Trees - 2

Announcement(s)

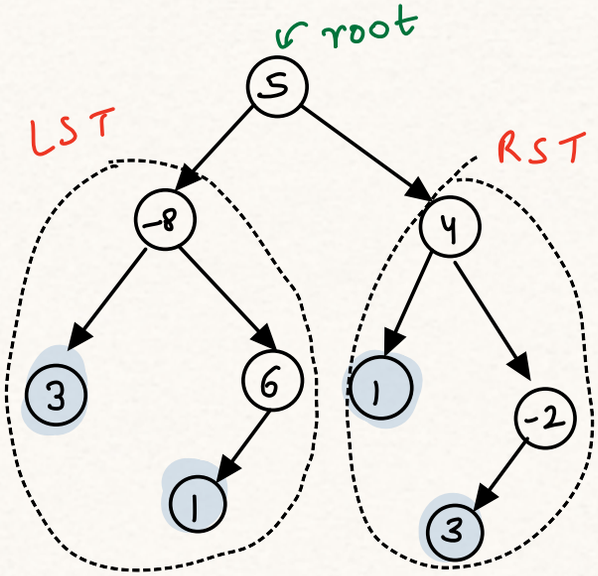
1 → Next week NO CLASSES.

2 → Contest Discussion on 6th March

3 → (14th - 16th - 18th) Problem Solving (Optional) or Remedial

4 → 21st March => DSML Content

Q1 Sum of all nodes in Given Binary Tree



5 + (-8) + 4 + 3 + 6 + 1 + (-2) + 1 + 3 = 13

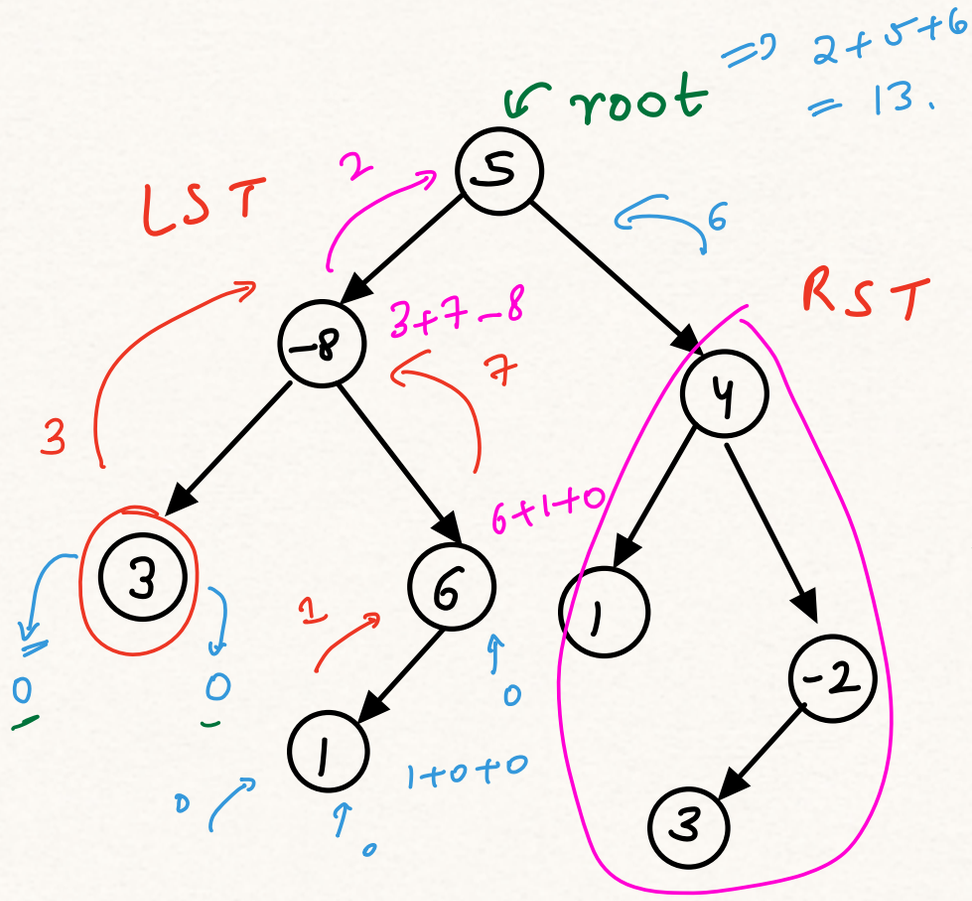
Method 1:

Do any traversal and update global sum.

```
# Assumption = Returns sum.
def sum_tree(root):
    if root == None:
        return 0
    ls = sum_tree(root.left)
    rs = sum_tree(root.right)
    return ls + rs + root.data
```

TC: O(N)

SC: O(H)



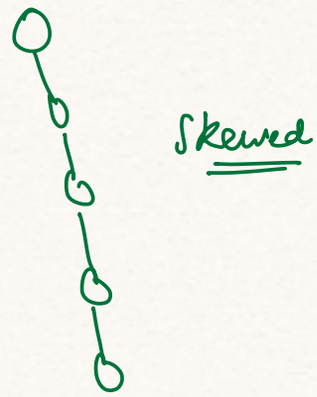
⇒ 2+5+6 = 13.

```
def sum_tree(root):
    if root == None:
        return 0
    ls = sum_tree(root.left)
    rs = sum_tree(root.right)
    return ls+rs+root.data
```

Base Condition

L
R
D

Postorder.



Skewed

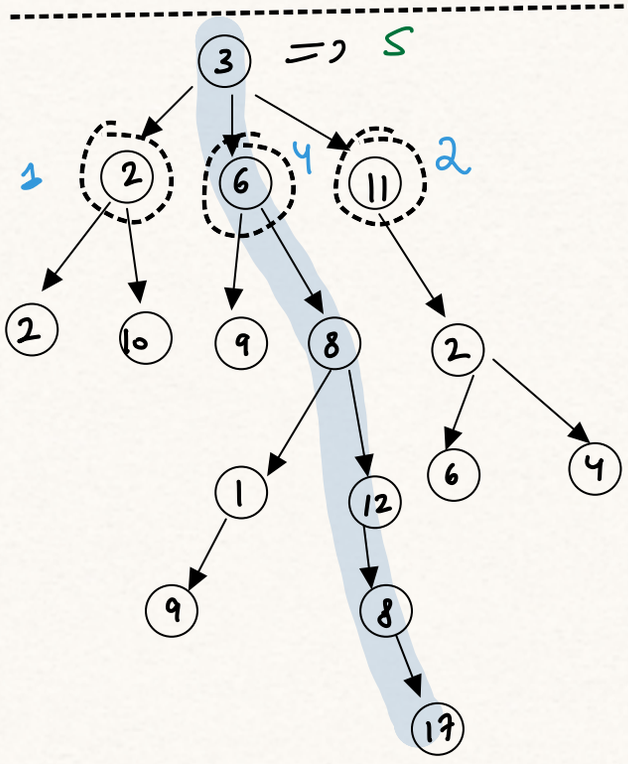
5	-8	~~3~~	6

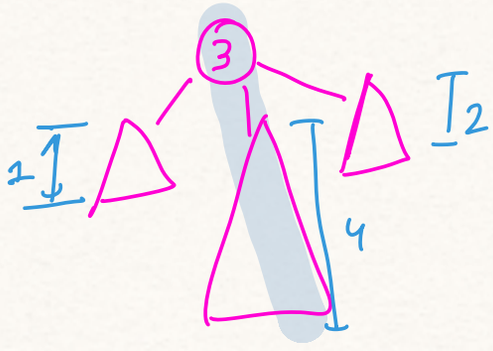
Idea:

Height of Given Tree.

