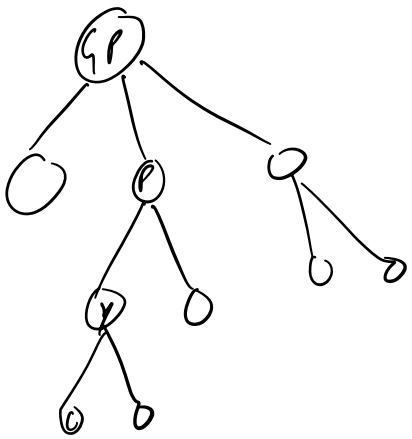
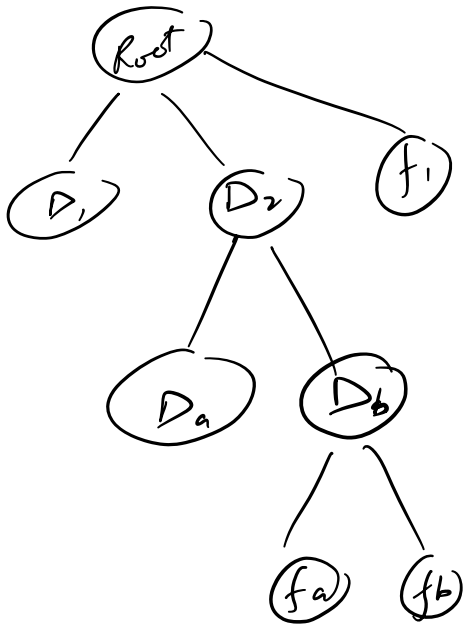
Trees

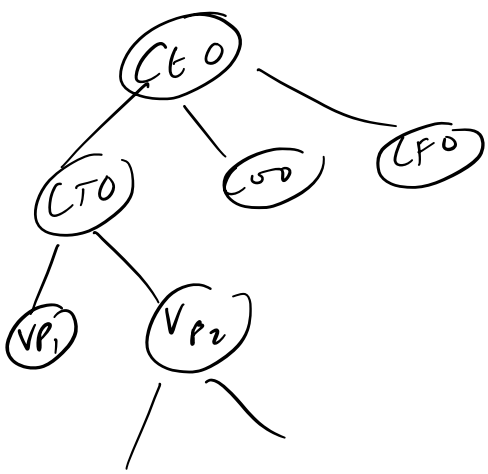
family tree



File structure

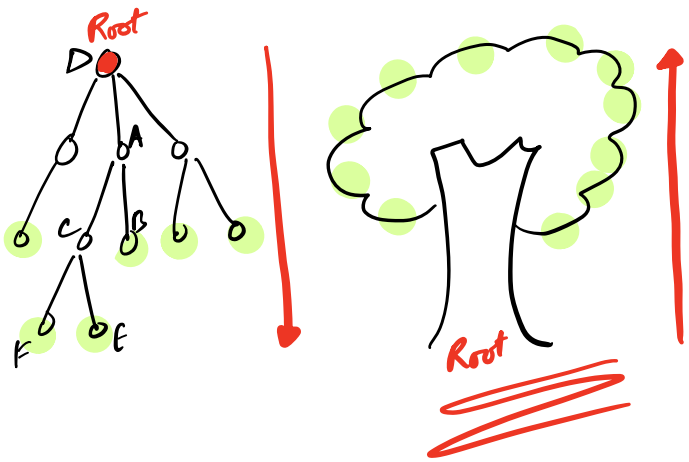


Org structure



Tree is a hierarchical DS !

⊙ Terminology



Node: An entity which holds info.

Edge: Connects two nodes!

Root: No parent

Leaf: No children

A is a parent of D

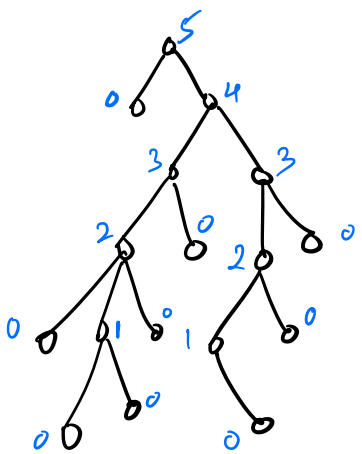
D is a child of A.

