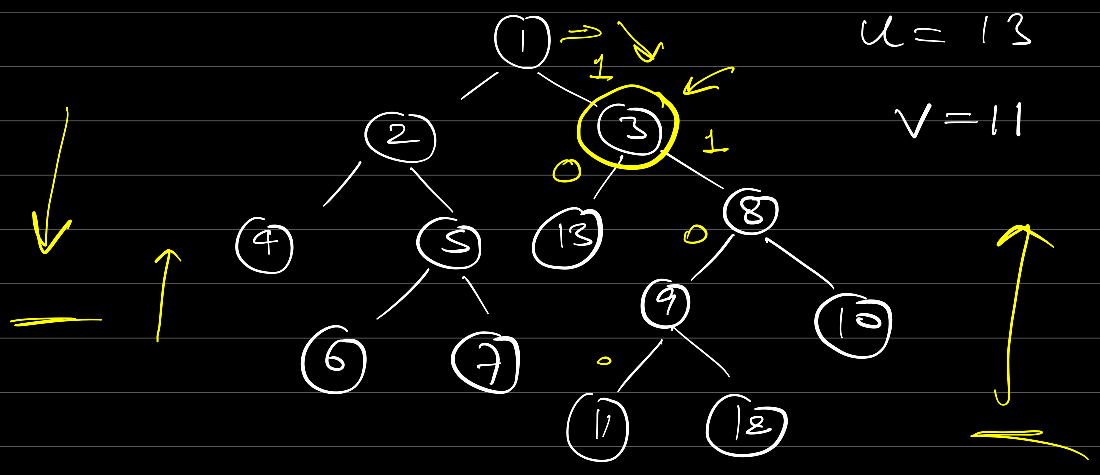
Bit approach for LCA



u = 13

v = 11

LCA(13, 11) = 3

13 ⇒ 1 0 0 0

11 ⇒ 1 1 0 0 } O(N) + O(H)

mismatch (i)

(0 → i-1) ⇒ path of the LCA

In Time / Out Time

⇓ Pre

start processing the subtree

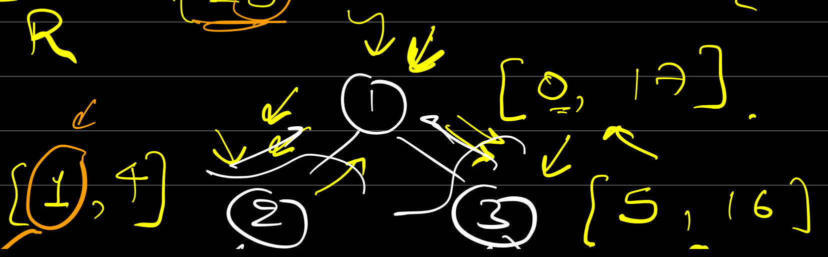
N, L, R

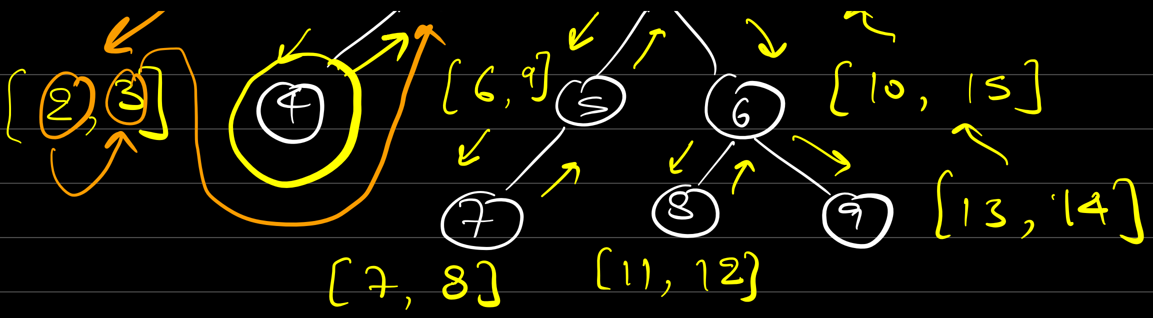
End processing the subtree

L, R, N ⇒ Post

[S.T., E.T]

t = 0





```
      ⇒ time = 0;
        void update ( root ) {

                if ( root == NULL ) { return ; }

IN                ⇒ inTime ( root ) = time ; //Update intime of root
                    time ++;
IN    -OUT          update ( root . left );
IN    -OUT          update ( root . right );

      -OUT          outTime ( root ) = time ;
                    time ++;
        }
```

