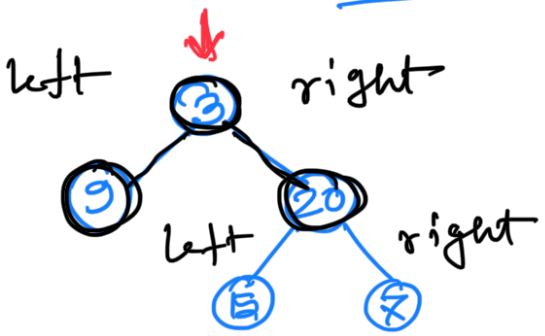
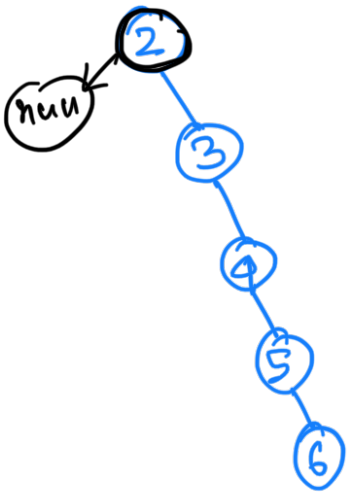
Minimum Depth Binary Tree



⇒ 3 (1)
→ 9 (1)
→ null = 0 } min = 0+1 = 1
→ null = 0 }
→ 20 (2)
→ 15 (1)
→ 7 (1)

Test Case 2



Recursion →

```
int minDepth(TreeNode root) {
    if(root == null)
        return 0;
    if(root.left == null &&
       root.right == null)
        return 1;
    if (root.left != null & root.right
                == null)
        return minDepth(root.left)+1;
    if (root.right != null &&
        root.left == null)
        return minDepth(root.right)
                +1;
    return Math.min(
        minDepth(root.left),
        minDepth(root.right)) +1;
}
```

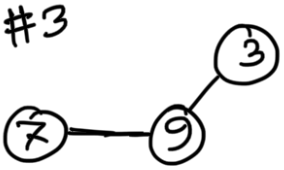
#1

root == null
return 0;

#2

root.left = null
&& root.right = null

#3



root.left != null &&
root.right == null

return 1;

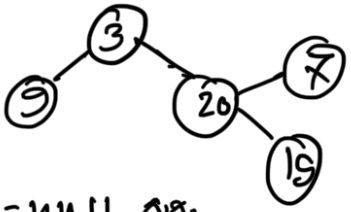
ret mindepth(root.left) +1;

#4

if(root.left == null &&
root.right != null)

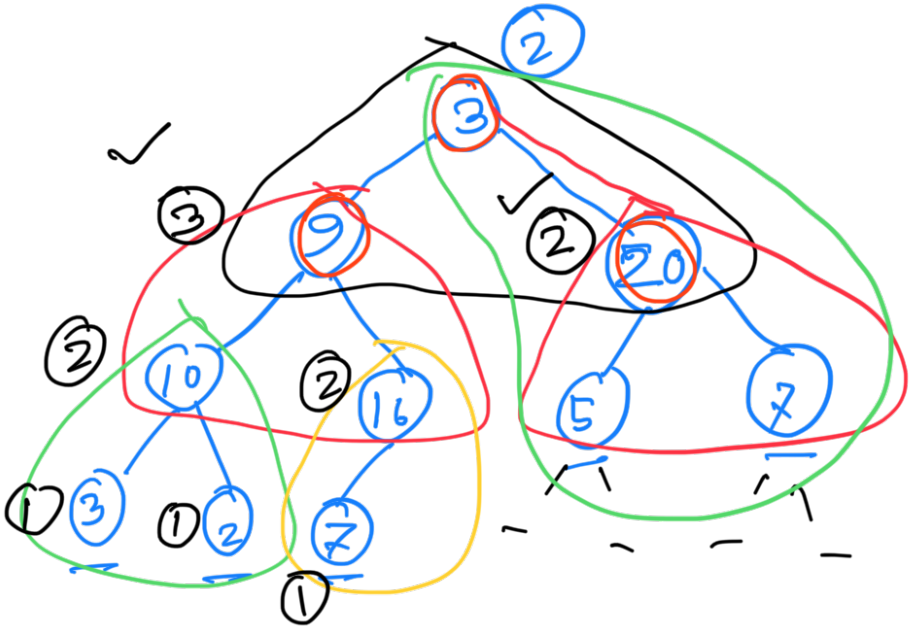
ret mindepth(root.right) +1;