B & C are siblings.

Ancestors (F): C A D

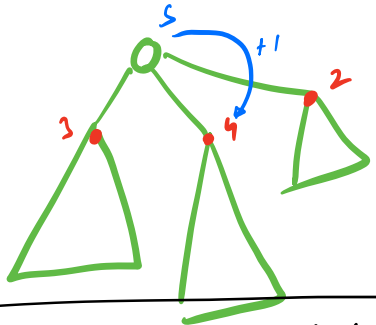
Descendant (A): C D F E

tree : Heights ..



⊙ Height (Node): Length of the longest path from this node to any of it's descendants!

Height (tree): height (root)

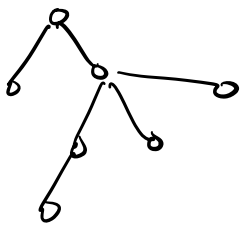


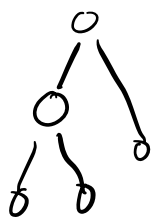
H (Node) = 1 + Max Height of Child

⊙ If a tree has N nodes : 7

? Edges = 6

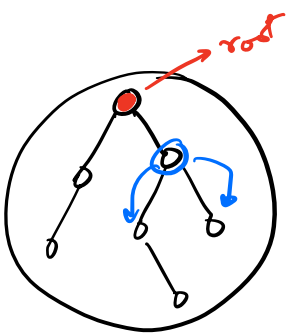
 # Edges = N-1

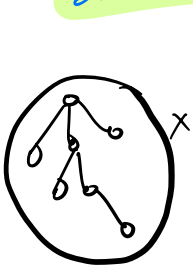


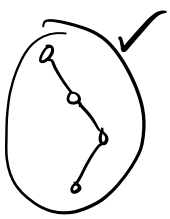


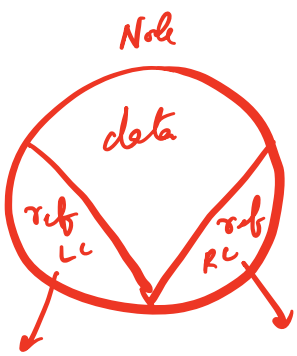
N	E
1	0
+1	+1
+1	+1
+1	+1
+1	+1

⊙ Structure of Binary tree → Every node has **AT MOST 2 children**



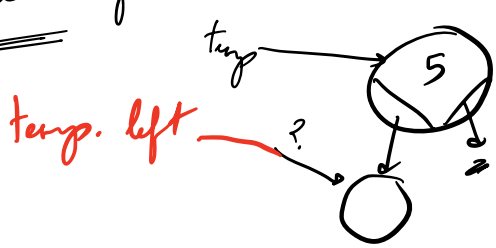


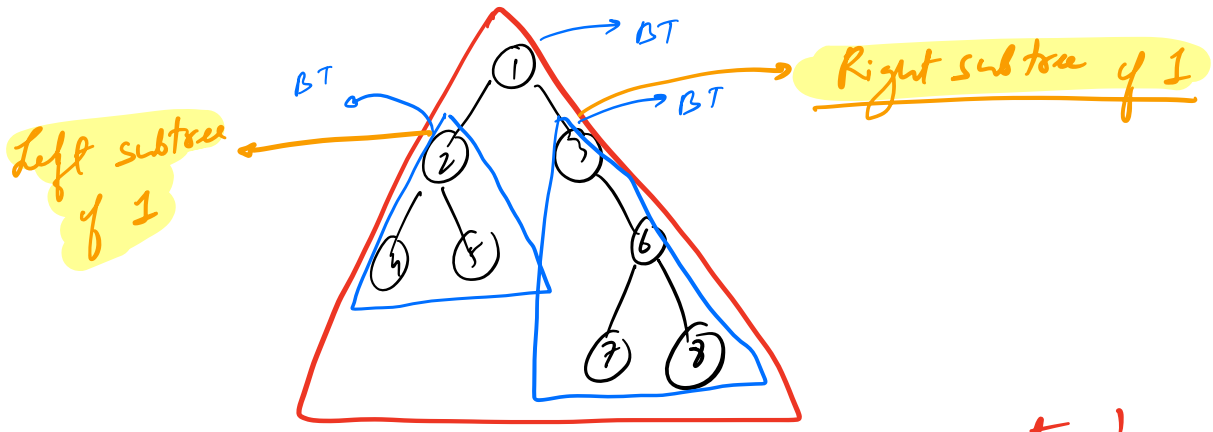





```
class Node {
  int data;
  Node left;
  Node right;
  Node(x) {
      data = x;
      left = NULL;
      right = NULL;
  }
}
```