Height = max distance from node to the farthest leaf.

Length of longest path from node to any of its descendant leaf nodes. (No. of edges)

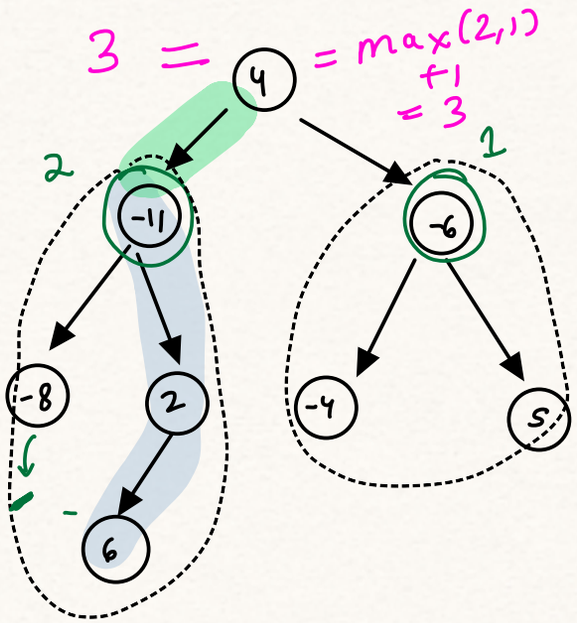




Height (node) = max(height of all its direct children) + 1.

Height (Tree) = Height (root)

Q2. Given a binary tree, find its height. (no. of edges in path)



Main Logic:

ht(root) =
max(ht(root.left), ht(root.right)) + 1

Base Condition

(1) Leaf => return 0

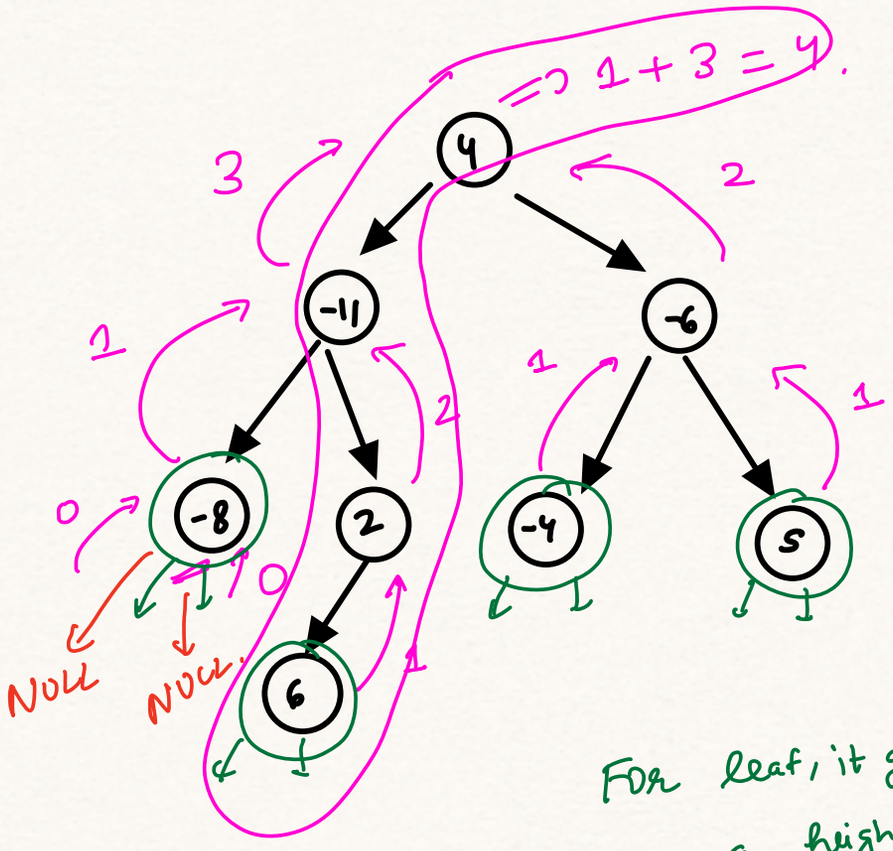
(2) None

```
def ht(root):
    if root == None:
        return 0
    lh = ht(root.left)
    rh = ht(root.right)
    return 1 + max(lh, rh)
```

NOTE:

In your assignment, height is defined as no. of nodes.

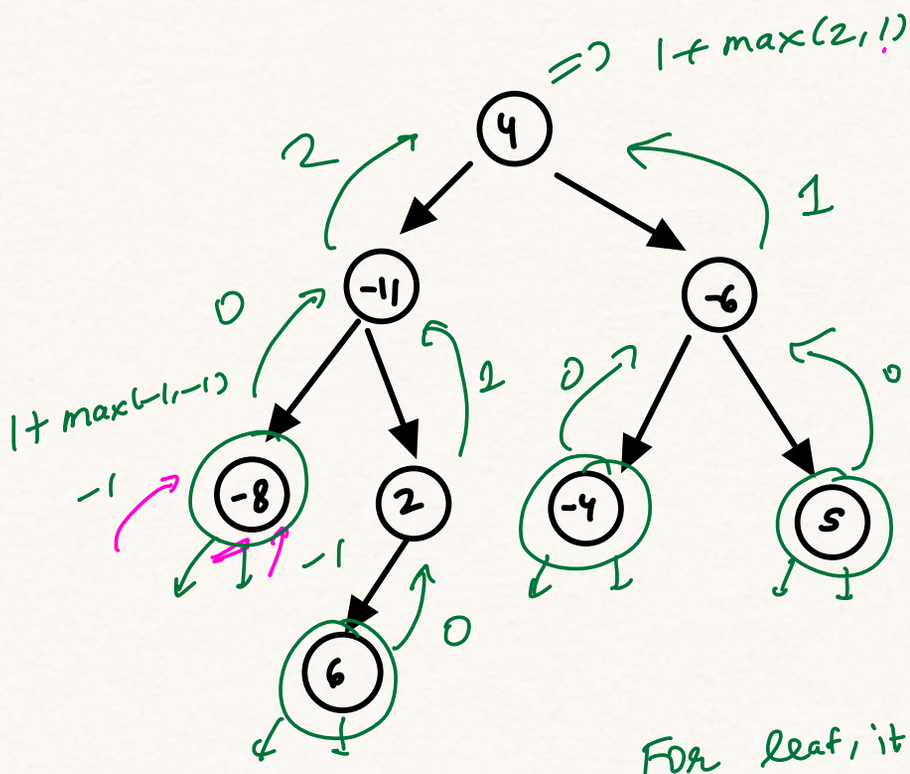
Try to tweak the above code to make that definition work.




```
def ht(root):
    if root == None:
        return 0
    lh = ht(root.left)
    rh = ht(root.right)
    return 1 + max(lh,rh)
```

In terms of no of nodes

For leaf, it gives a height of 1.




```
def ht(root):
    if root == None:
        return -1
    lh = ht(root.left)
    rh = ht(root.right)
    return 1 + max(lh,rh)
```

In terms of no of edges.

