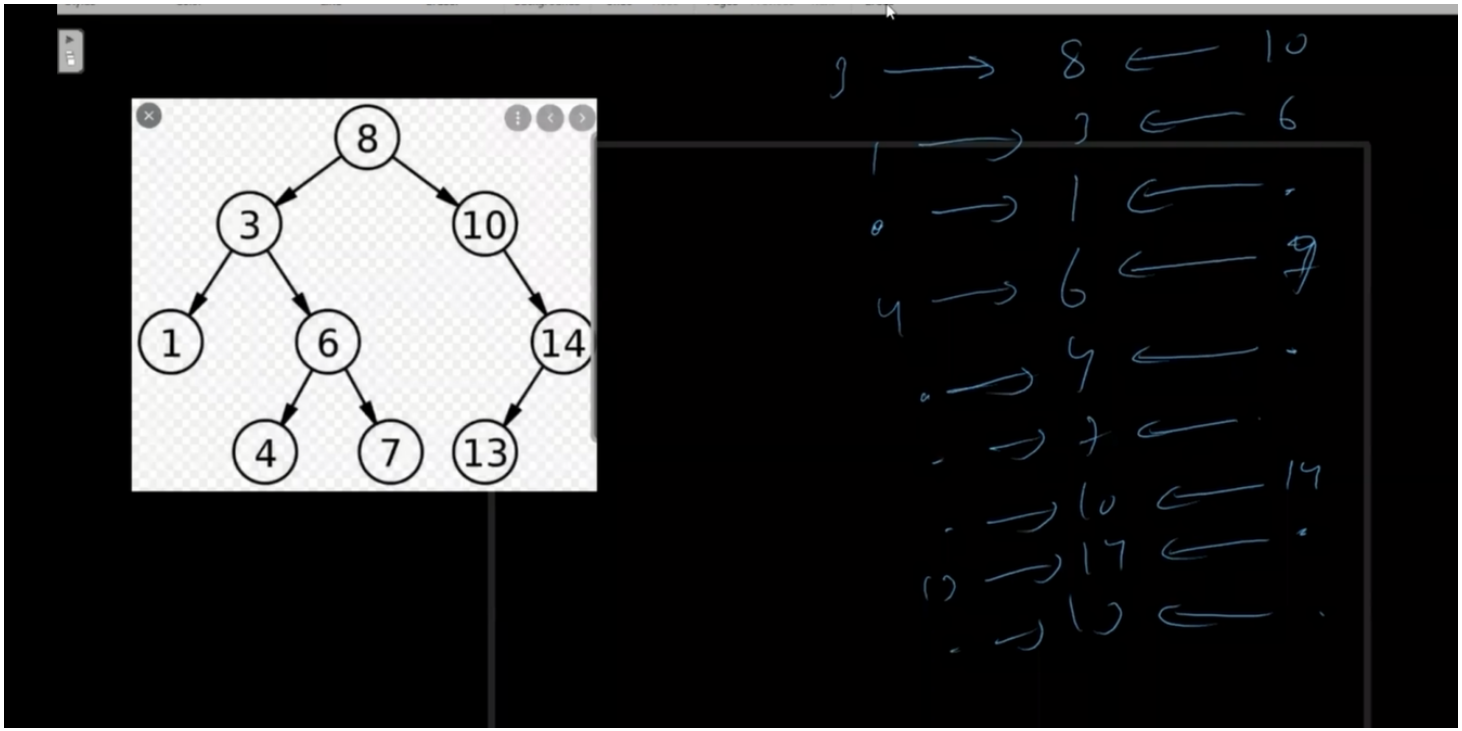
Trees

08 July 2022 08:34

x

1. Print a Binary Tree




```
public static void display(TreeNode node) {
    if (node == null)
        return;

    String str = "";
    str += ((node.left != null ? Integer.toString(node.left.val) : "."));
    str += (" -> " + Integer.toString(node.val) + " <- ");
    str += ((node.right != null ? Integer.toString(node.right.val) : "."));

    System.out.println(str);
```

```java
        display(node.left);
        display(node.right);
    }
```

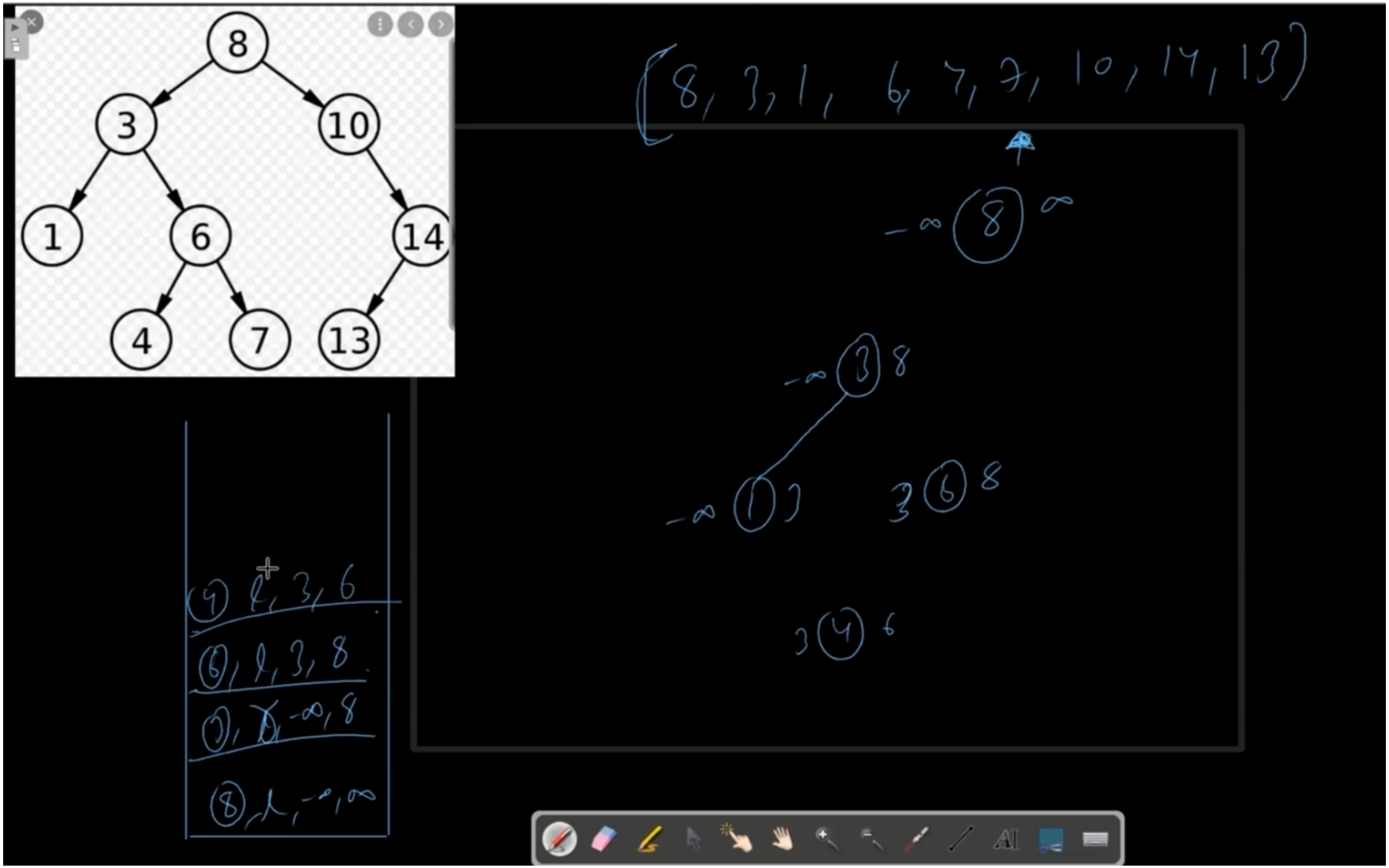
```java
    public static void display1(TreeNode node) {
        if (node == null)
            return;

        StringBuilder sb = new StringBuilder();
        sb.append((node.left != null ? node.left.val : "."));
        sb.append(" -> " + node.val + " <- ");
        sb.append((node.right != null ? node.right.val : "."));

        System.out.println(sb.toString());

        display1(node.left);
        display1(node.right);

    }
```

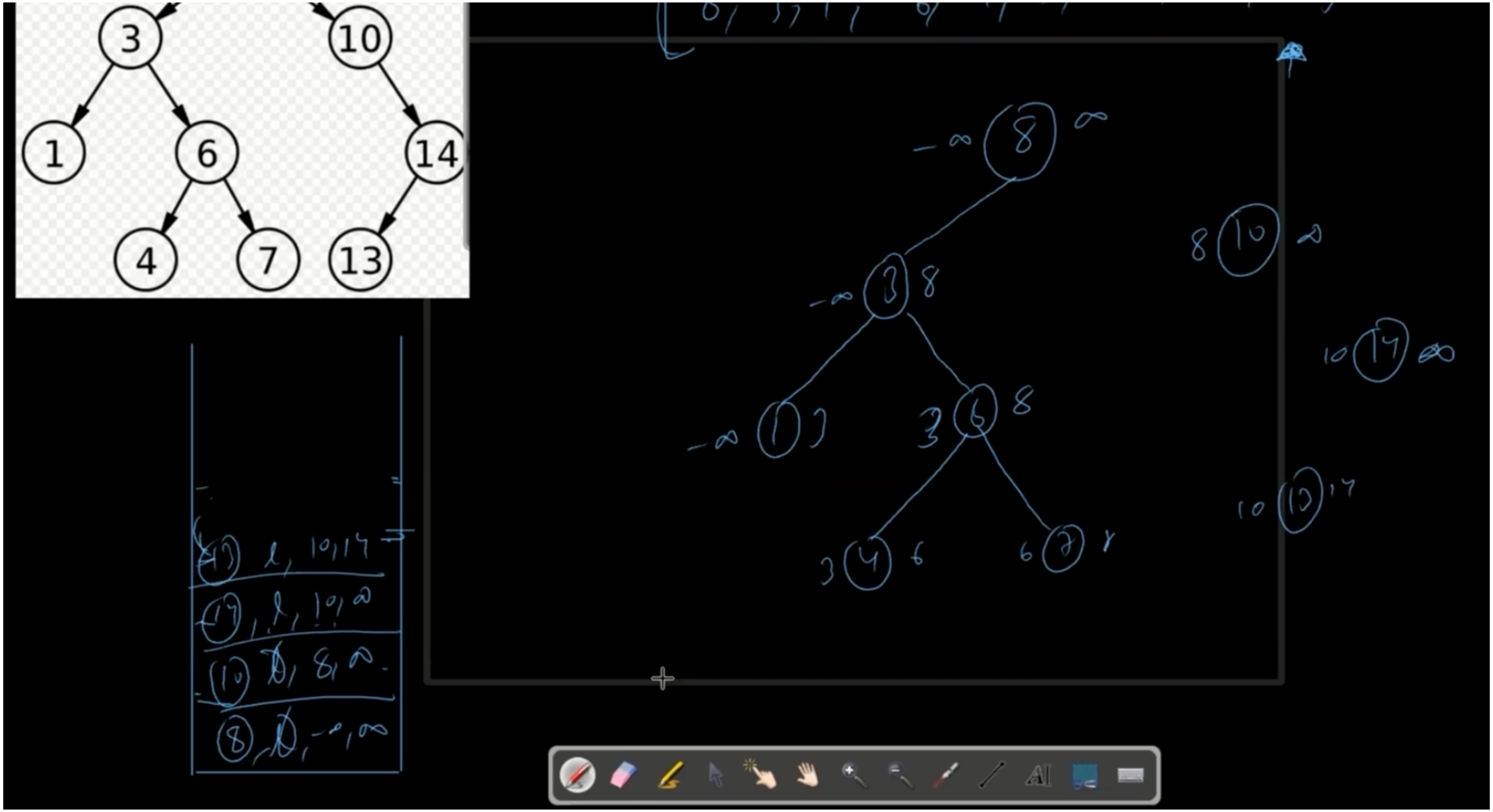
2. Construct a BST from inorder of a BST

