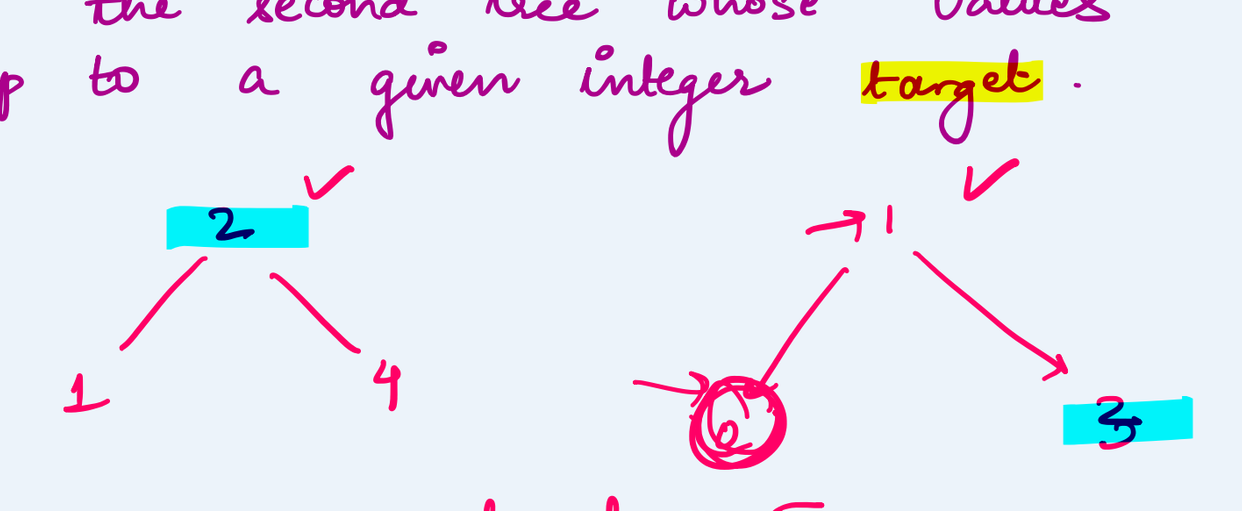
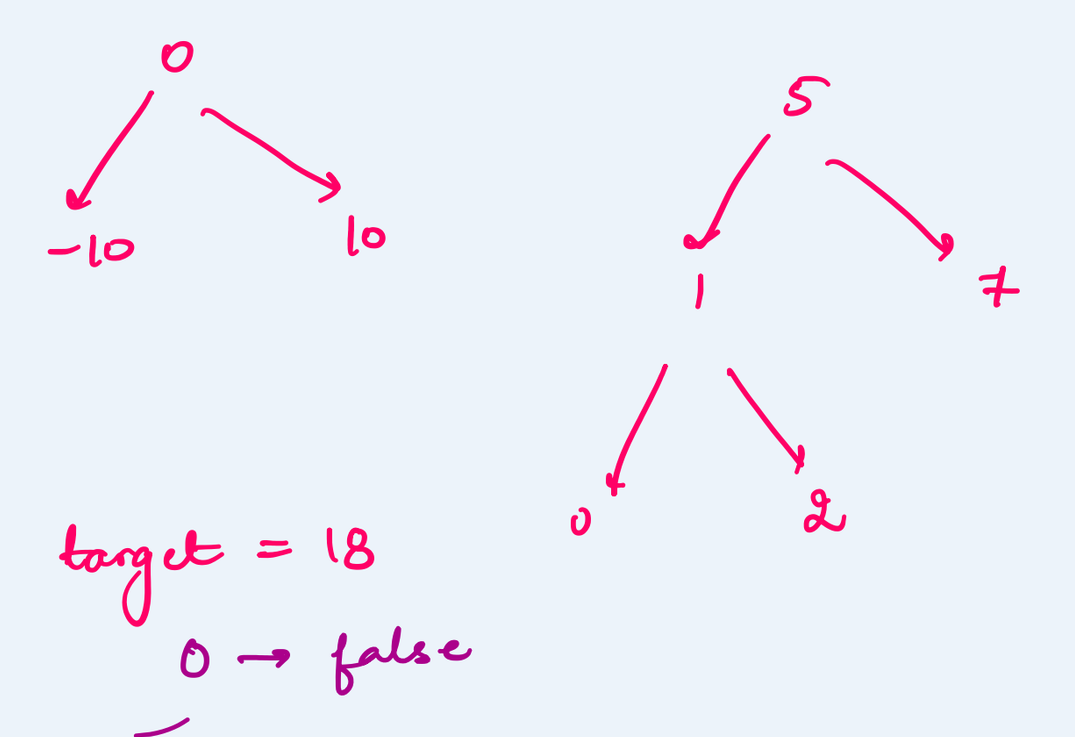
Saturday, 17 April 2021 9:01 PM

