Today's Content

- → Binary Tree
- → Traversal
- → Iterative traversals
- → Construct B.T from Preorder to Inorder

DSA certification

DSA certified → DSA → company

↓

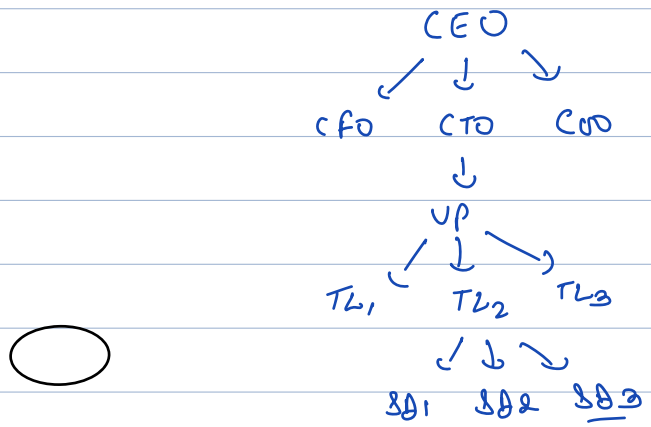
Mock Interview of DSA + contest of DSA → After last class of DSA → 30 days. 1 Mock Interview.

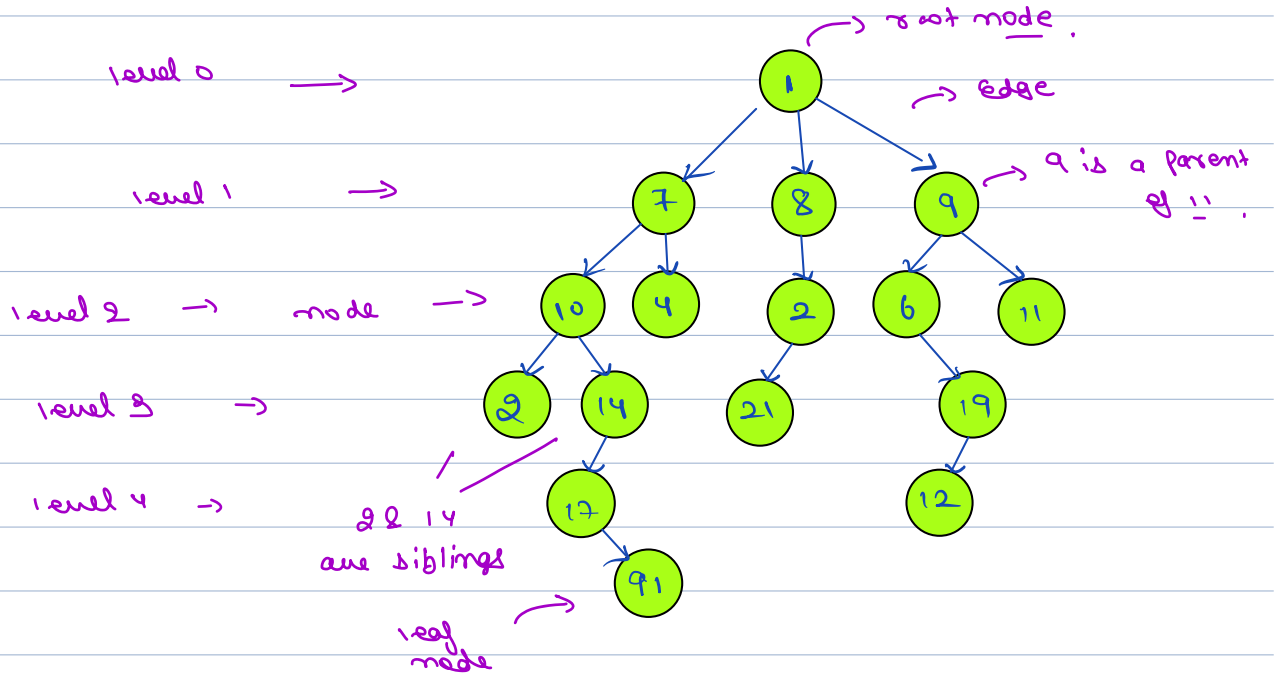
↳ last contest of DSA.

Trees Basics & terminologies

// Linear data structures → arrays, stacks, LL.