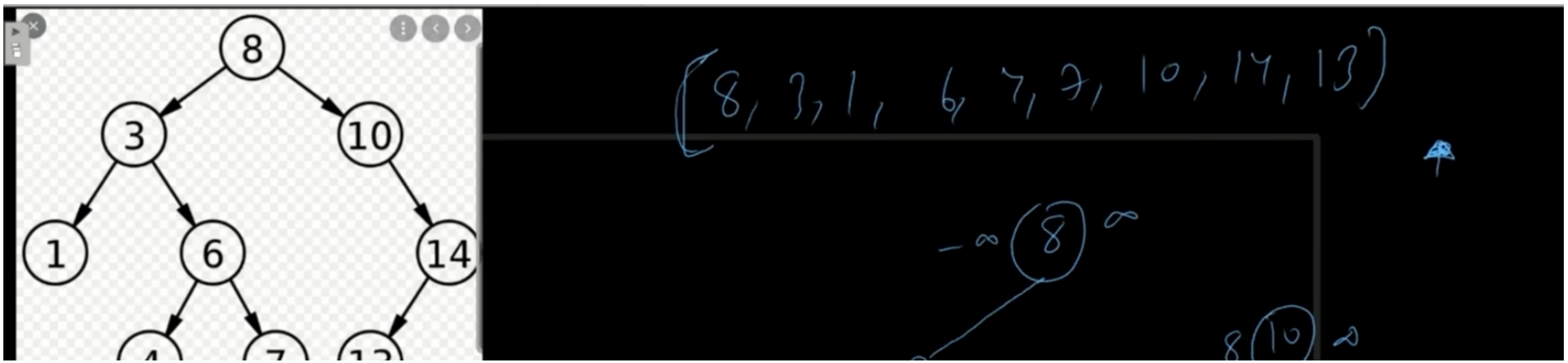
```java
    // Construct Tree form inorder . i.e sorted array

    // Basic faith :
    // Maine sabse middle wale ko root banaya.
    // Jo middle se left wale index hain wo sare left subtree ka part banaege kyunki
    // wo middle se chote hain because array is sorted

    // Jo middle se right wale hain wo right subtree ka part banenge

    // Isi kaam ko karne ke liye mai index ko mask karke bhej raha hun
    // Therefore starting index and ending index are used.

    // Best is to do stack dry run for the following test case :
    // 10, 20, 28, 31, 40, 42, 56, 68, 90

    public static TreeNode construct_BST_from_sorted_array(int[] arr, int si, int ei) {

        if (si > ei)
            return null;
        int mid = (si + ei) / 2;
        TreeNode root = new TreeNode(arr[mid]);

        root.left = construct_BST_from_sorted_array(arr, si, mid - 1);
        root.right = construct_BST_from_sorted_array(arr, mid + 1, ei);

        return root;
    }
```

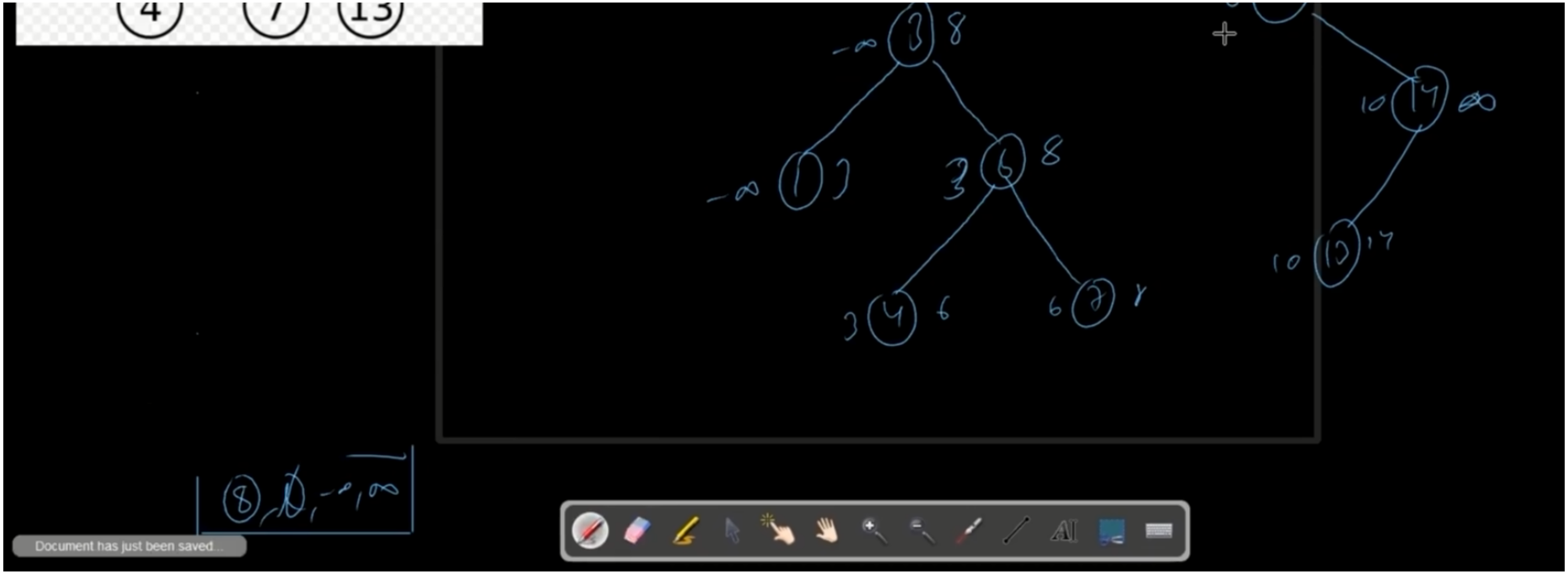
```java
public static TreeNode construct_BST_from_sorted_array(int[] arr) {
    return construct_BST_from_sorted_array(arr, si: 0, arr.length - 1);
}
```