Q Given **two binary search trees**, return true iff there is a node in the first tree and a node in the second tree whose values sum up to a given integer **target**.

Eg:

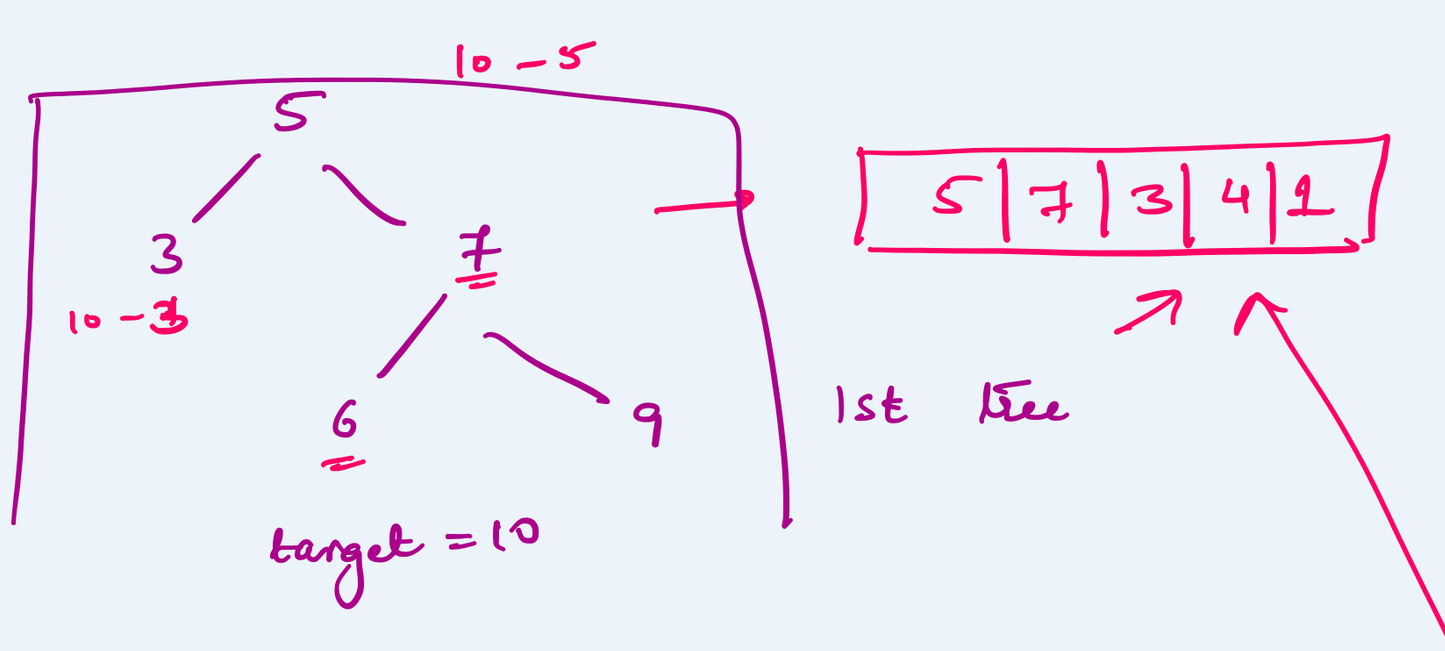


Eg:



Approach ①: Traverse the first tree and store (target − val) of all nodes in a hashset.