// Hierarchical :-



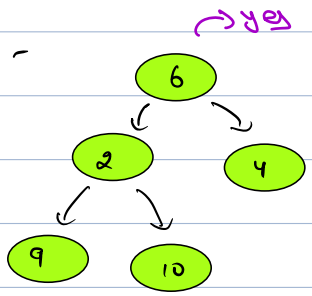


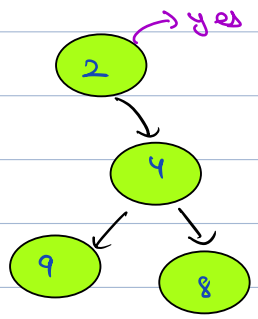
root → It is a node with no parent.

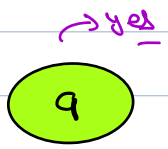
leaf → It is a node with no children.

Binary Tree :- for every node, no. of children <= 2.

Ex 1 :-

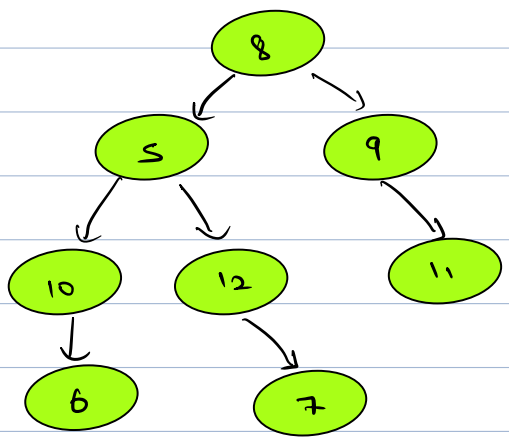






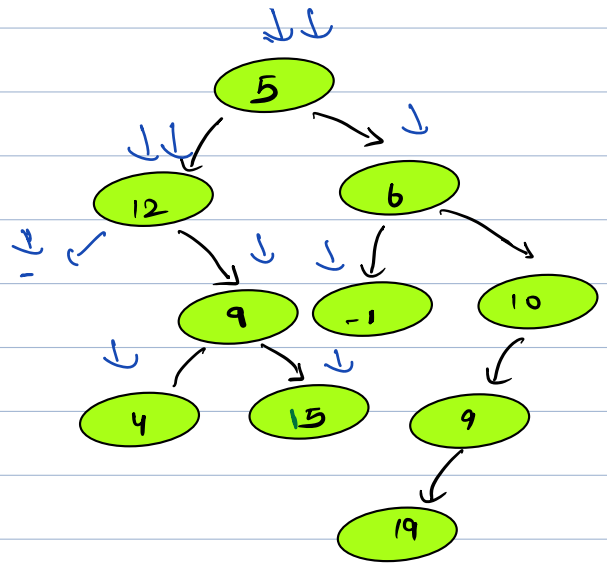

```
class Node {
    int data;
    Node left;
    Node right;

    Node(x) {
        data = x;
        left = null;
        right = null;
    }
}
```



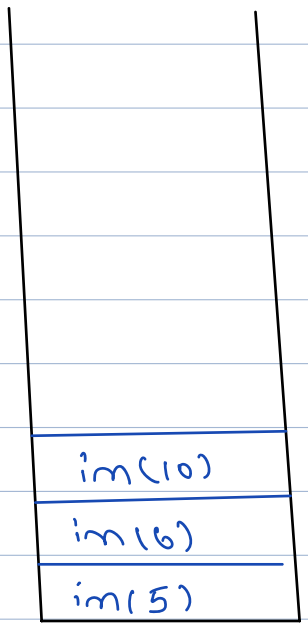
Tree Traversals :-

- → Inorder
- → Preorder
- → Postorder



1) Inorder (L → R)

