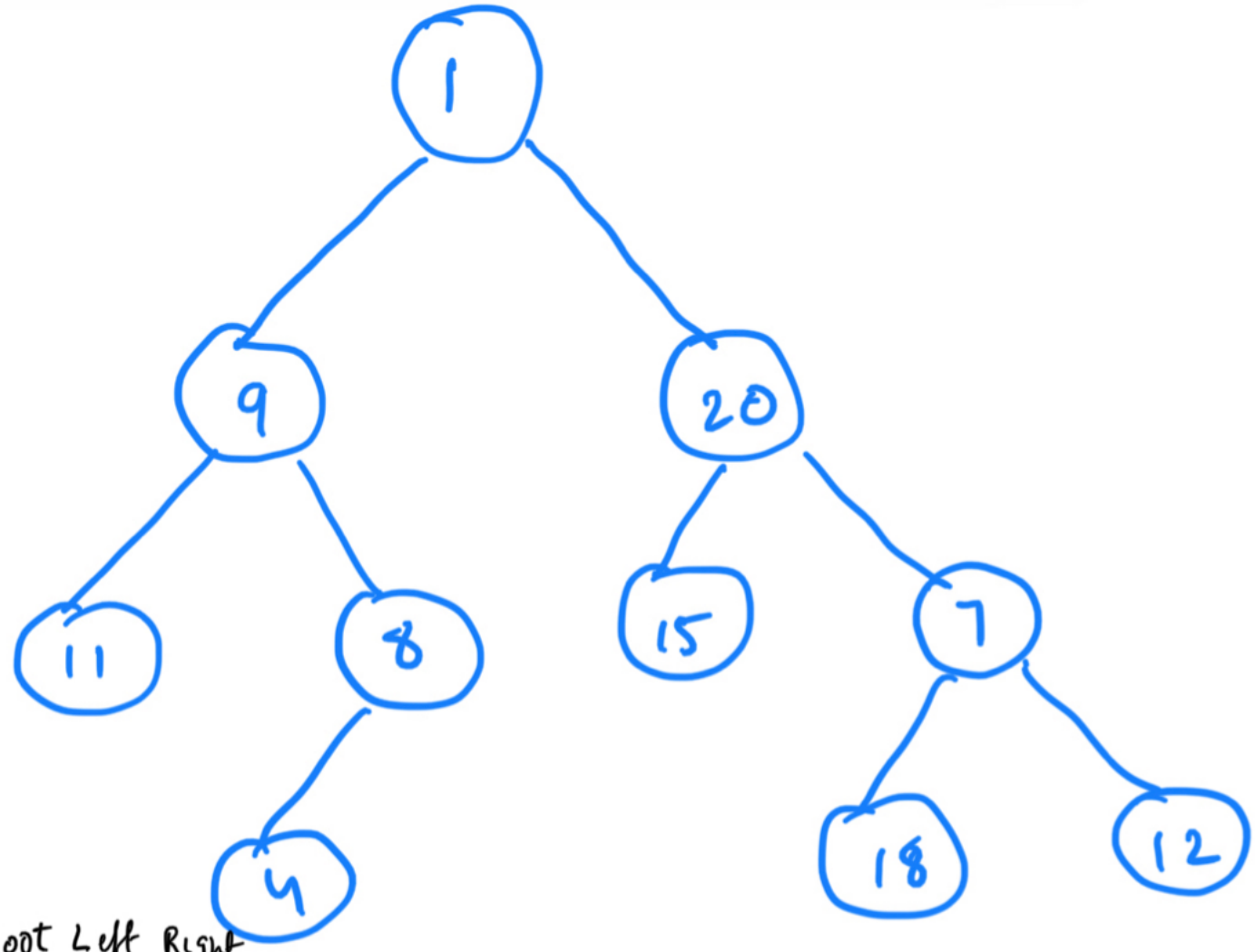
Trees-2



Root Left Right

Preorder: 1 9 11 8 4 20 15 7 18 12

Inorder: 11 9 4 8 1 15 20 18 7 12

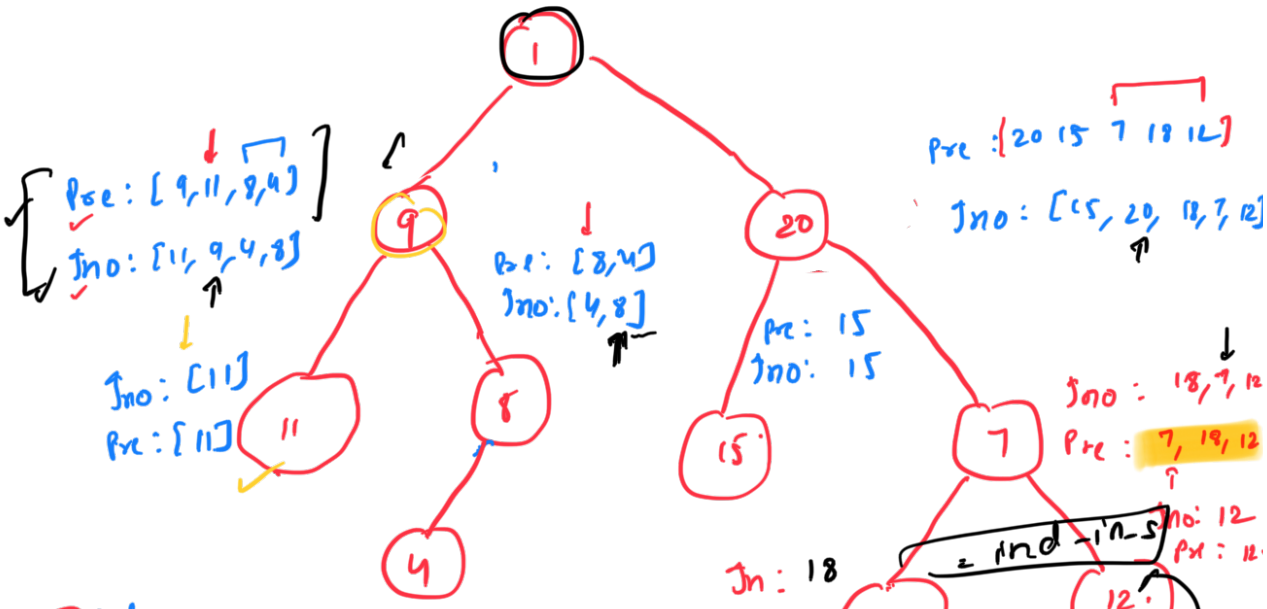
Left Root Right

Question: Construct tree from Inorder & Preorder

Root L R — Preorder:

L Root R — Inorder:

	0	1	2	3	4	5	6	7	8	9
Preorder:	1	9	11	8	4	20	15	7	18	12
Inorder:	11	9	4	8	1	15	20	18	7	12
	0	1	2	3	4	5	6	7	8	9



Pre: Root Left Right

Pre: [| | | | | |]

Subarray:

Ino: [| pre_st | Root | pre_e |]

in_s

L R+1 pre_e

Ino (LST): [in_s , ind-1]

L = ind - in_s

L ind+1, in_e]

Root: pre[pre_s]

Length: ind - in_s

Pre[LST]: [pre_s+1, pre_s + 1 + ind - in_s]

Pre(LST): [pre_s+1, pre_s + ind - in_s]

Pre(RST): [pre_s + ind - in_s + 1, pre_e]

start = pre_s+1

	0	1	2	3	4

end

A [1:3]

L R

R - L + 1

Start = 1

Length = 2

end = Start + length

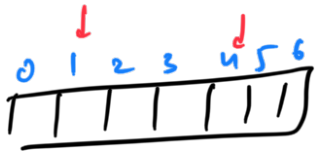
Length = ind - in_s

end = pre_s + 1 + ind - in_s - 1

= pre_s + ind - in_s

1 + 2 = 3

end = start + length - 1



start = 1

end = 4

end - start + 1

Assumption: constructTree (pre, pre_s, pre_e, ino, ino_s, ino_e) construct a tree using the given preorder & inorder and returns the root of the tree

```
Node constructTree (ino, in_s, in_e, pre, pre_s, pre_e){
    if( in_s > in_e || pre_s > pre_e){
        return NULL;
    }

    int root_val = pre[pre_s];
    Node root = new Node (root_val); }

    for(ind = in_s ; ind < in_e ; ind++){
        if( in[ind] == pre[pre_s]
            break;
    }

    R = pre_s + ind - in_s ;

    root.left = constructTree ( ino, in_s, ind-1,
                                pre, pre_s+1, R);

    root.right = constructTree ( ino, ind+1, in_e,
                                 pre, R+1, pre_e);

    return root;
```

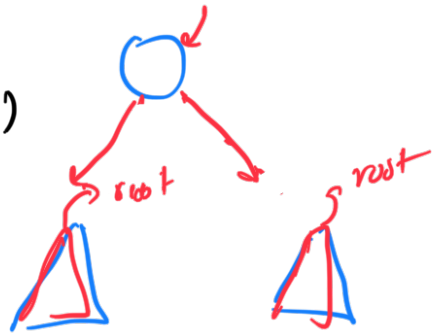
Hashmap = O(1)

O(n)

}

T.C: # Function calls × T.C per Function : O(n)

n

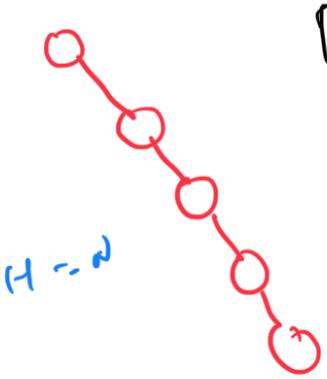


$\boxed{T.C: \quad O(n^2)}$

$\boxed{S.C: O(H)}$ Recursion Stack

O(N) worst case

O(H)



H = N

New T.C: O(N)

S.C: O(N) → Hashmap

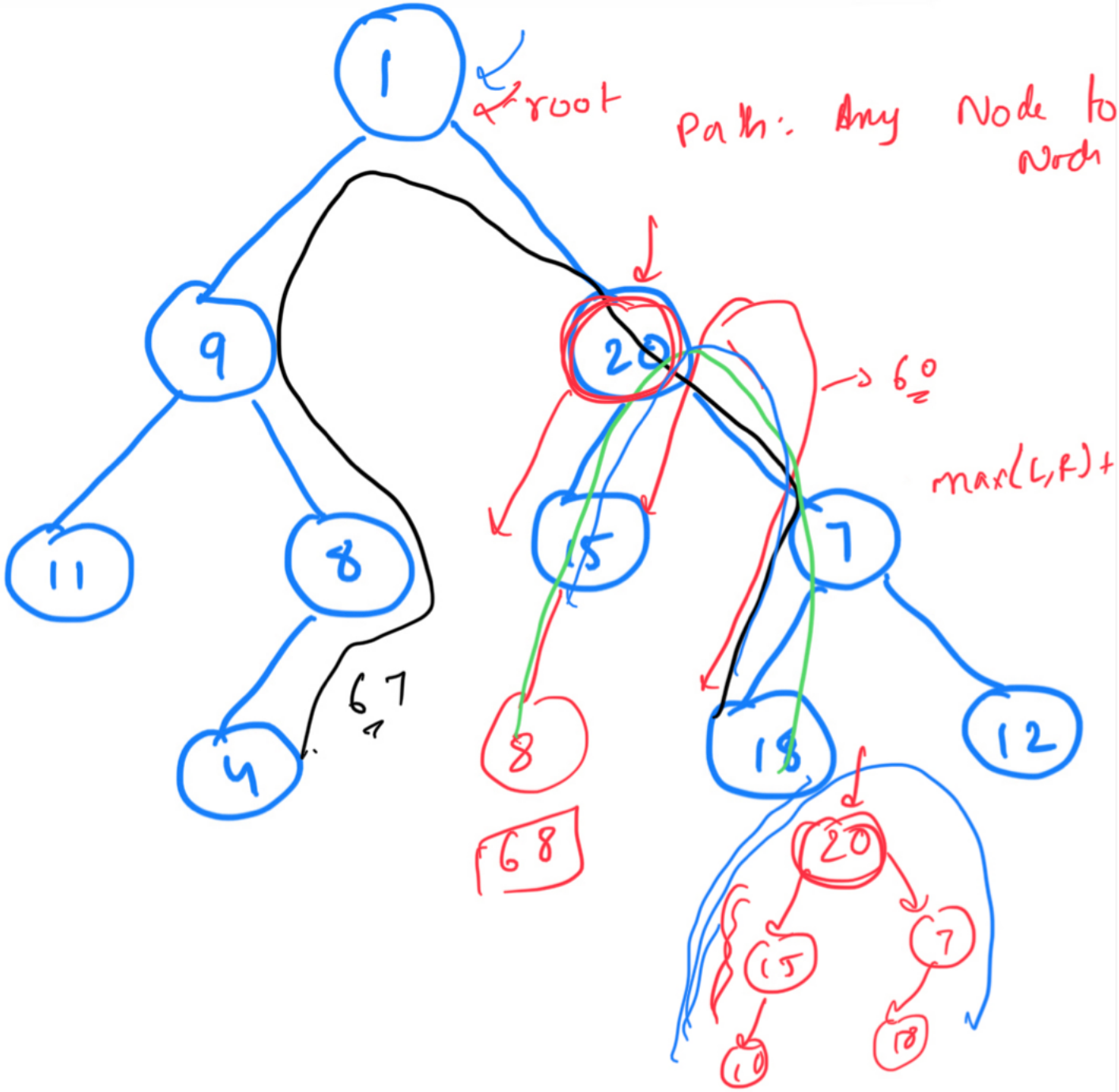
Space Complexity: the extra space apart from input and output.

