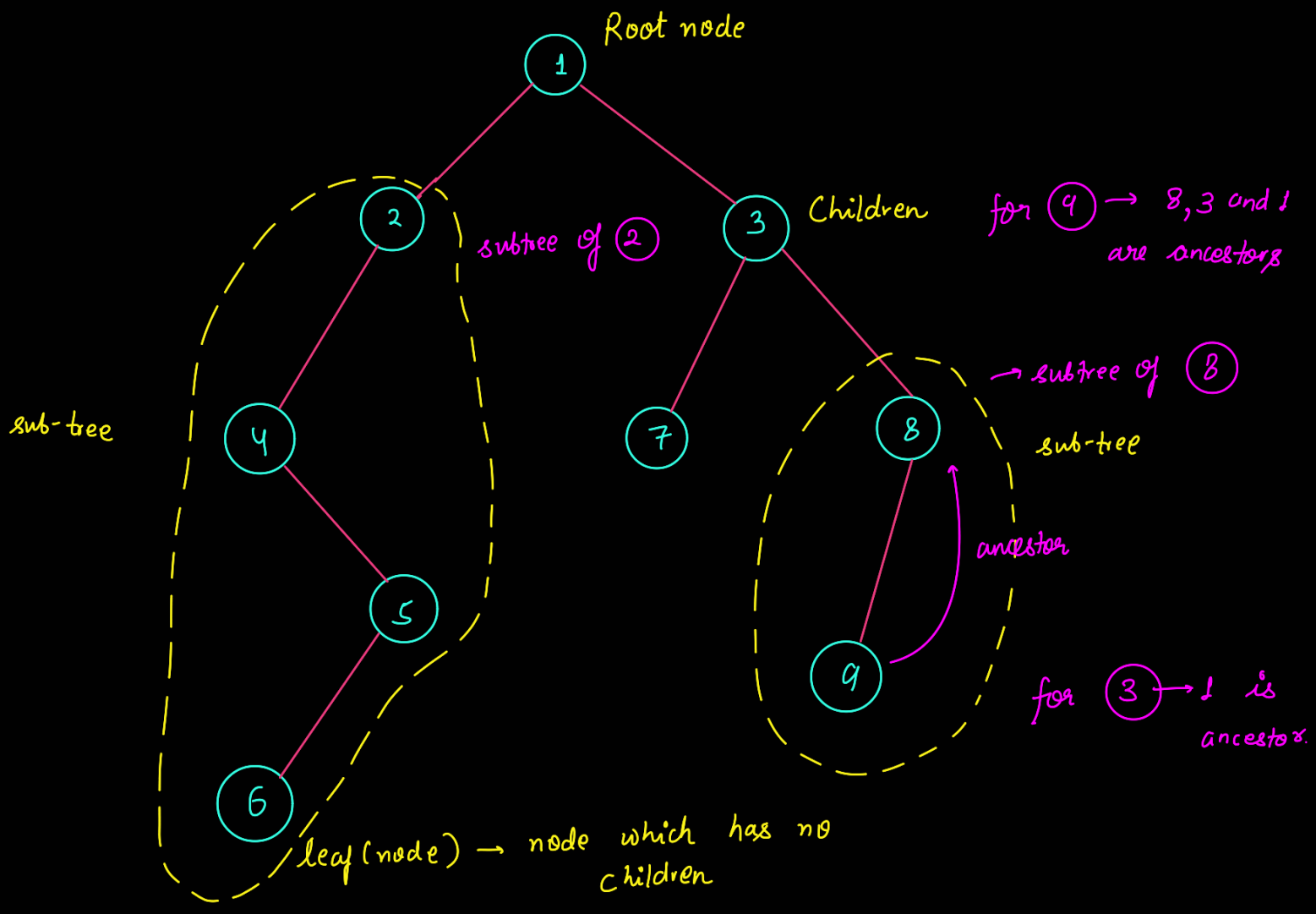
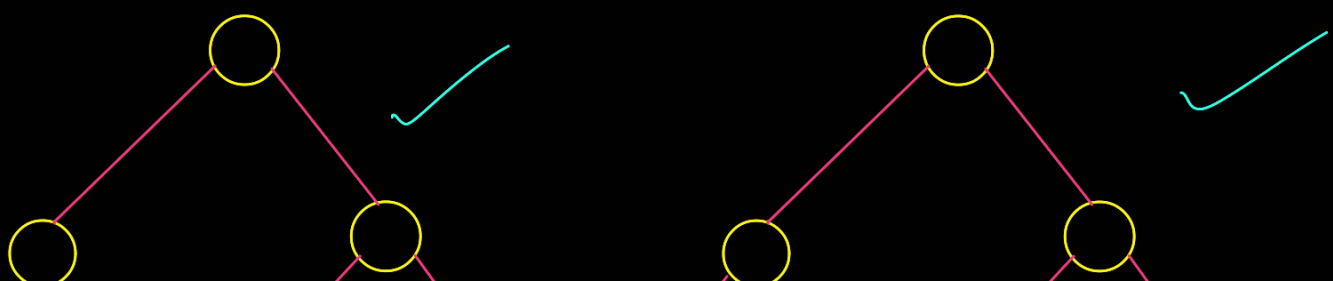
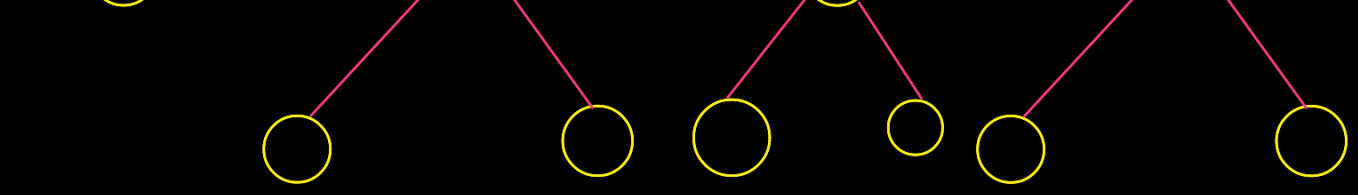
TREES:

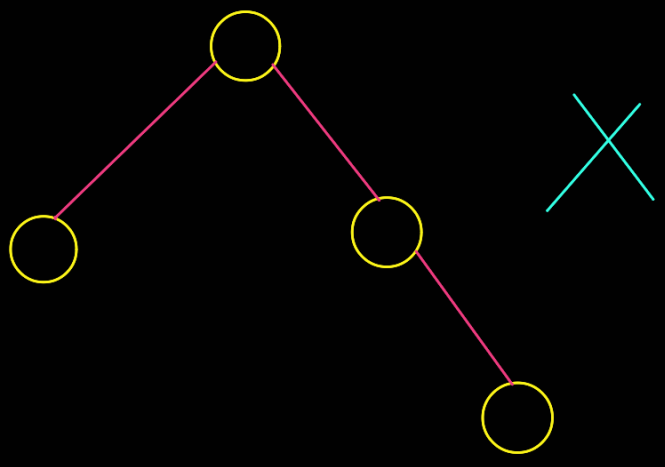


Types of Binary Trees:

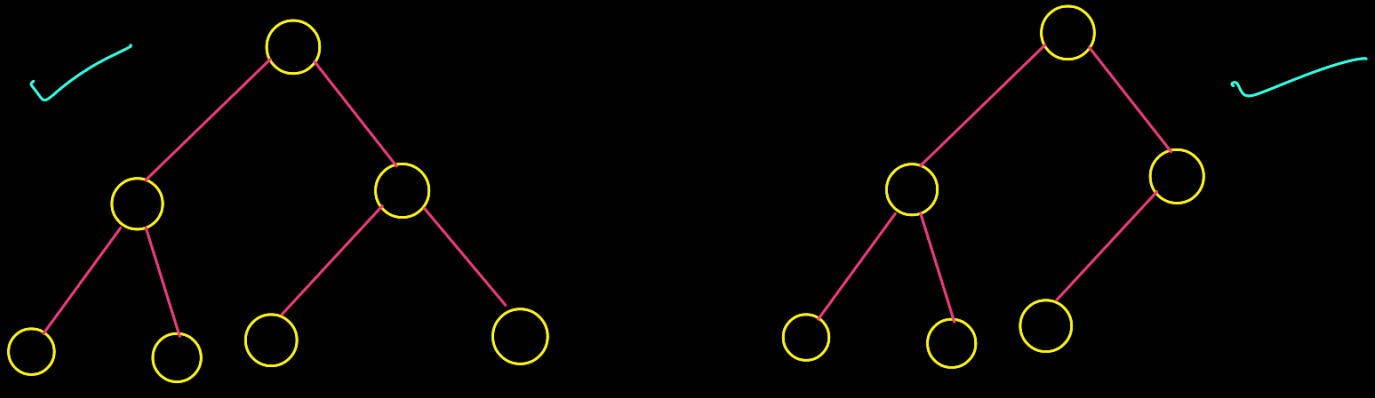
1. Full Binary Tree: either has 0 or 2 children

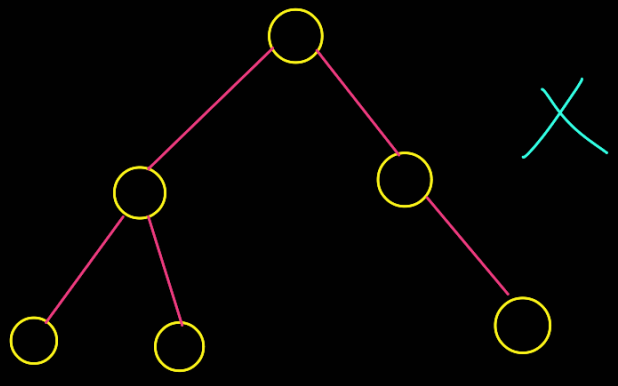




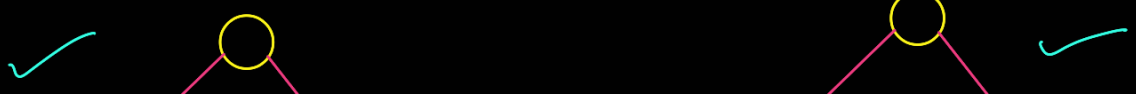


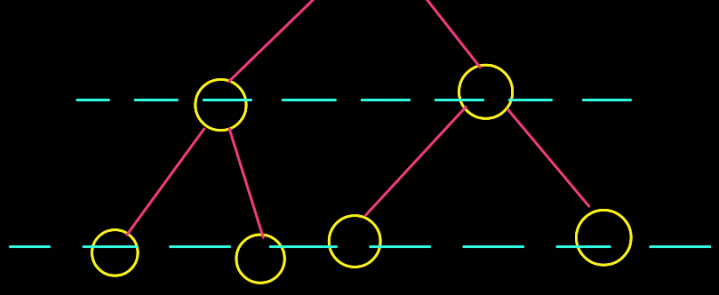
② Complete Binary Tree ⇒ 1) all levels are filled except possibly the last level.
2) the last level has all nodes on left as possible.

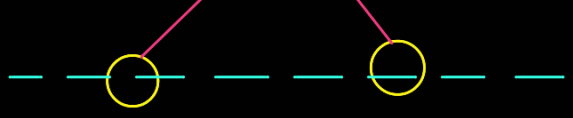


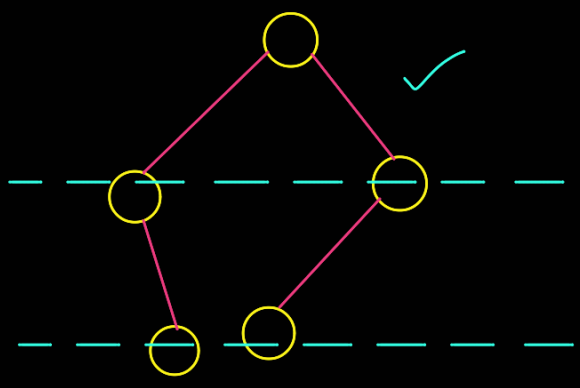


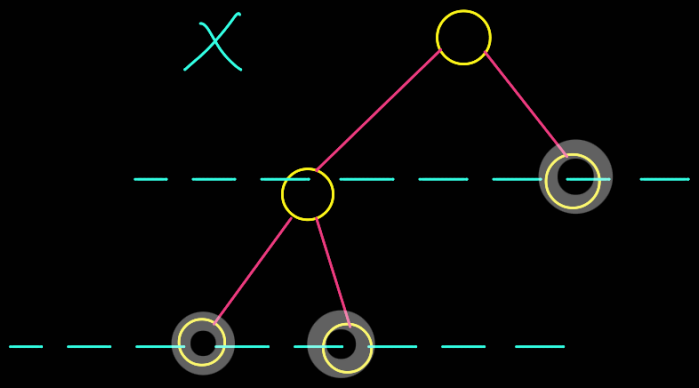
③ Perfect Binary tree : all leaf nodes are at same level.







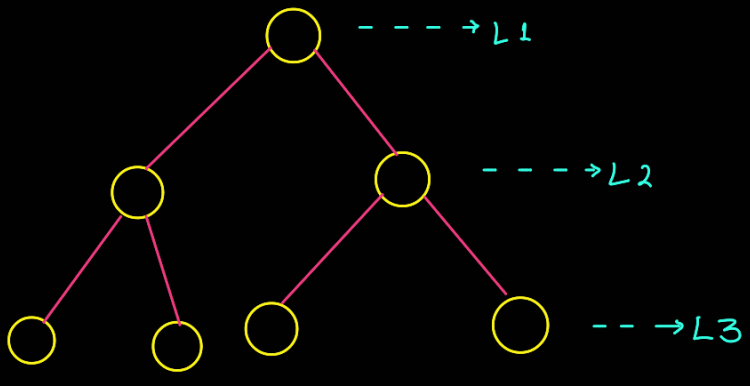




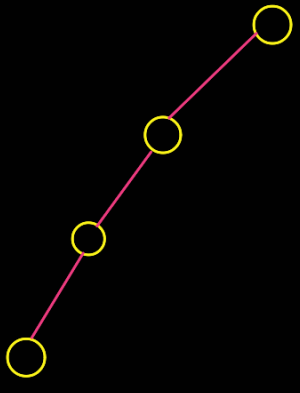
4. Balanced Binary tree : height of tree can at max be $\log_2(N)$
where $N \rightarrow$ no. of nodes

$n = 8$ nodes.