Node temp = new Node(5);

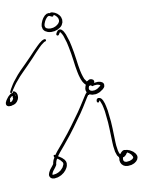




→ Structurally BT is recursive in nature!

→ Solving tree problems recursively is generally easy!

Tree traversal

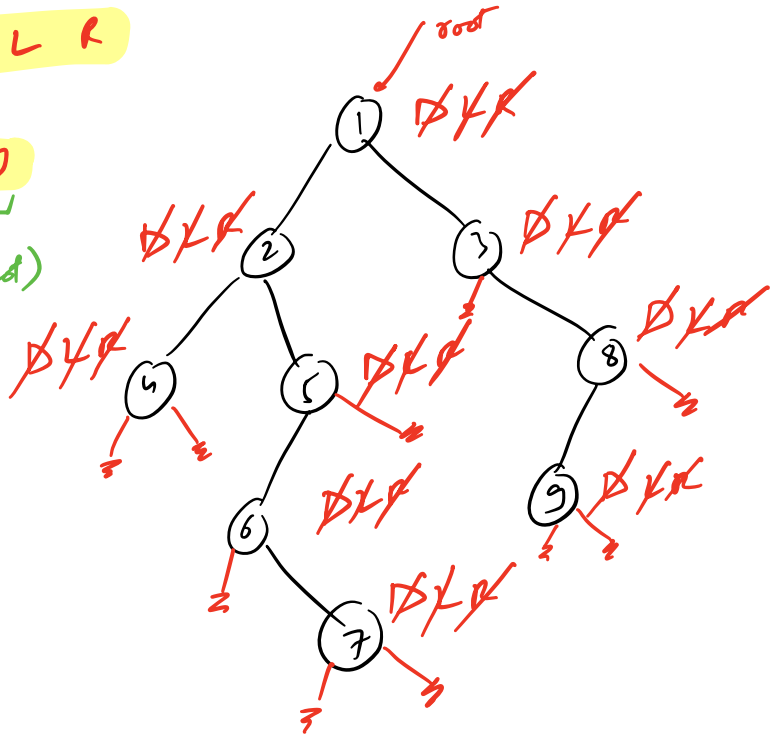


- Pre order : D L R
- In order : L D R
- Post order : L R D

1) Preorder → D L R

1, 2, 4, 5, 6, 7, 3, 8, 9

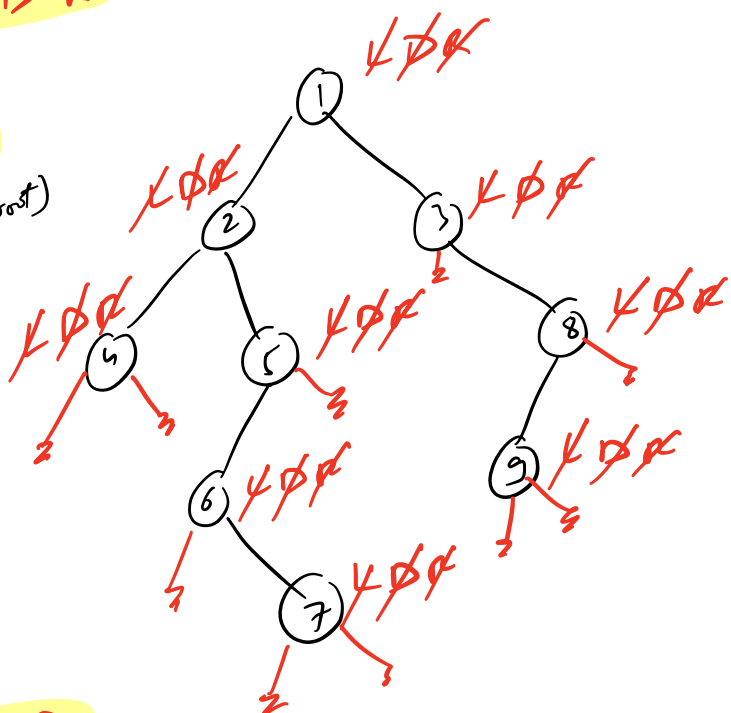
Root | LST(Root) | RST(root)



2) Inorder → L D R

4, 2, 6, 7, 5, 1, 3, 9, 8

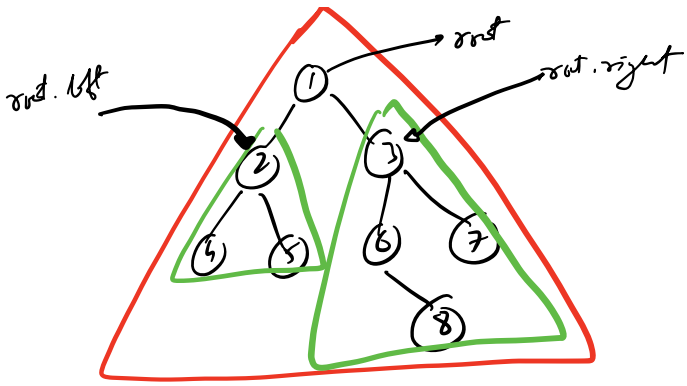
LST(root) | root | RST(root)