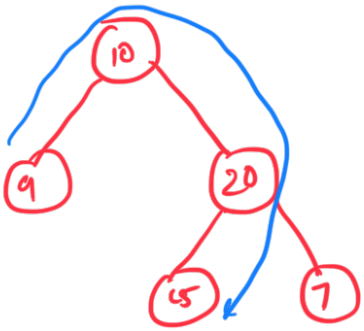
→ Recursion Space

→ Tree

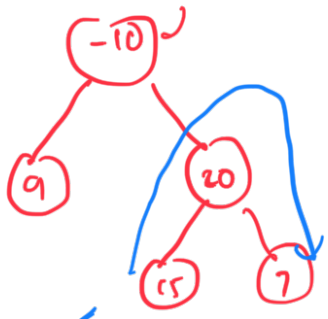
→ Preorder, Inorder.

Question: Maximum Path in a B.T

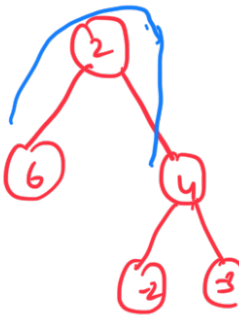

1
root
Path: Any Node to Node
9
20
60
max(L,R)+
11
8
15
7
67
4
8
18
12
68
20
15
7
10
18



54/2



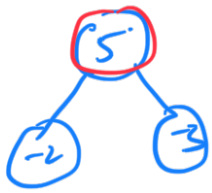
42/2



12/2

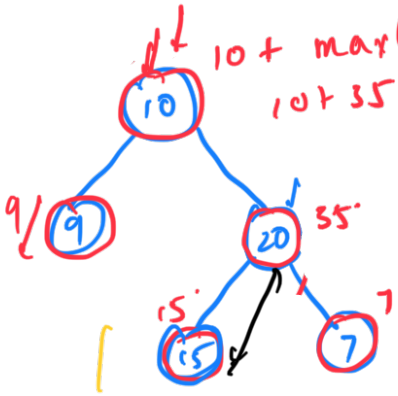
1) Need not always pass through root

2) Need not always end at a leaf



Max Sum = $\boxed{5}$

Find maximum sum path where atmost one child of a node is involved

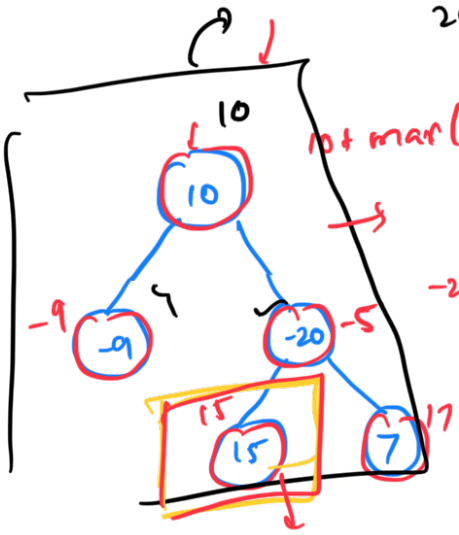


$10 + \max(9, 55)$

$10 + 55 = \boxed{45}$

root.data + max(left, right)

$20 + 15 = 35$



$10 + \max(-9, -5, 0)$

$-20 + 15 = -5$

~~ans = INT_MIN~~;

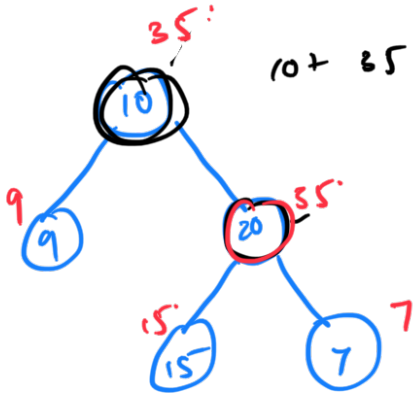
(root) {

int maxSum(...)
if(root == NULL) return 0;

{
left = maxSum(root.left);
right = maxSum(root.right);
max_sum_one_child = root.data + max(left, right, 0)
max_sum_two_children = root.data + max(left, 0) + max(right, 0)
ans = max(ans, max_sum_two_children);
return max_sum_one_child;
}

Height

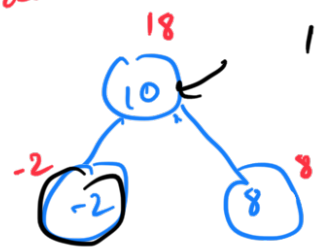
Original Problem:



10 + 35 + 9 = 54

20 + 15 + 7 = 42

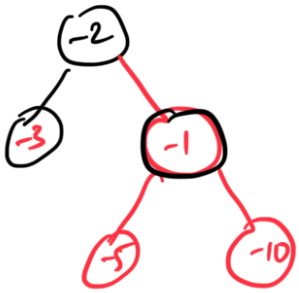
= root.data + left + right;



