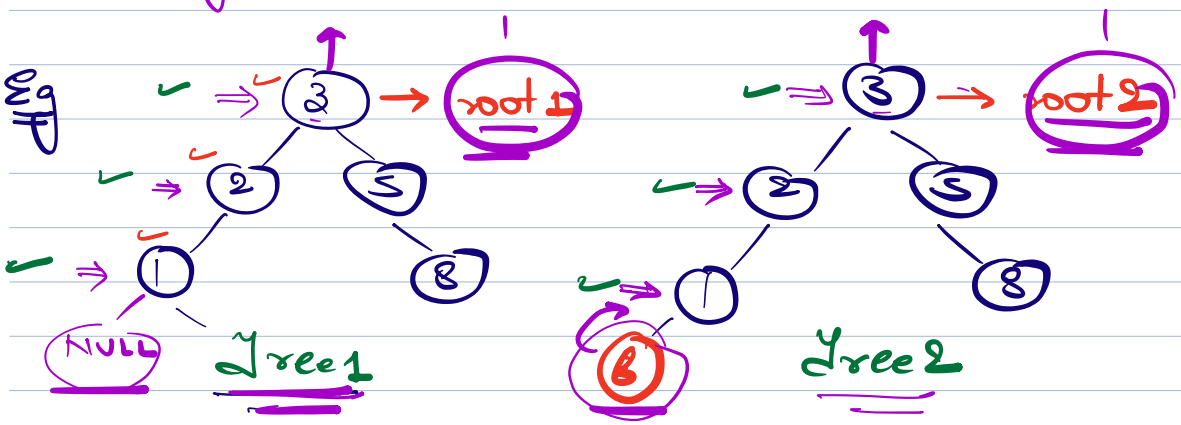
Search in Binary Tree

⇒ Iterating over the tree

		T.C.
#nodes = N	⇒ PreOrder	O(N)
	⇒ Inorder	O(N)
	⇒ PostOrder	O(N)

Q1 Given two Binary trees, check if they are identical or not.??

Eg



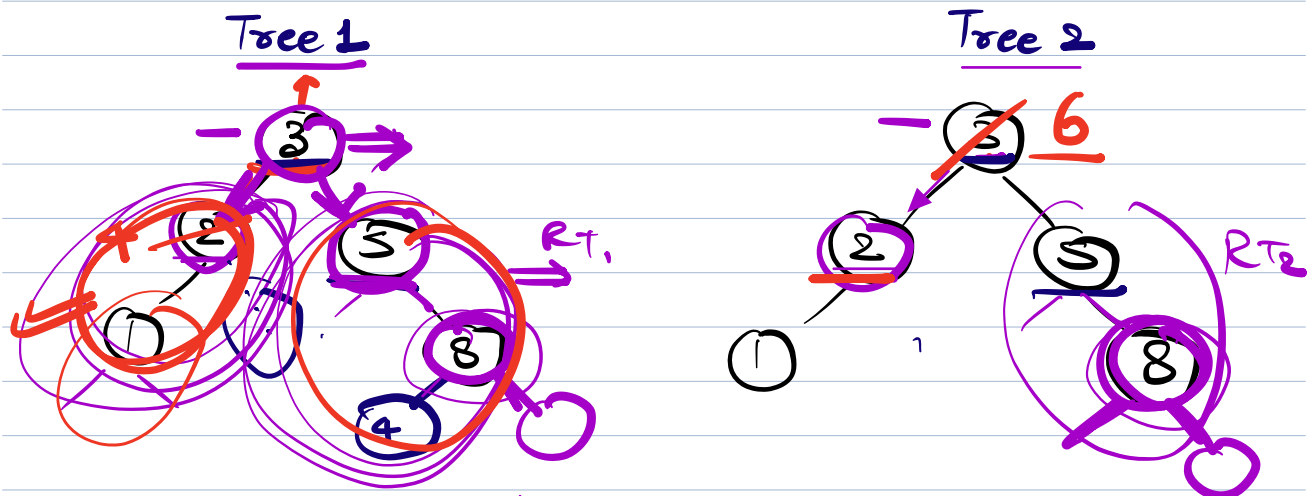

```
boolean check ( root1 , root2) {
```