12 4 9 15 5 -1 6 19 9 10

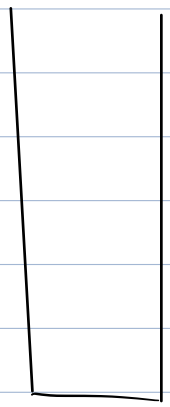


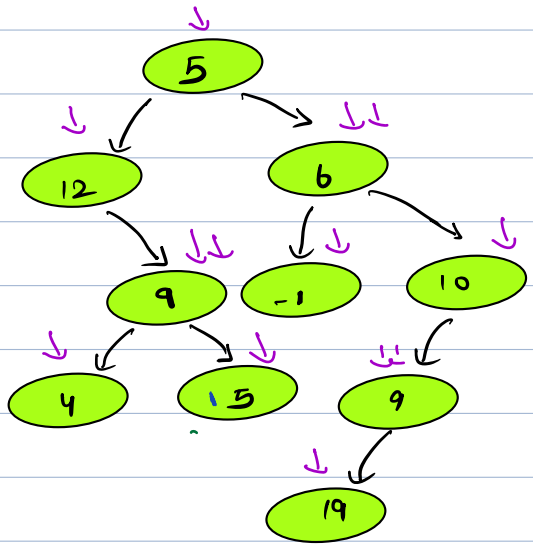

```
void inOrder( node root) {
    if (root == null) {return;}
    inOrder (root.left);
    print (root.data);
    inOrder (root.right);
}
```

T.C → O(n)

S.C → O(H)
- H → Max = n O(n) (skewed)
- H → min O(log n) (Balanced)

2) PreOrder D L R



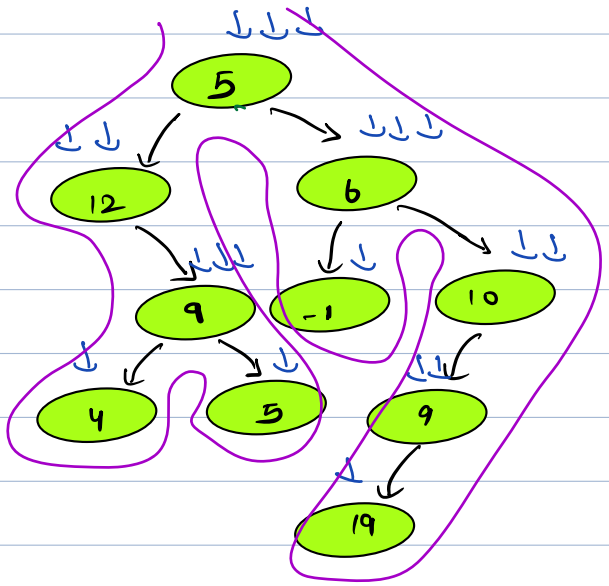


5 12 9 4 15 6 -1 10 9 19

T.C → O(n)
S.C → O(h)

```
void PreOrder(Node root) {
    if (root == null) {return;}
    print(root.data);
    PreOrder(root.left);
    PreOrder(root.right);
}
```

3) Post Order :- LRD



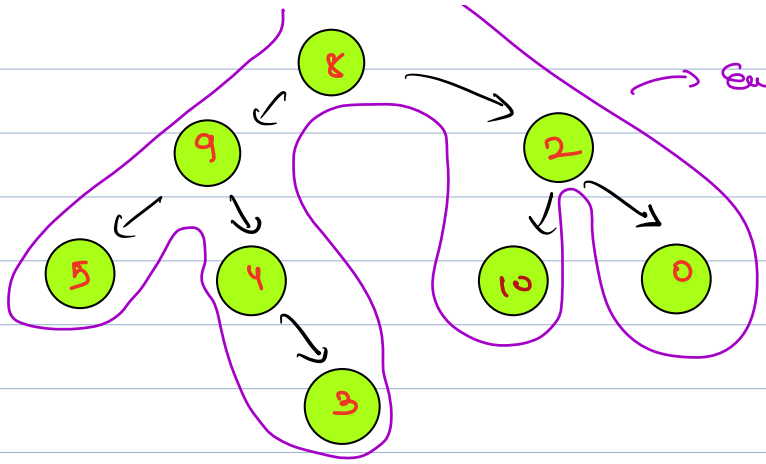
4 5 9 12 -1

19 9 10 6 5

T.C → O(n)

S.C → O(h)

```
void PostOrder(Node root) {
    if (root == null) {return;}
    PostOrder(root.left);
    PostOrder(root.right);
    print(root.data);
}
```

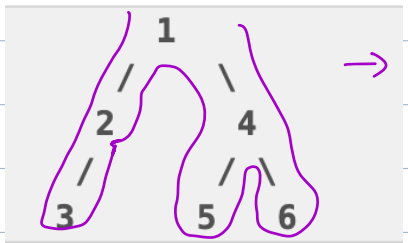


→ Euler Path.

