```java
public List<Integer> preorderTraversal(TreeNode root)
{//記憶: 除了 add 外, 總共四行(即那四行藍的)
   List<Integer> result = new ArrayList<>();
   Deque<TreeNode> stack = new ArrayDeque<>();
   TreeNode p = root;
   while(!stack.isEmpty() || p != null) {
      if(p != null) {
         stack.push(p);
         result.add(p.val);  //加到前兩行中間
         p = p.left;
      } else {
         TreeNode node = stack.pop();
         p = node.right;
      }
   }
   return result;
}
```

```java
public ArrayList<Integer> preorderTraversal(TreeNode
root) {
   ArrayList<Integer> res = new ArrayList<Integer>();
   helper(root, res);
   return res;
}

private void helper(TreeNode root, ArrayList<Integer>
res) {
   if(root == null)
      return;
   res.add(root.val);
   helper(root.left,res);
   helper(root.right,res);
}
```

```java
public List<Integer> inorderTraversal(TreeNode root) {
   List<Integer> result = new ArrayList<>();
   Deque<TreeNode> stack = new ArrayDeque<>();
   TreeNode p = root;
   while(!stack.isEmpty() || p != null) {
      if(p != null) {
         stack.push(p);
         p = p.left;
      } else {
         TreeNode node = stack.pop();
         result.add(node.val); //加到後兩行中間
         p = node.right;
      }
   }
   return result;
}
```

```java
public ArrayList<Integer> inorderTraversal(TreeNode
root) {
   ArrayList<Integer> res = new ArrayList<Integer>();
   helper(root, res);
   return res;
}

private void helper(TreeNode root, ArrayList<Integer>
res) {
   if(root == null)
      return;
   helper(root.left,res);
   res.add(root.val);
   helper(root.right,res);
}
```

```java
public List<Integer> postorderTraversal(TreeNode root)
{
   LinkedList<Integer> result = new LinkedList<>();
   Deque<TreeNode> stack = new ArrayDeque<>();
   TreeNode p = root;
   while(!stack.isEmpty() || p != null) {
      if(p != null) {
         stack.push(p);
         result.addFirst(p.val);  // Reverse preorder
         p = p.right;             // Reverse preorder
      } else {
         TreeNode node = stack.pop();
         p = node.left;           // Reverse preorder
      }
   }
   return result;
}
```

```java
public ArrayList<Integer> postorderTraversal(TreeNode
root) {
   ArrayList<Integer> res = new ArrayList<Integer>();
   helper(root, res);
   return res;
}

private void helper(TreeNode root, ArrayList<Integer>
res) {
   if(root == null)
      return;
   helper(root.left,res);
   helper(root.right,res);
   res.add(root.val);
}
```

計數法:

```
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    if(root == null) return res;
    List<Integer> item = new ArrayList<>();
    LinkedList<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    int curNum = 1, nextNum = 0;

    while(!queue.isEmpty()) {
        TreeNode cur = queue.poll();
        item.add(cur.val);
        curNum--;

        if(cur.left != null) {
            queue.offer(cur.left);
            nextNum++;
        }

        if(cur.right != null) {
            queue.offer(cur.right);
            nextNum++;
        }

        if(curNum == 0) {
            curNum = nextNum;
            nextNum = 0;
            res.add(item);
            item = new ArrayList<>();
        }
    }
    return res;
}
```

每層結尾用 null 標記法:

```
 public List<List<Integer>> levelOrder(TreeNode root) {
      List<List<Integer>> listSet = new ArrayList<List<Integer>>();
      if (root == null)   return listSet;
      Queue<TreeNode> queue = new LinkedList<>();
      queue.add(root);
      queue.add(null);    // flag of end-of-level . 每层结束后 queue.add(null)

      while (!queue.isEmpty()) {
         List<Integer> list = new ArrayList<Integer>();
         TreeNode curr = queue.poll();
         while (curr != null) {
            list.add(curr.val);
            if (curr.left != null)  queue.add(curr.left);
            if (curr.right != null) queue.add(curr.right);
            curr = queue.poll();
         }
         listSet.add(list);
         if (!queue.isEmpty())   queue.add(null);
      }
      return listSet;
}
```

DFS 法:

```
public List<List<Integer>> levelOrder(TreeNode root) {
      List<List<Integer>> res = new ArrayList<List<Integer>>();
      levelHelper(res, root, 0);
      return res;
}

   public void levelHelper(List<List<Integer>> res, TreeNode root, int height) {
      if (root == null) return;
      if (height >= res.size()) {
         res.add(new LinkedList<Integer>());
      }
      res.get(height).add(root.val);
      levelHelper(res, root.left, height+1);
      levelHelper(res, root.right, height+1);
   }
```