For leaf, it gives a height of 0

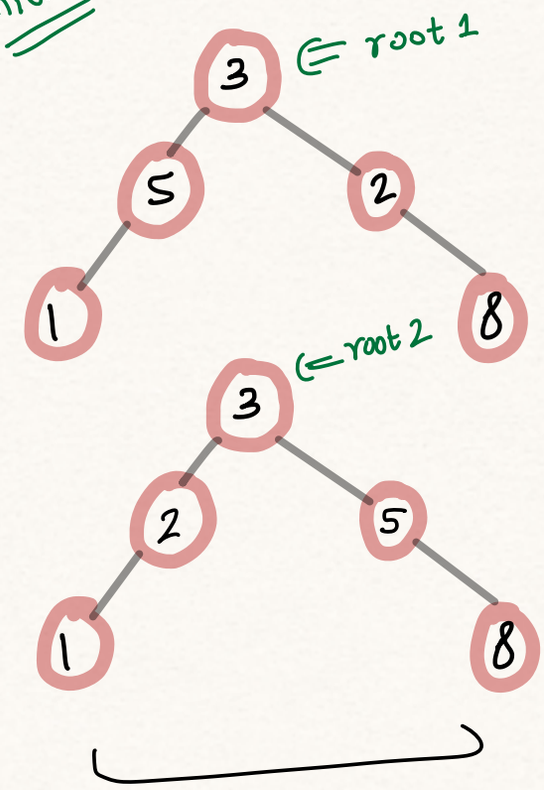
None Empty Tree

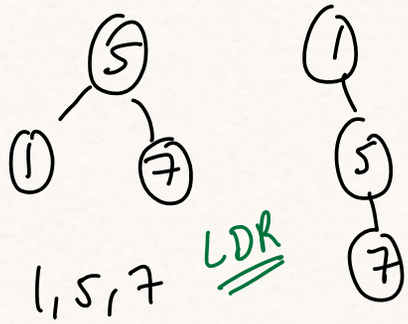
-1

TC: O(N)

Amazon
Microsoft Q3. Given 2 binary trees, check if they are IDENTICAL.

{ structure, data } same.



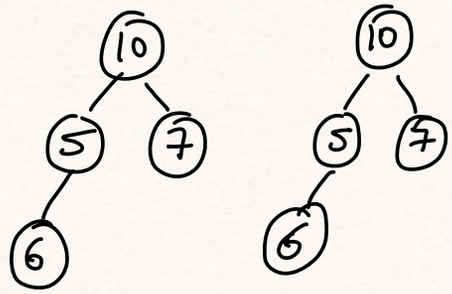


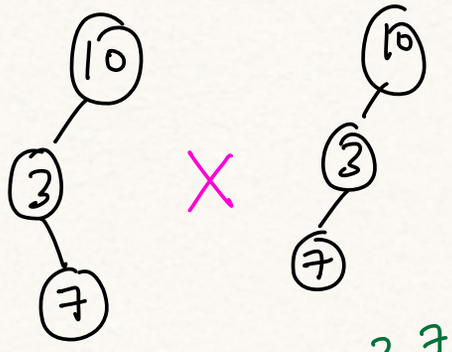
LDR

1,5,7

Inorder

1,5,7



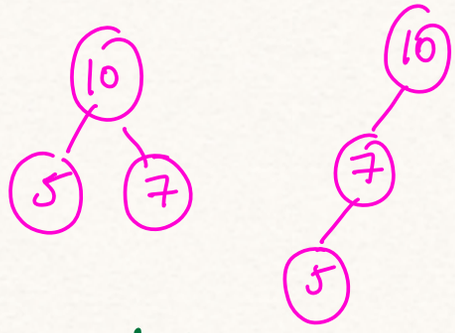


10,3,7

10,3,7

Preorder

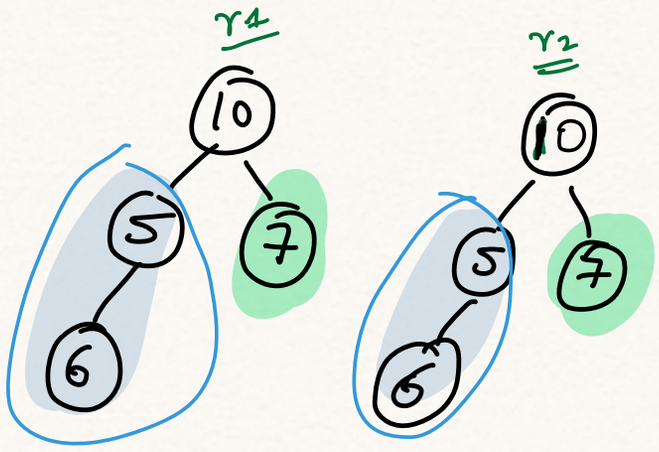
DLR

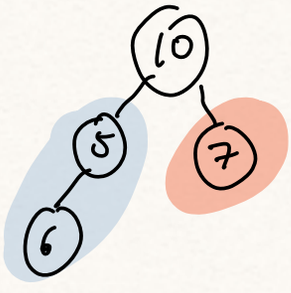


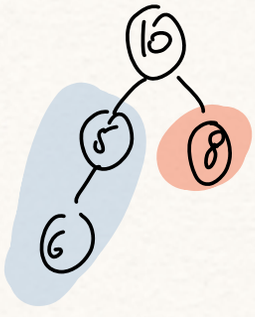
Postorder

LRD

5,7,10







&& ⇒ and

|| ⇒ or

```
# return True/False.
def check(r1, r2):
    if r1 == None and
       r2 == None:
        return True
    if r1 == None or
       r2 == None:
        return False
    if r1.data != r2.data
        return False
    ls = check(r1.left, r2.left)
    rs = check(r1.right, r2.right)
    return ls && rs
```

Base Condition ✓

Edge Cases

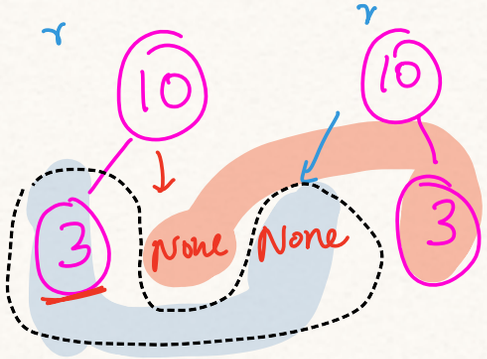
only 1 of them is None

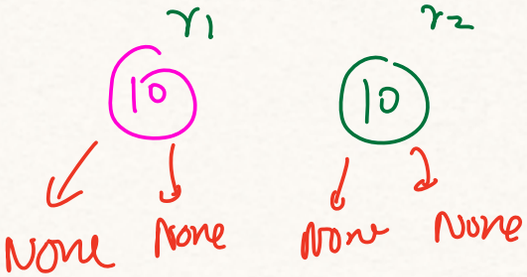
r2 ==None r2 ==None.

10 — None. — False.

None — 10 — False.