3. Construct a BST from preOrder

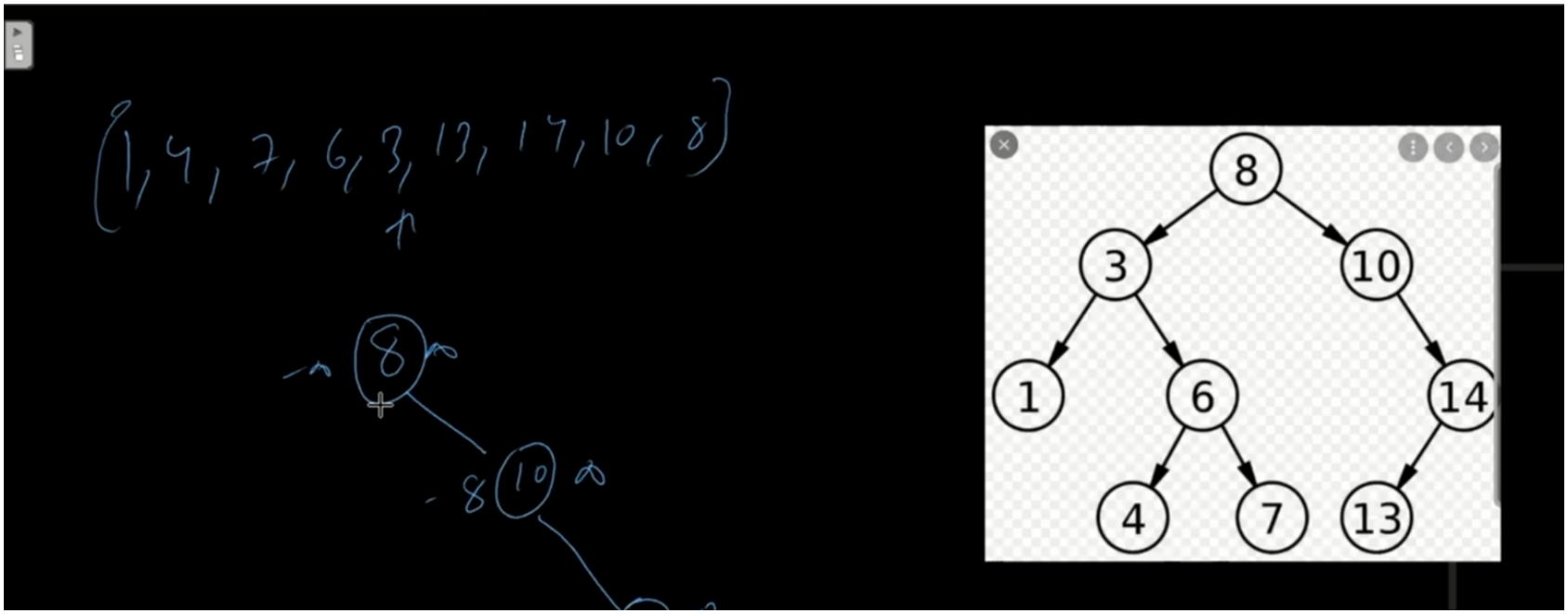


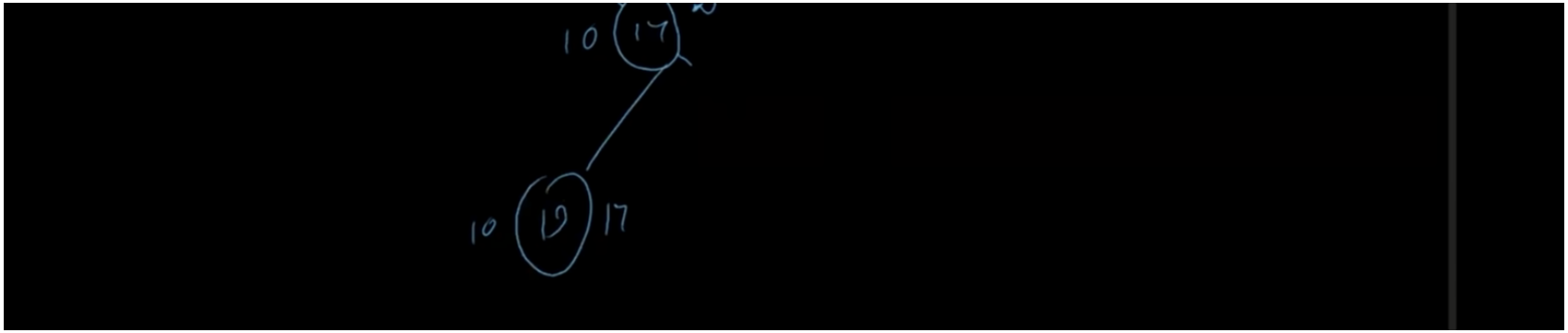

3
10
1
6
14
4
7
13

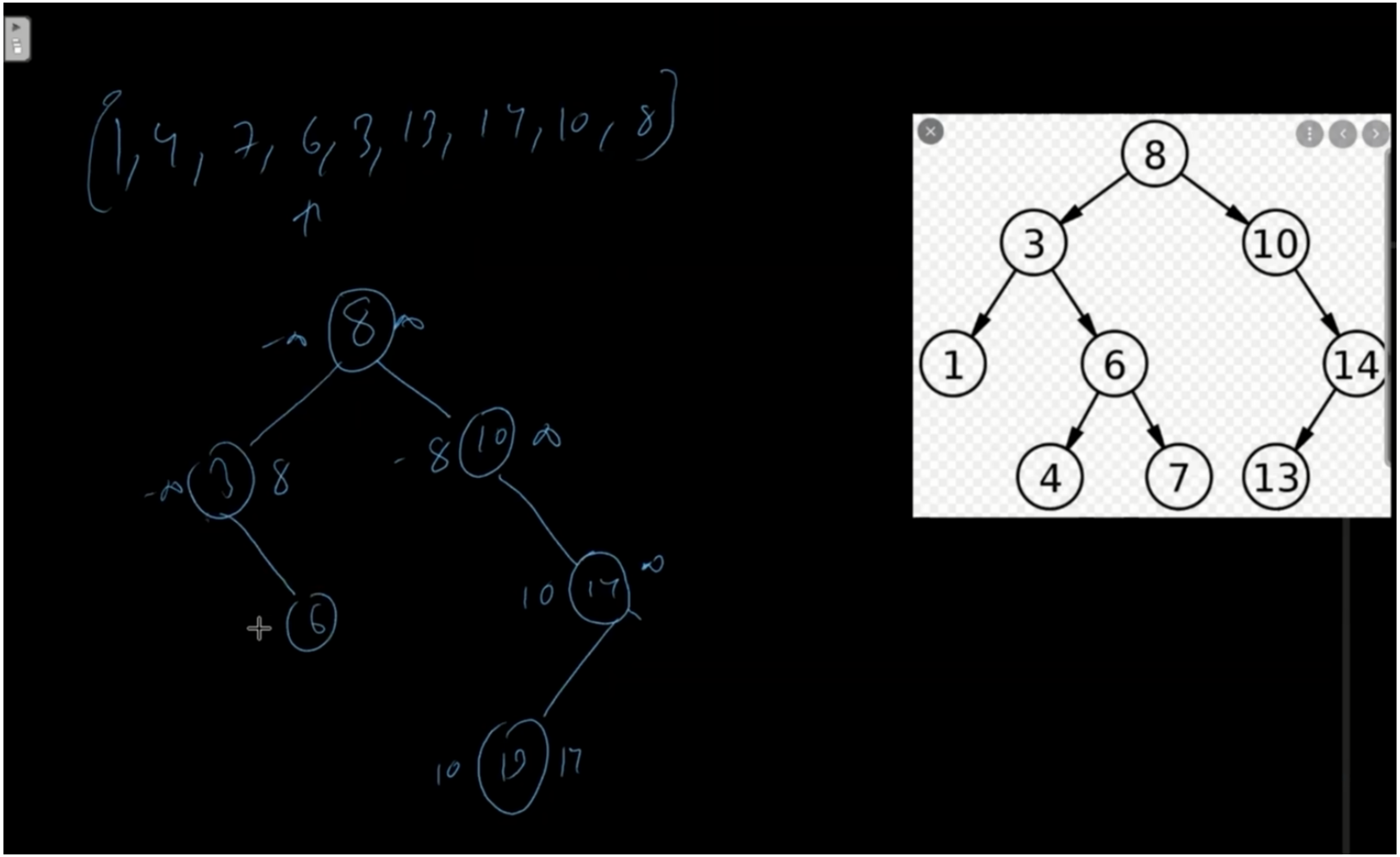

8
3
10
1
6
14
4
7
13
(8, 3, 1, 6, 4, 7, 10, 14, 13)



