Serialisation of BT

Deserialisation of BT

Kth smallest in BST

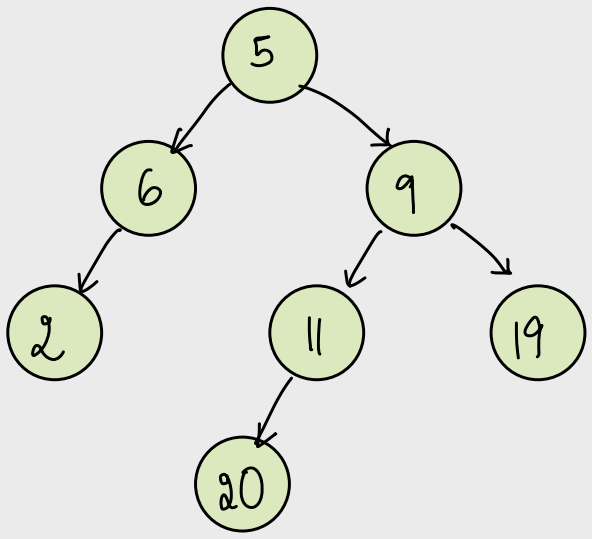
Trim BST

Iterative Preorder

Serialization of BT using level order traversal

Ex 1.

5 6 9 2 N 11 19 N N 20 N N N N N



list<int>

{ 5, 6, 9, 2, -1, 11, 19, -1, -1, 20, -1, -1, -1, -1, -1 }

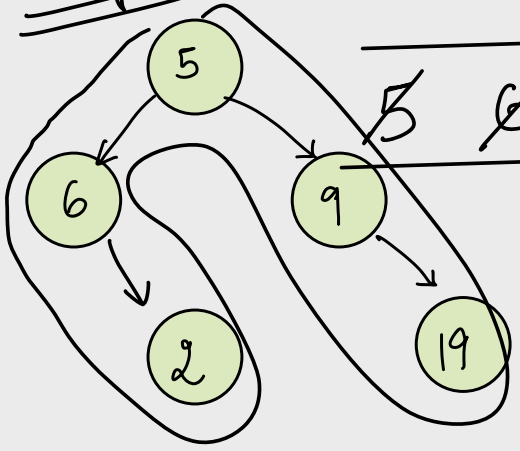
In place of NULL

① -1 ? say it's part of the data.

② INT.MIN / INT.MAX

③ $ / @ / .. ⇒ list<string>

Try



queue<Node>

5 ~~6~~ ~~9~~ NULL ~~2~~ NULL ~~19~~ NULL NULL NULL NULL

list<int>

5 6 9 -1 2 -1 19 -1 -1 -1 -1

```
list<int> l
queue<Node> q
q.push(root)
while(q.size()>0){
    Node t = q.front()
    q.pop()
    if(t==NULL){
        l.add(-1)
        continue
    }
    l.add(t.data)
    q.push(t.left)
    q.push(t.right)
}
```