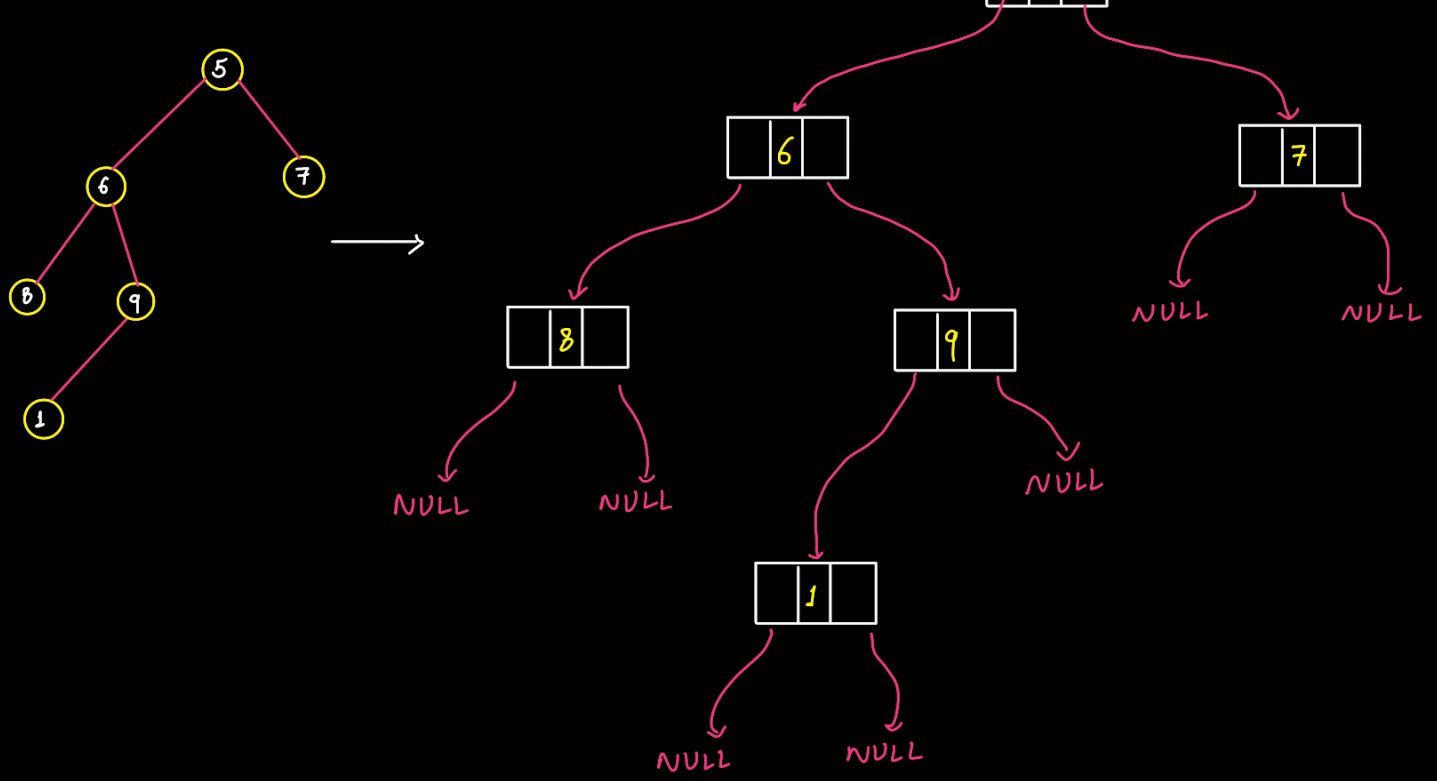
levels $= \log_2 8 = 3$



⑤ Degenerate trees → (linked list)



☆ Representation of Binary tree :

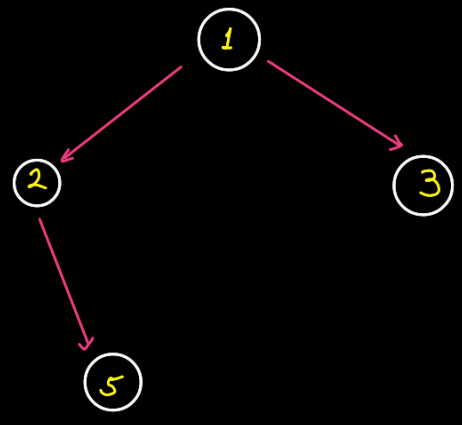


Code →

```
Struct node {
    int data;
    Struct node* left;
    struct node* right;
    node (int val) {
        this → data = val;
        this → left = this → right = NULL;
    }
};
```

eg →

```
main ( ) {
    Struct node * root = new node(1);
    root → left = new node (2);
    root → right = new node (3);
    root → left → right = new node (5);
}
```

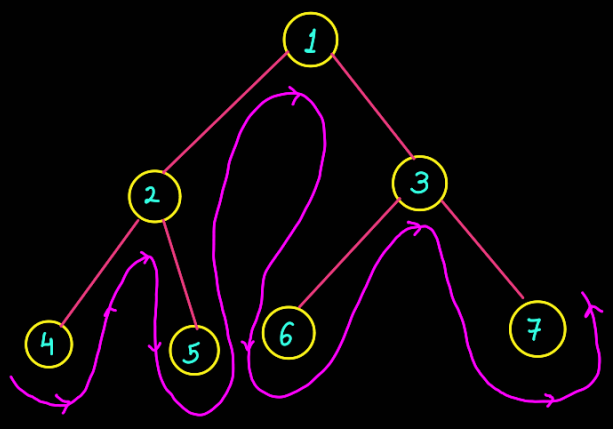


Tree Traversals : (BFS / DFS) → (Breadth first search / depth first search)

DFS → types

- inorder traversal (left root right)
- pre-order traversal (root left Right)
- post-order traversal (left right root)

(I) Inorder Traversal : (left root right)



⇒ 4 2 5 1 6 3 7

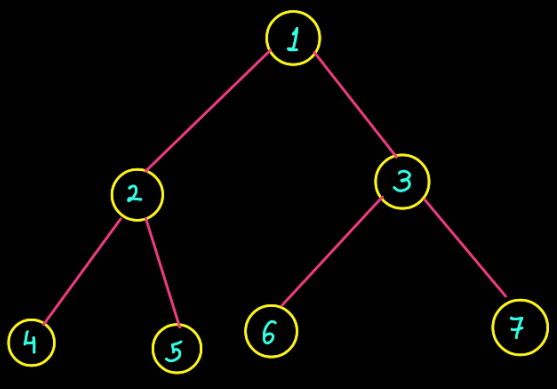
(extreme left Pe Left Root Right)

↓

root

↓

(extreme right Pe left root right)

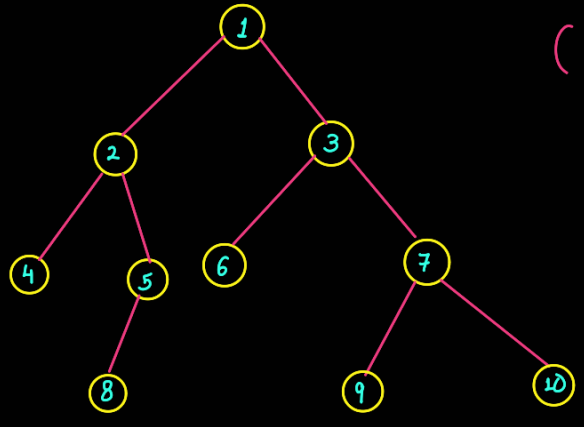


Pre-order traversal → (Root left Right)

1 2 4 5 3 6 7

post order → (left Right Root)

4 5 2 6 7 3 1



(L Ro R)

Inorder : 4 2 8 5 1 6 3 9 7 10

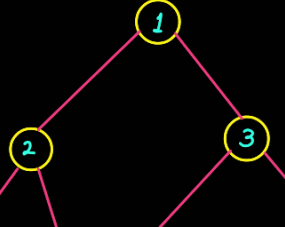
(Ro L R)