None = None. — True.





a or b

0 1

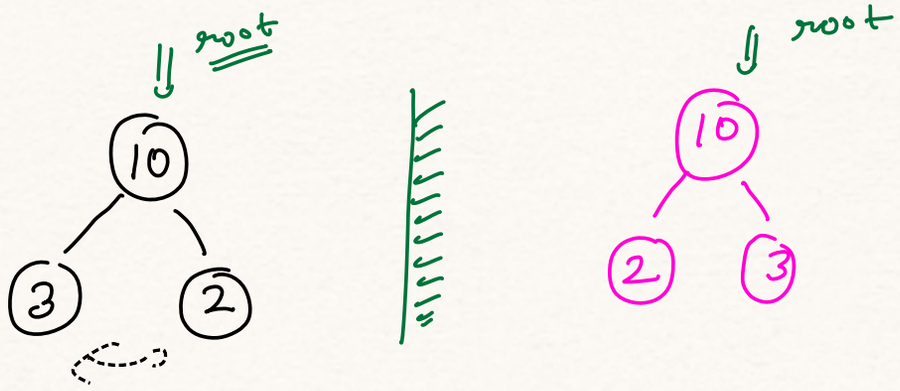
1 0

1 1

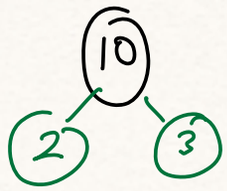
0 0

✓ Q4. Given a binary tree, invert the binary tree.

Homebrew

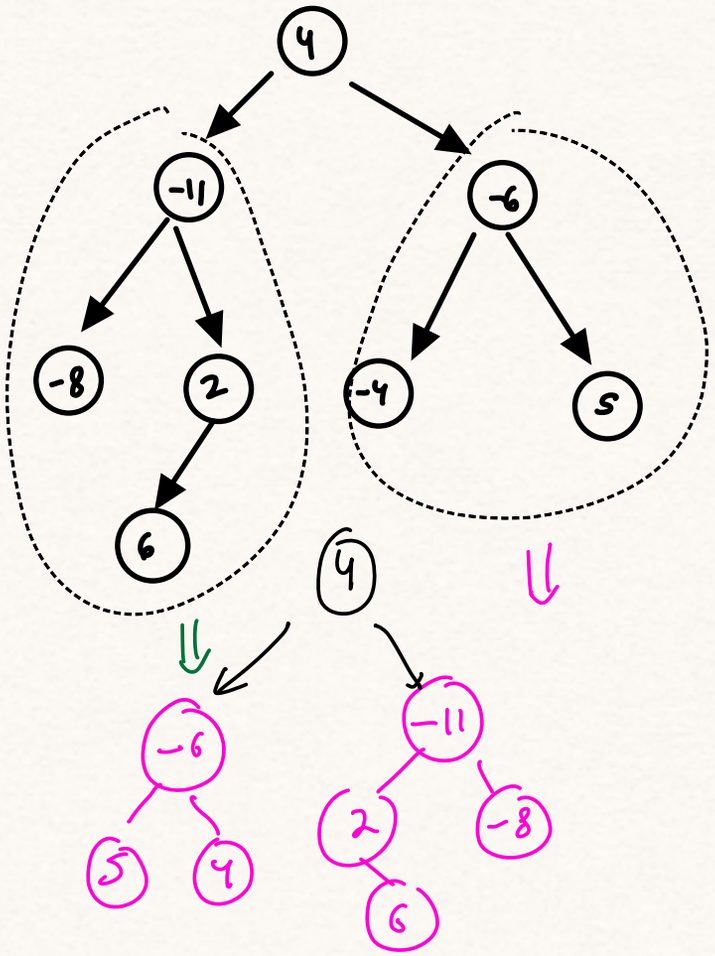


Modify the existing binary tree.

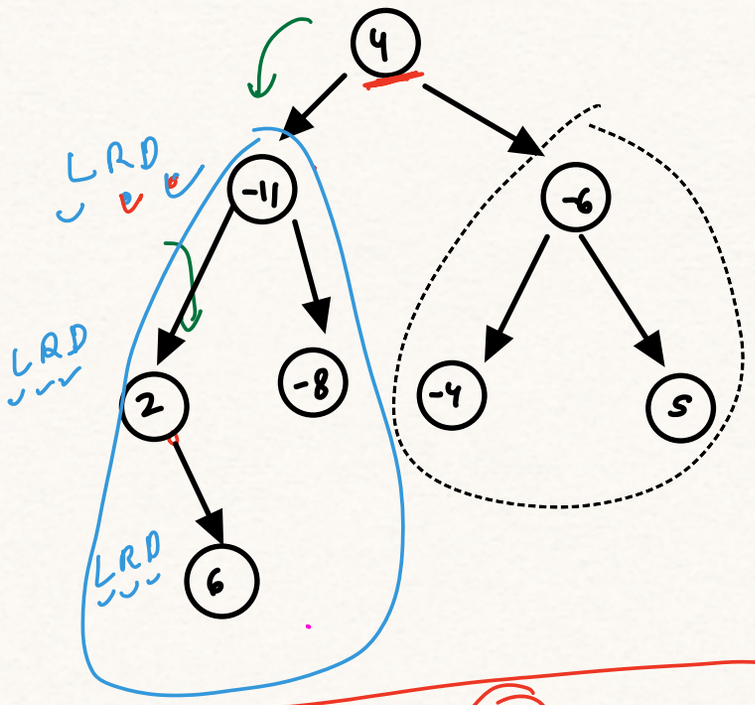


root.left, root.right = root.right, root.left

a, b = b, a




```
def invert (root):
    if root == None:
        return None
    linv = invert (root.left)
    rinv = invert (root.right)
    root.right = linv
    root.left = rinv
    return root
```




```
def invert(root):
    if root == None:
        return None
    linv = invert(root.left)
    rinv = invert(root.right)
    root.right = linv
    root.left = rinv
    return root
```

L
R
D

changes to root.

Post order Traversal

```
def invert(root):
    if root == None:
        return
    invert(root.left)
    invert(root.right)
    root.left, root.right = root.right, root.left
    return
```