Inorder (LDR) :- 5 9 4 3 8 10 2 0

Preorder (DLR) :- 8 9 5 4 3 2 10 0

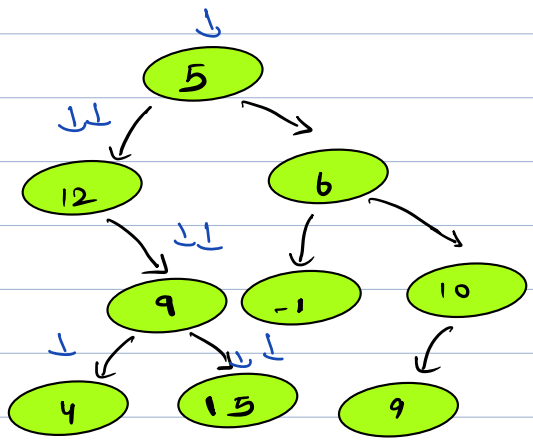
Postorder (LRD) :- 5 3 4 9 10 0 2 8



→ 3 2 1 5 4 6 ←

Inorder Iterative :- (L D R).

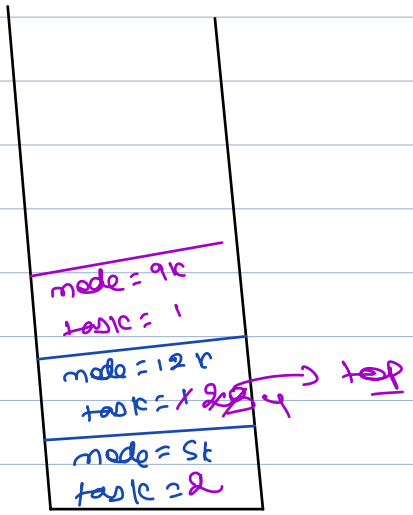
1. call left child
2. Print data
3. call right child.