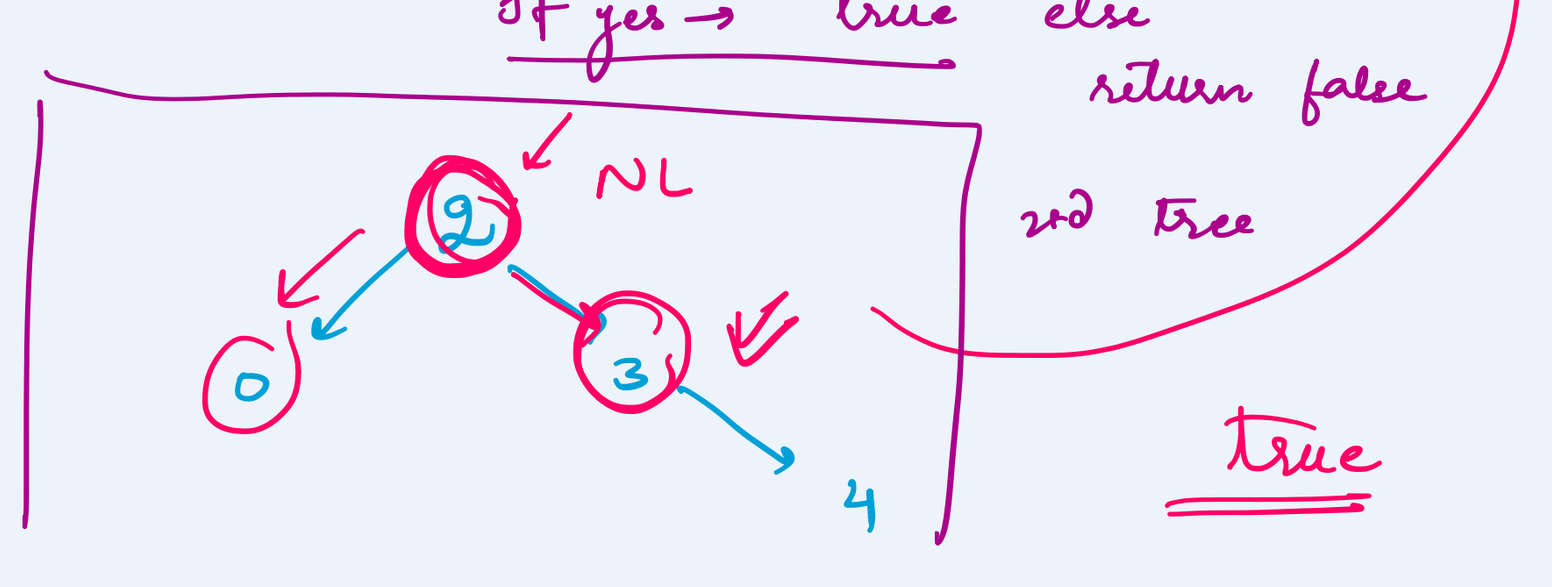
1.