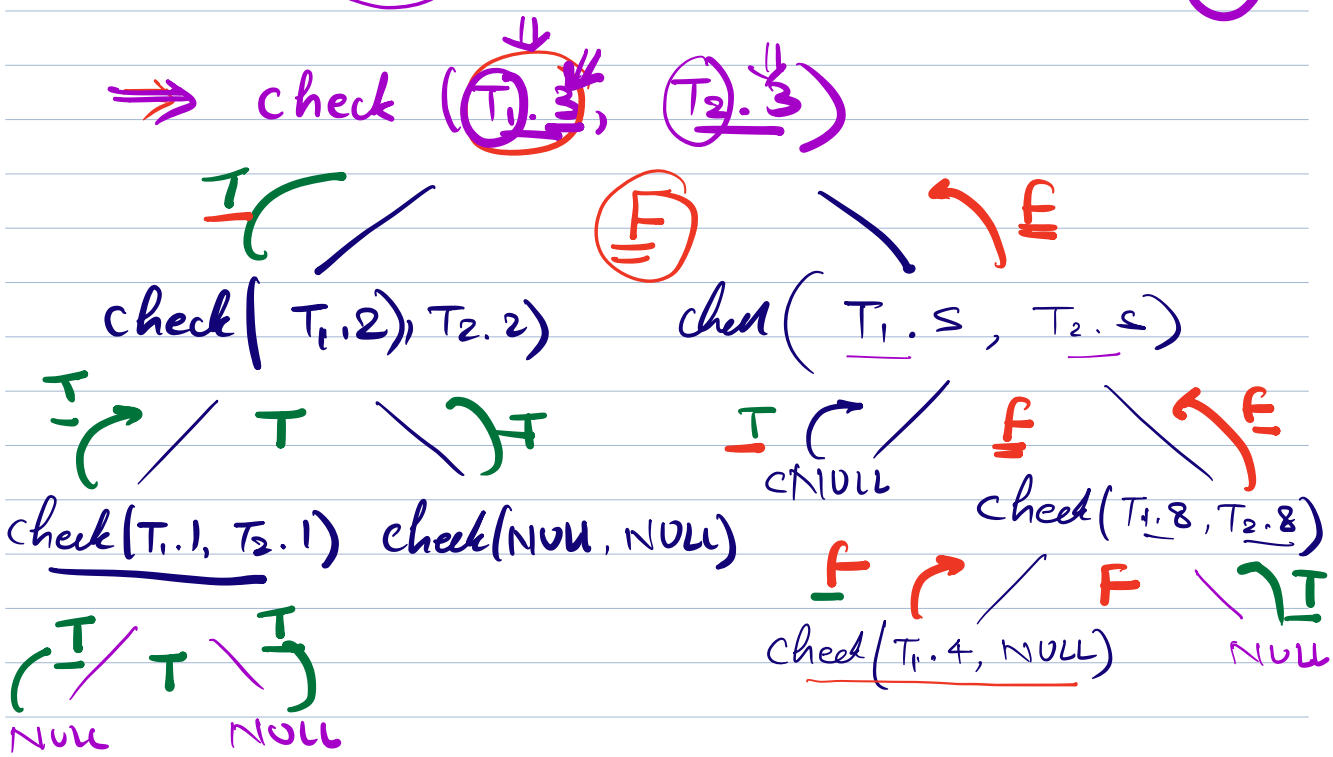
1
```
if ( root1 == NULL && root2 == NULL) {
    return true;
}
```