c to

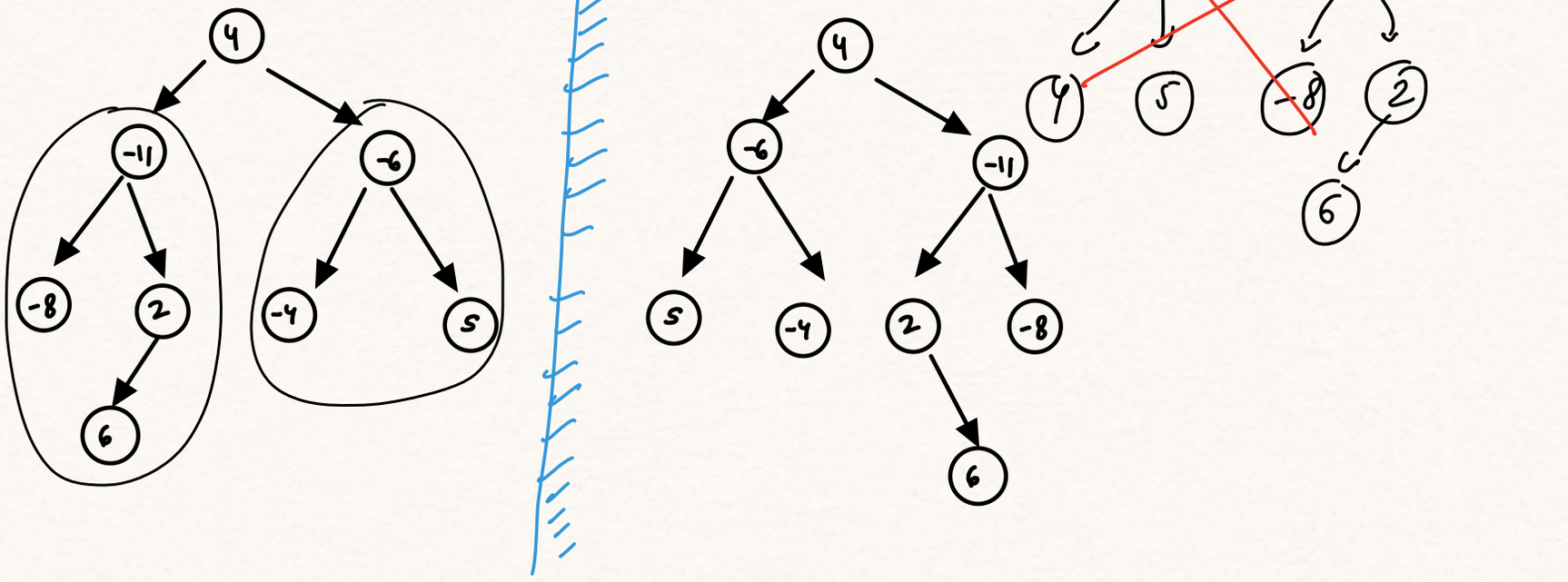
TC: O(N)

SC: O(H) ⇒ O(N)

Worst Case

✓ Q4. Given a binary tree, invert the binary tree.

Homebrew



BST — Binary Search Trees

⇓

{ No duplicates }

- Balanced BST ⇒ $\log_2 N$

⇓

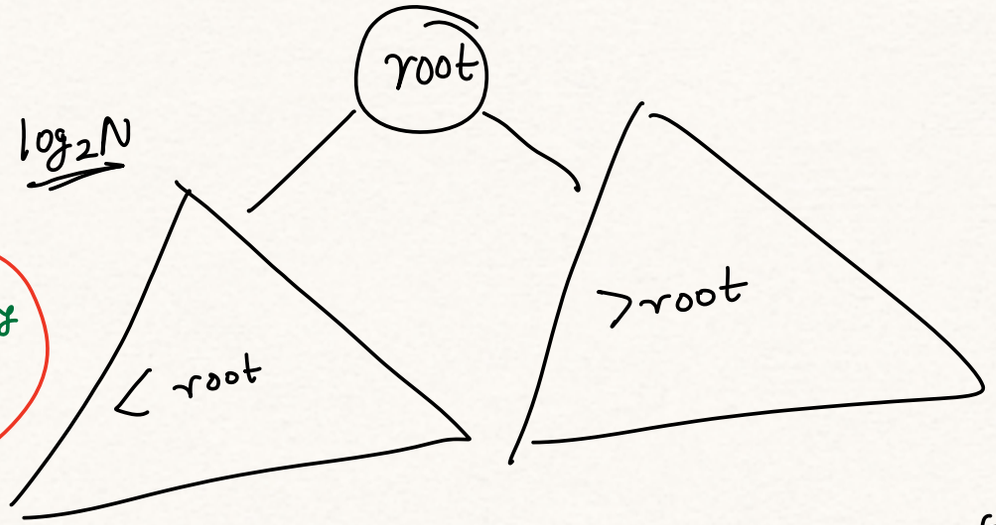
self-balancing trees.

⇓

dictionaries

sets

{ AVL
Red-Black

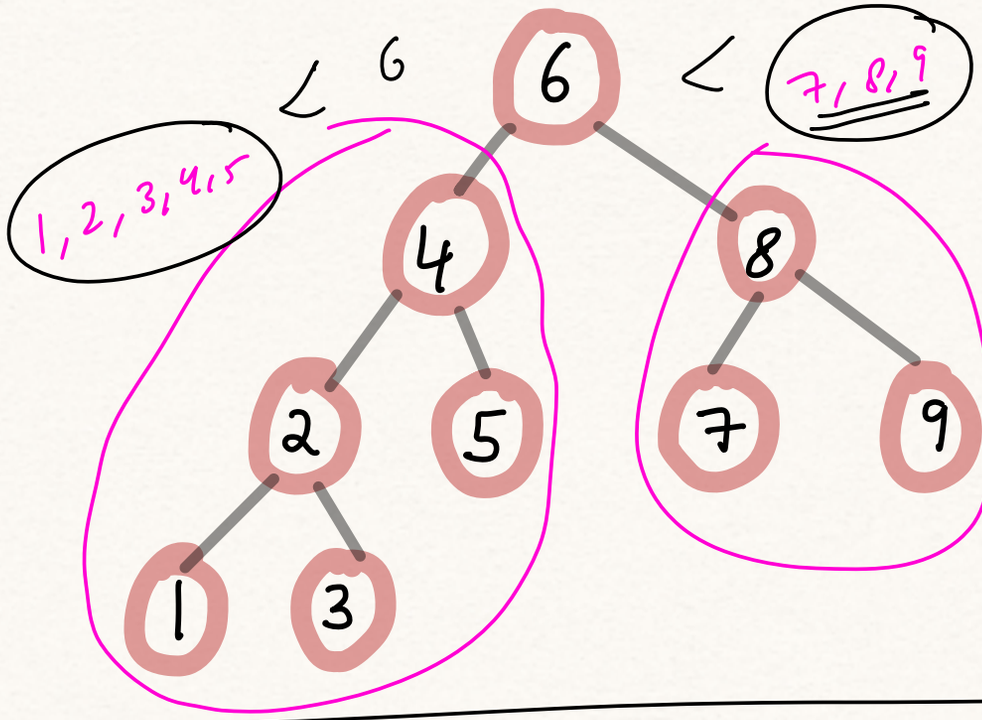


This BST condition

∀ nodes

node.data > Call of its left subtree)

node.data < Call of its right subtree)