1 → 2 NULLs

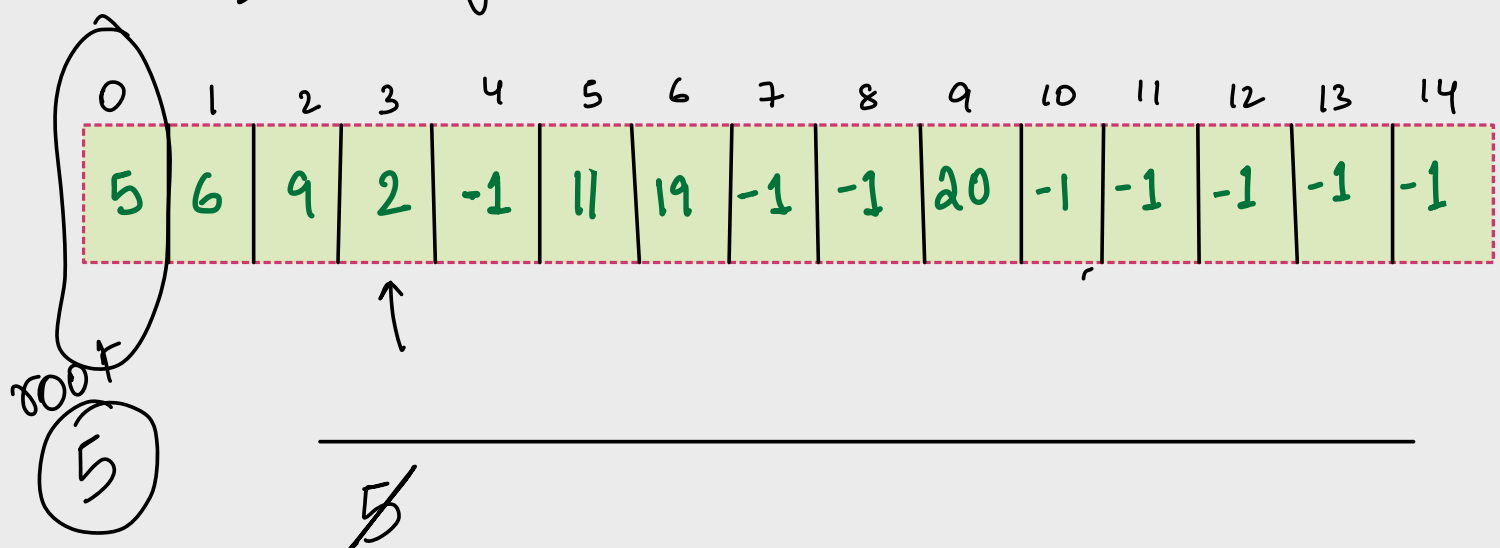
N → 2XN NULLs

N + 2N = 3N ⇒ O(N)

TC: O(N)

SC: O(N)

Deserialize using level order.



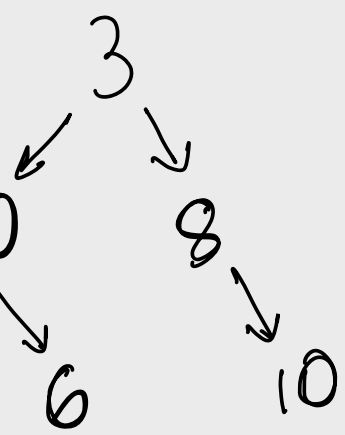

```
Node root = new Node(A[0])
i = 1
queue<Node> q
q.push(root)
while(q.size() > 0){
    Node t = q.front()
    q.pop()
    if(A[i] != -1){
        t.left = new Node(A[i])
        q.push(t.left)
    }
    i++
    if(A[i] != -1){
        t.right = new Node(A[i])
        q.push(t.right)
    }
    i++
}
return root
```

Ex: 3 10 8 -1 6 -1 10 -1 -1 -1 -1 i

nodes.

~~3~~ ~~10~~ ~~8~~ ~~6~~ ~~10~~

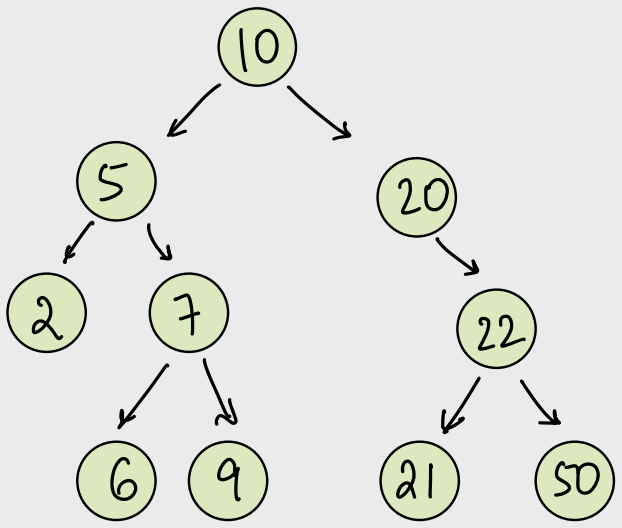
root



TODO:

- Preorder
- Postorder
- Inorder

Que. Kth smallest in BST



K = 3 | 6

K = 10 | 50

Solution 1:

Do inorder, store in array.