2
```
else if ( root1 == NULL || root2 == NULL) {
    return false;
}
```

3
```
else if (root1.val != root2.val) {
    return false;
```

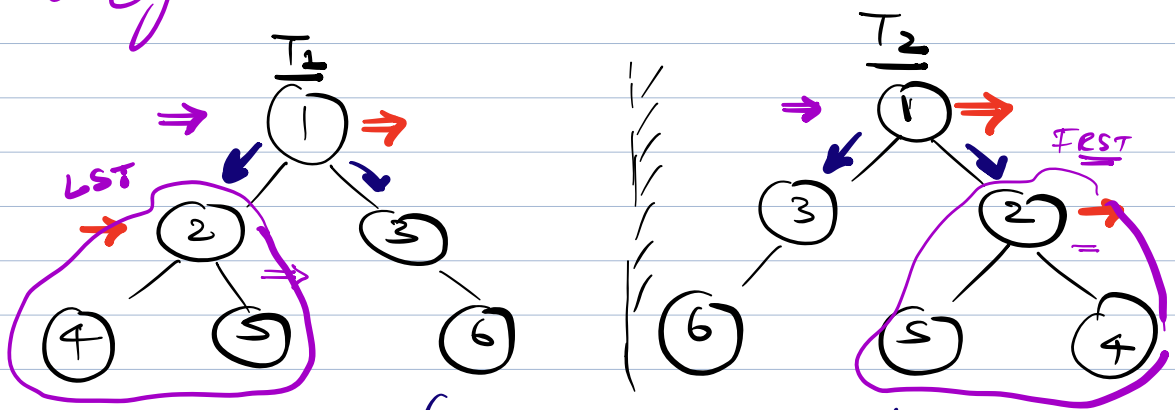
```
return ( check( root1.left, root2.left);
                && check( root1.right, root2.right);
);
}
```





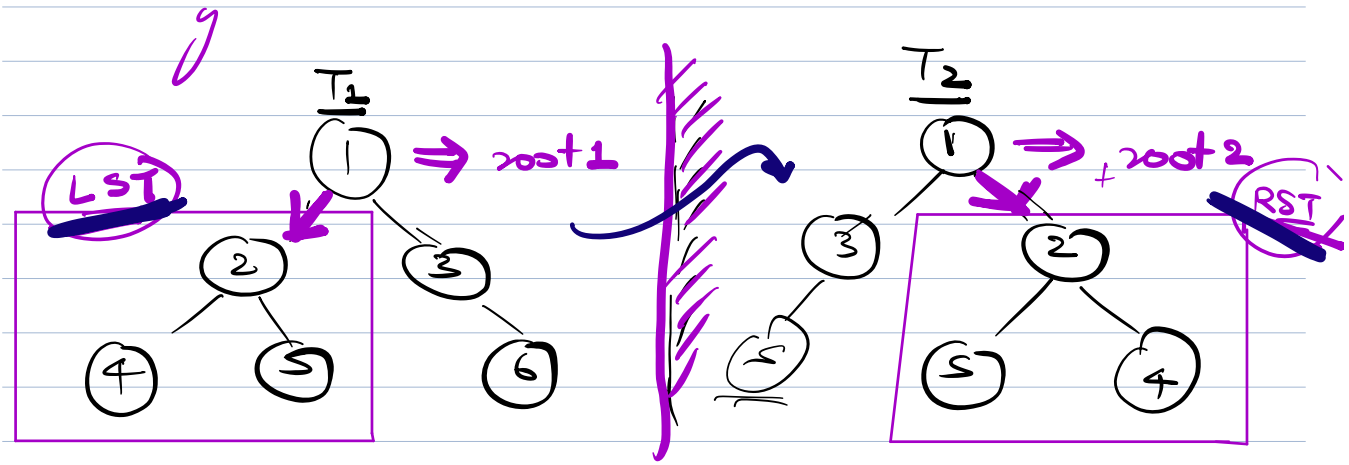
⇒ Using pre order

Q2 Given 2 BT. Check if they are Symmetrical.




```
boolean check ( root1 , root2) {

1 { if ( root1 == NULL && root2 == NULL) {
        return true;
  }

2 { else if ( root1 == NULL || root2 == NULL) {
        return false;
  }

3 { else if (root1.val != root2.val) {
        return false;
  }

  return ( check(root1.left, root2.right);
           &&
           check(root1.right, root2.left);
  );
```

5



check (root1 , root2)