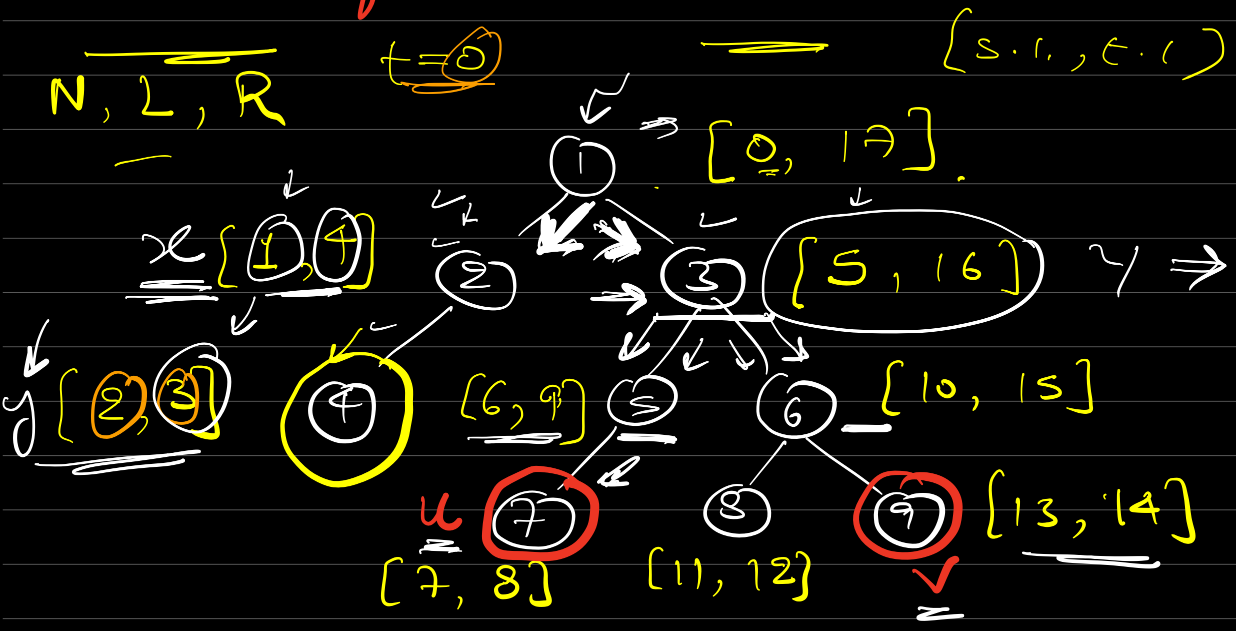
1) Update the node

```
        Node ( ) {                    root . inTime = Time
            value
            left
            right
              inTime
              OutTime
        }
```

2) HashMap (1 for InTime / 1 for OutTime)

Q⇒ Given the InTime / OutTime of every node in a binary tree.