Preorder : 1 2 4 5 8 3 6 7 9 10

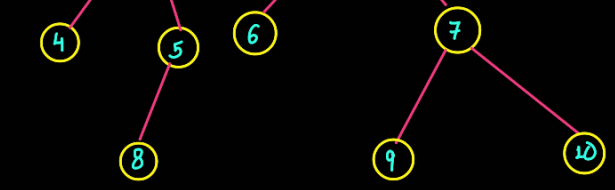
(L R Ro)

Post order : 4 8 5 2 6 9 10 7 3 1

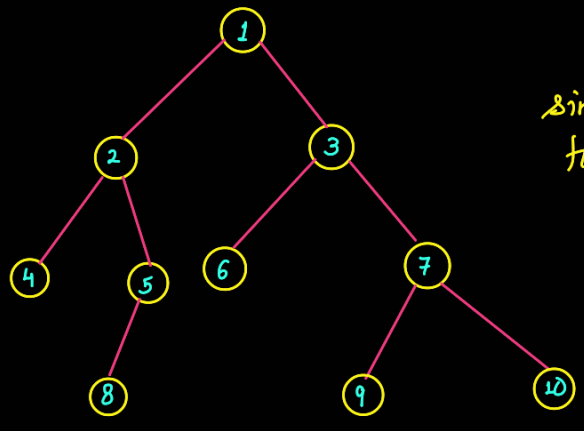
Breadth first Traversal → horizontally.



BFS : 1 2 3 4 5 6 7 8 9 10



BFS (level order traversal)



O/P ⇒ 1 2 3 4 5 6 7 8 9 10

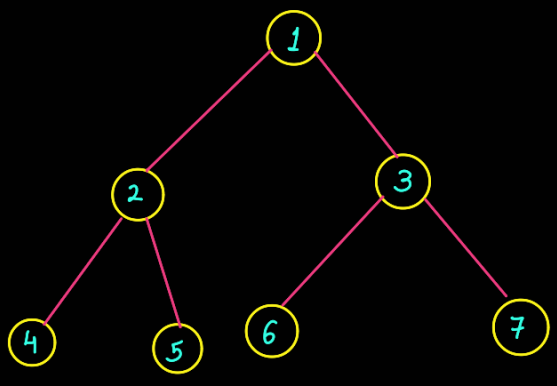
simple steps to follow →

1. keep printing the nodes one by one and simultaneously, if the curr. node has any left / right, add it to the queue.)

```
vector < vector <int >>  level ( node * root) {
    vector <vector <int >>ans. ;
   queue <node> q;
   q.push (root);
  while ( ! q.emty()) {
     int size = q.size();
     vector <int> level;
   for (int i =0; i<size; i++){
      TreeNode * node = q.front();
          q.pop();
      if (node → left) q.push (node → left);
    if (node → right) q.push (node → right);
    level.push_back (node → val);
     }
```

```
        ans.push_back(level);
    }
    return ans;
}
```

Iterative traversals:



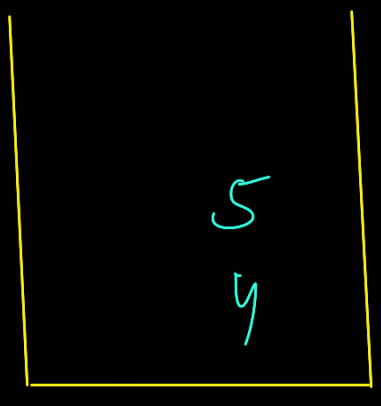
Pre-order traversal → (Root Left Right)

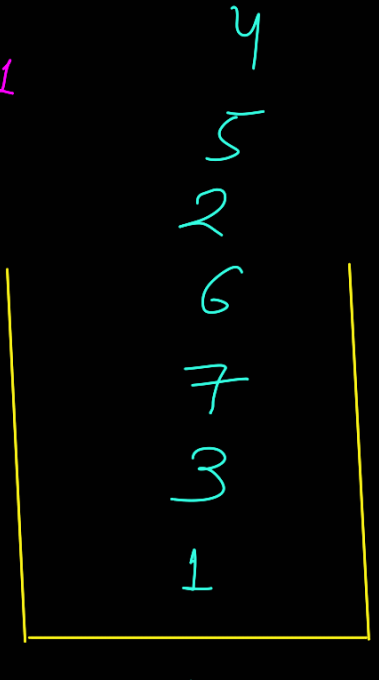
1 2 4 5 3 6 7

hint → use stack and push right node before left node because you have to process left node first.

post order → (Left Right Root)

4 5 2 6 7 3 1





st2

st1

curr = st2.top(); st2.pop();

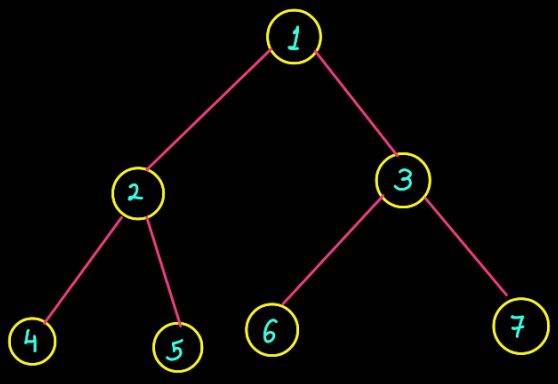
curr → left ? st1.push(curr→left);

curr → right ? st1.push(curr→right);

(LRoR)

inorder:

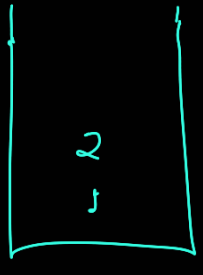
4 2 5 1 6 3 7



curr = null.

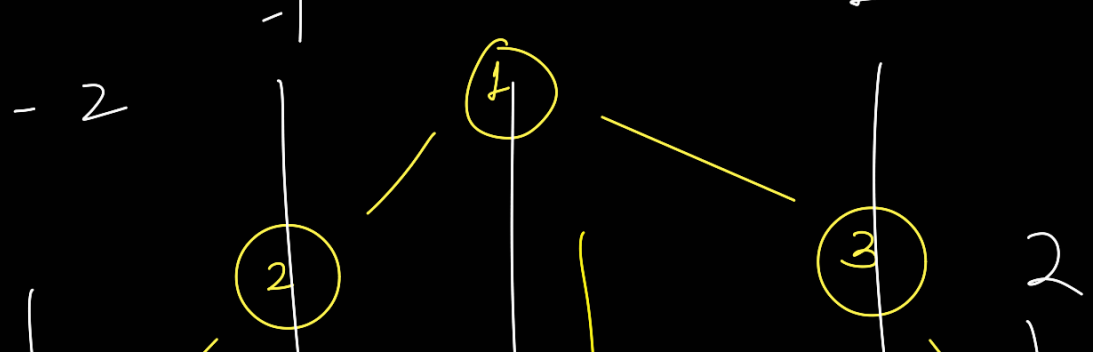
temp = null.