⇓

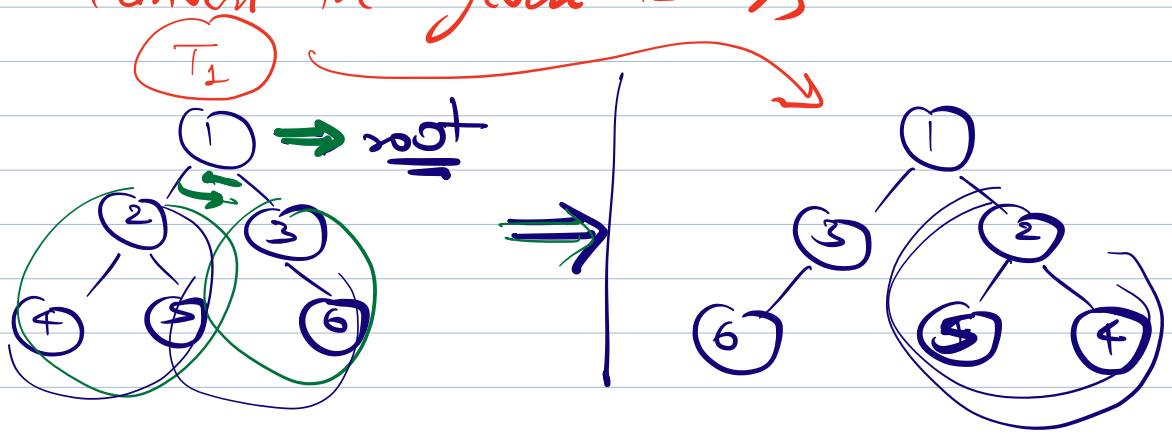
Ass: Returns true if tree rooted at root1 is symm. to tree rooted at root2

Q3 Given a BT. Return the inverted BT

⇓

mirror image

(Invert the given BT);



Approach 1

```
Node invert (root) {
    if (root == NULL) {
        return;
    }
    swap(root.left, root.right);
    invert(root.left);
    invert(root.right);
    return root;
}
```

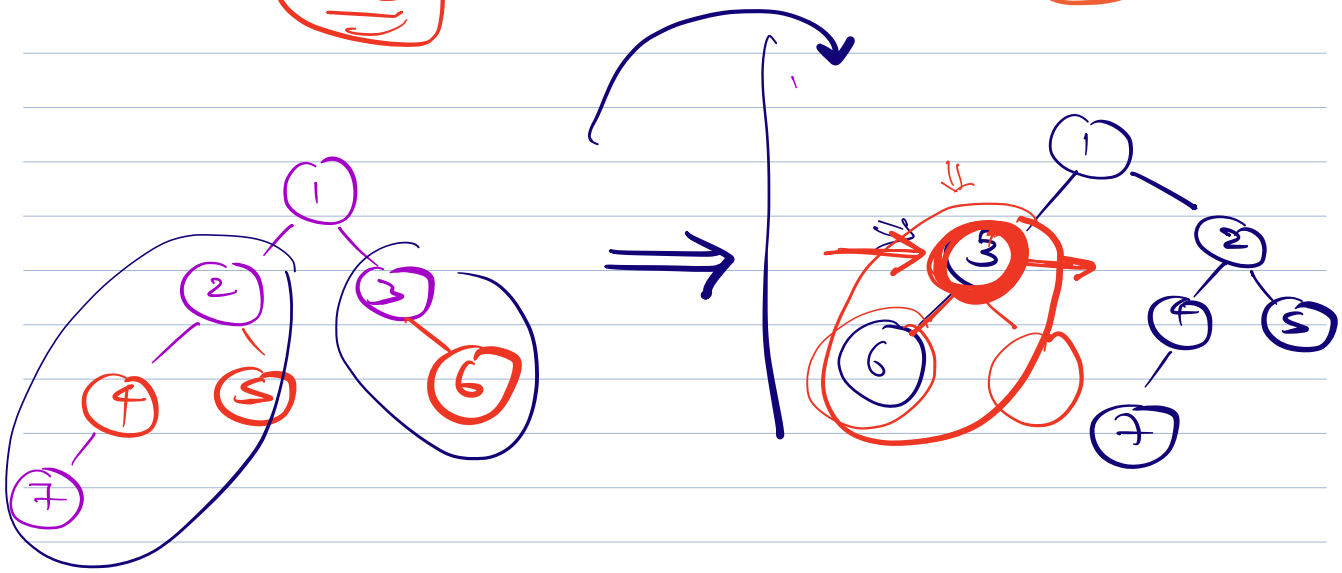
Post

⇒ Pre Order Traversal

⇓

Post

root.left = root.right



Search T.C.

Array : N

LL : N

Binary Tree : N

★ Binary Search Tree

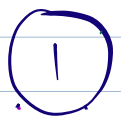
Prop

1) values of all element of LST ≤ Root.value

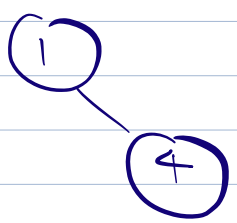
2) values of all ele. of RST > Root.value.