10 + 8 - 2

10 + max(left, 0) + max(right, 0);

Interviewer



Atleast Node: -1

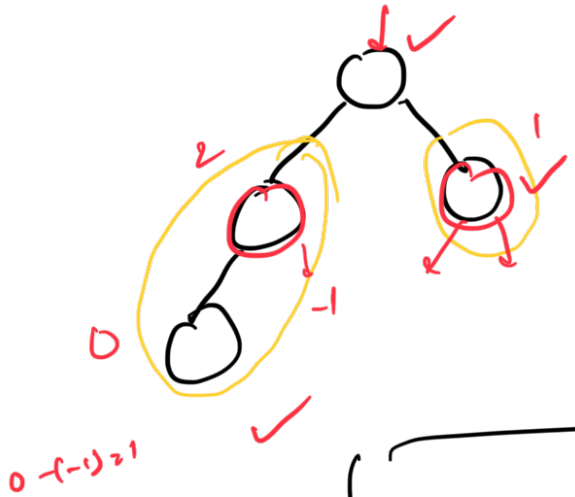
Ans: 0

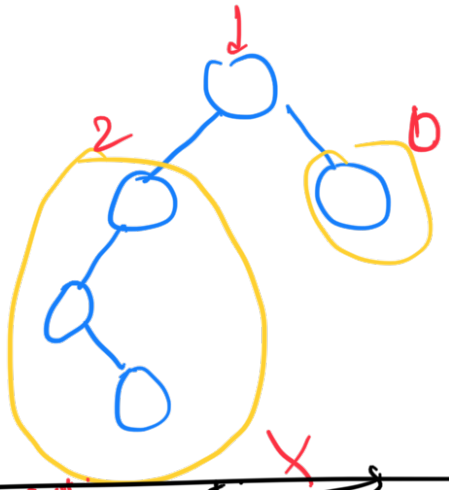
Diameter of a tree

Question: Diam...

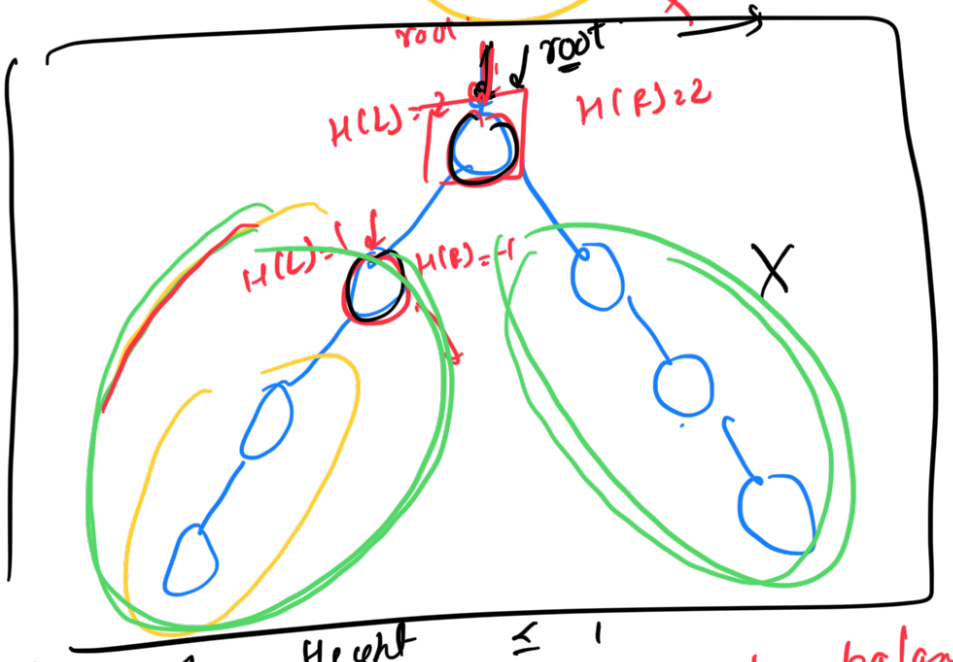
Question: check if a tree is height-balanced.

Absolute difference of heights of LST and RST of every node is ≤ 1





1 = 0



1) Difference of Height ≤ 1
2) LST and RST should be balanced

Assumption: isHtBal(root) and returns whether the tree is height balanced or not

Fail-Fast

boolean isHtBal(root)?

...NULL) return true;

boolean

if(root = ...

Preorder

Rid of this?

left_ht = getHeight(root.left);
right_ht = get_Height(root.right);

if(|left_ht - right_ht| < 1 && isHtBal(root.left) && isHtBal(root.right))

False

return true;
else return false;

}

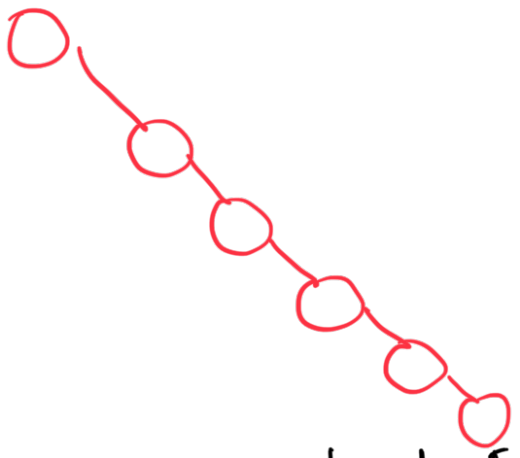
T.C:

No of Function Calls × T.C Per Function

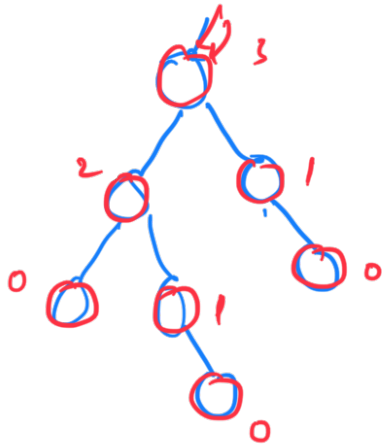
N × H

N

worst Case: $O(N^2)$



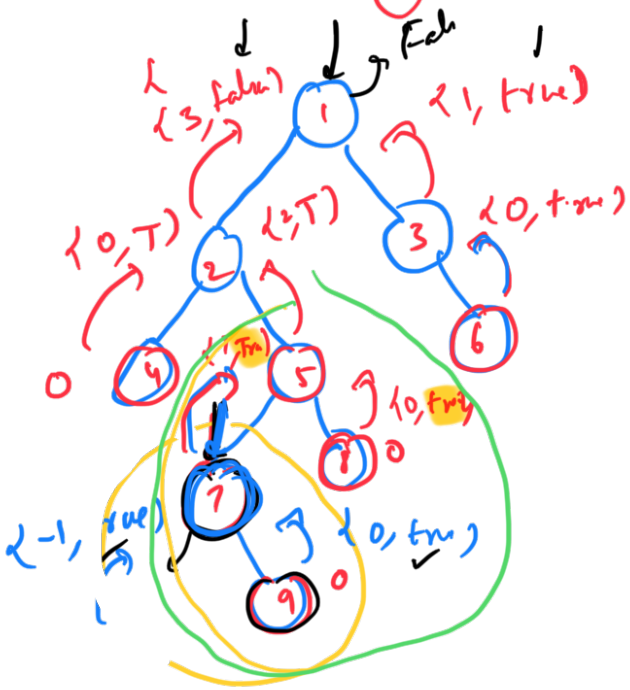
3-1 > 1



O(N)

Postorder

Postorder



H(LST) = -1
H(RST) = 0

1) Height
2) isBal

... int, boolean

1) pair pair <int, ...> max(-1, 0) + 1

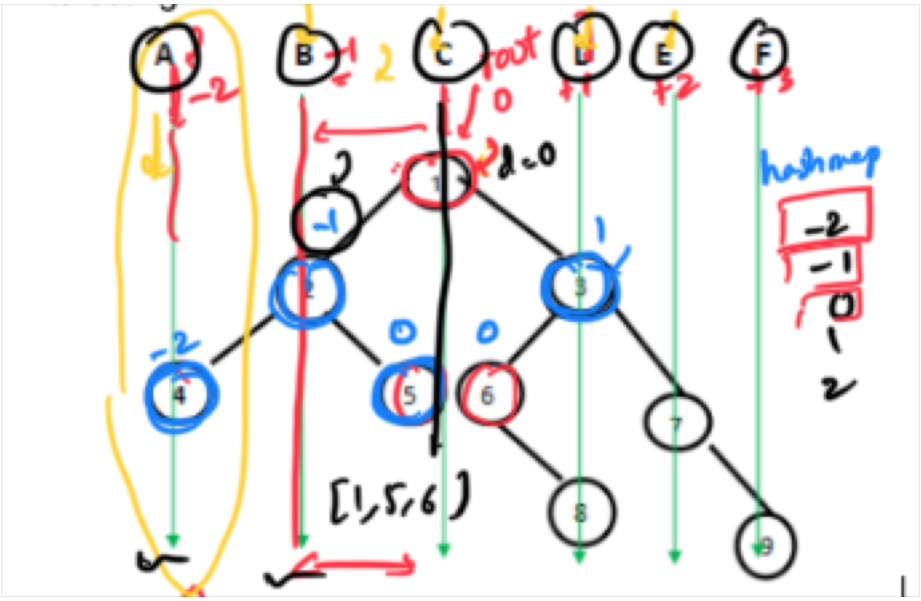
2) class Info {
    int ht;
    boolean isBal;
}

```
Info isHtBal(root) {
    if(root == NULL) {
        return <-1, true>;
    }
    left = isHtBal(root.left);
    right = isHtBal(root.right);

    if(|left.ht - right.ht| <= 1 && left.isBal == true && right.isBal == true) {
        return Info(max(left.ht, right.ht) + 1, true);
    } else return Info(ht, false)
}
```

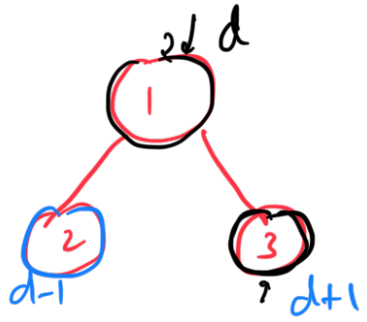
T.C: O(n)
S.C: O(H) => O(N)

Question: Vertical Order Traversal



Output: Nodes with same Horizontal distance from root will lie on the same line

[
A [4]
B [2]
C [1, 5, 6]
D [3, 8]
E [7]
F [9]
]



Hash Map
< Key, value >
Key → Horizontal from Root
value → List / Array

unordered-map
Tree Map

Use a Tree Map: keys are stored in sorted order:

map<int, List> mp;

vertical Order (root, 0)

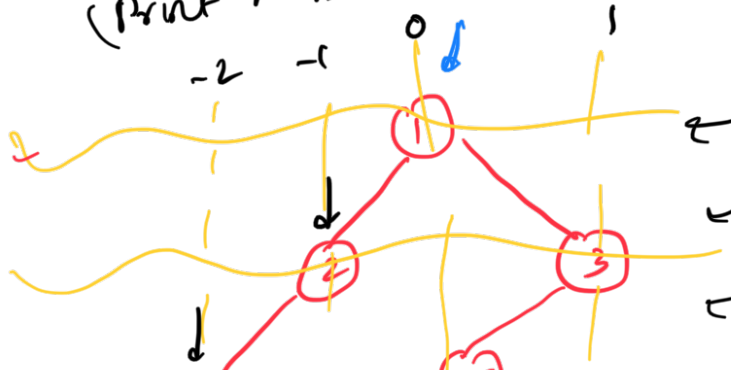
void verticalOrder (root, d) {
if(root == NULL) return;

mp[d].push(root.data);
→ verticalOrder(root.left, d-1);
verticalOrder(root.right, d+1);
}