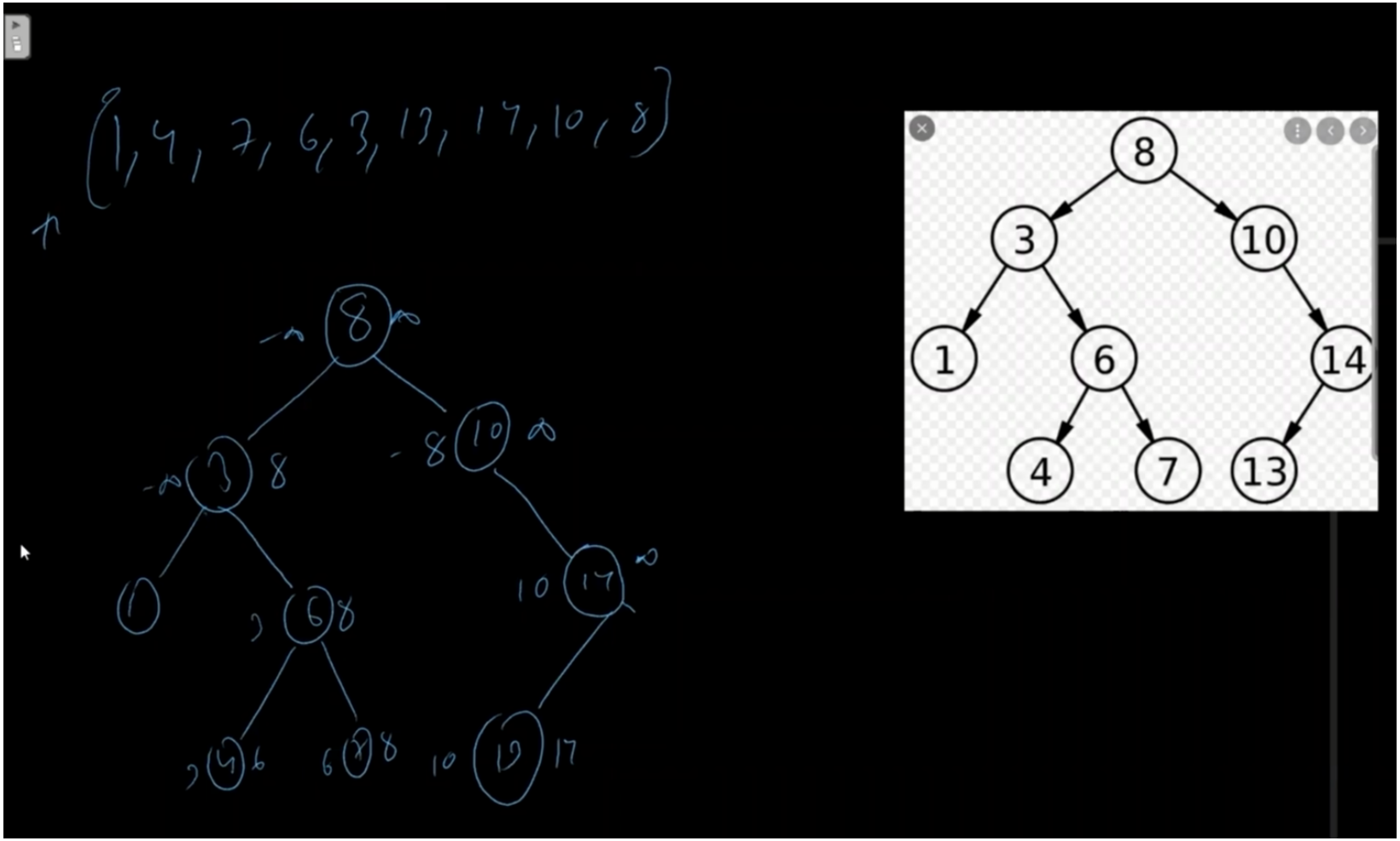
4. Construct Tree from Postorder



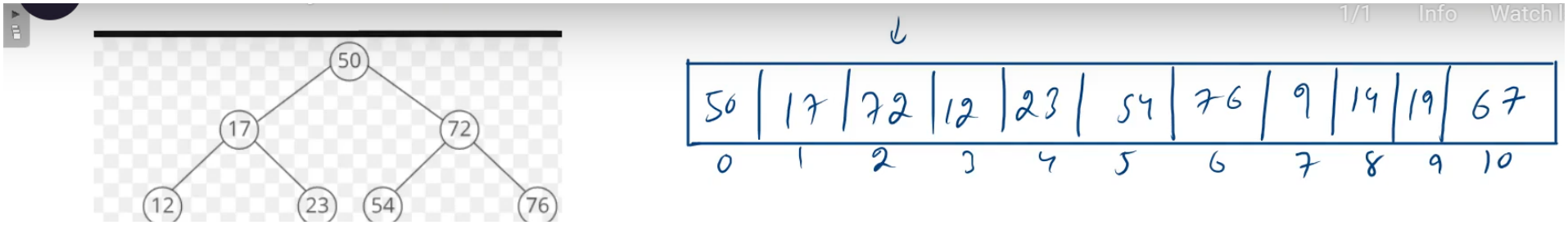

10
14
10
19
17

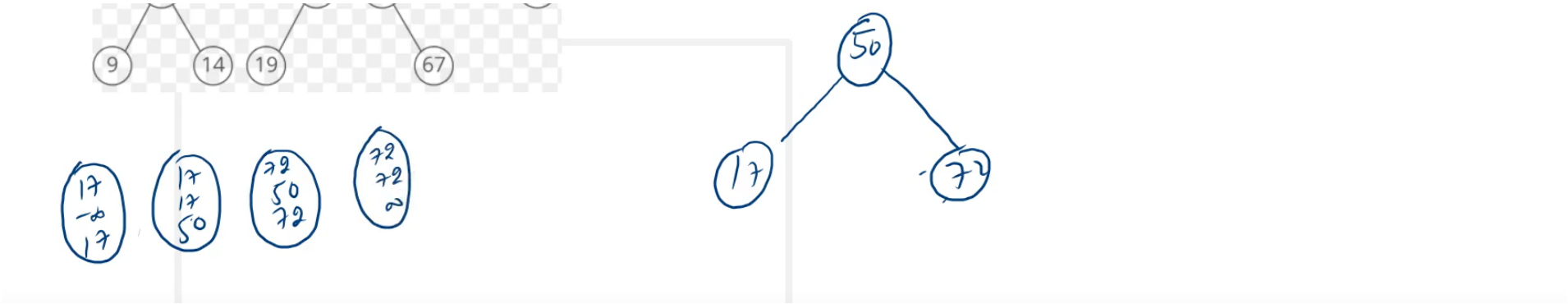

(1, 4, 7, 6, 3, 13, 14, 10, 8)
8
3
8
10
6
10
14
10
19
17

8
3
10
1
6
14
4
7
13

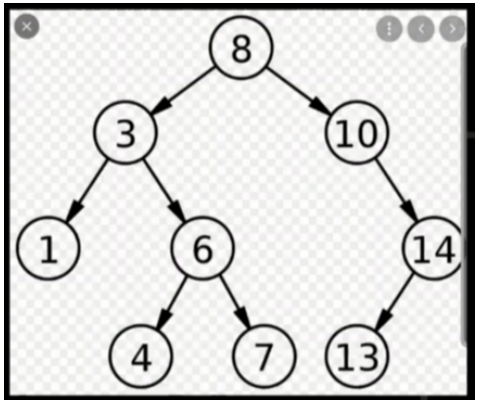


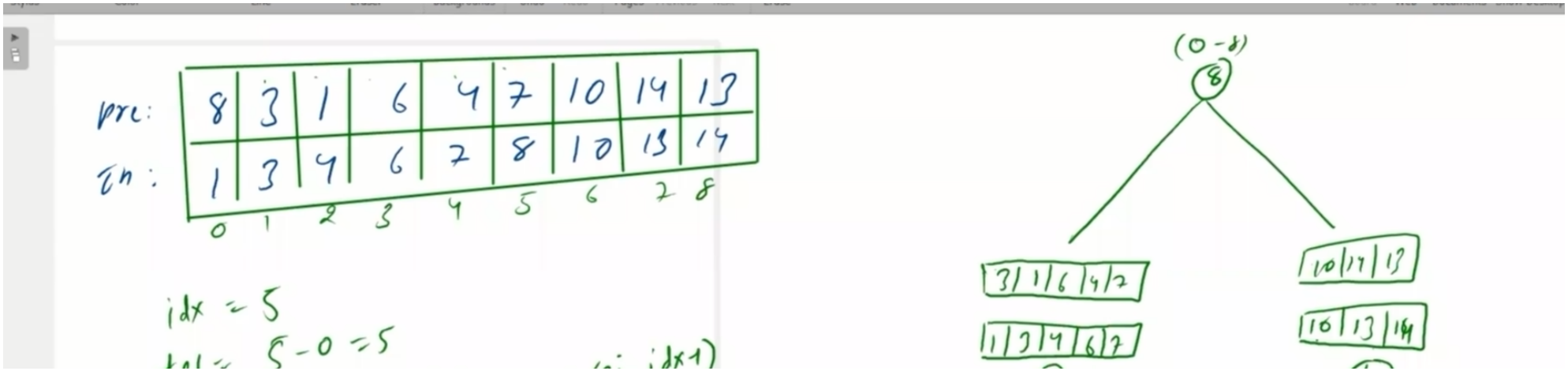
5. Construct BST from level order Traversal

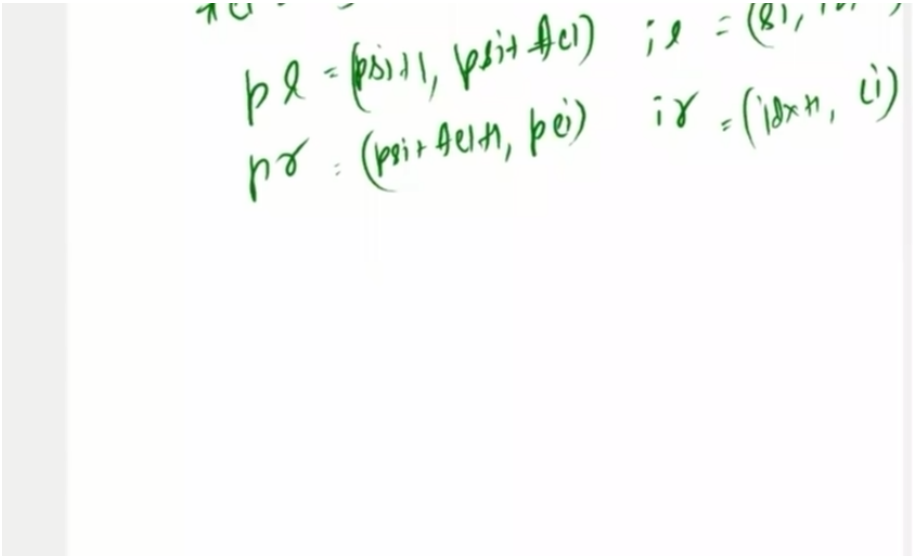


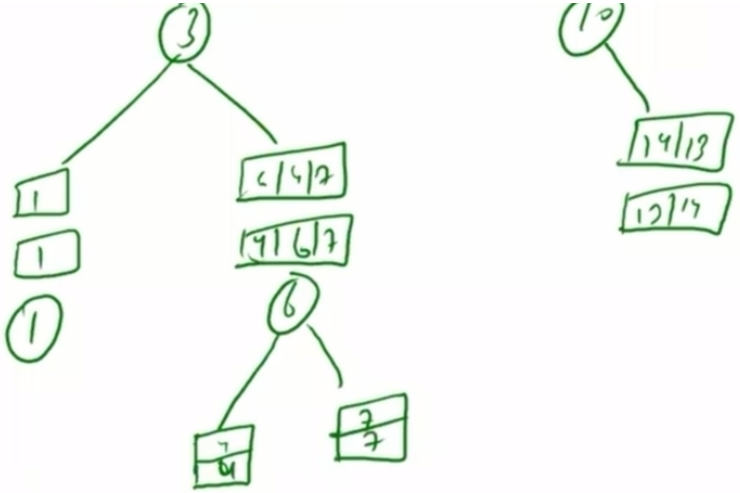


6. Binary Tree From Inorder and Preorder

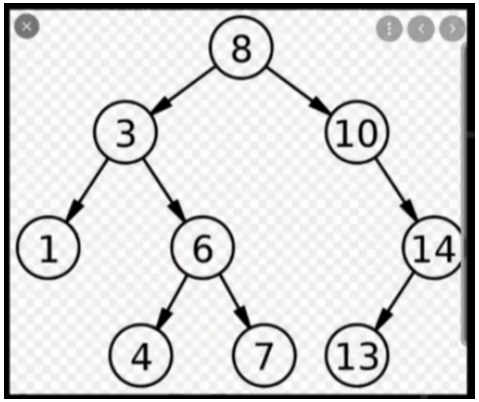


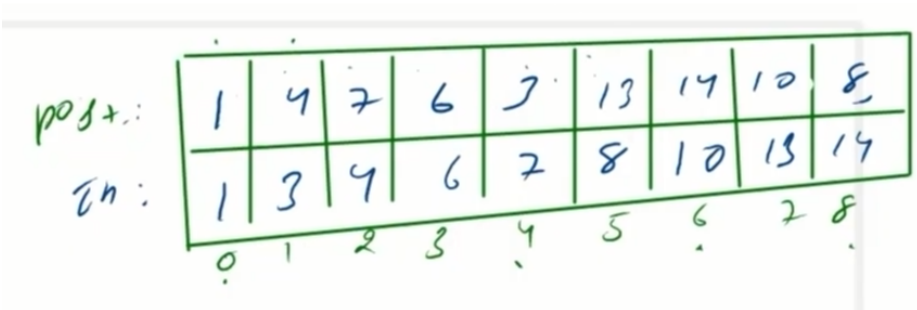






7. Binary Tree from Post and Inorder





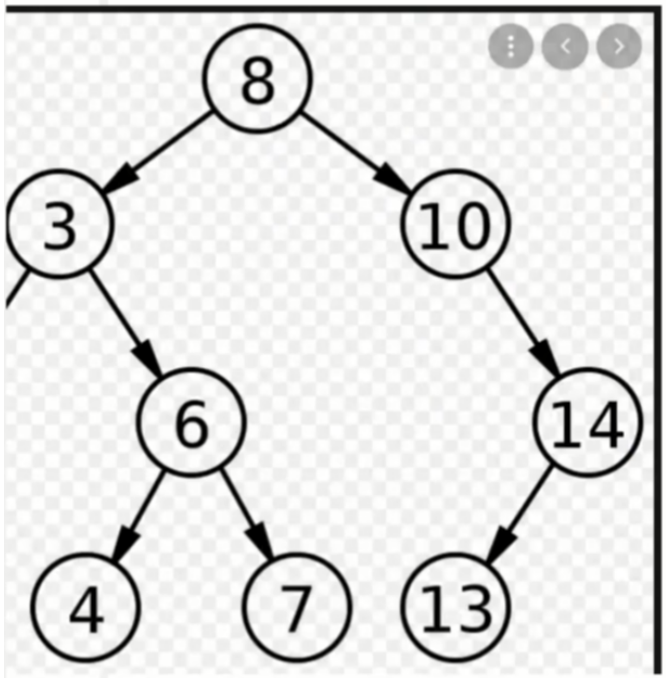
post:	1	4	7	6	3	13	14	10	8
In:	1	3	4	6	7	8	10	13	14
	0	1	2	3	4	5	6	7	8