↳ for all nodes of tree

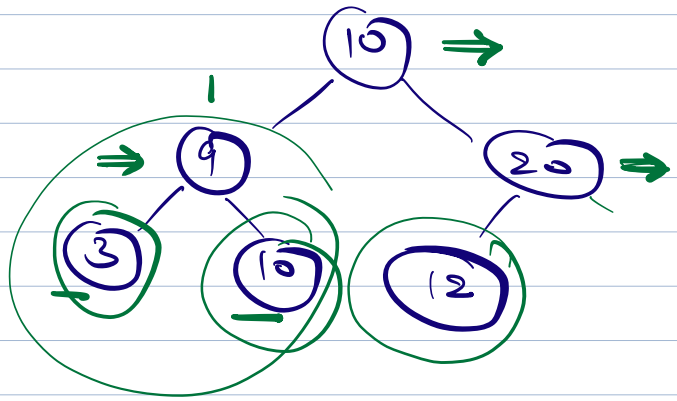
Eg 1)



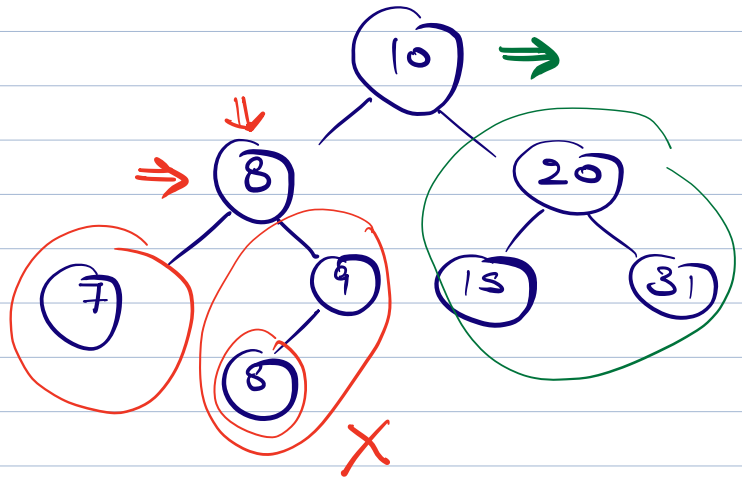
2)



3)