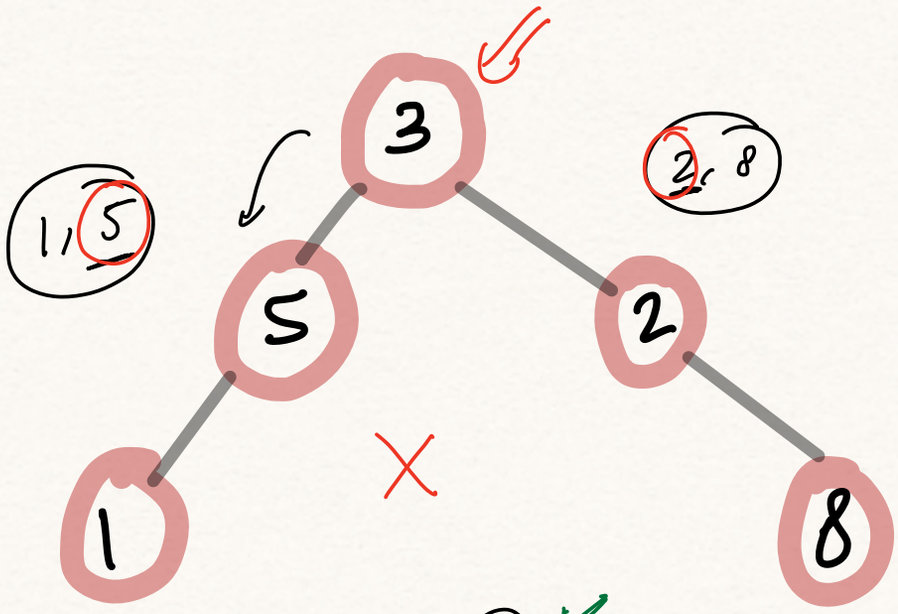
Inorder

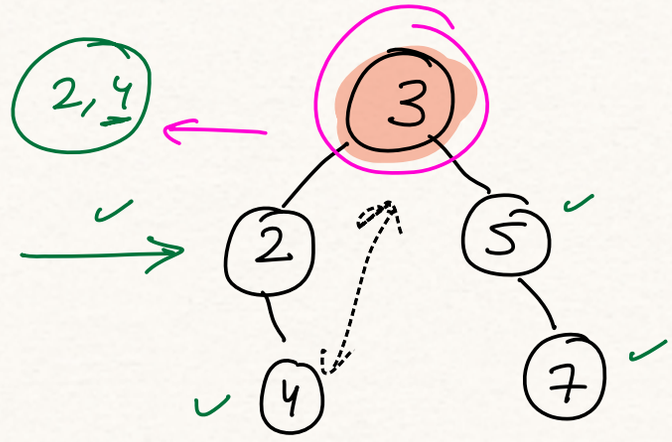
1, 2, 3, 4, 5, 6, 7, 8, 9

For all nodes.

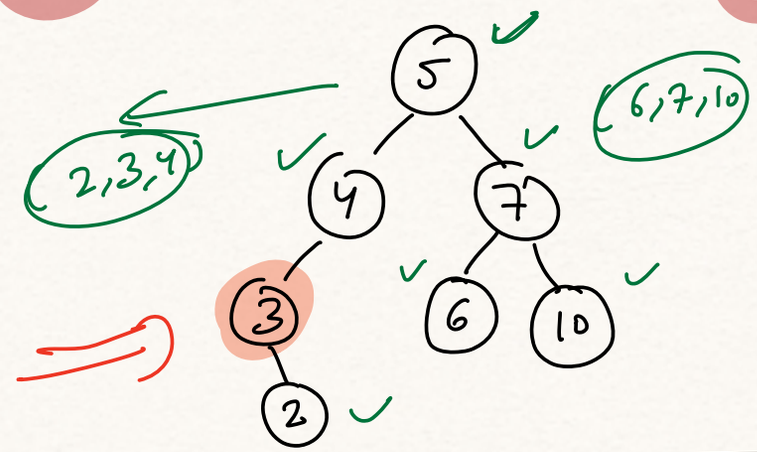
left subtree < node.data < right subtree.

L < D < R.

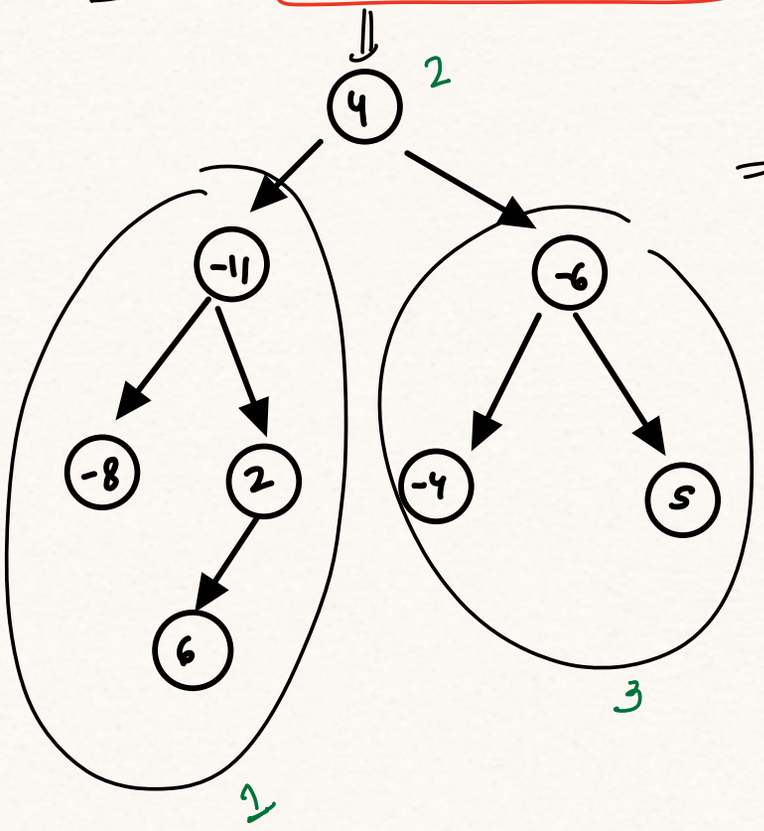




Is this a BST?



Q4 Given a binary tree, check if it is BST or not.



⇒ Inorder LDR

Left Data Right

Root

If inorder is sorted

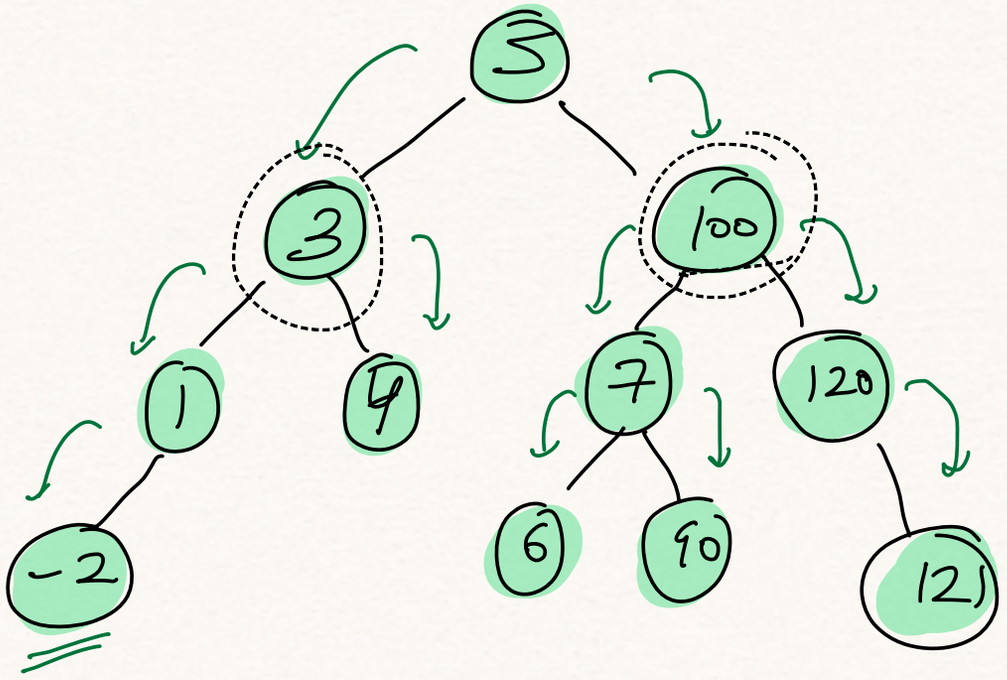
⇒ BST.