2. Traverse the second tree and check if any of its elements exist in the hashset.

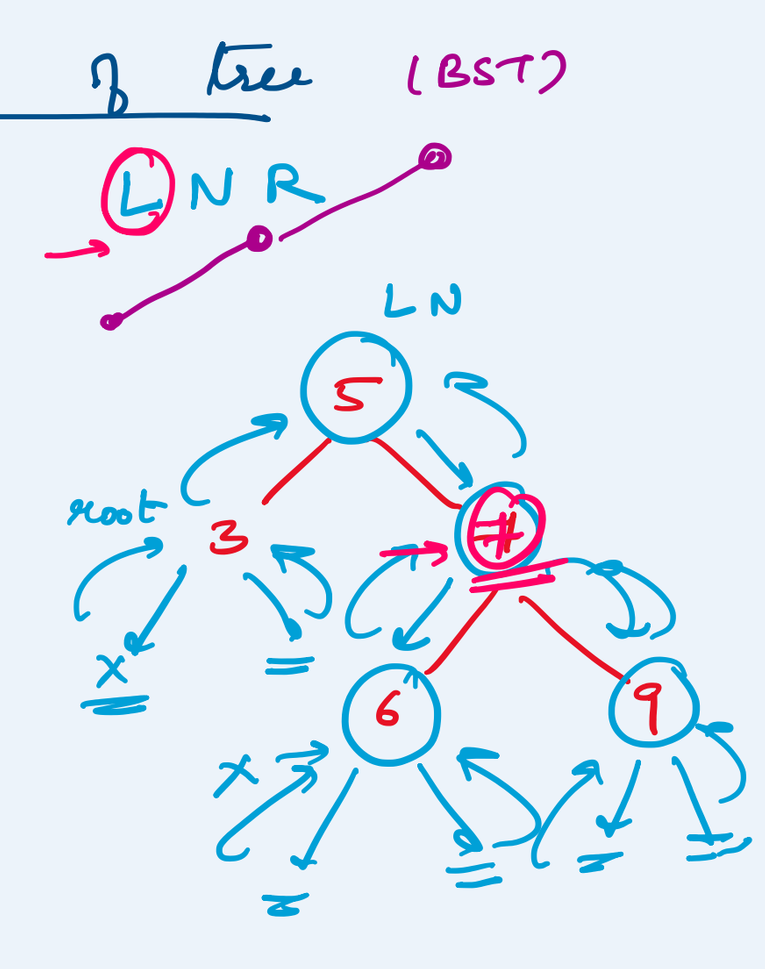
If yes → true else return false

$T(N_1 + N_2)$



→ Recursive approach

Iterative traversal of tree (BST) (Inorder)