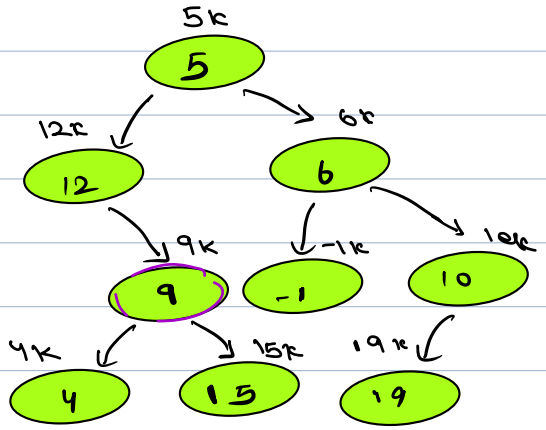


12, 4, 9, 15, 5

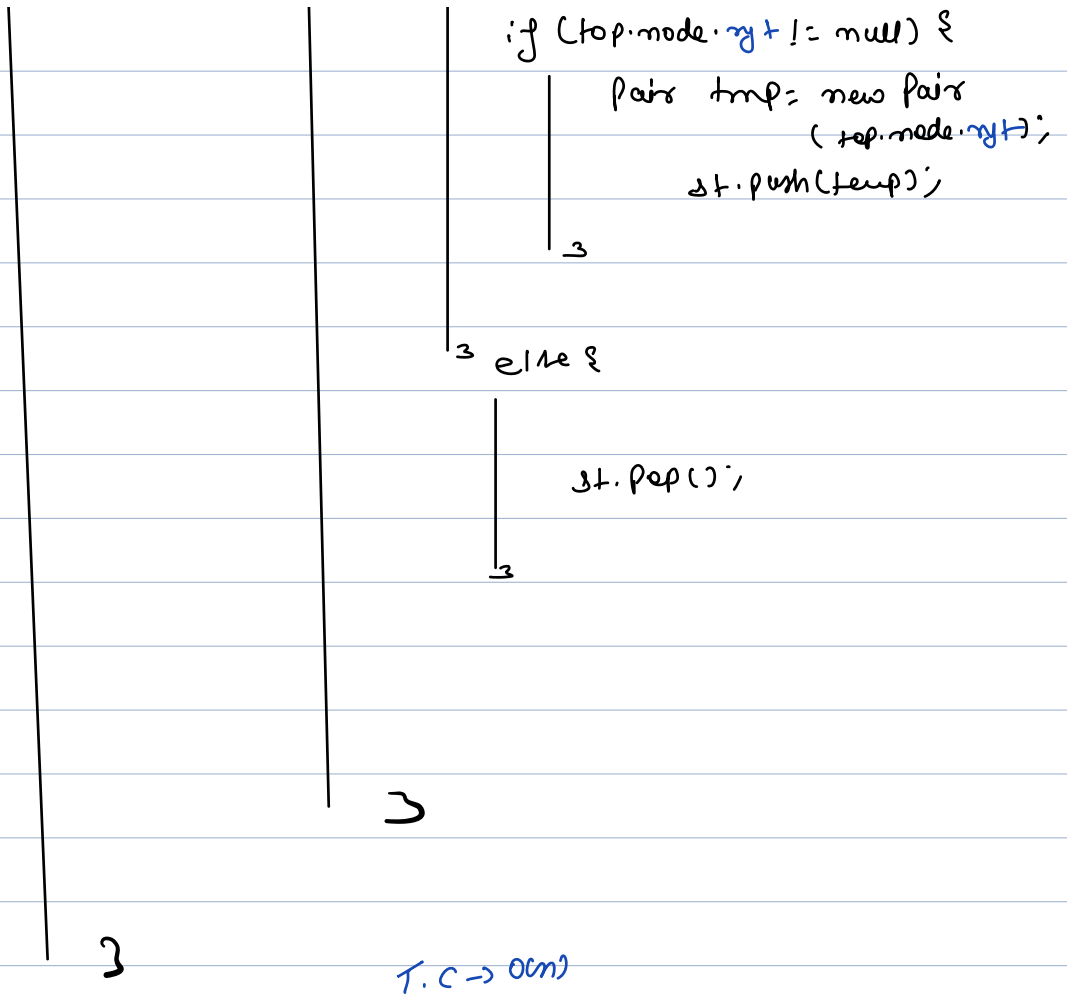

```
class Pair {
    Node node;
    int task;
    Pair (Node t) {
        node = t;
        task = 1;
    }
}
```

12






```
public void inorder (Node root) {          // 5k
    Stack<pair> st;
    pair p = new pair (root);              // 5k
    st.push(p);
    while (st.size()>0) {
        pair top = st.peek();
        if (top.task == 1) {
            top.task++;
            if (top.node.left != null) {
                pair tmp = new pair
                        (top.node.left);
                st.push(temp);
            }
        }
        else if (top.task == 2) {
            top.task++;
            print (top.node.data);
        }
        else if (top.task == 3) {
            top.task++;
```

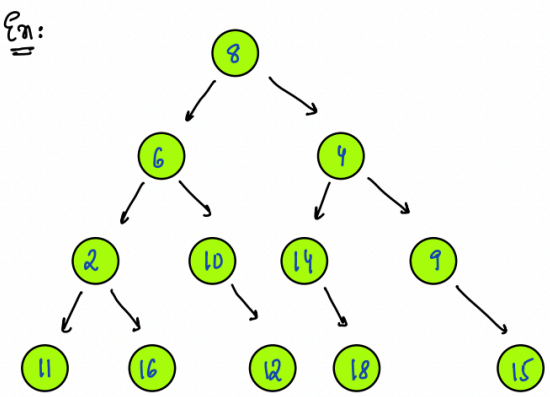

if (top.node.rt != null) {
Pair tmp = new Pair
(top.node.rt);
st.push(temp);
}
else {
st.pop();
}
}

T.C → O(n)

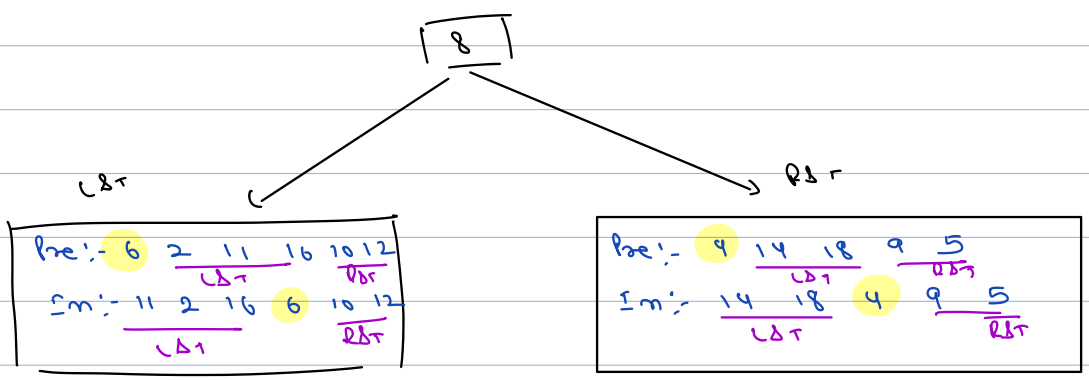
S.C → O(H)