Trees + BST

7-8

Adv Batch

[-8, -11, 6, 2, 4, -4, -6, 5]



Inorder

3 different ideas.

Adv batch

visit check

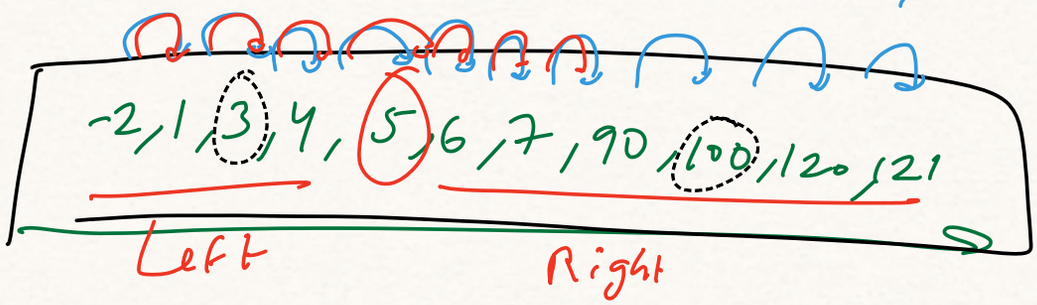
TC: O(N) + O(N)

SC: O(H) rec.

+ O(N)

storing inorder.

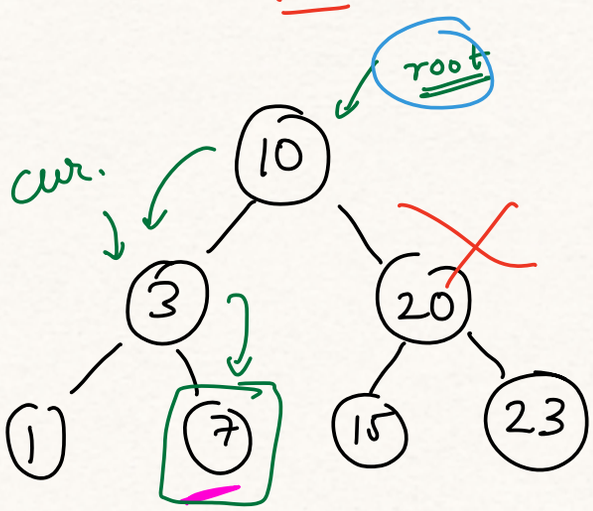
arr =



Q 5. Given a BST, search for the given value K. (Target)

Search

K = 7



✓ Way 1:

Inorder.

1, 3, 7, 10, 15, 20, 23

O(N) + o(log N)

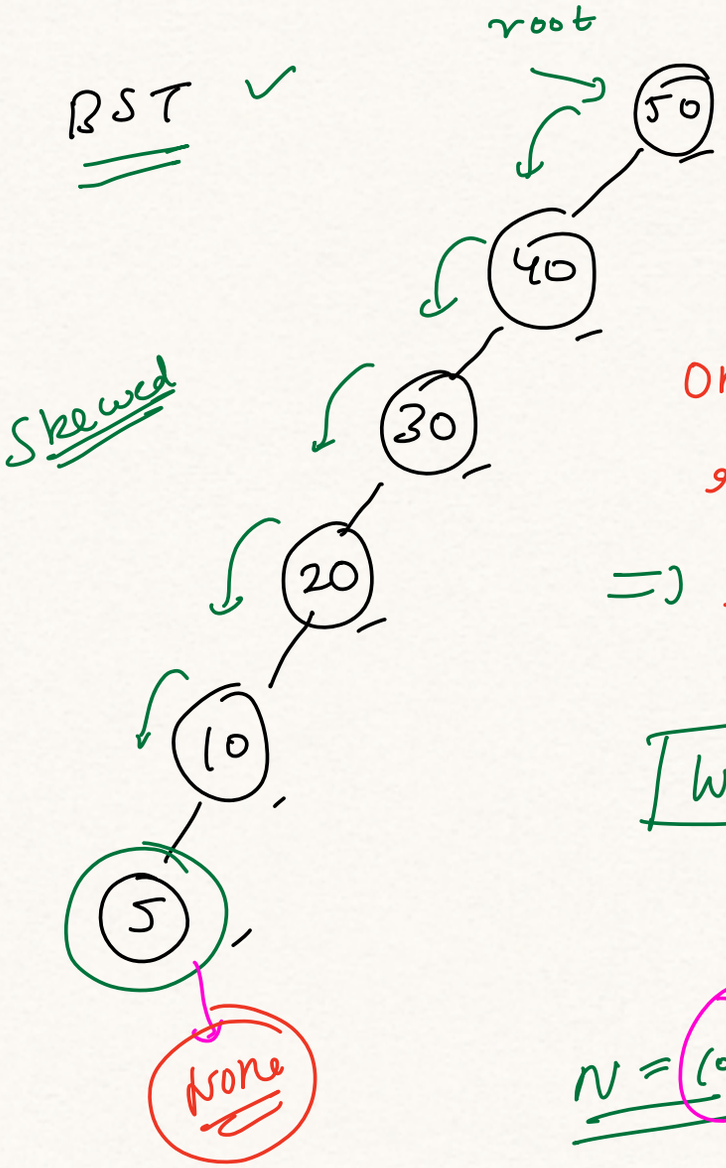
store search using binary search

Way 2: Without inorder

Binary Search

K = 7

1, 3, 7, 10, 15, 20, 23



BST ✓

Skewed

$N = 6$

search for $K = 7$

False (-1)

Once we reach None

⇒ return false.

$TC = O(H)$

Worst case $= O(N)$

Worst case $H = N$

Balanced BST

$O(\log_2 N)$

$N = 10^8$ ⇒ 1 sec

$\log_2 N$ ⇒ 27...