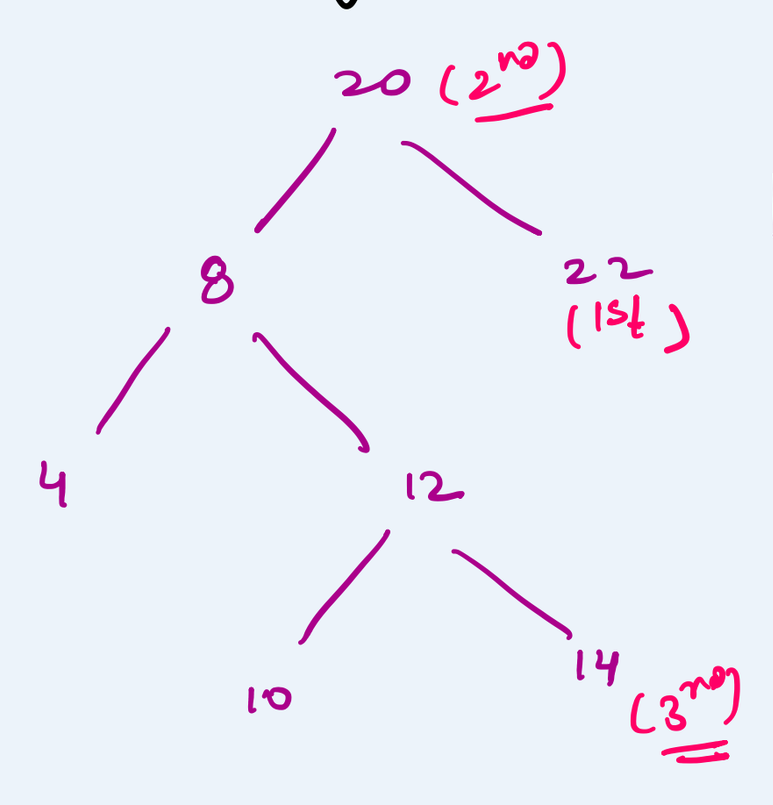


stack

3 5 6 7 9

```
boolean targetSum ( root 1, root 2, target) {
        Stack<I> stack = new stack();
        Set < Integer> s = new Set ();
        while ( ! stack .isEmpty () || root1 != null) {
    left    while (root1 != null) {
                stack.push (root1);
                root1 = root.left;
            }
    node    root1 = stack.pop();
            s.add ( target - root1.val);
            root 1 = root1.right;
        }
}

If ( s.contains( root2.val) {
        return true;
}
```

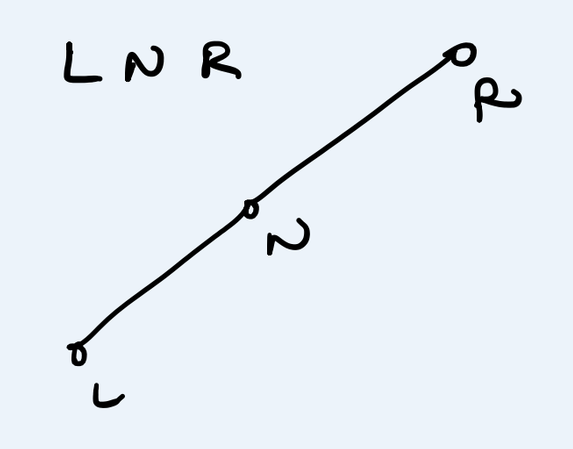
Q Given a BST, and a positive integer k, find the k^{th} largest element in the BST.