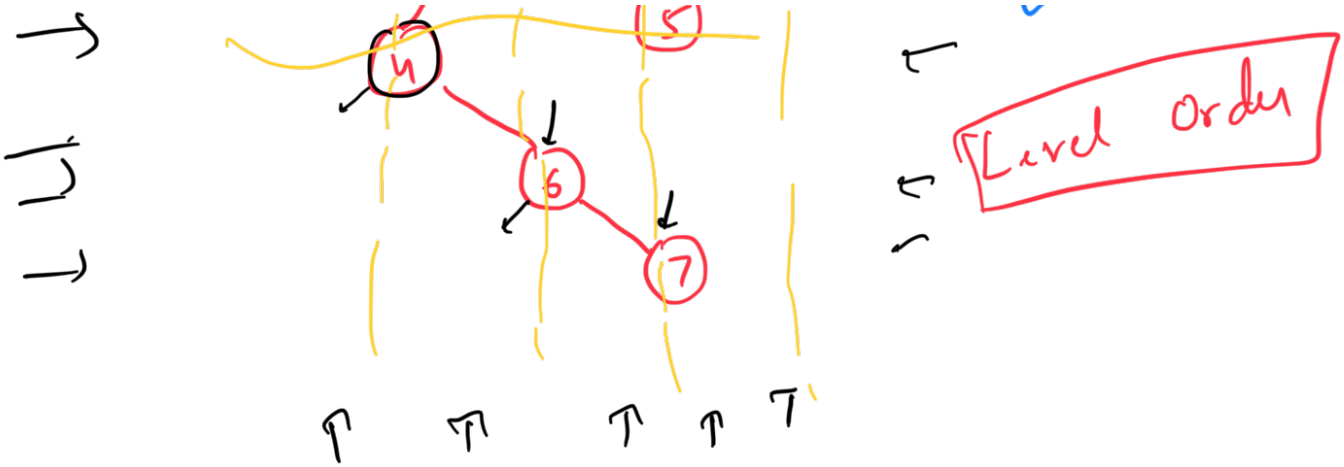
(Print Map):

0: [1, 5, 7]