idx = 5

tl = 5 - 0 = 5

pl = (ps, ps + tnl - 1) il = (si, idx - 1)

pr = (ps + tnl, pe - 1) ir = (idx + 1, ei)



post:	1	4	7	6	3	13	14	10	8
in:	1	3	4	6	7	8	10	13	14
	0	1	2	3	4	5	6	7	8

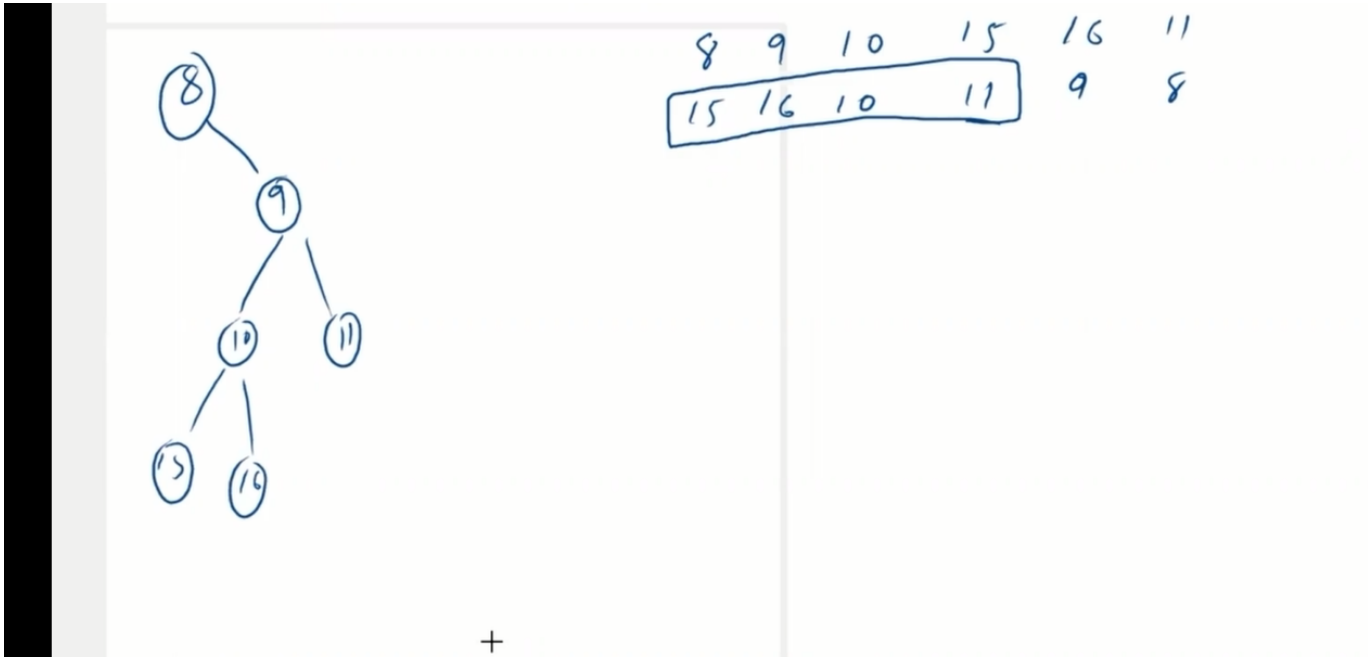
idx = 5

tl = 5 - 0 = 5

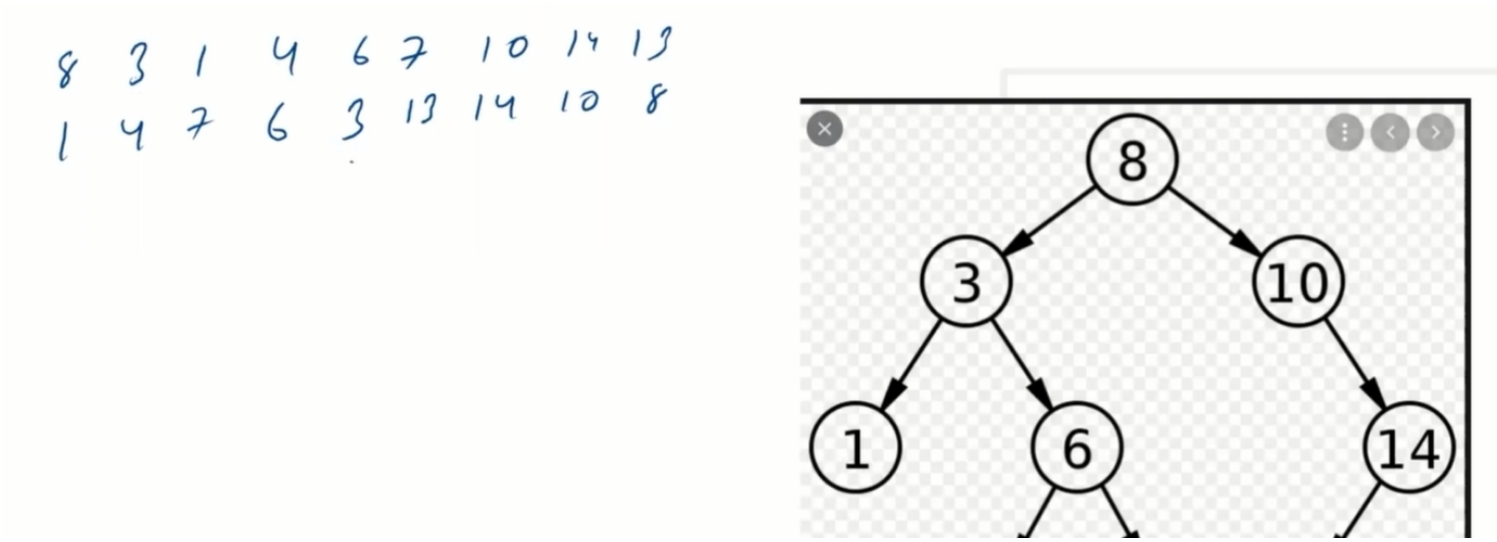
pl = (ps, ps + tnl - 1) il = (si, idx - 1)

pr = (ps + tnl, pe - 1) ir = (idx + 1, ei)

8. Binary Tree from Pre and Post



In this case where the inorder and the post order given are not of full binary tree, the tree formed can loose its shape, since in preorder we are not able to decide that 9 belongs to left subtree or right subtree. Also in postorder the elements left to 9 just show that they are below 9 , so no idea about 9 being the left or right. Therefore tree formed can not be of exact shape.

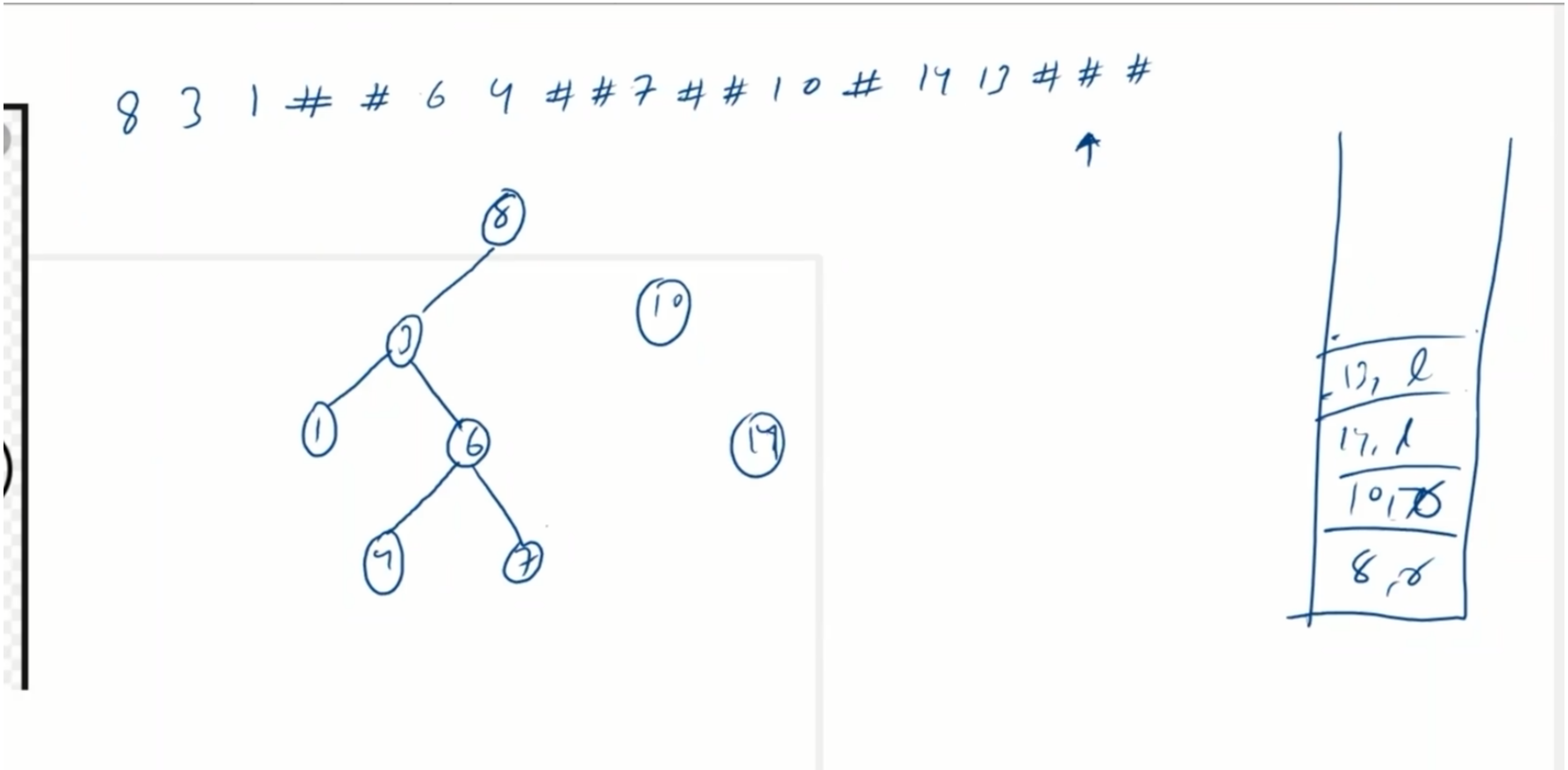


But in case of full binary tree, the tree formed is of exact shape.