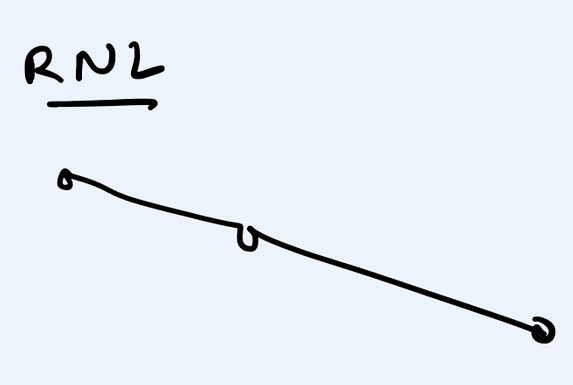


k = 3
ans → 14

k = 5
ans = 10

Inorder traversal

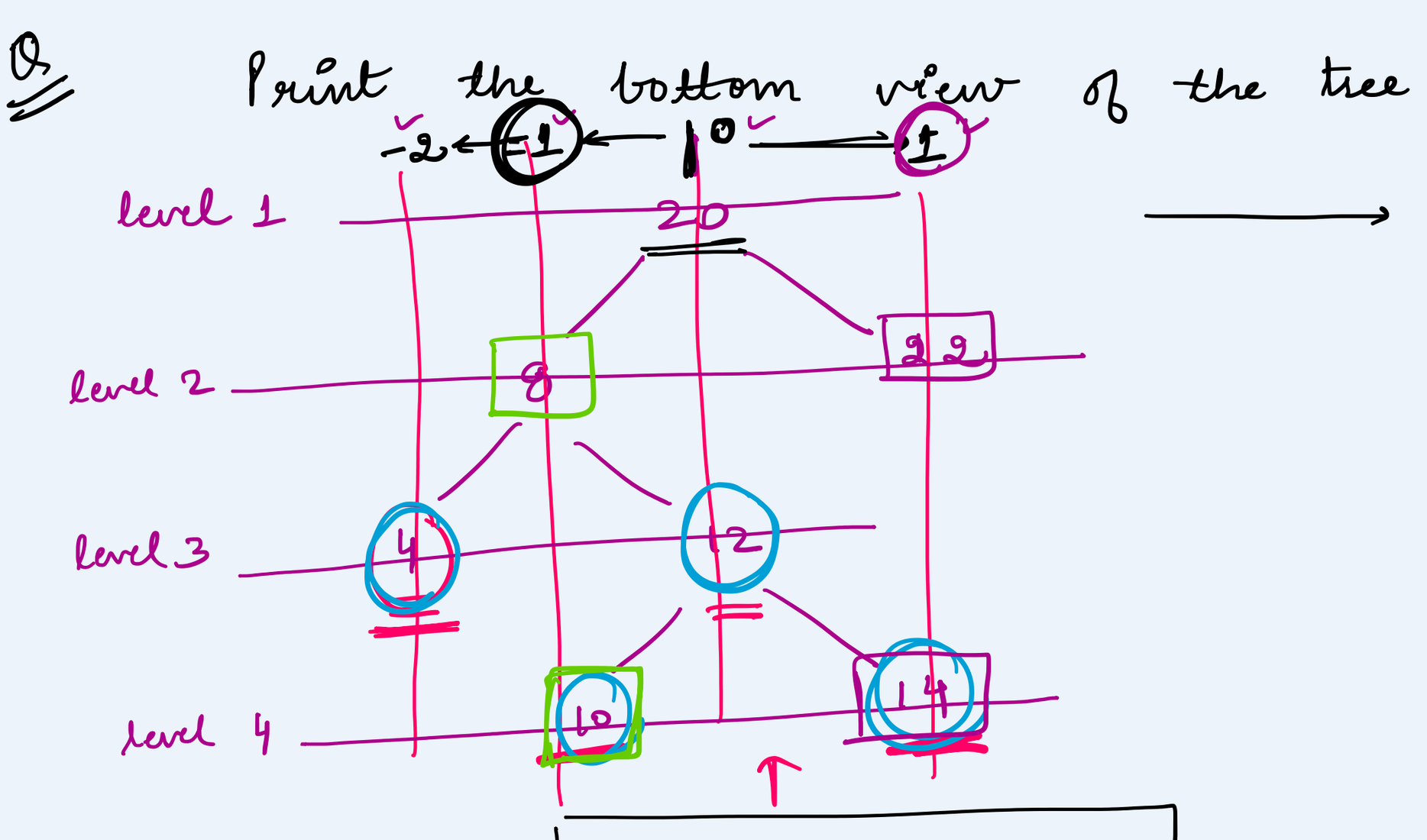





```
kth largest ( k) {
        reverse Inorder ( root, k, 0);
}

reverse Inorder ( root, k, c) {
        If ( root == null || c >= k) {
                return;
        }
R       reverse Inorder ( root.right, k, c);
N       c++;
        If ( c == k) {
            syso ( root.data);
            return;
        }
L       reverse Inorder ( root.left, k, c);
}
```