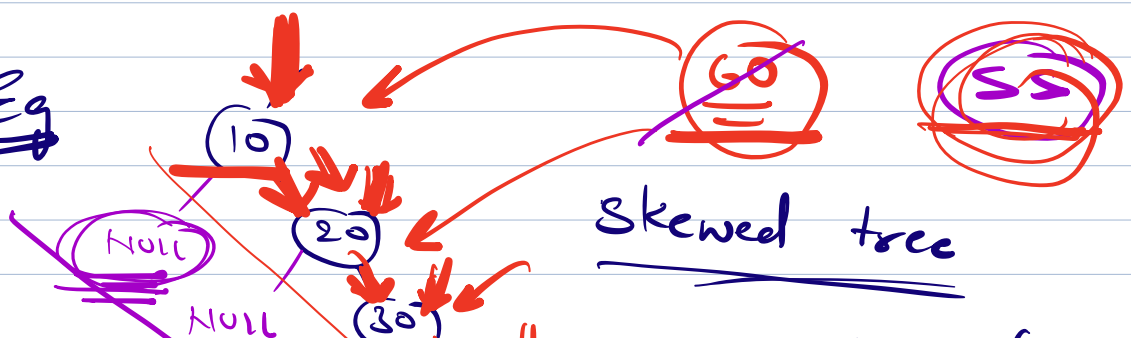
4) X

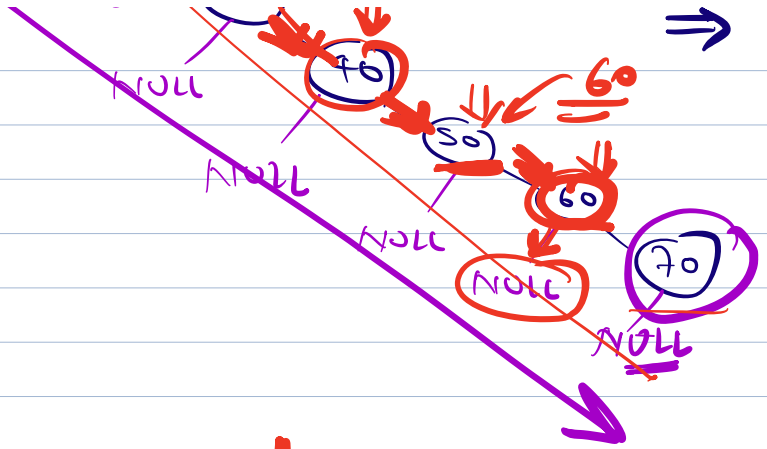


T.C. of search in BST with N nodes??

⇒ log(N) X

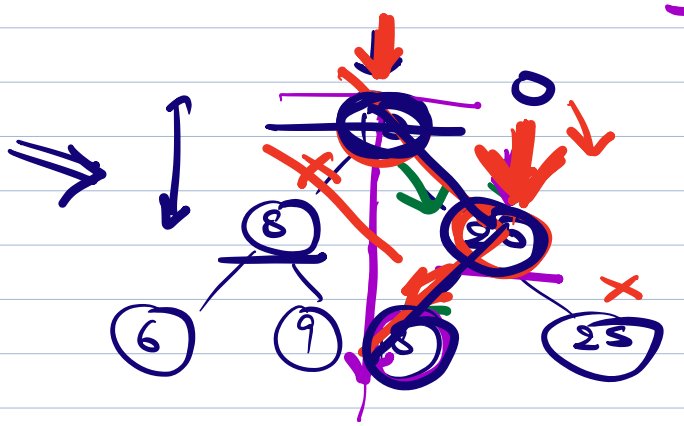
Eg





$\Rightarrow O(N)$

$H = O(N)$



18

Height of the Tree

O(Height)