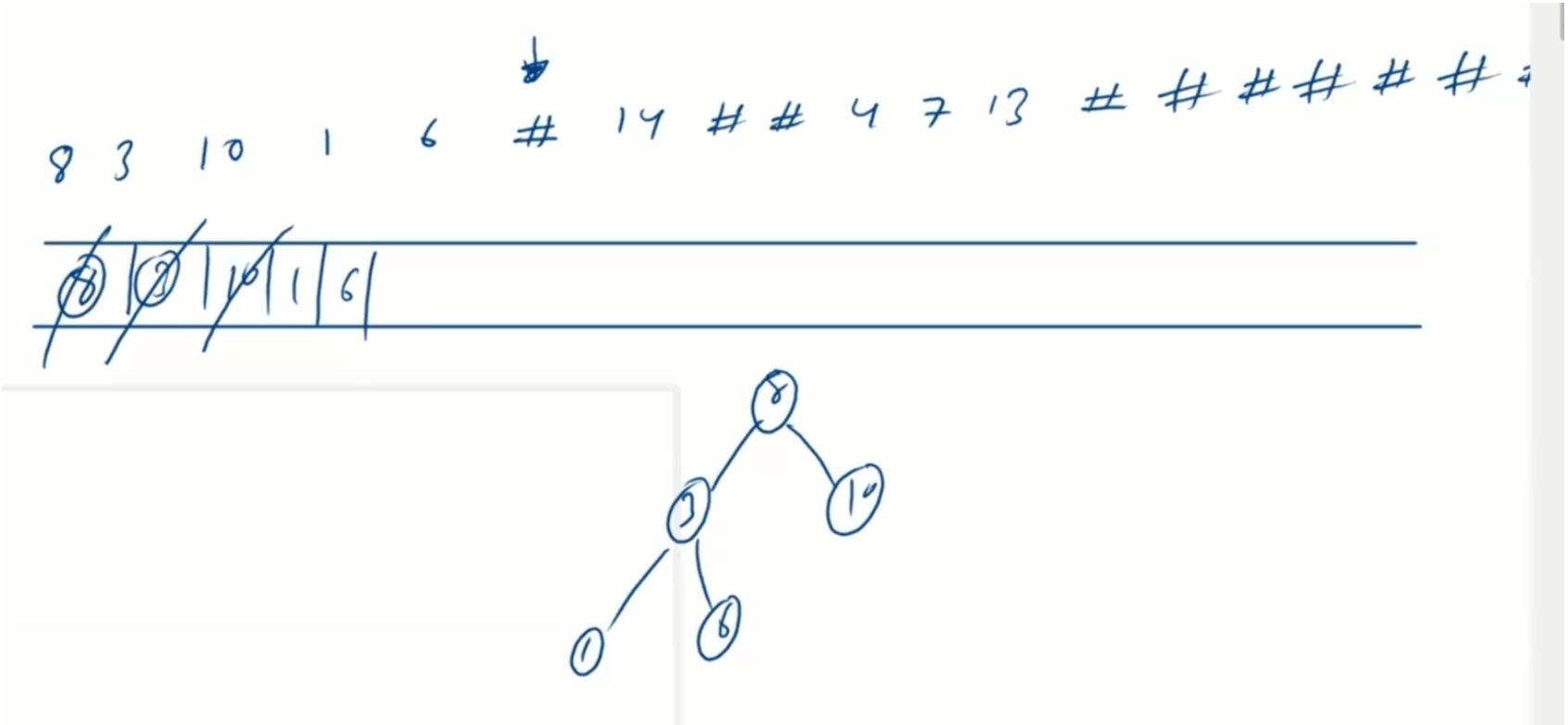
In both the cases tree will be formed.