3) Post Order → L R D

HW

CODE:



INORDER: [L D R]

```
void inorder (Node root) {
    if (root == NULL) { ret; }

L   inorder (root.left);
D   print (root.data);
R   inorder (root.right);

}
```

PRE ORDER: [D L R]

```
void inorder (Node root) {
    if (root == NULL) { ret; }

D   print (root.data);
L   inorder (root.left);
R   inorder (root.right);

}
```

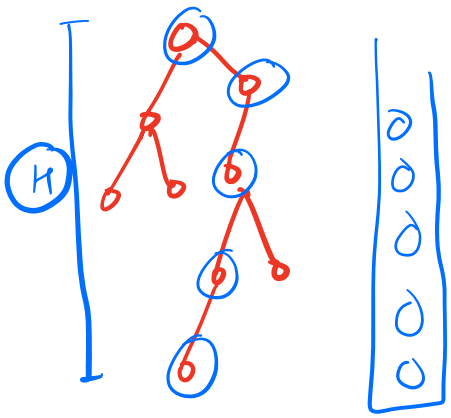
(

print (root.data);

POST ORDER

HW

TC / SC ?



f^n calls = O(N)

time taken / f^n → O(1)

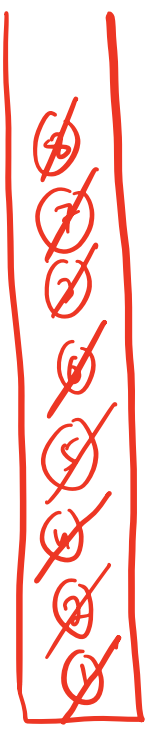
TC = O(N)