0	1	2	3	4	5	6	7	8	9
2	5	6	7	9	10	20	21	22	50

Return A[K-1]

TC: O(N)
SC: O(N)

Q queries

TC: O(N) + O(Q)

Solution 2:

inorder (K) variable

```
inorder(left)
print(data)
inorder(right)
```

TC: O(K)
SC: O(H)

O(Q * K)

Solution 3: Morris.

TC: O(K)
SC: O(1)

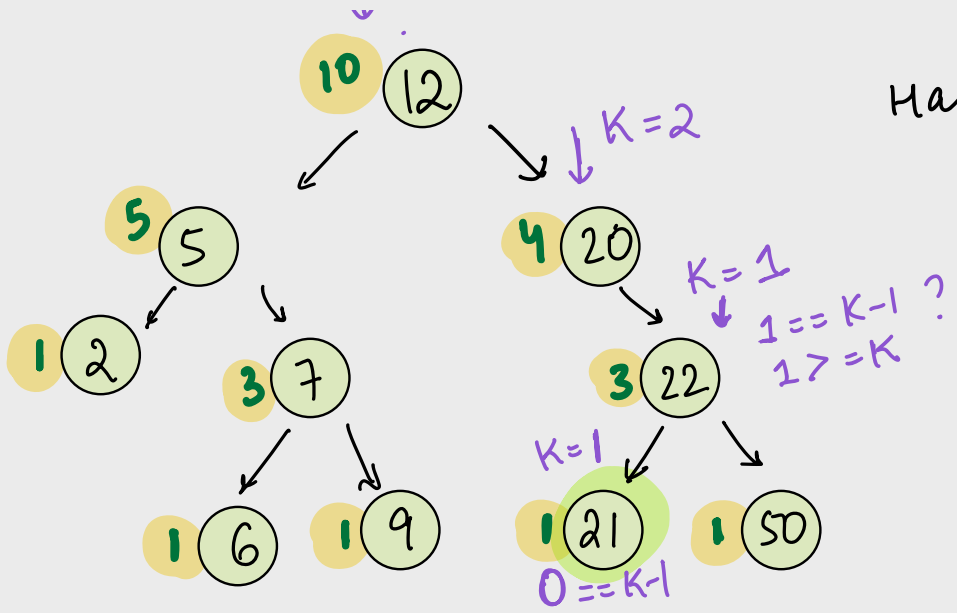
Q queries

TC: O(Q * K)
SC: O(1)

Find the answer for Q queries.

Using inorder { Store in array ⇒ O(N)
Answer queries ⇒ O(1)
Total → O(N+Q) } Solution 1.

∴ K=8



HashMap < Node, int > mp.

12	10
5	5
20	4
2	1
7	3
22	3
6	1
9	1
21	1
50	1
NULL	0

for Q queries.

```
int Kth( Node root, int K){
    Node curr = root
    while(curr != NULL){
        if(mp[curr.left] == K-1){
            return curr.data
        }
        else if(mp[curr.left] >= K){
            curr = curr.left
        }
        else{
            curr = curr.right
            K = K - mp[curr.left] - 1
        }
    }
    return -1
}
```