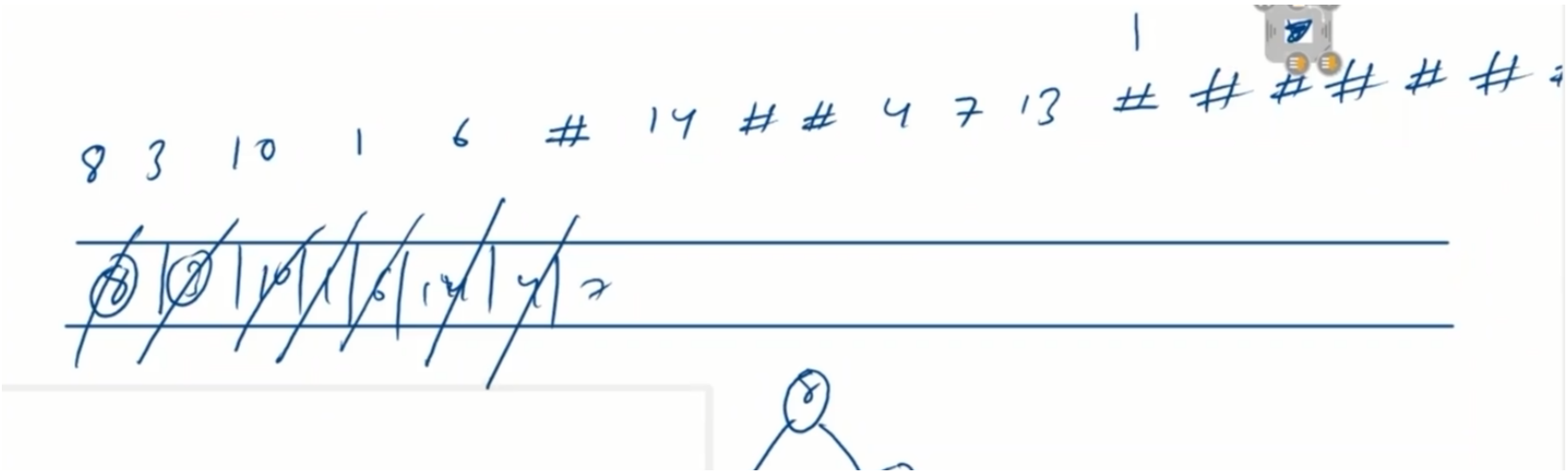
9. Serialize and Deserialize

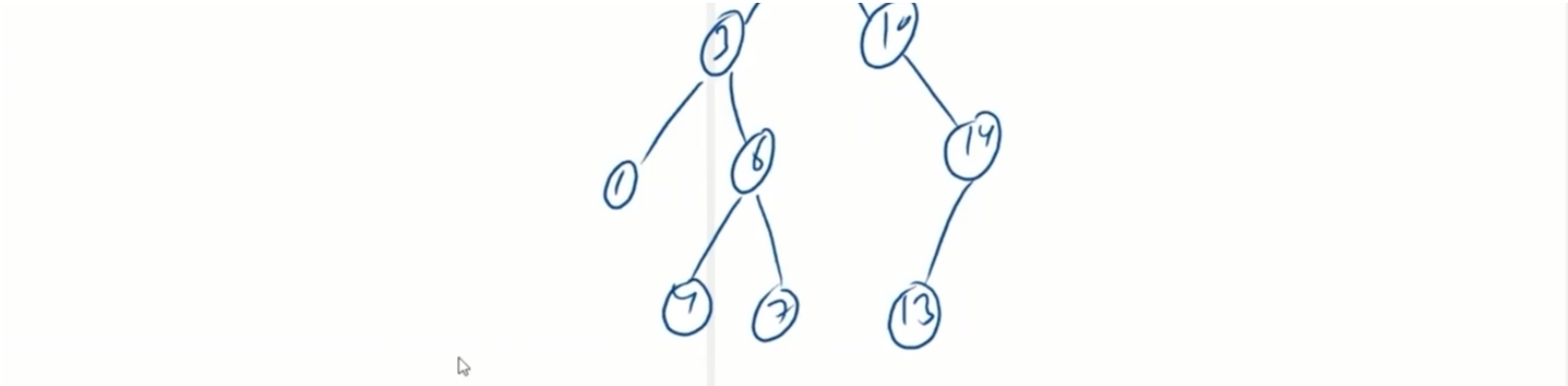
Method 1 : Pure Recursive



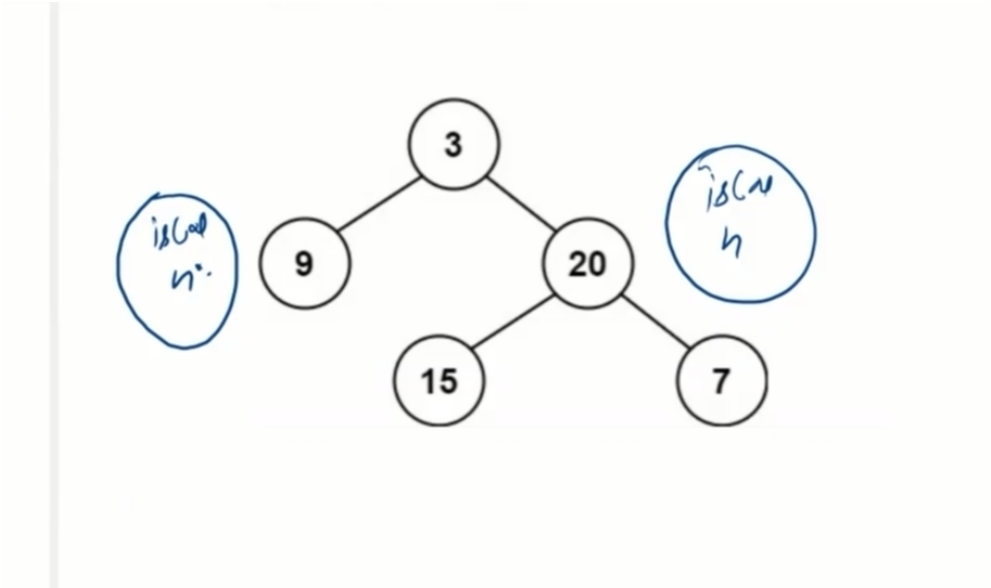
Method 2: Level Order Traversal







10. Balanced Binary Tree



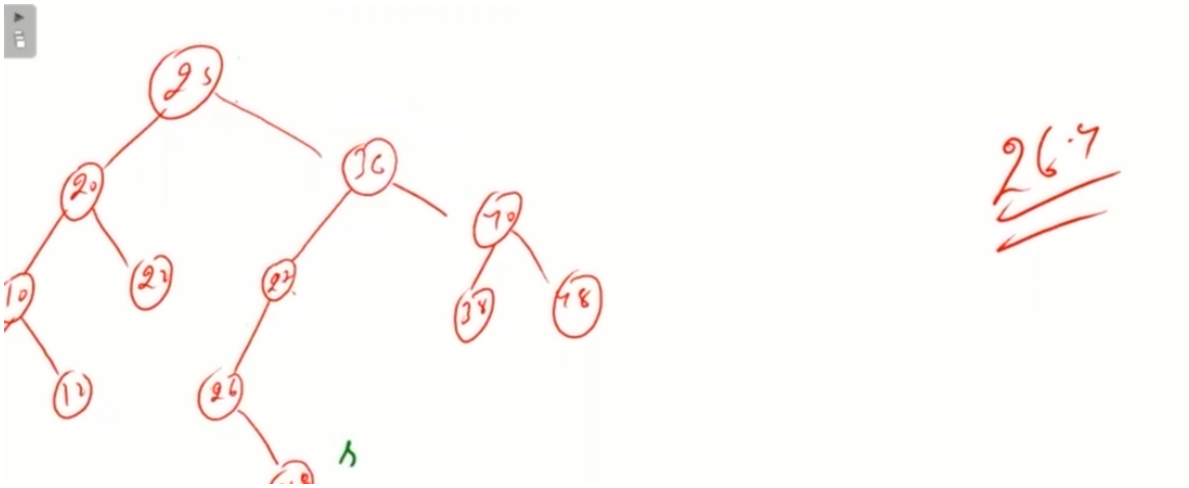
11. Largest BST Subtree

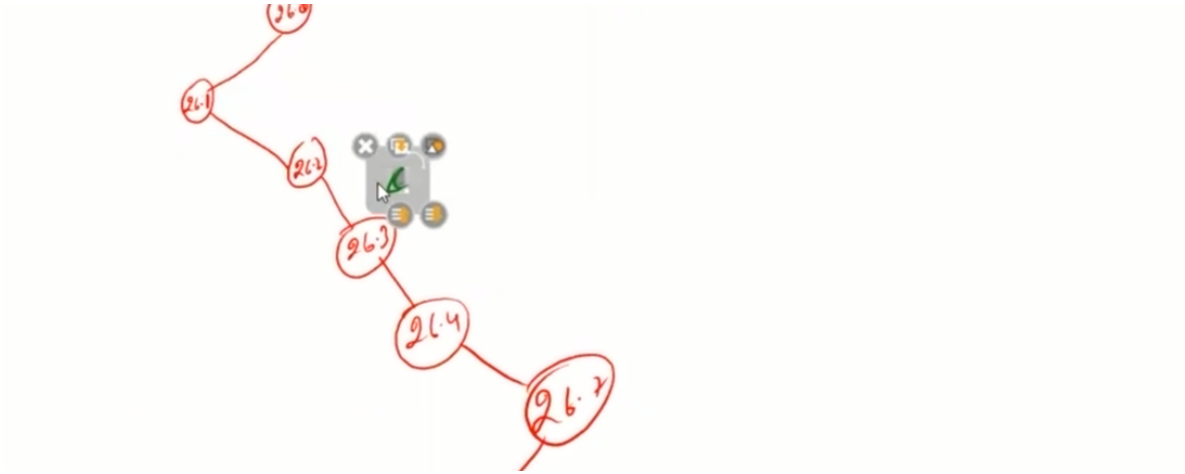
```
Input:
        1
      /   \
     4     4
   /   \
```

```
  6     8
Output: 1
Explanation: There's no sub-tree with size
greater than 1 which forms a BST. All the
leaf Nodes are the BSTs with size equal
to 1.
```

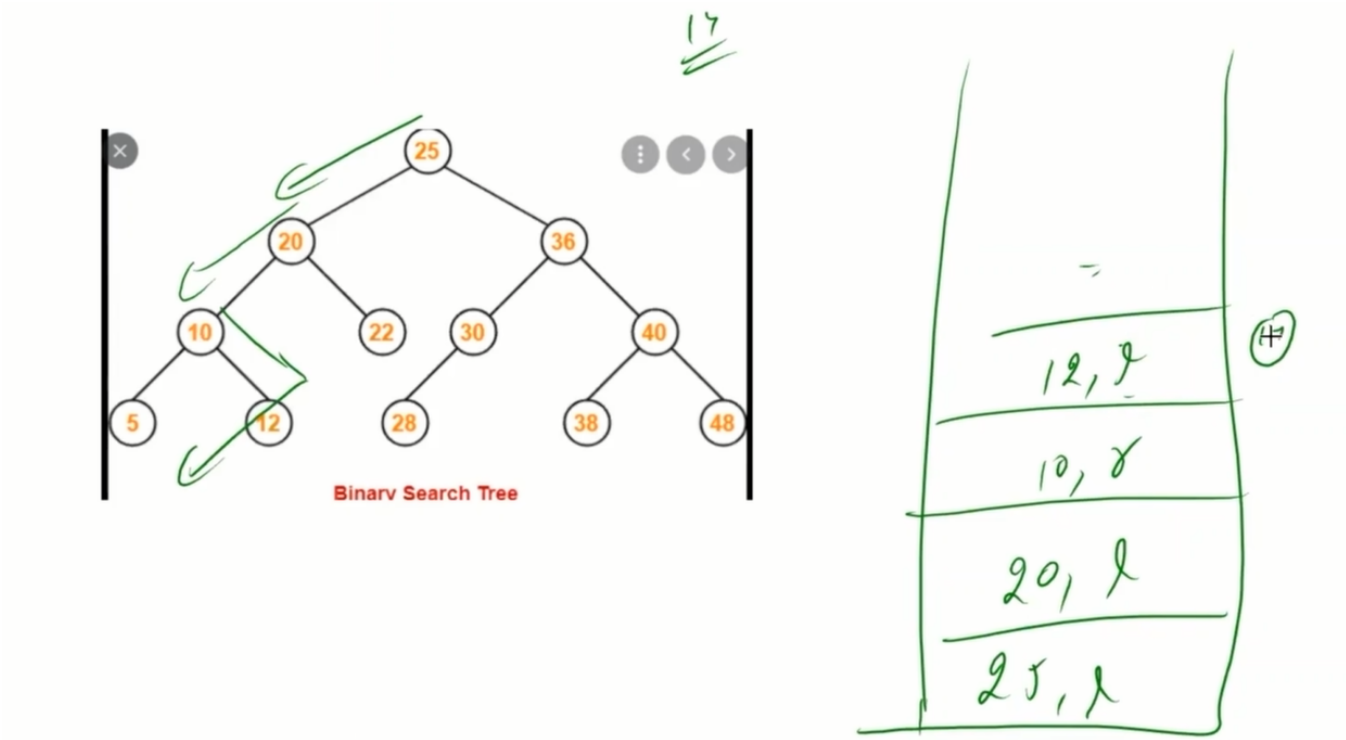
```
Input: 6 6 3 N 2 9 3 N 8 8 2
            6
        /       \
       6         3
        \      /   \
         2    9     3
          \  /  \
          8 8    2
Output: 2
Explanation: The following sub-tree is a
BST of size 2:
       2
    /    \
   N      8
```

12. Predecessor and Successor in BST





13. Insert a Node in BST



14. Inorder Successor in BST ||

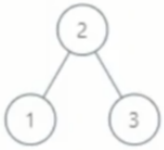
Given a `node` in a binary search tree, return *the in-order successor of that node in the BST*. If that node has no in-order successor, return `null`.

The successor of a `node` is the node with the smallest key greater than `node.val`.

You will have direct access to the node but not to the root of the tree. Each node will have a reference to its parent node. Below is the definition for `Node`:

```
class Node {
    public int val;
    public Node left;
    public Node right;
    public Node parent;
}
```

Example 1:




```
Input: tree = [2,1,3], node = 1
Output: 2
Explanation: 1's in-order successor node is 2. Note that both the node and the return value is of Node type.
```

```java
class Solution {
    public Node inorderSuccessor(Node node) {

        Node succ = null;
        Node right = node.right;
        if(right != null){
           while(right.left != null){
             right = right.left;
           }

            return right;
        }

         while(node.parent != null && node.parent.left != node){
             node = node.parent;
         }

         return node.parent;
    }
}
```

15. Left, Right view of Tree

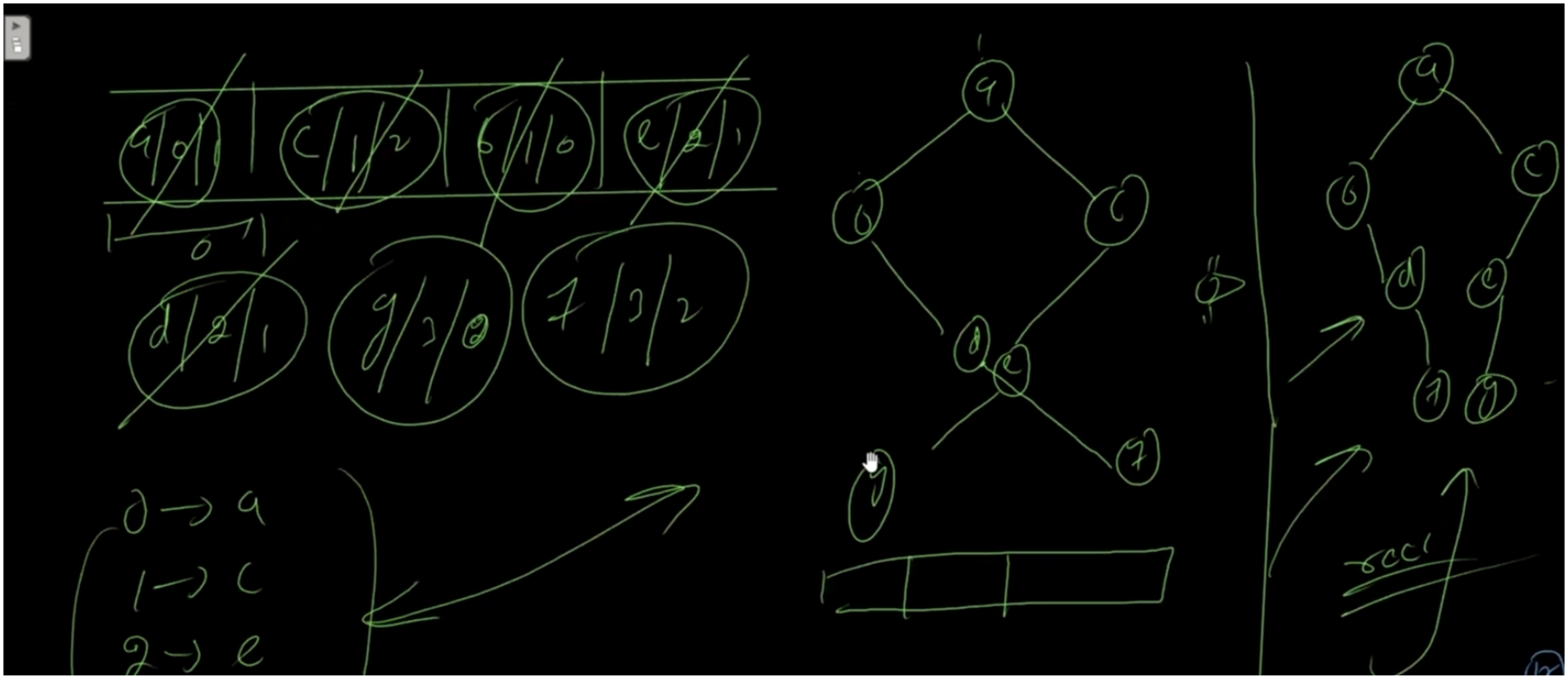
When we normally solve the left or right view , we do not consider that *Left child will be at an angle of -45° and right child will be at an angle of +45°.*