Find out the LCA of two nodes u & v for the BT



① if (in[x] < in[y]) x start before y

② if (out[x] > out[y]) x ends after y

① & ②

u[s, e] & v[s, e]

Steps

```
if (root.left is ancestor of both u & v){
        root = root.left;
}
else if (root.right is ancestor of both u & v){
        root = root.right;
}
else {
    return root;    // root is the LCA
}
```

Q Given a Binary Tree

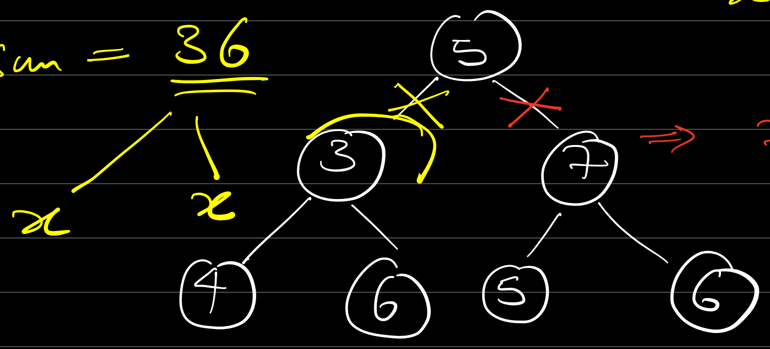
Remove 1 edge s.t. sum of 2 remaining BT is same

$x + x = 36$

$2x = 36$

$x = \frac{36}{2}$

#Sum = 36



$\Rightarrow 7+5+6$

$= 18$

Post

$5+3+4+6$

$= 18$

Total sum of all nodes of tree

Sol^n ① BF

$\forall$ edge, I will calculate the sum of 2 subtree's generated by removing this edge

T.C. $((n-1) \times N)$

$= O(N^2)$

② Using postorder.

Calculate sum of every subtree.

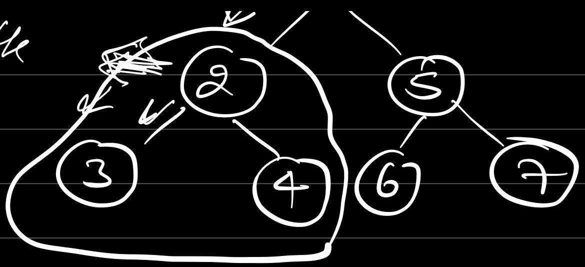
If sum = Total Sum/2

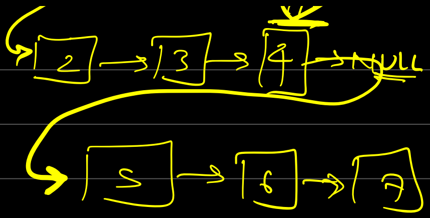
Break the tree here.