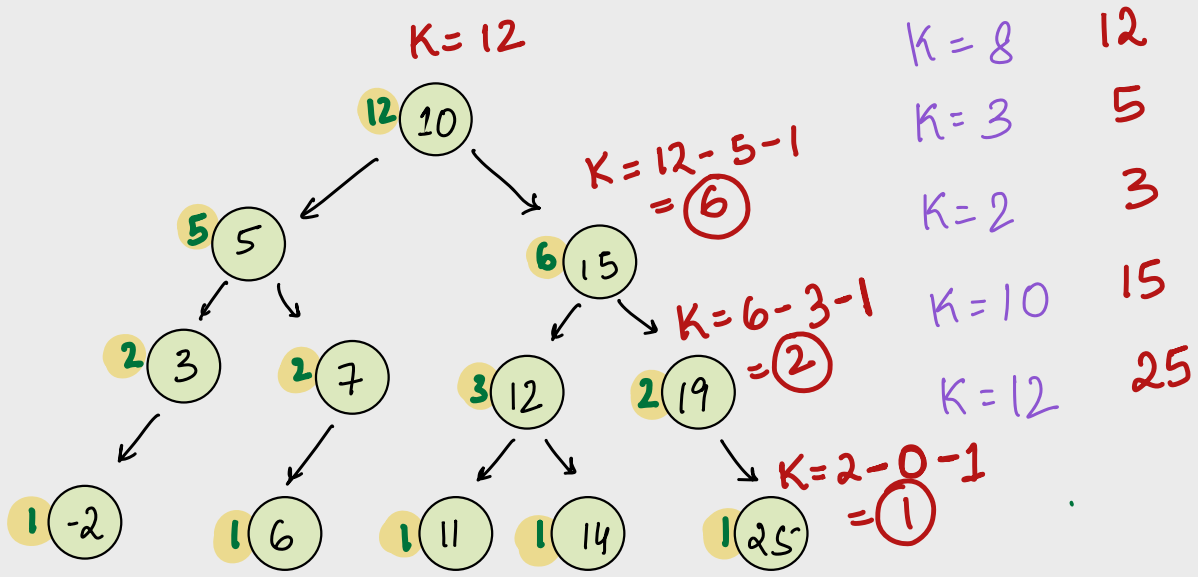
TC: O(N) + O(Q x H)

O(N + Q x H)

SC: O(N)

```
HashMap<Node, int> mp;

int size(Node root){
    if(root == NULL) return 0
    int ls = size(root.left)
    int rs = size(root.right)
    mp[root] = 1 + ls + rs
    return 1 + ls + rs
}
```