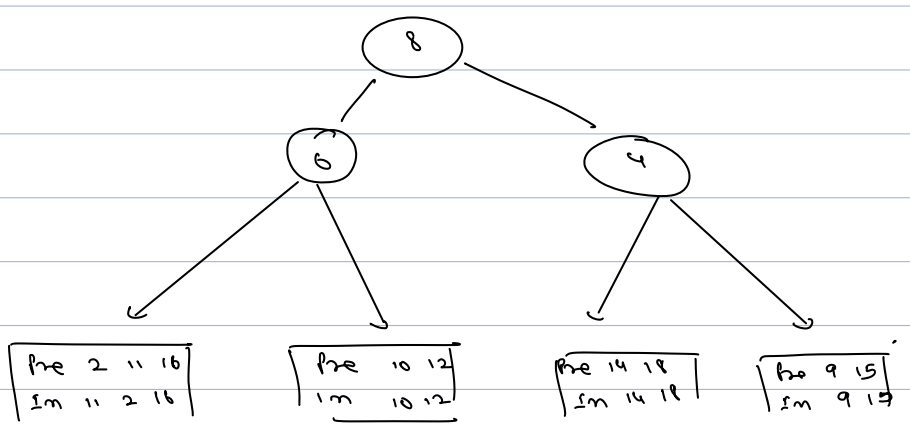
Ques) Given Preorder & inorder of a B.T. with distinct values. create it.

Eg:



	0	1	2	3	4	5	6	7	8	9	10	11
(DLR) Preorder :-	8	6	2	11	16	10	12	4	14	18	9	5
				LST						RST		
(LDR) Inorder :-	11	2	16	6	10	12	8	14	18	4	9	15
	LST							RST				





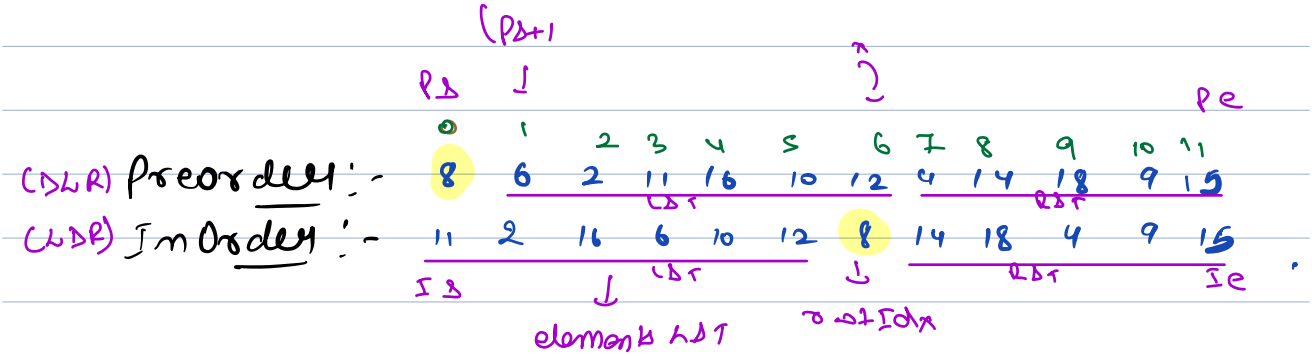
elementsLST = (is, rootIdx - 1);

=> rootIdx - x - is + x

=> rootIdx - is;

(ps+1, x) = elementsLST

=> x - ps - 1 + 1 = elementsLST => x = elementsLST + ps;




```
Node create (int ps, int pe, int is, int ie) {

    if (ps > pe || is > ie) return null;

    root.data = pre[ps];
    Node root = new Node(root.data);
    int rootIdx = find(is, ie, root.data);   // make it const with hashmap
    int elementsLST = rootIdx - is;

    root.left = create(ps+1, elementsLST + ps, is, rootIdx - 1);

    root.right = create(elementsLST + ps + 1, pe, rootIdx + 1, ie);
```

return sort;

2

T.C → O(n)

S.C → O(1)