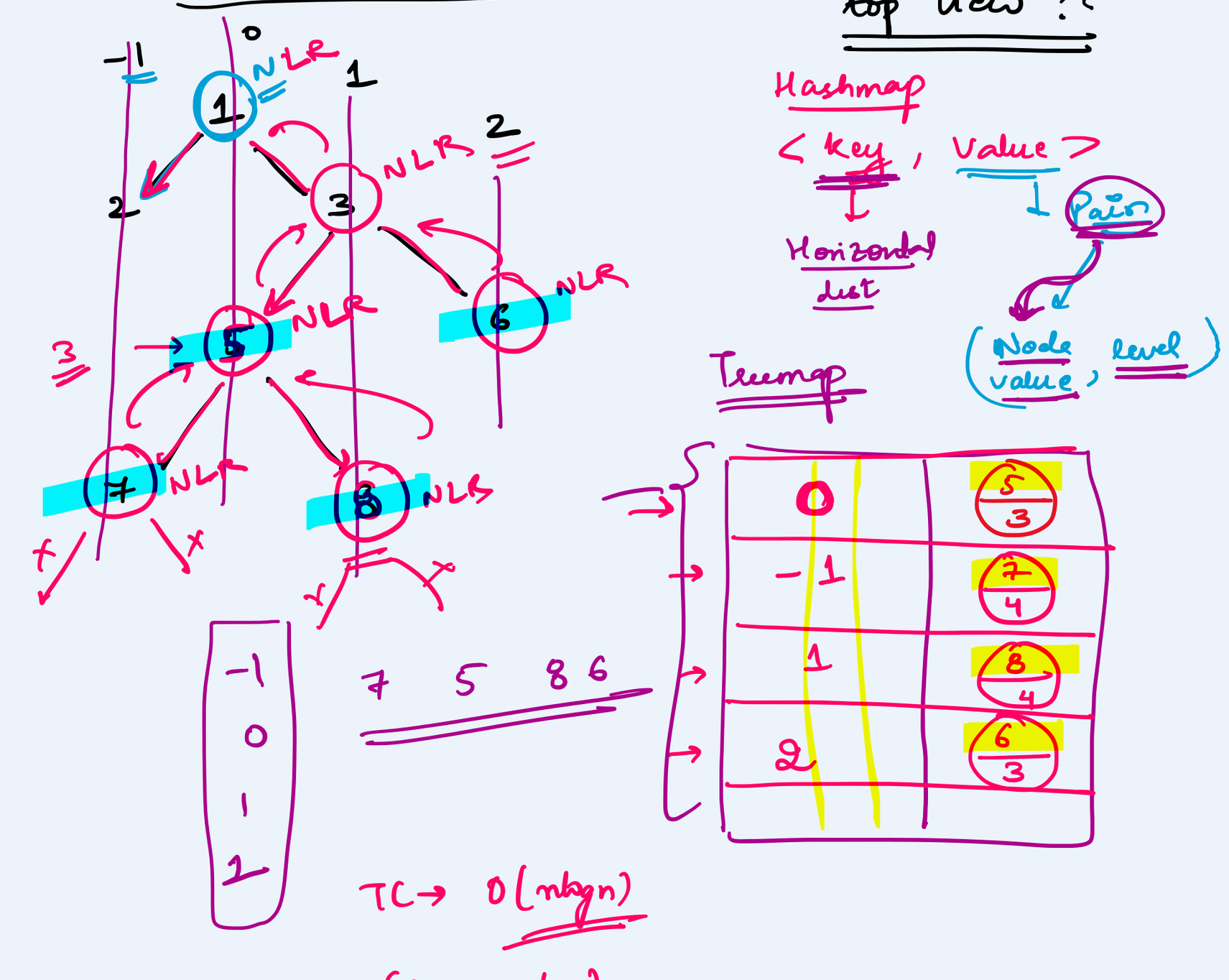
$TC \rightarrow O(n)$

Q Print the bottom view of the tree



ans → 4, 10, 12, 14

top view ??