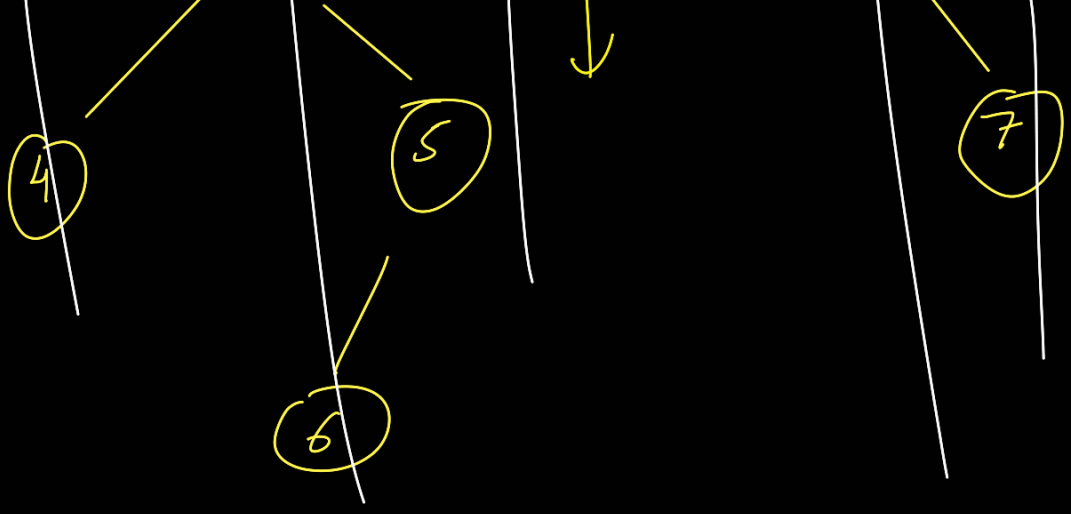
hint keep filling left nodes and moving to node = node → left until you find leaf node then print stack's top and move to node's right.

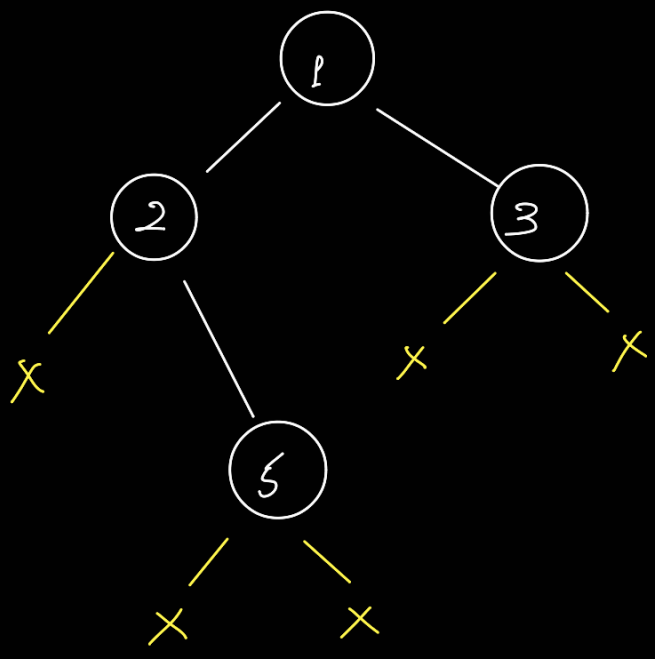


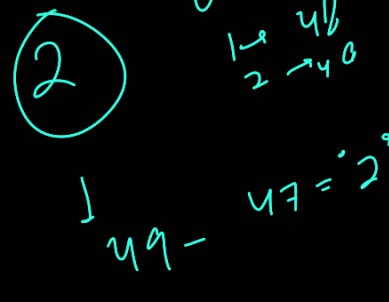
queue < ① , ② , value).

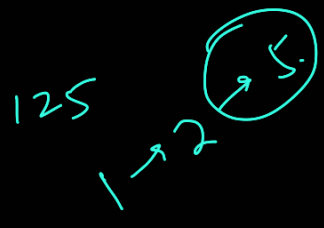
map < int, map < int, multiset < int > > >;











1 {} "1"

```
f( root, ans, temp){
  if (!root) ret;
  if (leaf) ans.push(stoi);
  temp += root->val; 1
  f( root->left, ans, temp); 0
  f( root->right, ans, temp); ✓
```

3 2 {} 12

```
f( root, ans, temp ){
  if (!root) ret;
  if (leaf) ans.push(stoi);
  temp += root->val; 12
  f( root->left, ans, temp); ✓
  f( root->right, ans, temp); ✓
```

remove

5 {} 125

```
f( root, ans, temp){
  if (!root) ret;
  if (leaf) ans.push(stoi);
  temp += root->val;
  f( root->left, ans, temp);
  f( root->right, ans, temp);
```

```
f( root, ans, temp) {
  if (!root) ret;
```

```
f( root, ans, temp){
  if (!root) ret;
```

```
f( root, ans, temp){
  if (!root) ret;
```

```
if (leaf) ans.push(sto);
temp += root->val;
f(root->left, ans, temp);
f(root->right, ans, temp);
```


```
if (leaf) ans.push(sto);
temp += root->val;
f(root->left, ans, temp);
f(root->right, ans, temp);
```

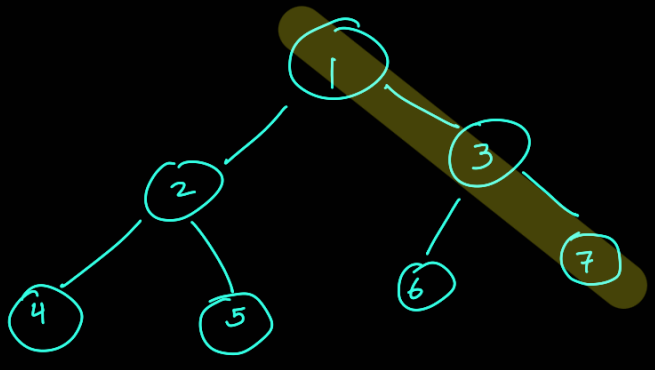
```
if (leaf) ans.push(sto);
temp += root->val;
f(root->left, ans, temp);
f(root->right, ans, temp);
```


```
f(root, ans, temp){
if (!root) ret;
if (leaf) ans.push(sto);
temp += root->val;
f(root->left, ans, temp);
f(root->right, ans, temp);
```


```
f(root, ans, temp){
if (!root) ret;
if (leaf) ans.push(sto);
temp += root->val;
f(root->left, ans, temp);
f(root->right, ans, temp);
```

```
f(root, ans, temp){
if (!root) ret;
if (leaf) ans.push(sto);
temp += root->val;
f(root->left, ans, temp);
f(root->right, ans, temp);
```


$x = 7$



1

```
f(root, x, ans, final)
if (!root) ret;
if (root->val == x){
    final = ans;
    return;}
ans.push(root);
helper(root->left, x, ans, final);
```

ans = 1

2 1 3

```
f(root, x, ans, final)
if (!root) ret;
if (root->val == x){
    final = ans;
    return;}
ans.push(root);
helper(root->left, x, ans, final);
```

1 2

0

```
helper(root->right, x, ans, final);
ans.pop();
```

```
helper(root->right, x, ans, final);
ans.pop();
```

null 124

```
f(root, x, ans, final)
if(!root) ret;
if(root->val==x){
    final=ans;
    return;}
ans.push(root);
helper(root->left, x, ans, final);
helper(root->right, x, ans, final);
ans.pop();
```