T.C. $= \underline{O(N)}$

Flatten a Binary Tree

root of the LST

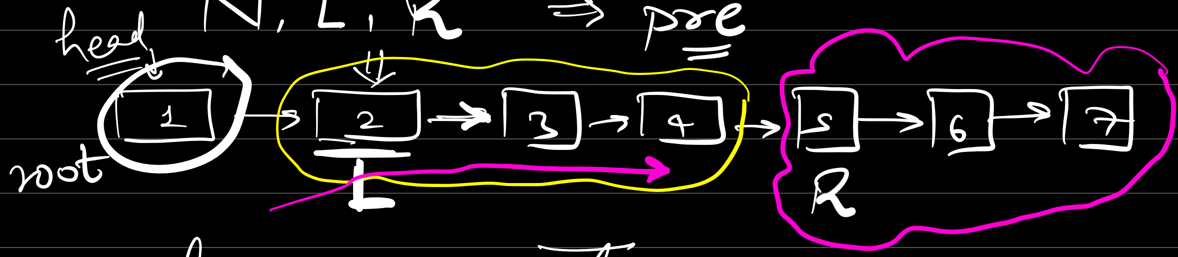




Convert to a LL

Root of BT should be the head of LL

N, L, R ⇒ pre



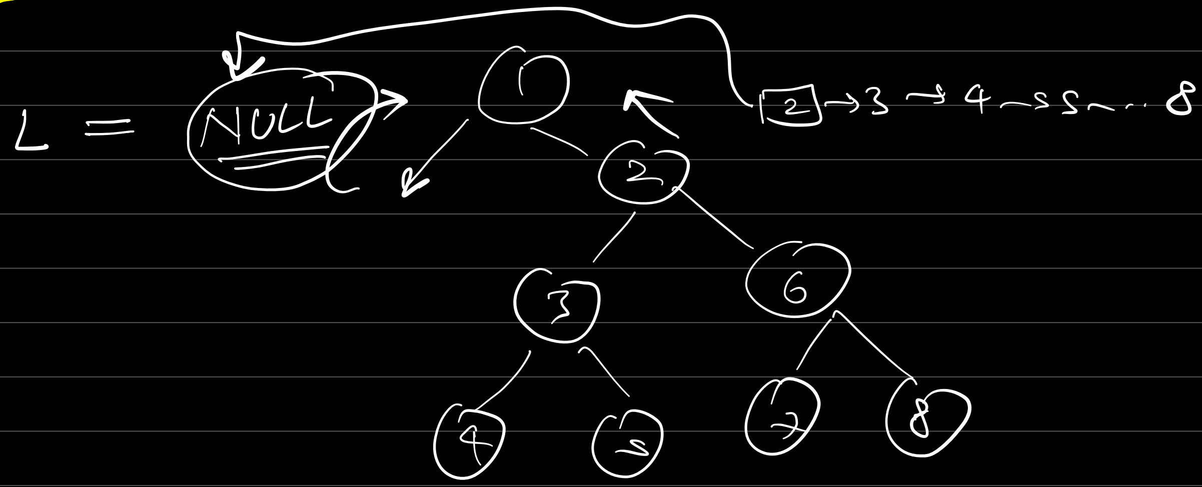
node ⇒ value
left = NULL
right ⇒ next

Code

```
node flatten (root) {
    ⇒ if (root == NULL) ret NULL;

    node L = flatten (root.left);
    node R = flatten (root.right);

    root.right = L;   root.left = NULL
    curr = root;
    if (L != NULL) { L.left = root; }   DLL
⇒   while ( curr.right != NULL) {
        curr = curr. right;
    → }
        curr. right = R; →
        if (R != NULL) { R. left = curr; } DLL
        return root;
}
```