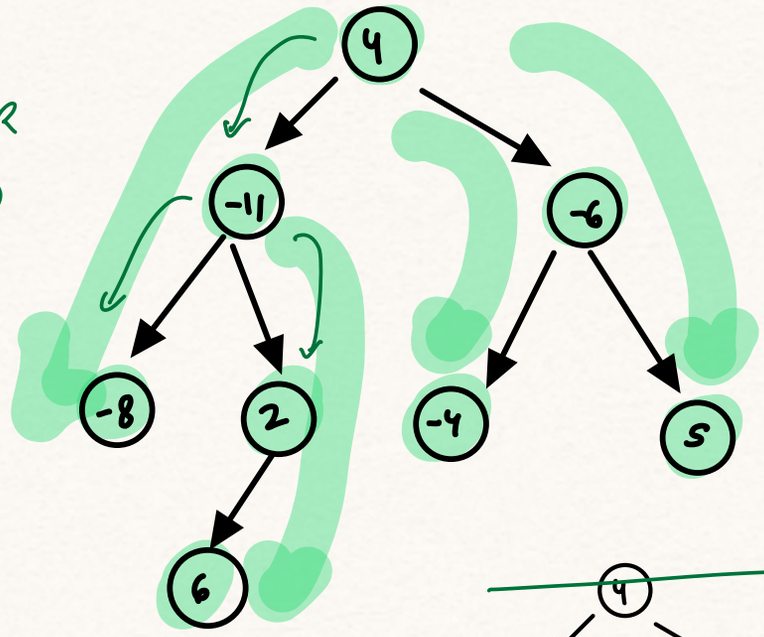
Doubts

DFS

preorder DLR

postorder LRD

inorder LDR

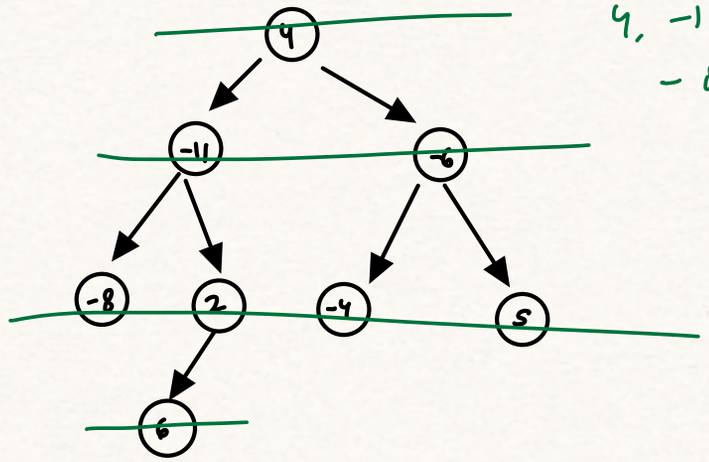


4, -11, -8

2, 6,

-6, -4, -5

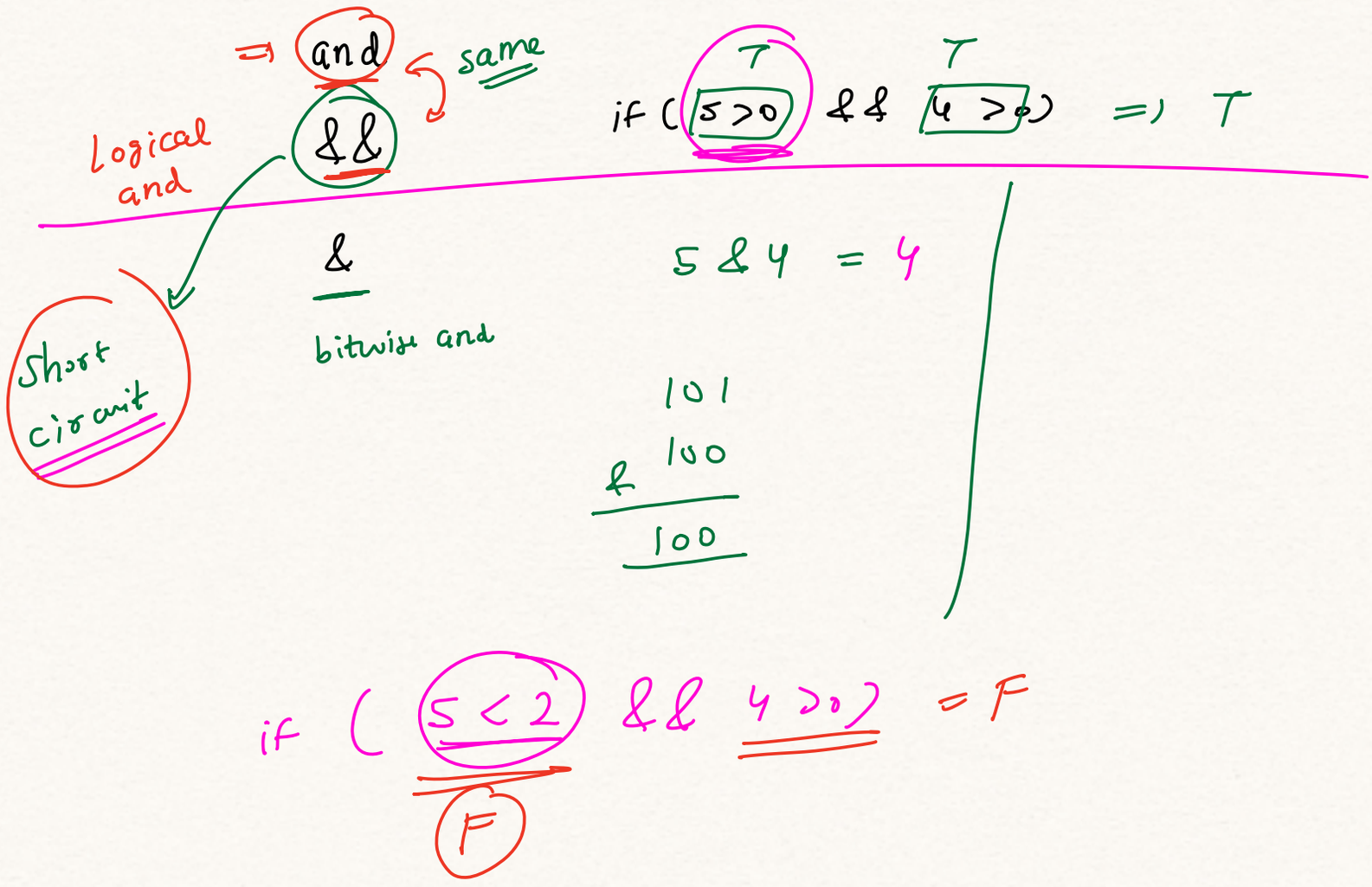
DFS



4, -11, -6

-8, 2, -4, 5,

6


=> and
same
&&
Logical and
Short circuit
if (5>0 && 4>0) => T
T
T
&
bitwise and
5 & 4 = 4
101
& 100
100
if (5<2 && 4>0) = F
F

Recursion-2

Revision Strategy

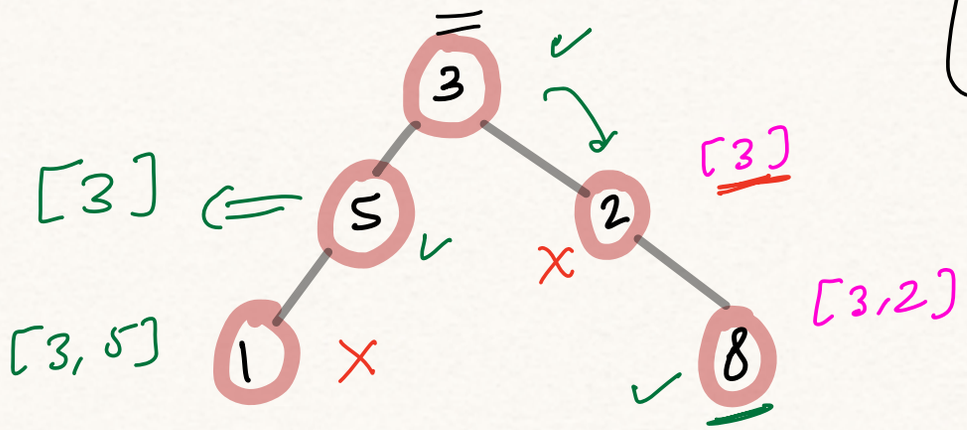
Solve problems daily

atleast 1

Bookmark

Problems

Try solving after few months again.



Shallow copy.

A = []

B = [1, 2, 3, 4, . . . , n]

A = B

O(1)