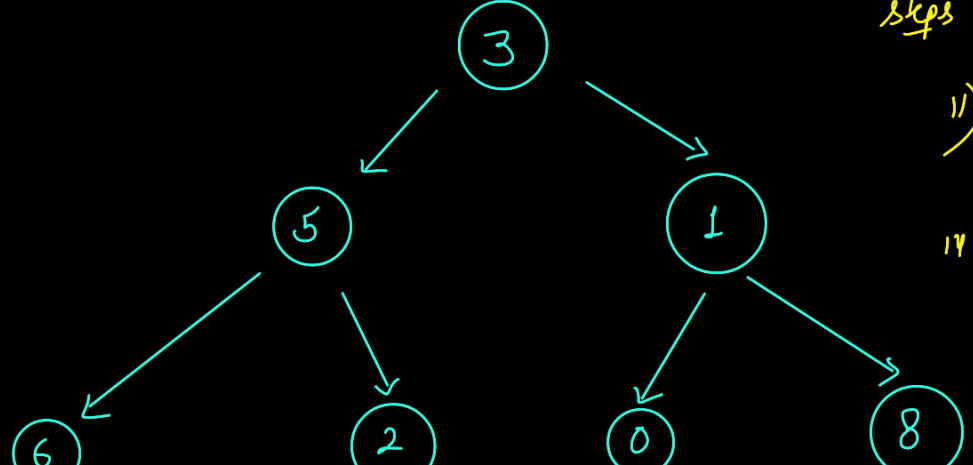
7 13

```
f(root, x, ans, final)
if(!root) ret;
if(root->val==x){
    final=ans;
    return;}
ans.push(root);
helper(root->left, x, ans, final);
helper(root->right, x, ans, final);
ans.pop();
```

```
f(root, x, ans, final)
if(!root) ret;
if(root->val==x){
    final=ans;
    return;}
ans.push(root);
helper(root->left, x, ans, final);
helper(root->right, x, ans, final);
ans.pop();
```

```
f(root, x, ans, final)
if(!root) ret;
if(root->val==x){
    final=ans;
    return;}
ans.push(root);
helper(root->left, x, ans, final);
helper(root->right, x, ans, final);
ans.pop();
```

[Print nodes at a distance of K in Binary Tree]:

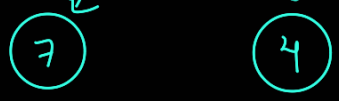


steps → i) parent store in map

ii) move

iii) if dis == k { v.push(curr)

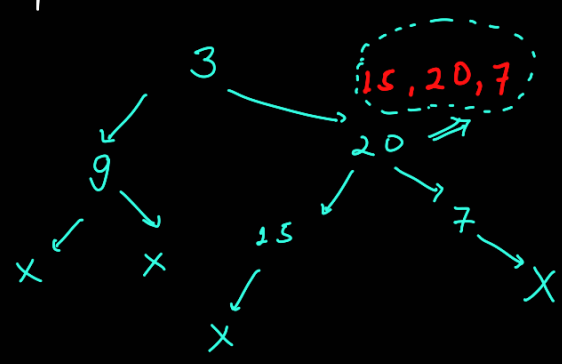
return v;



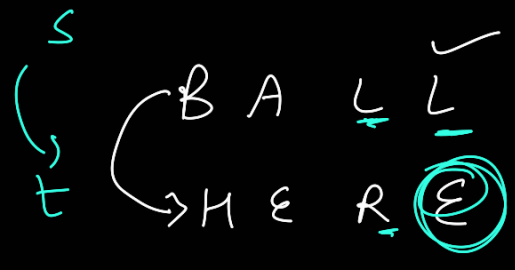
Ques: Imp - Topic → Construct Binary Tree From In-Order and Post / Pre-Order Traversals :

eg → pre Order : [3, 9, 20, 15, 7] Inorder : [9, 3, 15, 20, 7]

pstart, preend, instart, inend



15, 20, 7
Pre: 20, 15, 7
In: 15, 20, 7



Pre: 15
In: 15

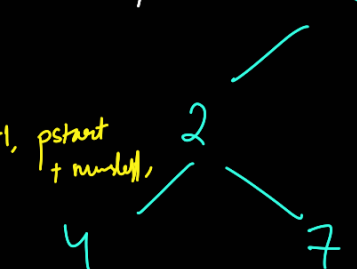
t
ps ↓ pstart ↓ pstart + numsleft ↓ pend (n-1)
pre = 1, 2, 4, 7, 3
inroot
In = 4, 2, 7, 1, 3
↑ instart ↑ inend

inroot = 1
numsleft = inroot - instart = 3

Step 1 → map inorder to index;

root = new Tre (preod [presta]) (1) f (preorder, 0, n-1, inorder, 0, n-1, map)

left = f (pre, pstart+1, pstart + numsleft, inorder,



3

right = f (pre, numsleft+1, pend, inorder, inroot+1, inend)

kyu → bcuz hume preorder se inorder dhundna hai.

4	0
2	1
7	2
1	3
3	4

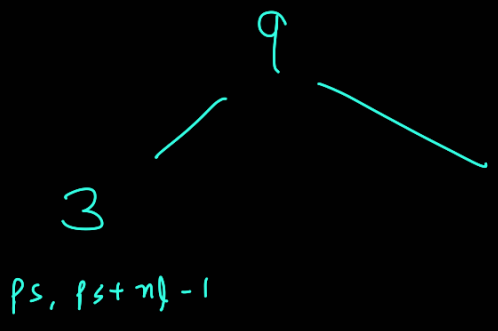
instart | inroot | inend

inorder = 3, 9, 15, 20, 7

postorder = 3, 15, 7, 20, 9

ps, pe, root, left, Root

①



[Morris Traversal : (Pre-order and inorder)

Recursive / Iterative traversal :

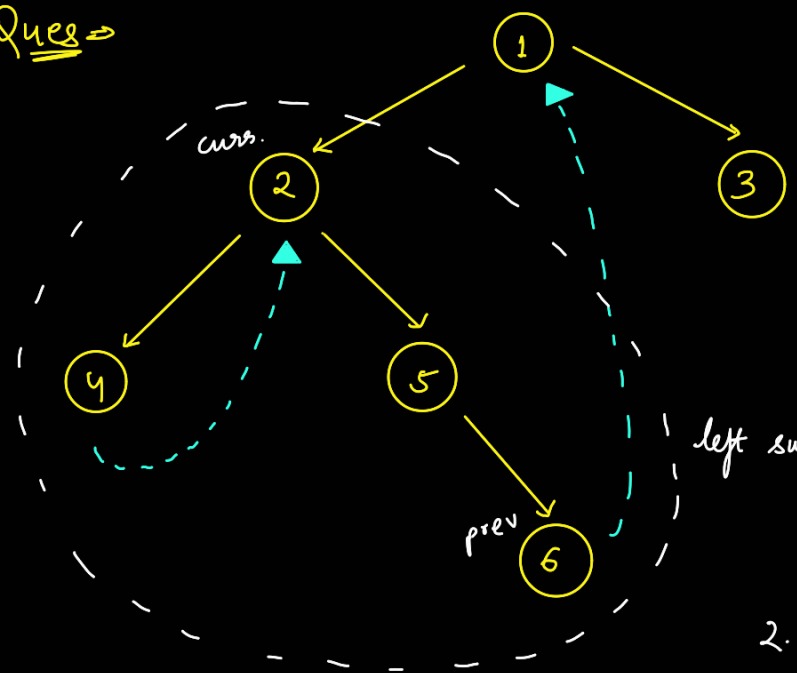
time : O(N)
space : O(N)

Morris traversal

time : O(N)
space : O(1)

* uses the concept of threaded binary tree.