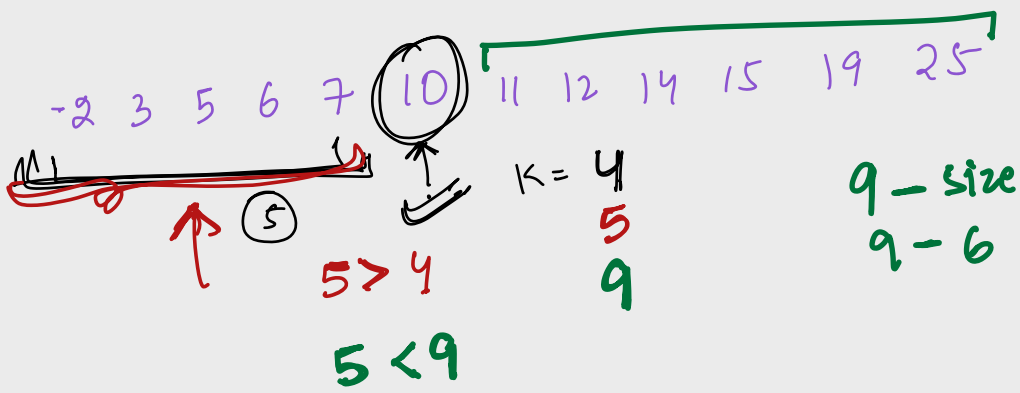
K = 8 12

K = 3 5

K = 2 3

K = 10 15

K = 12 25

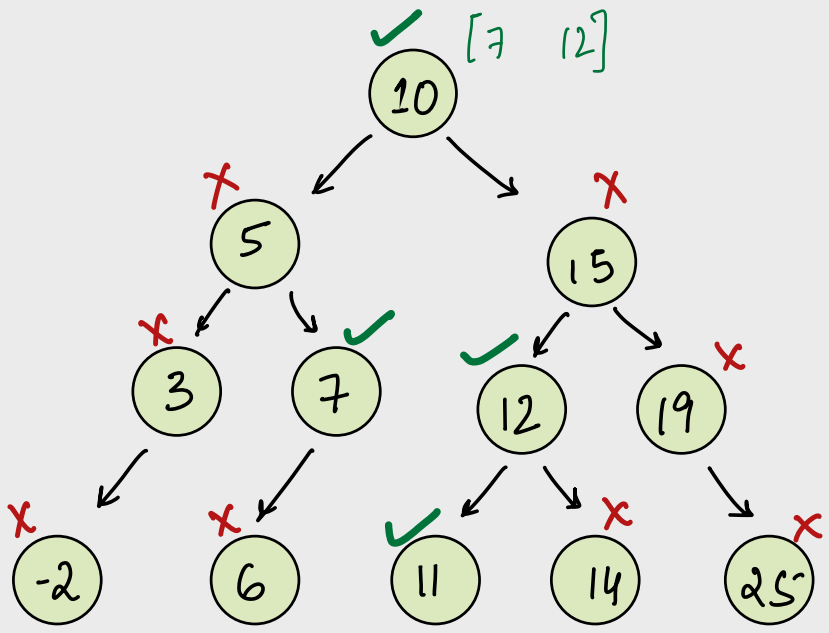


9 - size(LST) - 1

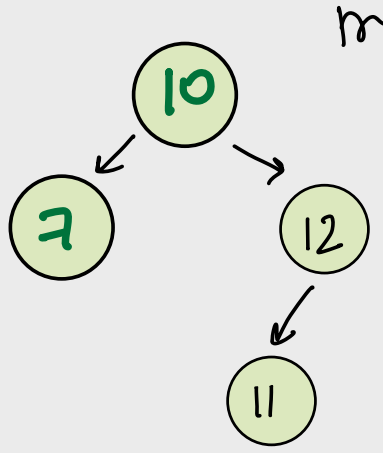
9 - 6 = 3

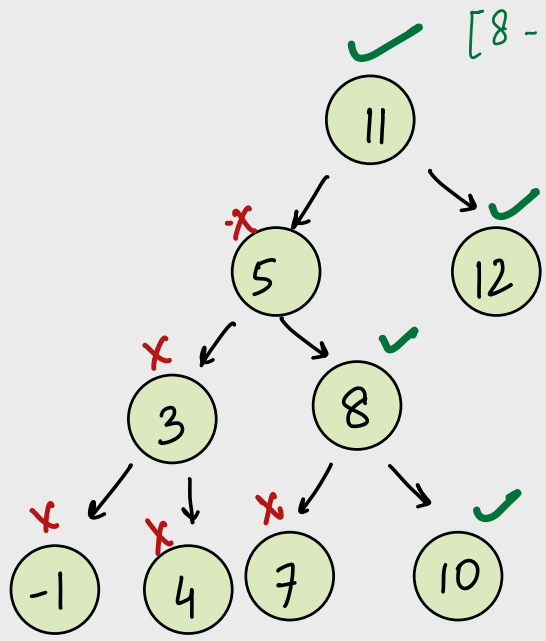
10:55

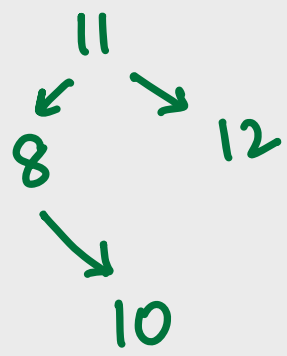
Q. Given BST, make sure all nodes are in given range [L, R]