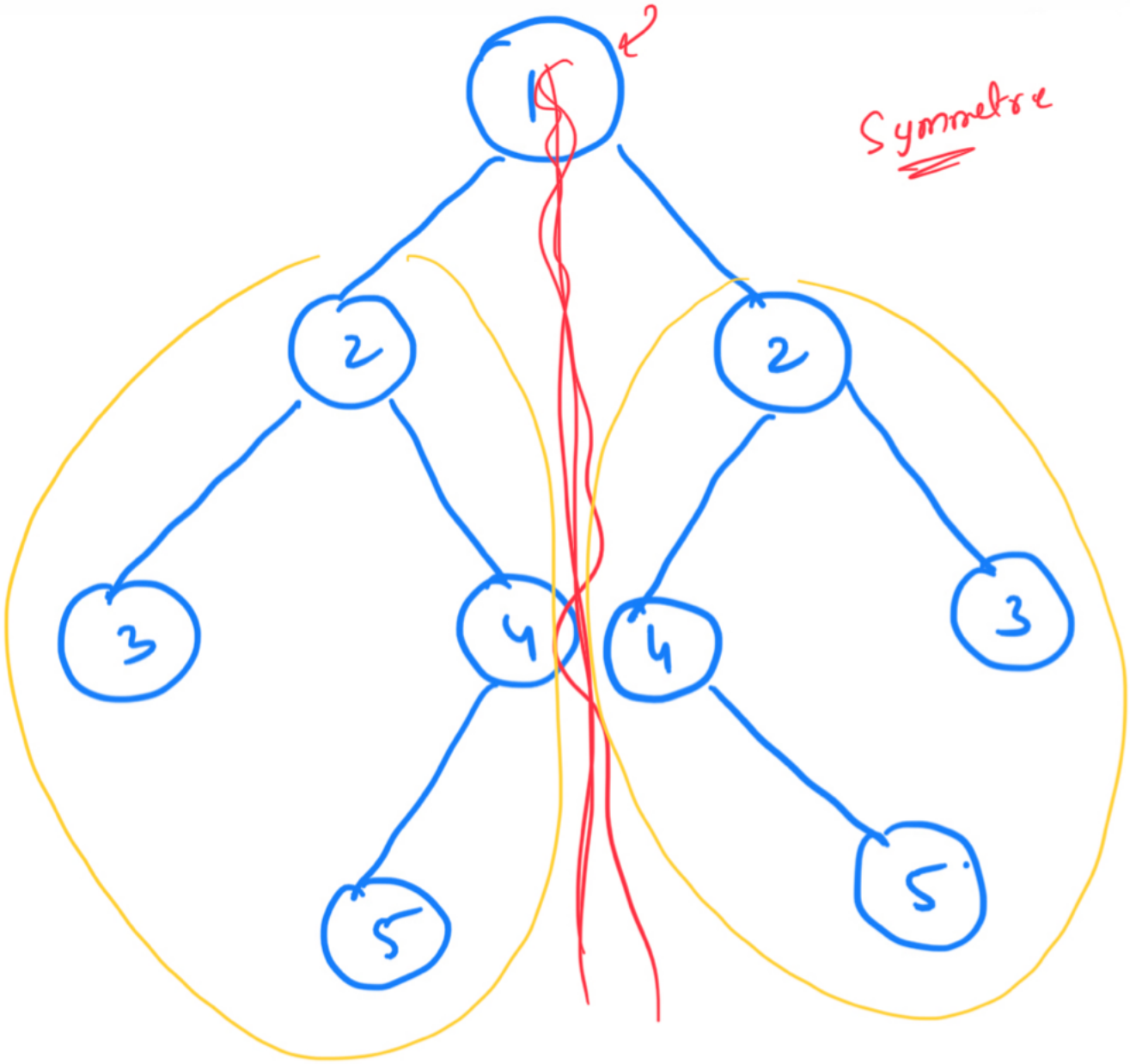
Left View

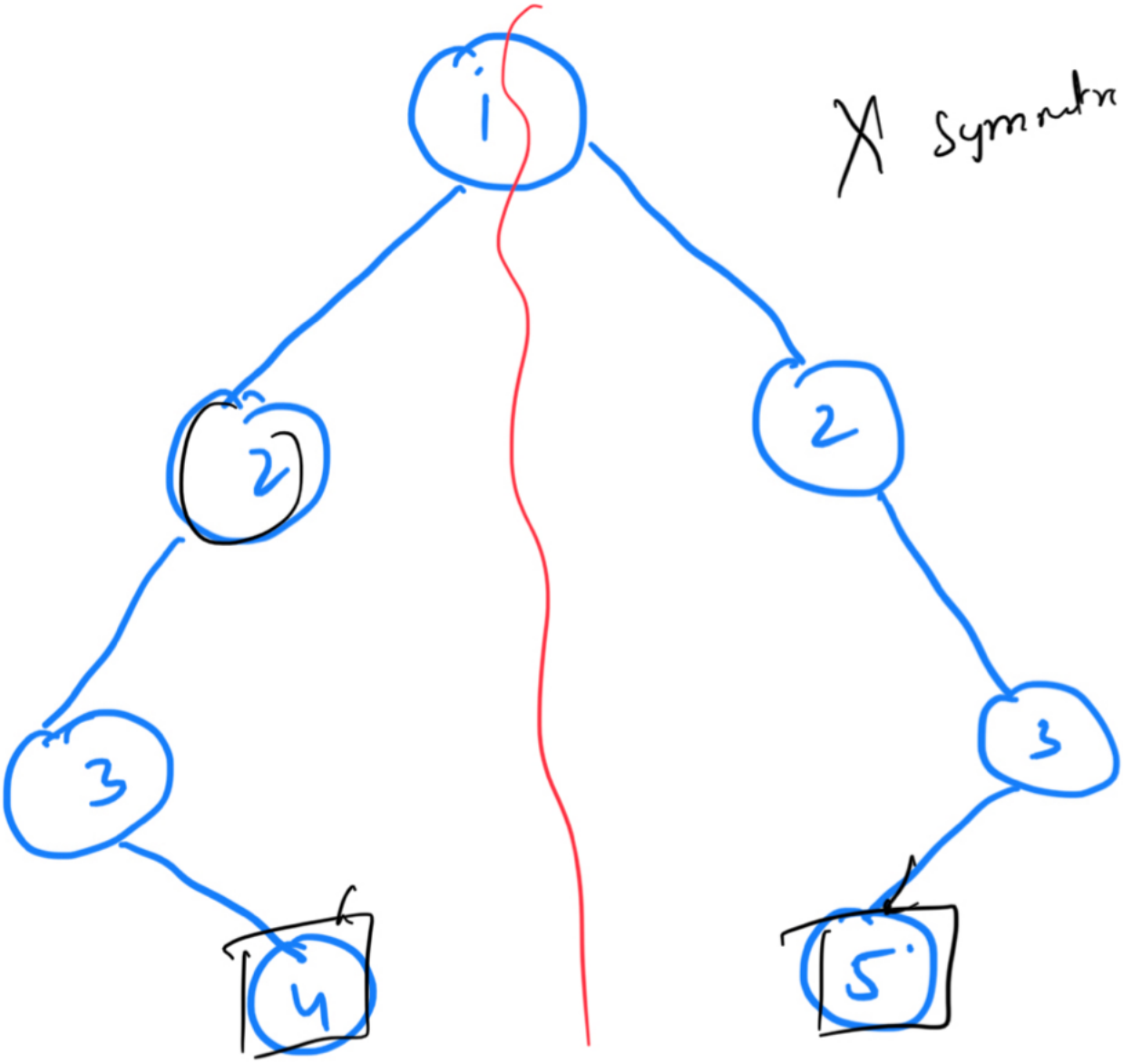


map:
-2: [-2
-1: [2, 6
0: [1, 7]
1 {
[1, 5, 7]