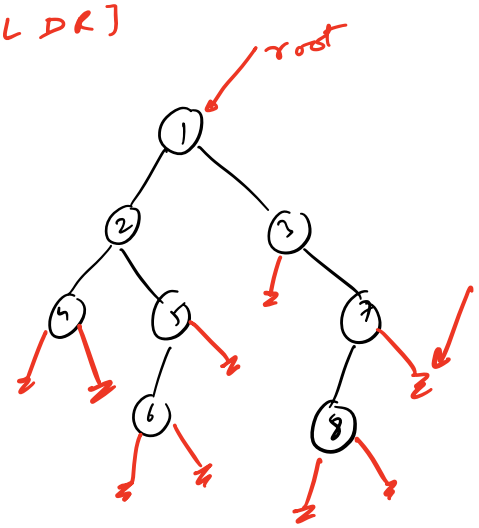
SC = O(H) → H: Height (tree)

→ N

Iterative Inorder traversal → [L D R]

4, 2, 6, 5, 1, 3, 8, 7



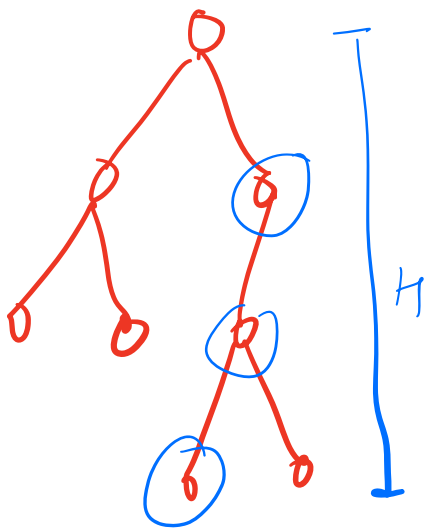


//CODE:

```
// Node root
Stack<Node> st;
Node temp = root;
while((temp != NULL) || (!st.isEmpty())) {
    if (temp != NULL) {
        st.push(temp)
        temp = temp.left;
    }
    else {
        temp = st.top();
        st.pop();
        print(temp.data);
        temp = temp.right;
    }
}
```

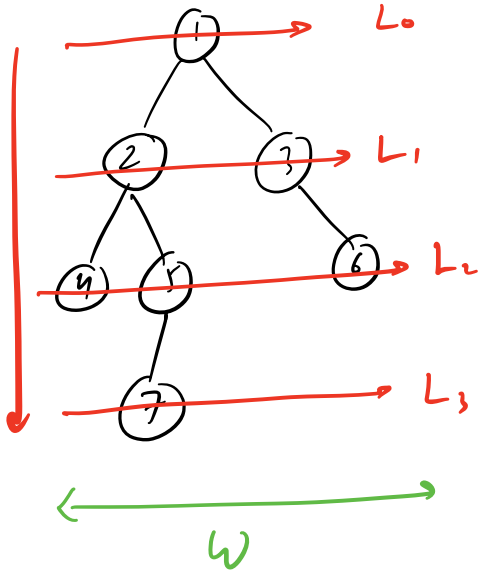
TC = O(N)

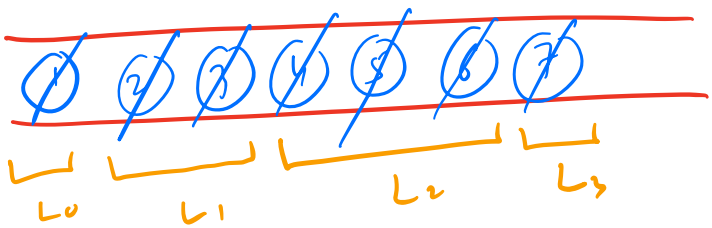
SC = O(H)



⑧ Level Order traversal ~~(BFS)~~ [LOT]