Ques ⇒



Left Root Right

Inorder → 4 2 5 6 1 3

☆ Morris-Traversal : O(1) space
O(N) time

Observations :

left subtree 1. If (left → null)
print (curr)
move → right

2. left → rightmost grey thread on left current

subtree

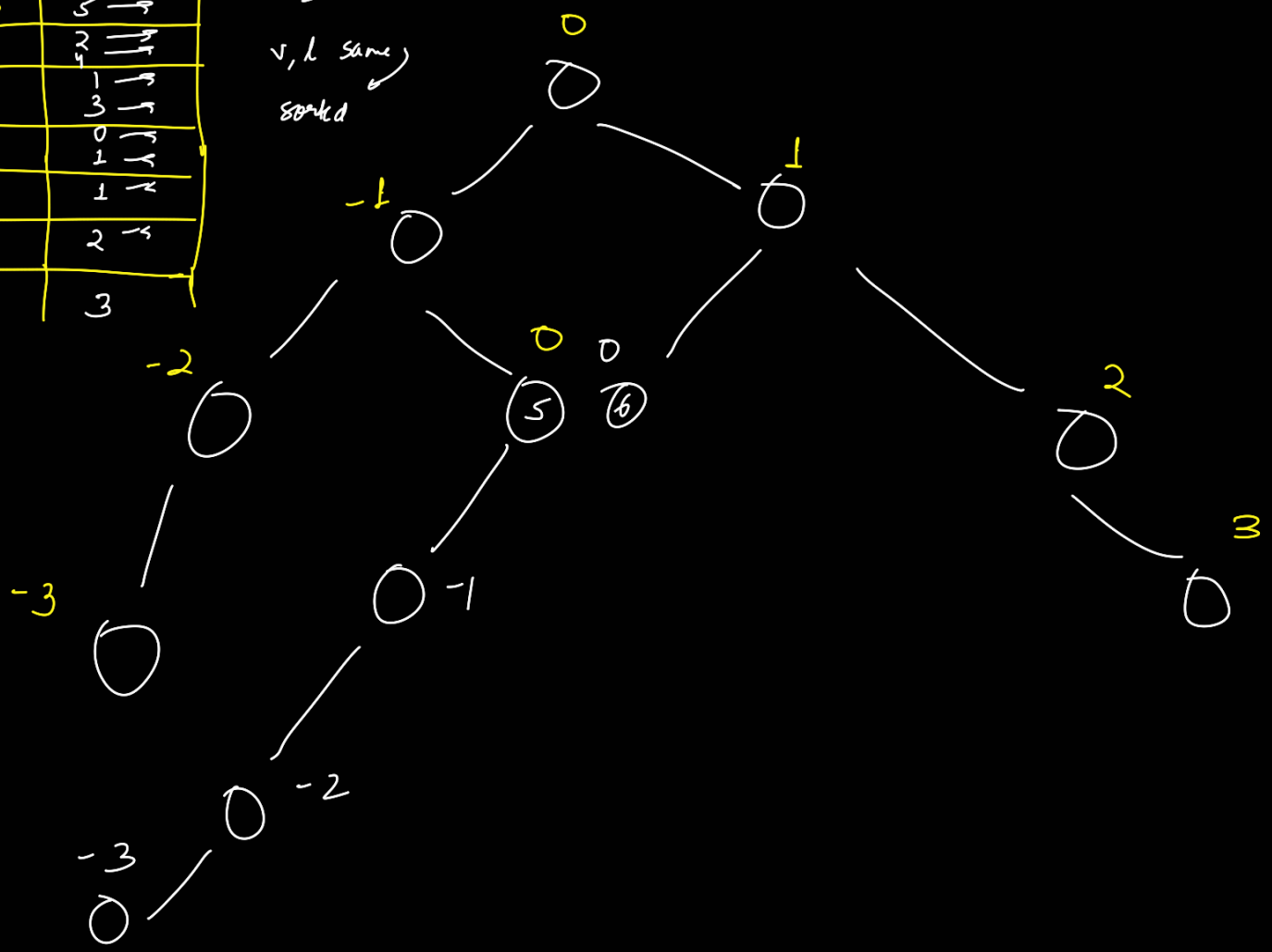
then move curr → left

If thread already exists, remove it and move curr → right.

sorted vertical	sorted level	value sorted
-3	3 → 5,6	multiset.
-2	2 → 4	
-1	1 → 3	
0	0 → 1	
1	1 →	
2	2 →	
3	3	

(v, l same) sorted

vertical, level, values.

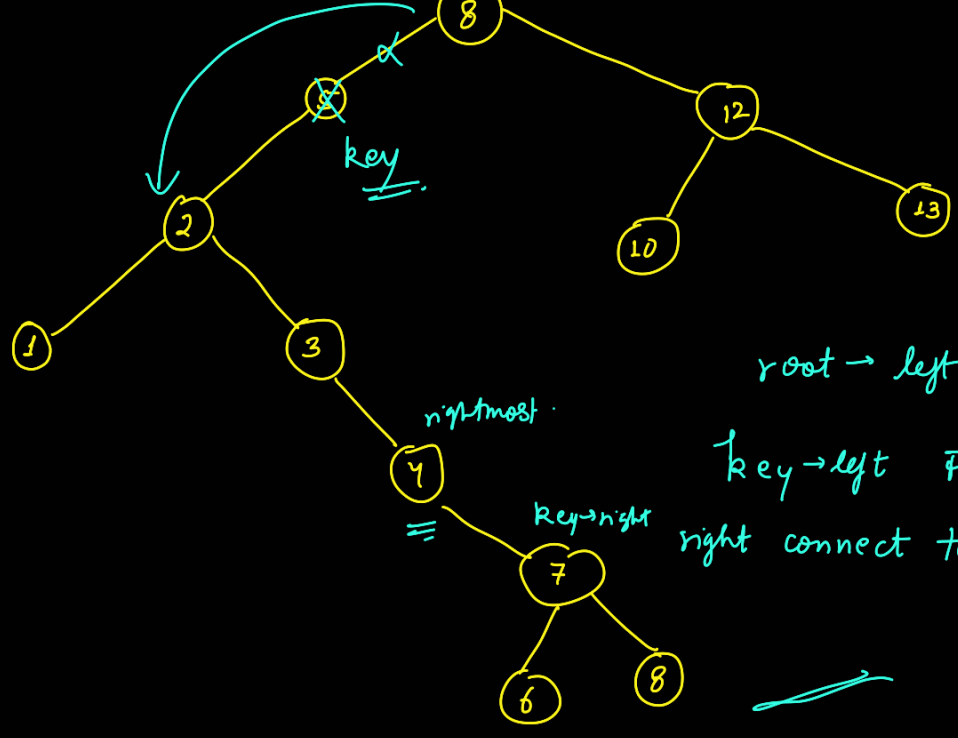


[Binary Search Tree]:

↳ i) L < N < R
ii) Left subtree → BST
iii) Right subtree → BST

↳ Height is generally kept $\log_2 N$

key = 5

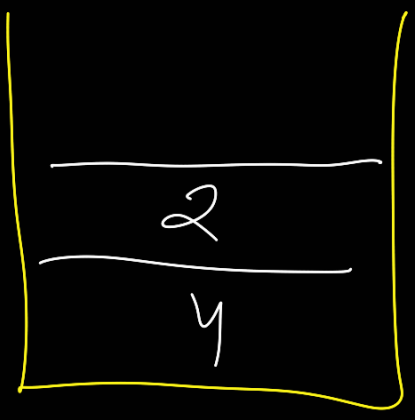


root → left = key → left

key → left 的 rightmost 的 right connect to key → right.