$TC \rightarrow O(n \log n)$

$SC \rightarrow O(n)$

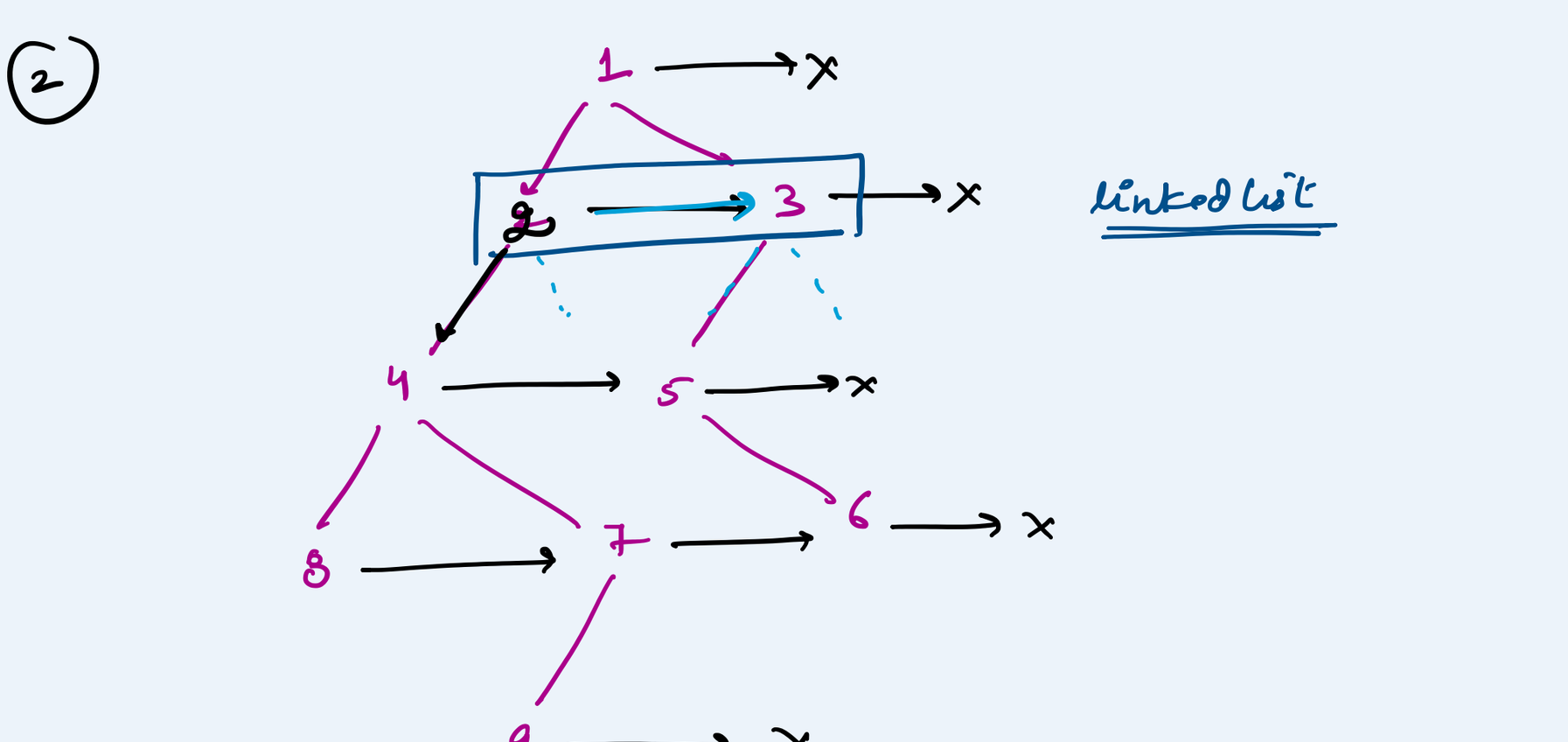
```
printBottom ( root) {
    → Map < Integer, Pair < Integer, Integer> map = new Treemap();
      print ( root, 0, 0, map);
      for (Pair <Integer, Integer> it : map.values()) {
            cyso ( it.value);
      }
}

print ( root, dist, level, map) {
        If (root == null)   return;
node    If ( ! map.containsKey (dist) || level >= map.get (dist) .level)
                map.put ( dist, new Pair ( root.key, level));
        }
        print ( root.left, dist - 1, level + 1, map);
        print ( root.right, dist + 1, level + 1 , map);
}
```

Q Given a binary tree

```
class Node {
        int val;
        Node left;
        Node right;
        Node next;
}
```

populate the next pointer to point to its next right node.

①



②