1, 2, 3, 4, 5, 6, 7





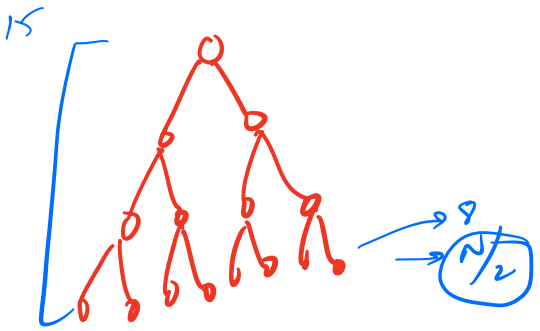

```
//Node root
Queue<Node> q;
q.enqueue(root);
while(!q.isEmpty()) {
    Node temp = q.front();
    q.dequeue();
    print(temp.data);
    if(temp.left != NULL) q.enqueue(temp.left);
    ——— right ——————————— right ———
}
```

→ if(root == NULL) ret;

TC = O(N)

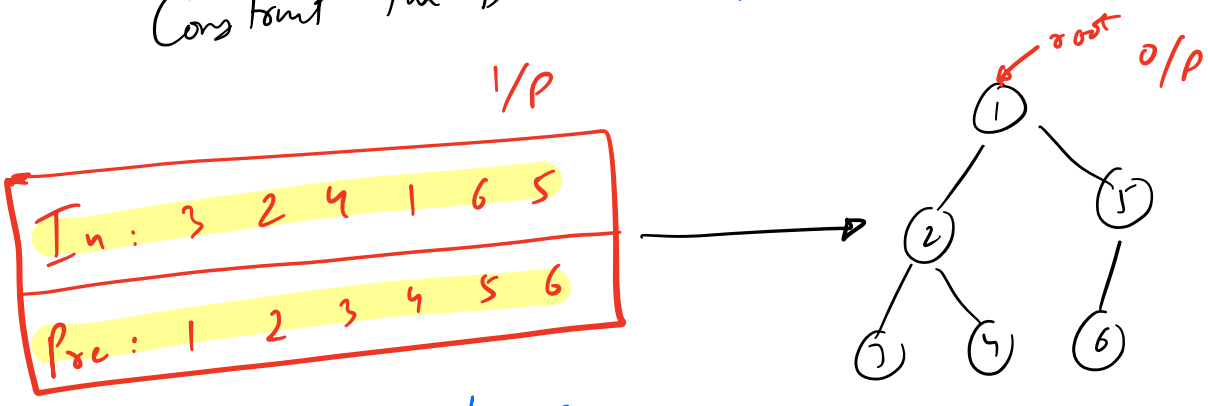
SC = O(W)

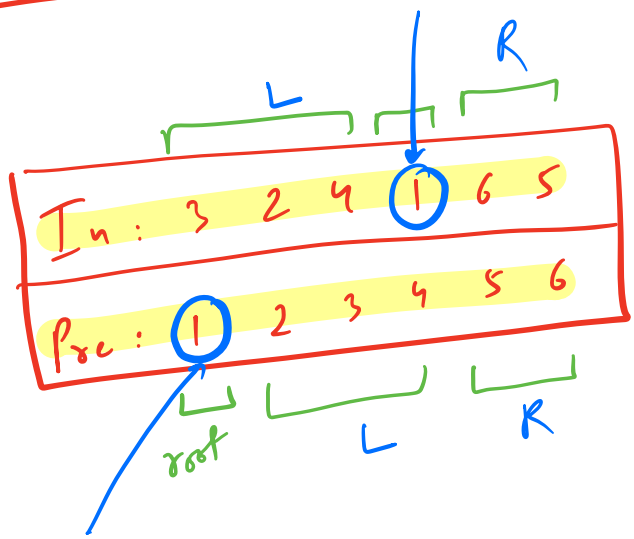
O(N)