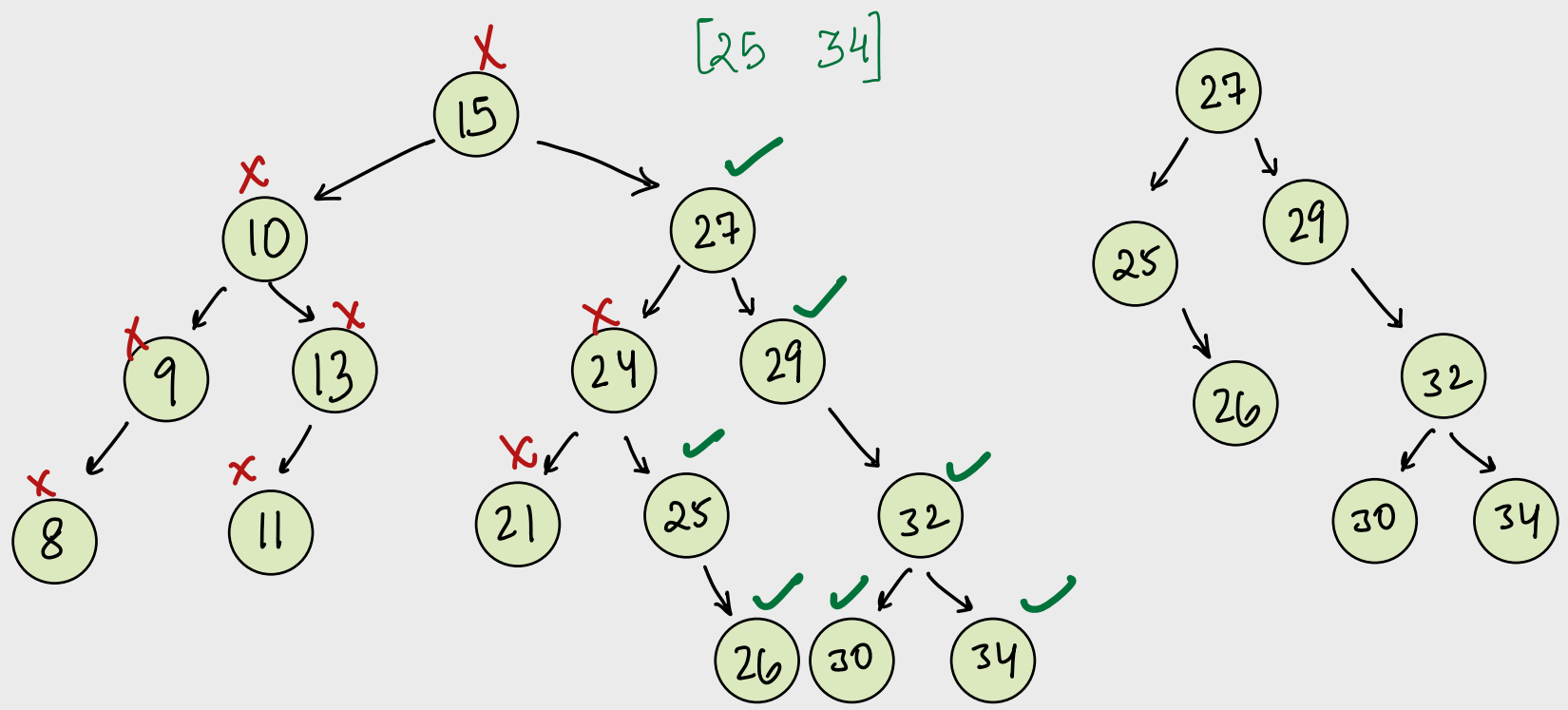
return this tree.







BST

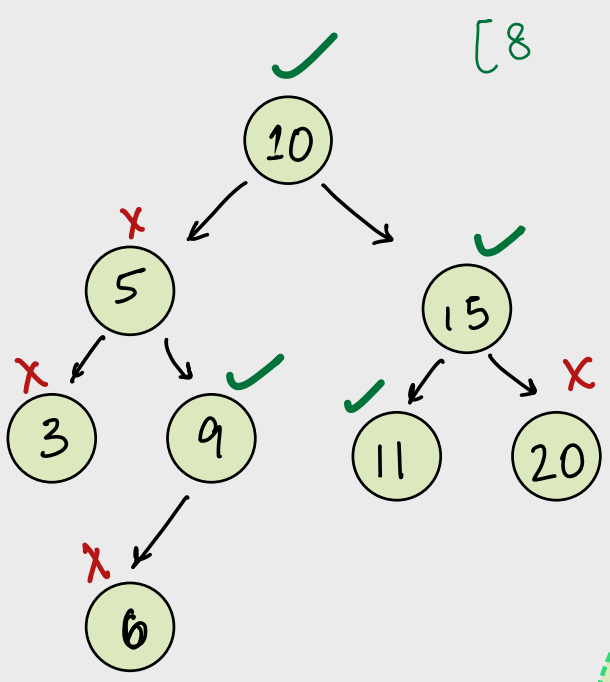


Issue: Construct the tree.

data data data

[L R]