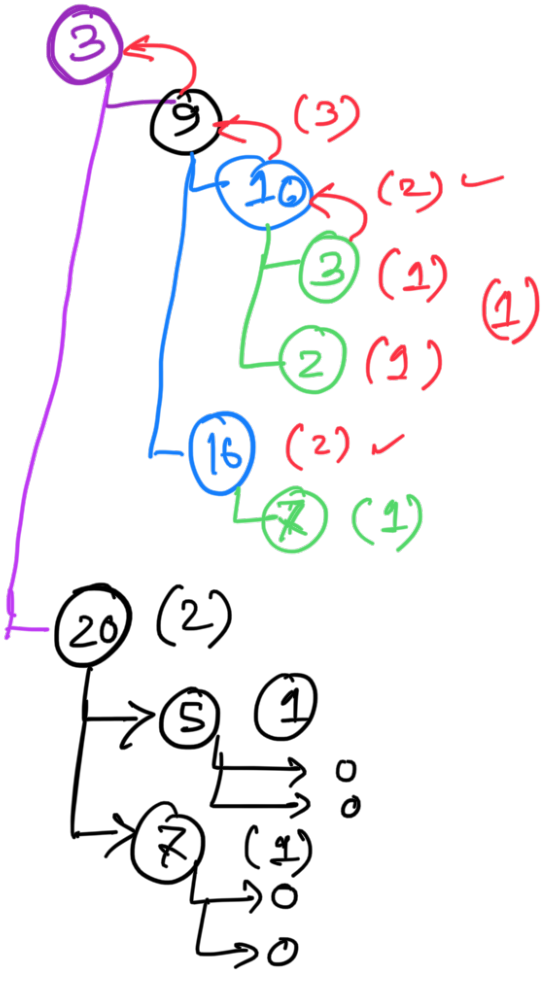
#5



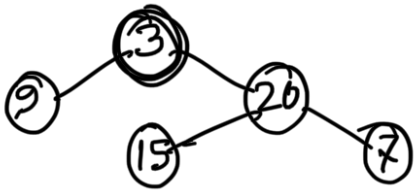
root.left != null &&
root.right != null

ret Math.min(
mindepth(root.left),
mindepth(root.right)) +1





Using Stack



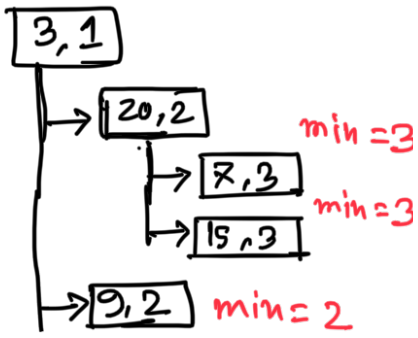
Pair {
  TreeNode node;
  int depth;
}

Min = 3
Min = 3
Min = 2

Stack

4	~~7~~, 3
3	~~15~~, 3
2	(~~20~~, 2)
5	(9, 2)
1	(3, 1)

Min = ∞




```
public int minDepth(TreeNode root) {
    if (root == null)
        return 0;
    int min = Integer.MAX_VALUE;
    Deque<Pair> stack = new ArrayDeque<>();
    stack.push(new Pair(root, 1));
    while (!stack.isEmpty()) {
        Pair pair = stack.pop();
        TreeNode node = pair.node;
        int depth = pair.depth;
        if(node.left == null && node.right == null)
            if(min > depth)
                min = depth;
        if(node.left != null)
            stack.push(new Pair(node.left, depth+1));
        if(node.right != null)
            stack.push(new Pair(node.right, depth+1));
    }
    return min;
}
```

when you find the leaf node