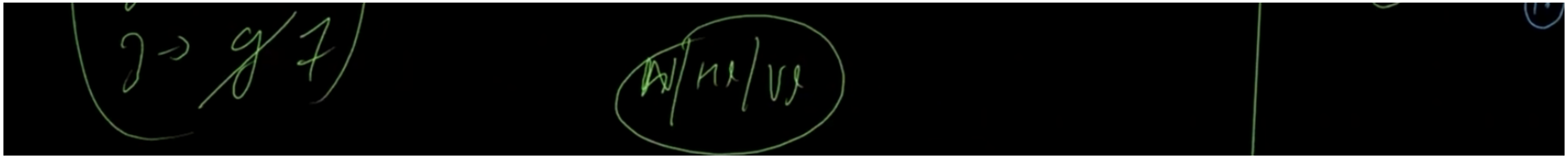
Therefore the right side Tree is considered which has the right view with a ,c, e,g

But if told that 45° angle is to be taken, the right view will be changed to a,c,e,f

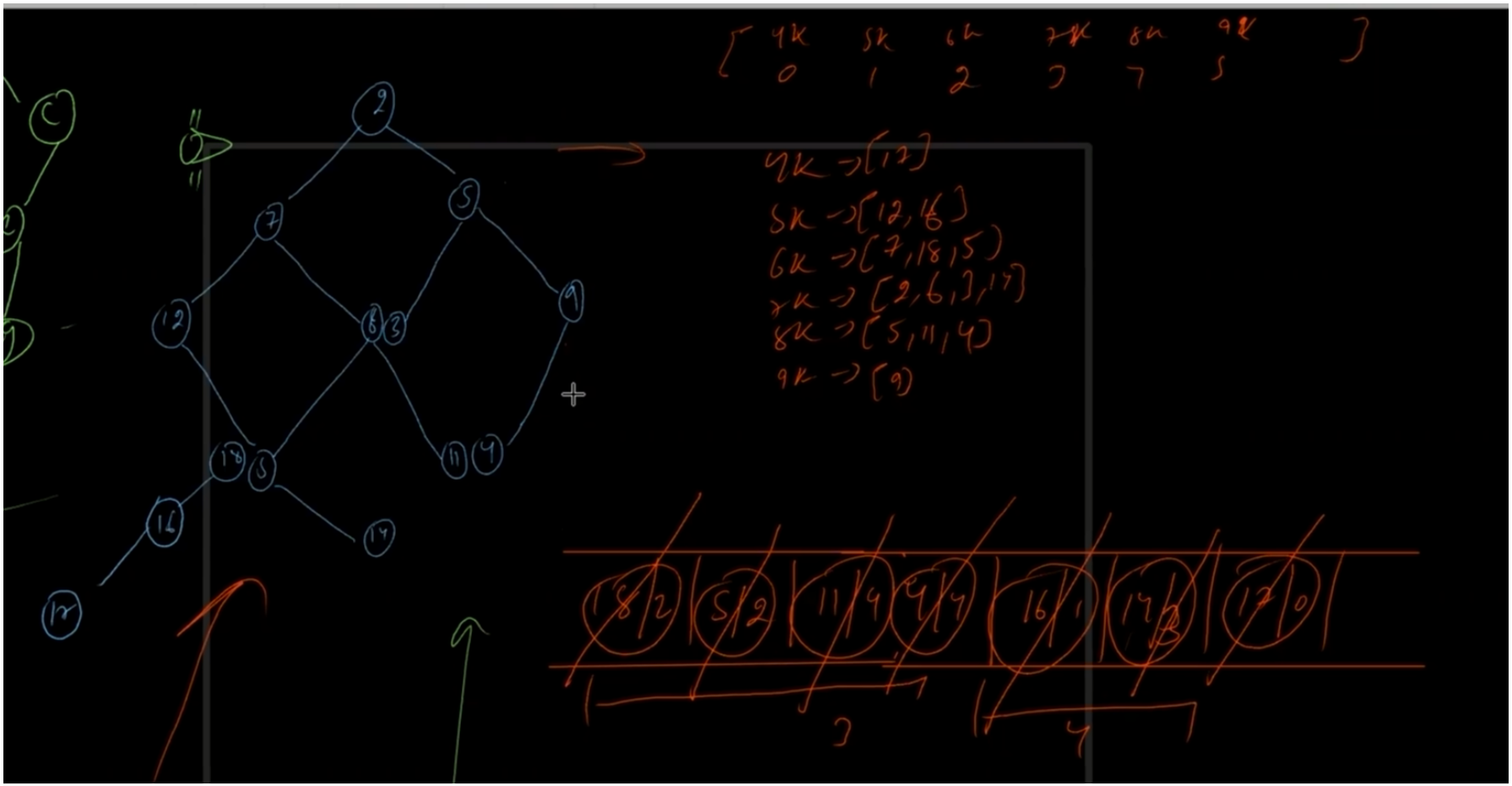
So for this, we have to keep the horizontal level as well as the vertical level in the class pair and then print the one with highest vertical level.

Also we can use priority queue for this.



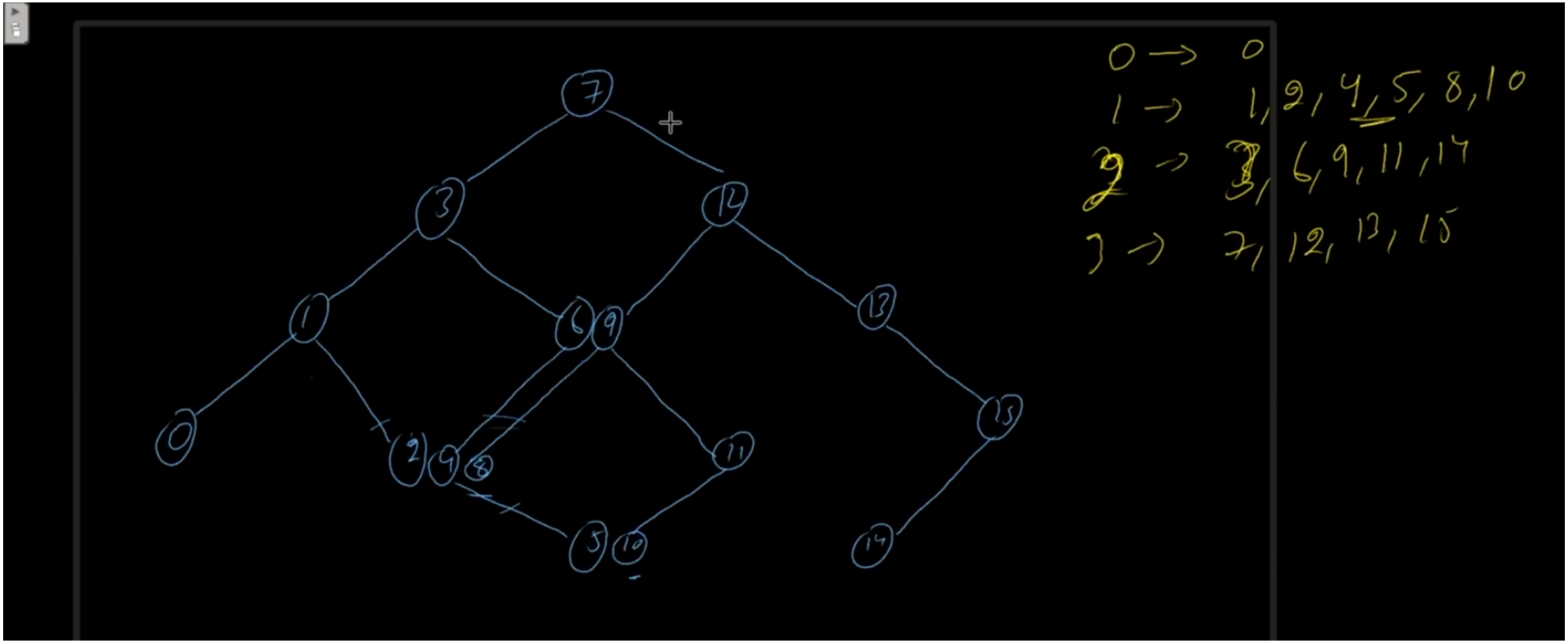


16. Vertical Order of the binary tree



17. Top view, Bottom View Based on same concept

18. Diagonal Order Traversal



19. Vertical order ii time complexity

$$T(N) = 1.2^1 + 2.2^2 + 3.2^3 + 4.2^4 + 5.2^5 + \cdots (n-1)2^{n-1} + n.2^n$$

$$2T(N) = 1.2^2 + 2.2^3 + 3.2^4 + 4.2^5 + 5.2^6 + \cdots (n-1)2^n + n.2^{n+1}$$

eq (i) − eq (ii)

$$T(N) - 2T(N) = 1.2^1 + 1.2^2 + 1.2^3 + 1.2^4 + 1.2^5 + \cdots 1.2^n - n.2^{n+1}$$

$$-T(N) =$$

Solving T(n) gets us the solution. Nlogn

$$eq\ (i) - eq\ (ii)$$

$$T(N) - 2T(N) = 1.2^1 + 1.2^2 + 1.2^3 + 1.2^4 + 1.2^5 + \ldots\ldots 1.2^n - n.2^{n+1}$$

$$-T(N) = \frac{2^1.(2^n-1)}{(2-1)} - n.2^{n+1}$$

$$T(N) = -\frac{2.(2^n-1)}{2}$$