array = 1, 3, 4, 2

nge = 3, 4, -1, -1



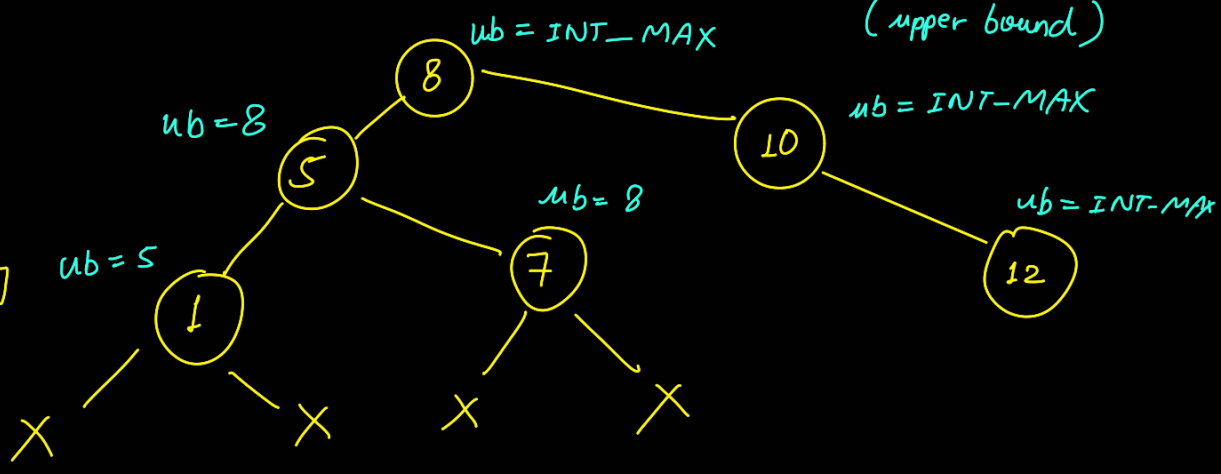

```
if a[i] > st.top() {
while (!st.empty) { and a[i] > st.top()
nge[i] = a[i];
st.pop();
st.push(a[i])
```

0 1 2 3 4 5 6 7 8 9

1, 2, 3, 7, 6, 5, 4, 9, 2, 0

2 3 2 7 9 9 9

[Ques]:



inorder: [8, 5, 1, 7, 10, 12]

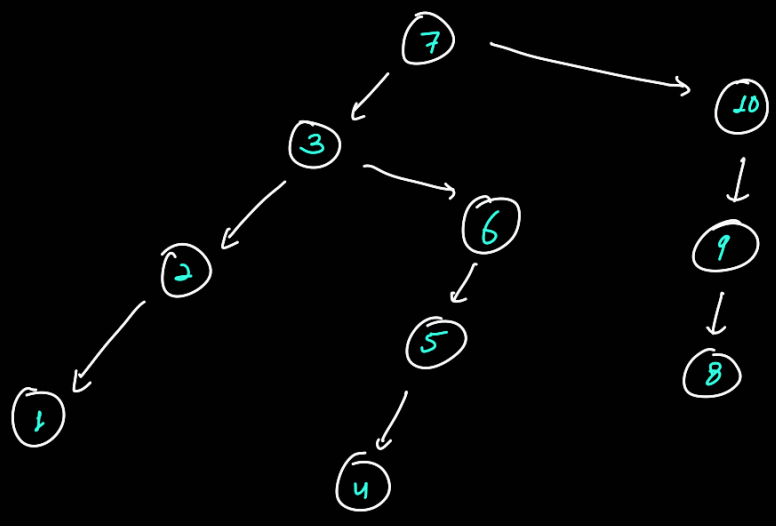
node → left → passing node → val as upperbound.
node → right → passing prev ub as new upperbound.

BST iterator of inorder

S.C ⇒ $O(H)$ → $\log_2 N$ and T.C. = $O(1)$

Left Root Right

Stack → save left daal do
If right ho to right jake save left daal do.



SC = O(H)
TC = O(N/N)
= 1

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Ques:

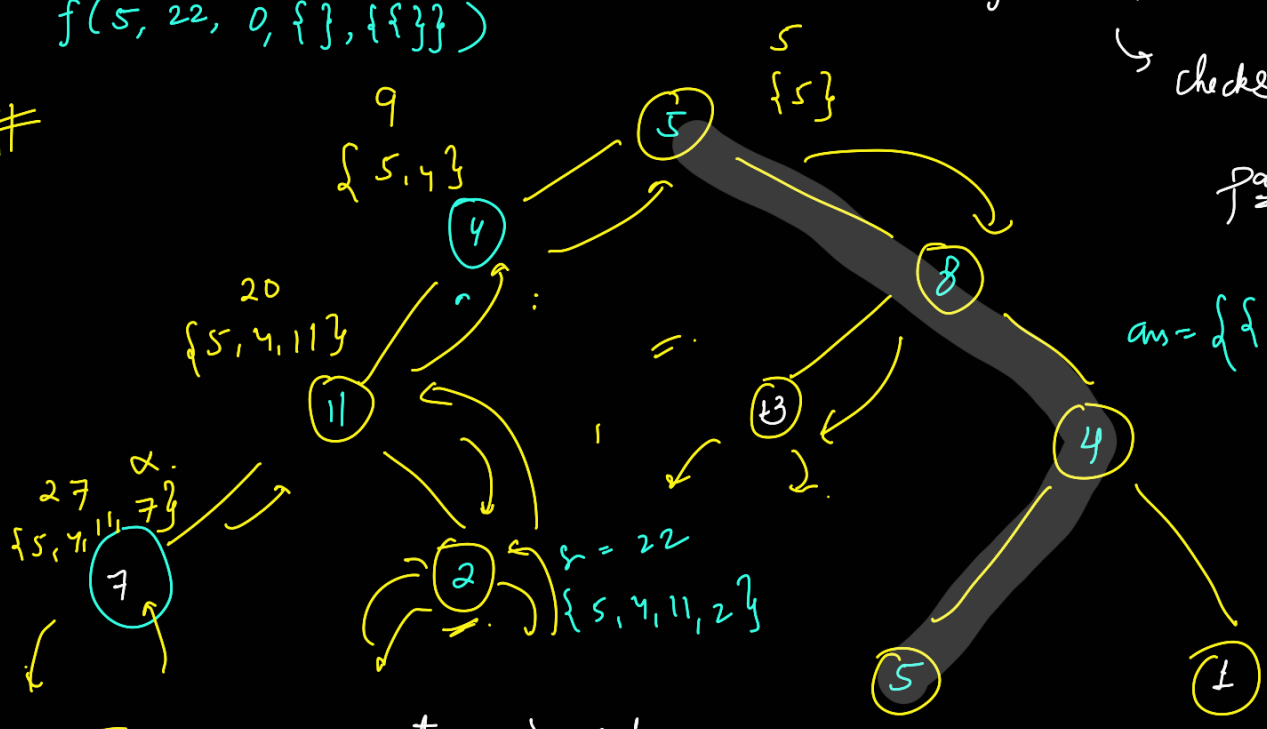
TC = O(n) + O(n) ⇒ O(n)

SC = O(n)

f(arr, matrix, sum, node)

f(5, 22, 0, {}, {{}})

#



↳ checks node → leaf tak possible paths.

ans = {{5,4,11,2}, }

requirements : i) vector

ii) 2d vector

iii) sum variable

base case : if root = NULL return

remember → while backtracking, remove the node from sum,