N

Depth

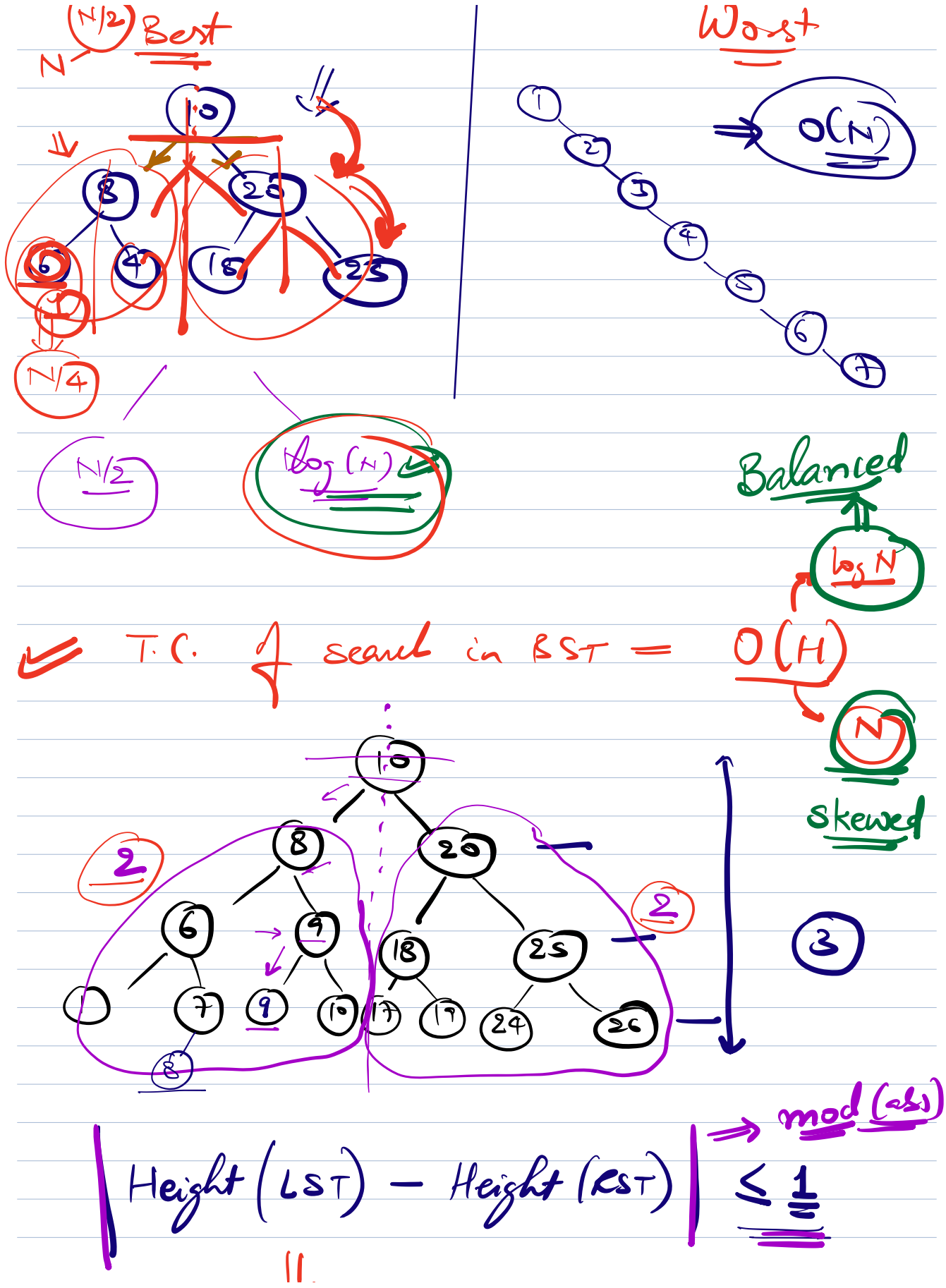
Node

Height

Face

Depth of the farthest leaf node

7 1 6


N/2
Best
10
8
20
6
4
18
25
N/4
N/2
log (N)
Worst
1
2
3
4
5
6
7
O(N)
Balanced
log N
T.C. of search in BST = O(H)
N
skewed
10
8
20
6
9
18
25
1
7
9
10
17
17
24
26
8
2
2
3
| Height (LST) − Height (RST) | ≤ 1
mod (abs)

Balanced ⇒ AVL, RB Tree

⇓

O(log N)