11 15

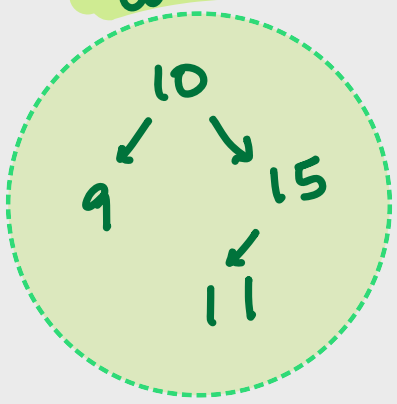


9 10 11 15

9 → 10 → 11 → 15

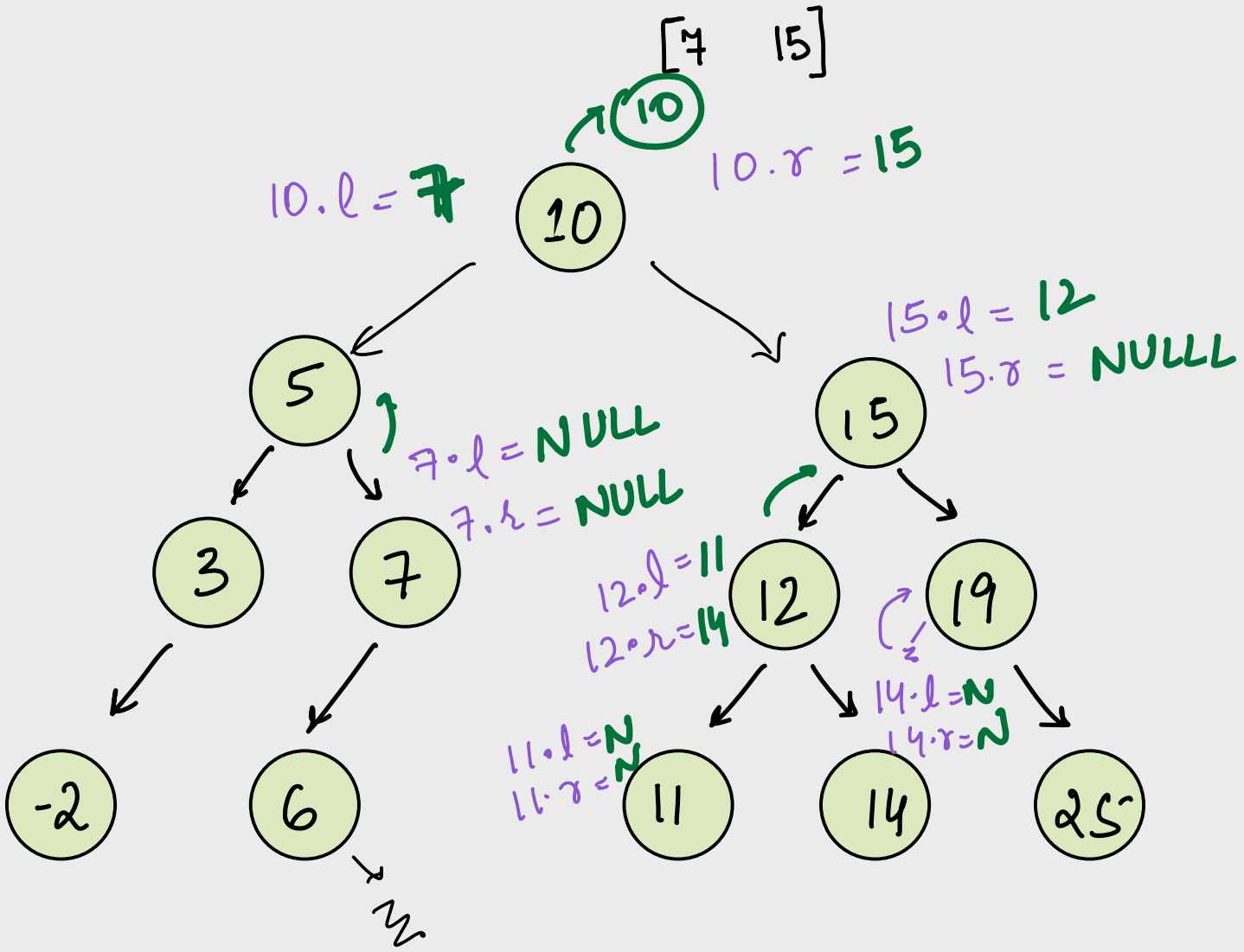
INCORRECT

Correct

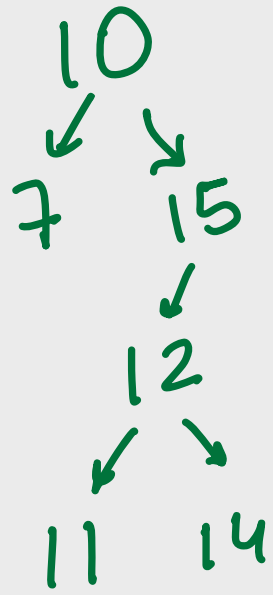



```
Node trim( Node root, int L, int R) {
    Assumption: Given BST, code will trim it & returns the new head node

    if(root == NULL) return NULL
    if( root.data >= L && root.data <= R) {
        root.left = trim (root.left, L, R)
        root.right = trim (root.right, L, R)
        return root
    }
    else if ( root.data < L ){
        return trim (root.right, L, R)
    }
    else {  // root.data > R
        return trim (root.left, L, R)
    }
}
```

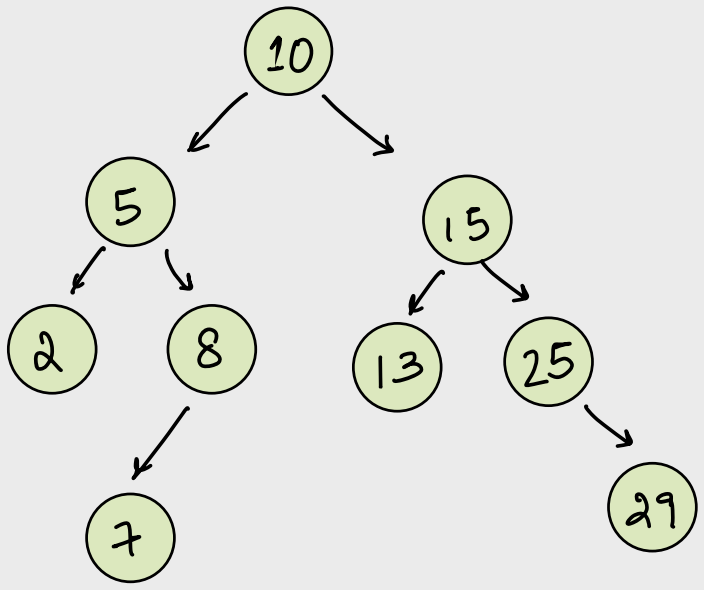

[7 15]
10
10
10.l = 7
10.r = 15