1. Right == NULL

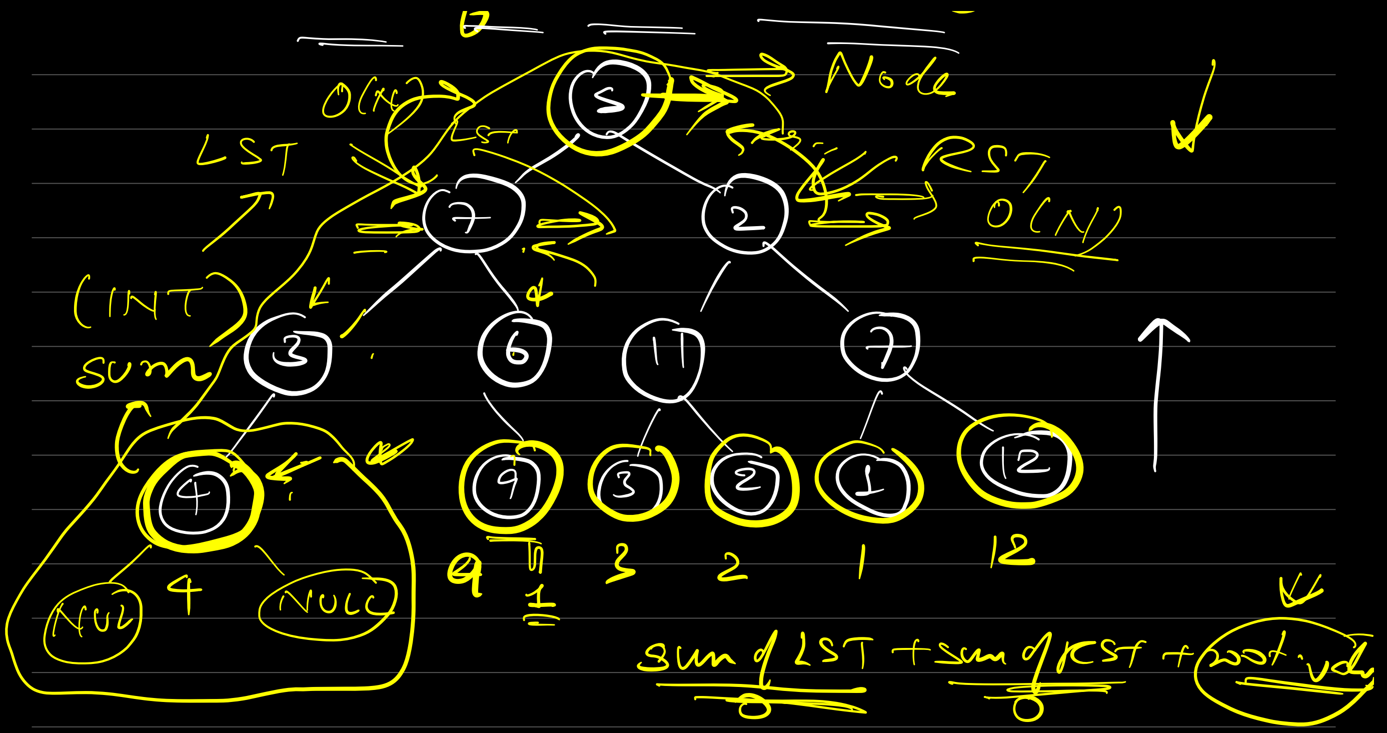
T.C. = O(N) ??? ✗

H.W: ⇒ How many times at max will you iterate over a given node

DLL

Code

Sum of all subtrees



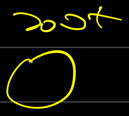
sum of LST + sum of RST + root.value

```
int calculateSum (root) {
        if (root == NULL) { return 0; }
        int sumLeft = calculateSum( root.left);
        int sumRight = calculateSum( root. right);
        return (( sumLeft + sumRight + root.value);
}
```

```
node LCA = null;
int calculateSum (root, LCA) {
        if (root == NULL) return 0;
        int sumLeft = (                    )
        int sumRight = (                    )
```

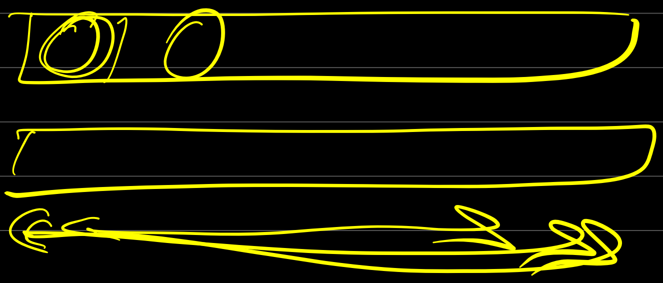
```
    sum Tree = sum left + sum Right;
    if (root.value == u ||root.value ==v){
          sum Tree += 1;
    }
    if (sum Tree == 2) {
          LCA = root;
    }
  return    sum Tree ;
}
```

)



BST

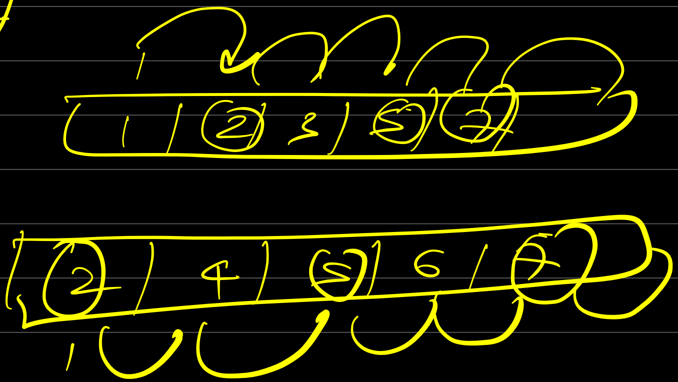
array



O(N^2)

arry

Sorting



Sum+=2

Sum+=2

Sum+=2

BST ⇒ Inorder

Iteratively Inorder