
             min(l.min, root.data)};
}
```

```
class pair {
    int min, max;
}
```

$TC = O(N)$
$SC = O(H)$

{-1, -1}

HW → Inorder Traversal in $SC = O(1)$? ⇐