Question: check if a tree is symmetric

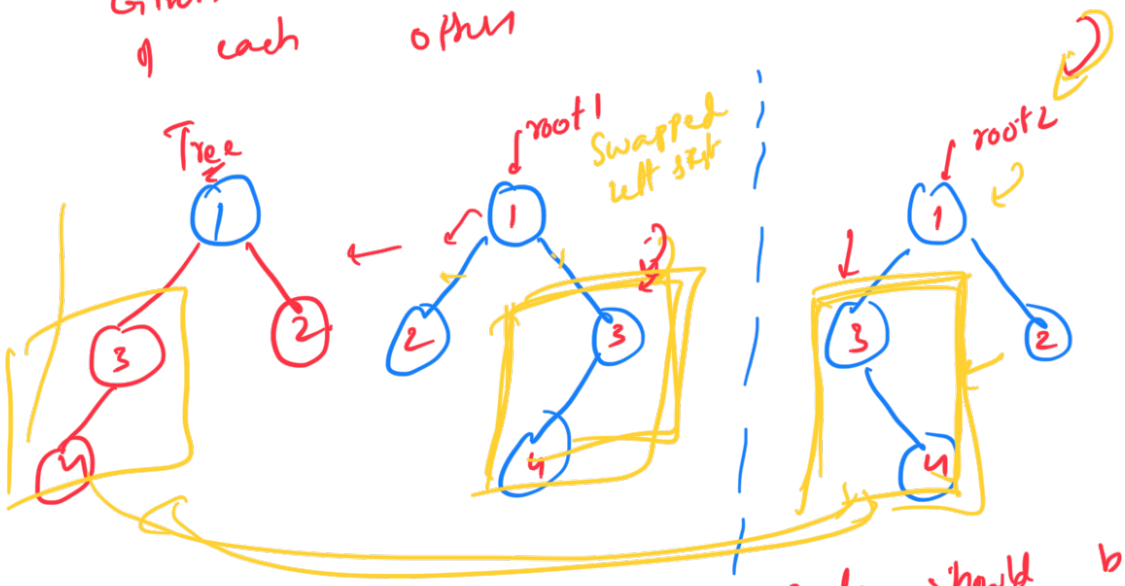

1
2
2
3
4
4
3
5
5
Symmetre



X symmetric

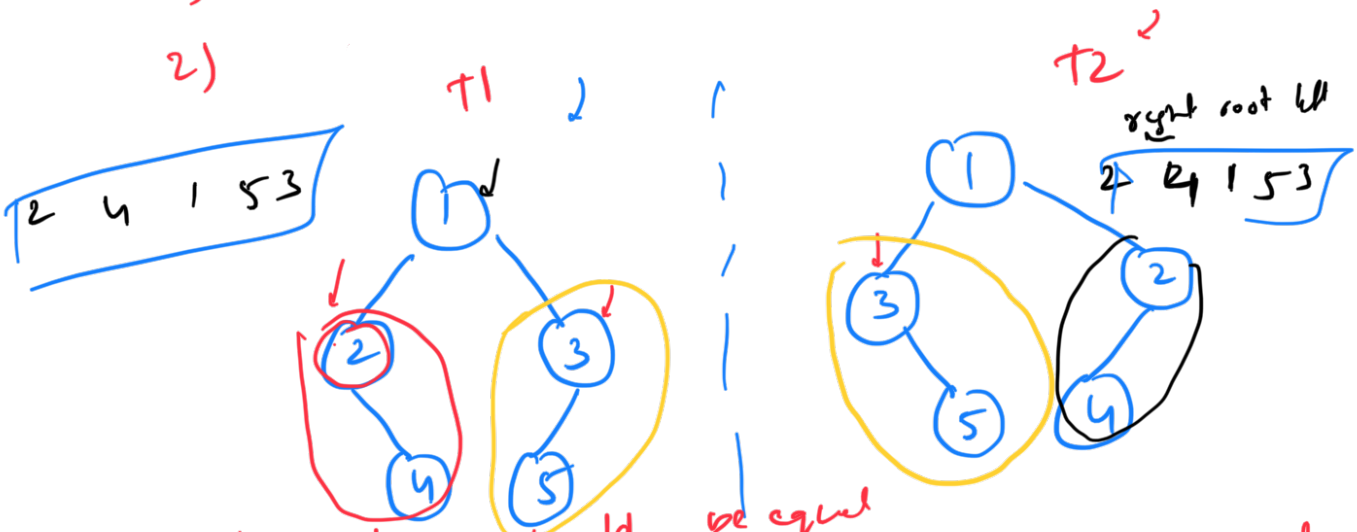
New Subproblem :

Given 2 trees, check if they are mirror images of each other



1) Values at root nodes should be equal

2)



1) Root values should be equal

2) Left child of T1 is mirror image of Right child of T2

3) Right child of T1 is mirror image of Left child of T2

Assumption : isMirror (root1, root2) returns if the 2 trees are mirror images of each other or not

(root1, root2) {

... NULL) return true

```
boolean isMirror(root1, root2)
    if( root1 == NULL && root2 == NULL || root2 == NULL) return true
    else if( root1 == NULL || root2 == NULL) return false

    if( root1.data == root2.data &&
        isMirror( root1.left, root2.right) && isMirror( root1.right, root2.left) )
            return true;
    else    return false;
}
```

isMirror(root.left, root.right).

T.C: O(N)

S.C: O(H) => O(N)

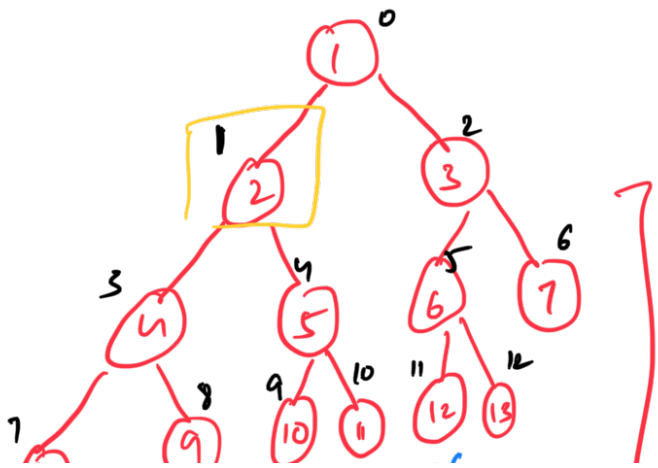
Unordered mp

Tree Hashmap mp;

Sorted Ordu

print key

-3	10	-6	.	.



unordu

O(n)

min dsla

[-2]

mp[-1] => O(n)

mp[0]

mp[1]

mp[2]

4+2 26

Level order Tree

Array

1	2	3	4	5	6
0	1	2	3	4	5

mp[3]

$\boxed{i} \Rightarrow$ parent children

Parent Children

0 => 1, 2

1 => 3, 4

2 => 5, 6

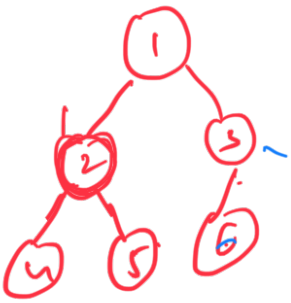
3 => 7, 8

4 => 9, 10

5 => 11, 12

2×2+1

2×2+2 26



i => 2i+1 => left

2i+2 => right

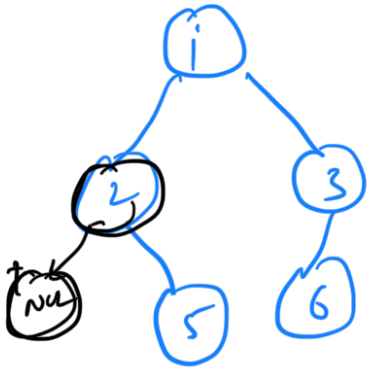
left = 2x

right = x+1

2×0+1

2+0+2

$\boxed{i \Rightarrow 2i+1,\ 2i+2.}$



```
bt_from_in_pre(inorder[], preorder[], in_s, in_e, pre_s, pre_e){
      if(in_s > in_e) return;

      int root_val = pre[pre_s];
      Node* root = new Node(root_val);

      for(int i = in_s; i <= in_e; i++){
            if(in[i] == root_val)
                  break;
      }

      r = pre_s + i - in_s;
      root->left = bt_from_in_pre(ino, pre, in_s, i - 1, pre_s + 1,
r);
      root->right = bt_from_in_pre(ino, pre, i + 1, in_e, r + 1,
pre_e);

return root;
}
```

Check if a tree is Height balanced

```
class Info{
      boolean isBal;
      int ht;
}

Info isBal(root){
      if(root == NULL){
            return new Info(true, -1);
      }
      Info left = isBal(root.left);
      Info right = isBal(root.right);
      if(left.isBal && right.isBal && |left.ht - right.ht| <= 1){
            return new Info(true, max(left.ht, right.ht) + 1)
      }
      else
            return new Info(false, -1);

}
```

Check if one tree is a mirror of other

```
boolean isMirror(root1, root2){
      if(root1 == NULL && root2 == NULL) return truel;
      if(root1 == NULL || root2 == NULL) return false;

      if(root1.data != root2.data)
            return false;
      return isMirror(root1.left. root2.right) &&
             isMirror(root1.right, root2.left);

}
```