5
15
15.l = 12
15.r = NULL
7.l = NULL
7.r = NULL
3
7
12
19
12.l = 11
12.r = 14
14.l = N
14.r = N
11.l = N
11.r = N
-2
6
11
14
25

TC: O(N)
SC: O(H)


10
7
15
12
11
14

Iterative Preorder

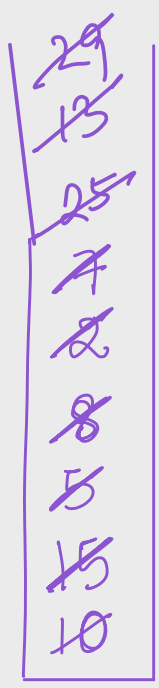
DLR




```
void preorder( Node root) {
    if (root == NULL) return root
    print( root.data)
    preorder(root.left)
    preorder(root.right)
}
```

Preorder :

10 5 2 8 7 15 13 25 29.



Print :

10 5 2 8 7 15 13 25 29.

```
void iterative (Node root) {
    stack<Node> s;
    s.push (root)
    while ( s.size() > 0) {
        Node t = s.top()
        s.pop()
        print (t.data)
        if (t.right) s.push(t.right)
        if (t.left) s.push (t.left)
    }
}
```

TC : O(N)

SC : O(N)