Doubt

Given ⇒ root

LIFO

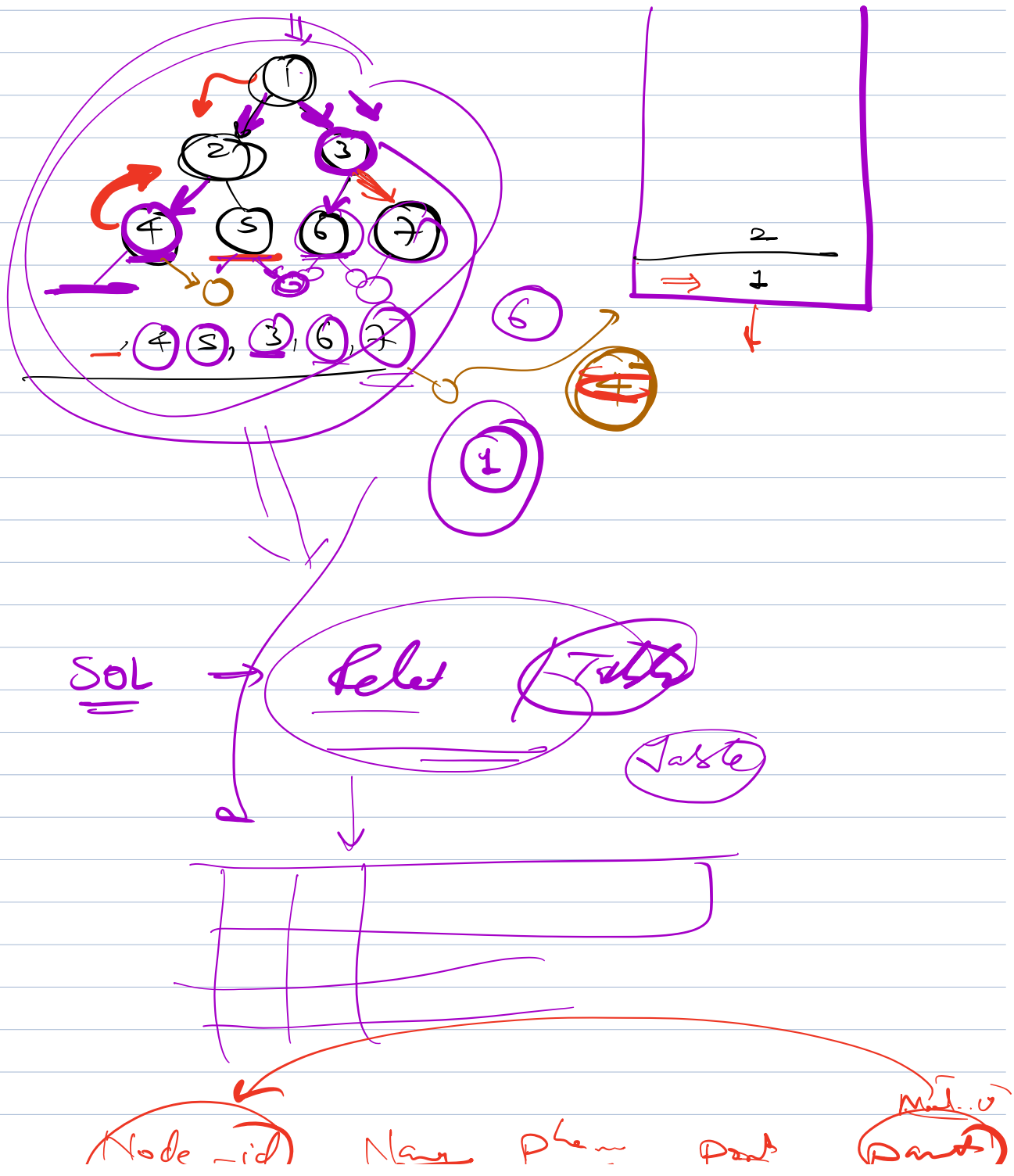
Pre → N, L, R,

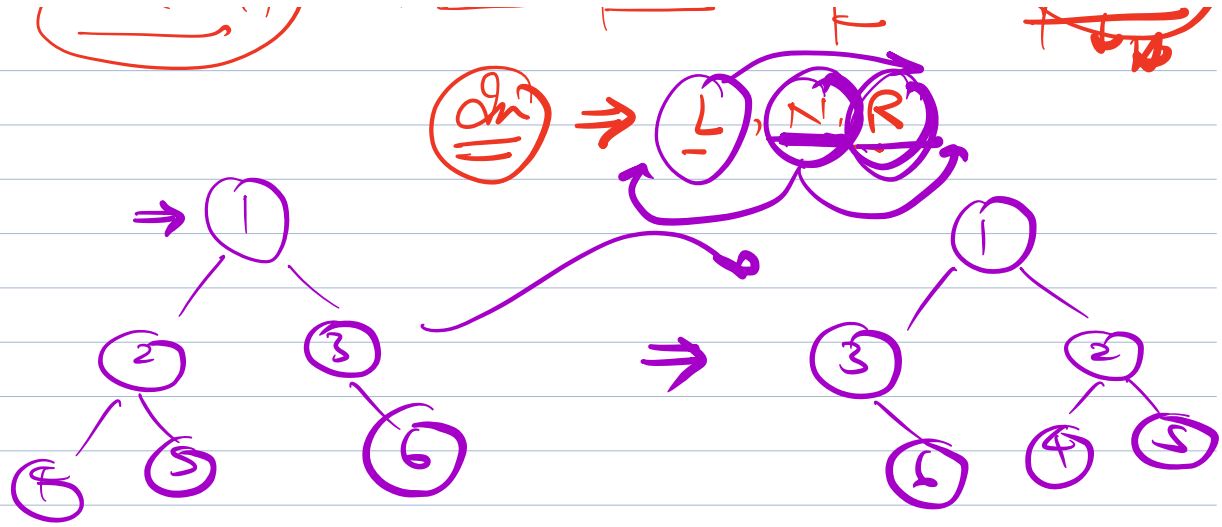
Post → L, R, N

In → L, N, R

```
stack<Node> st.
st.push(root);
print(root.data);

while (          ) {
    curr = st.top();
    if (curr.left != null) {
        curr = curr.left;
        st.push(curr);
        print(curr.data);
        continue;
    }
}
```

1, 2


1
2
3
4
5
6
7
4, 5, 3, 6, 7
2
1
6
4
1
SQL
Table
Table
Node_id


L
N
R
1
2
3
4
5
6
1
3
2
1
4
5