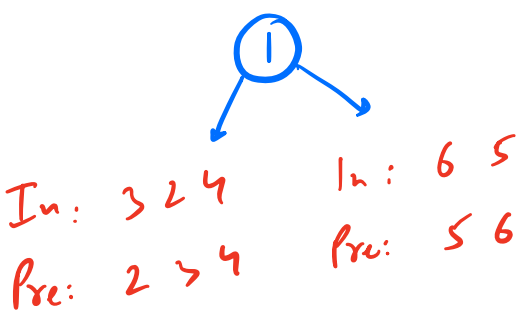


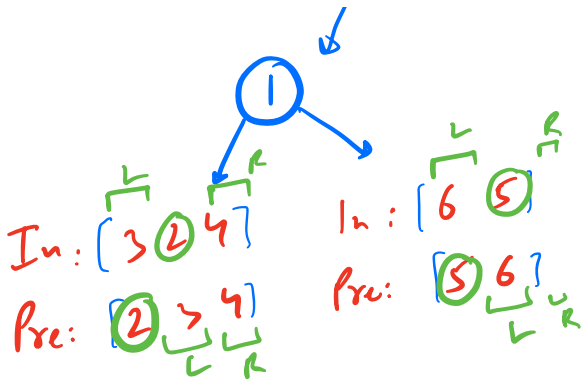
Q Given Inorder & Preorder traversal of a B.T Construct the B.T.

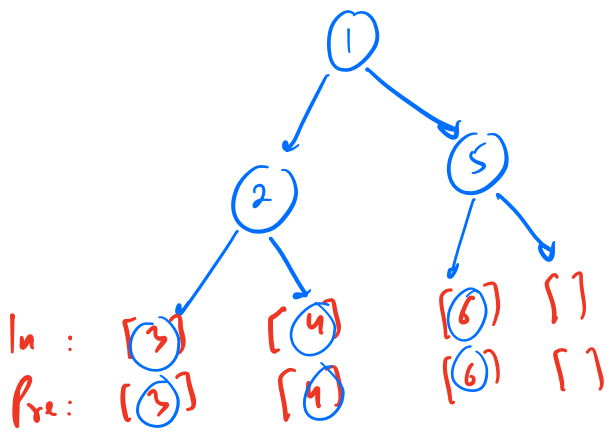
// Values are UNIQUE!

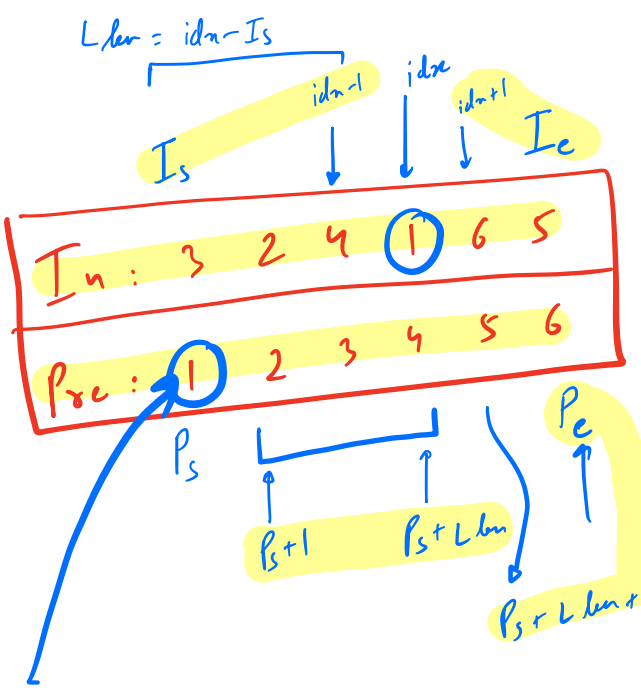












HM

In[i] → i

<3, 0>

<2, 1>

<4, 2>

<1, 3>

<6, 4>

<5, 5>

```
Node Construct ( In[], Pr[], Is, Ie, Ps, Pe ) {
    if ( Is > Ie ) ret NULL;

    // Pr[Ps] → root
→   find it in In[] ⟶ idx   [Using HM → O(1)]
    Llen = idx - Is;
    Node root = new Node ( Pr[Ps] );
    root.left
      = construct (In[], Pr[], Is, idx-1,
                   Ps+1, Ps + Llen);

    root.right
      = construct (In[], Pr[], idx+1, Ie,
                   Ps+Llen+1, Pe);

    ret root;
}
```

TC = O(N)

SC = O(N)