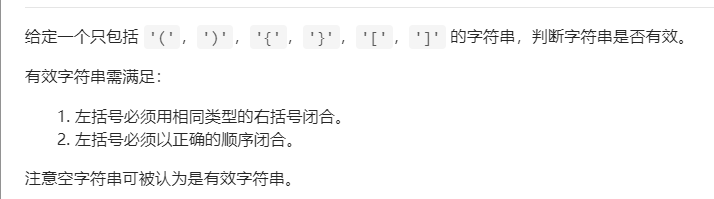
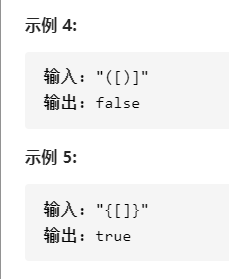
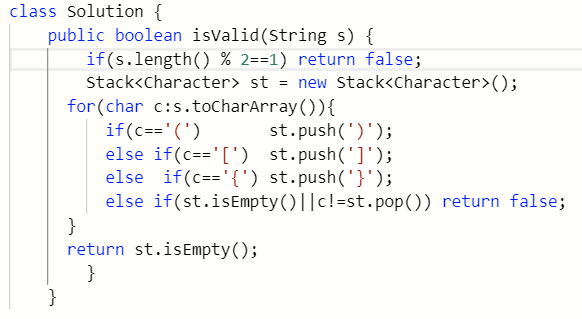
Stack 栈

1.20





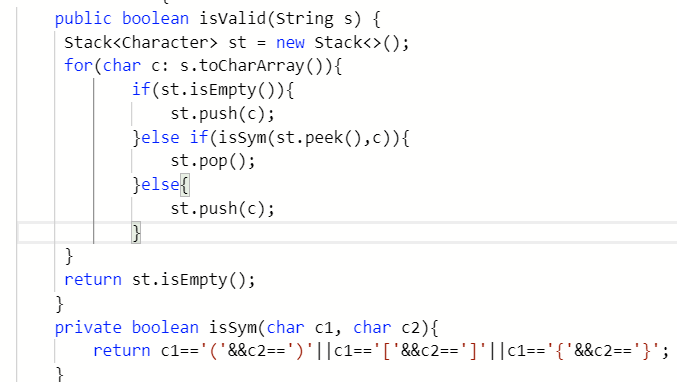
解法1：把输入string变成char数组，从第一个元素开始如果是‘(’,则在stack上push‘)’，以此类推，最有比较剩下每个元素是不是和stack上push过的一样



解法2：把输入的string的(),{},[]都替换成“”（循环一半的string长度），最后看这个string是不是empty的



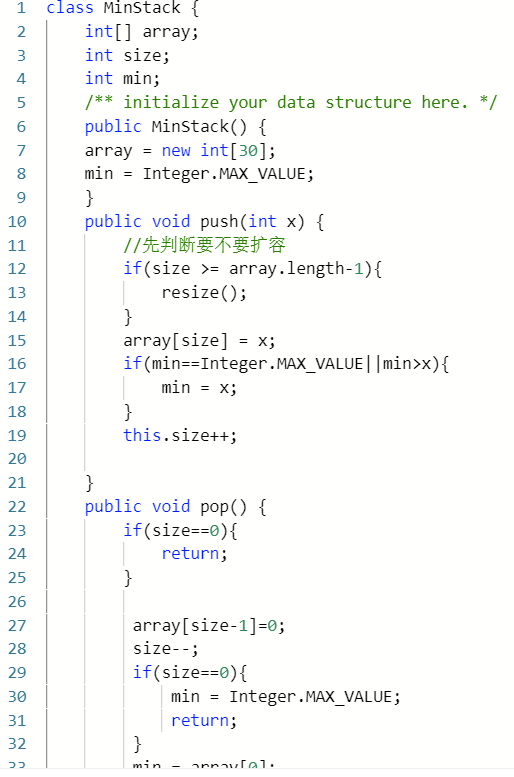
解法3：把输入的string变成char数组，只要stack空就放元素进去，之后找下个元素是不是和stack上这个元素配对，比如stack上是“(”下个元素是“)”，是的话就把这个元素pop出，不是的话就继续往stack里放，最后判断stack的size是不是0

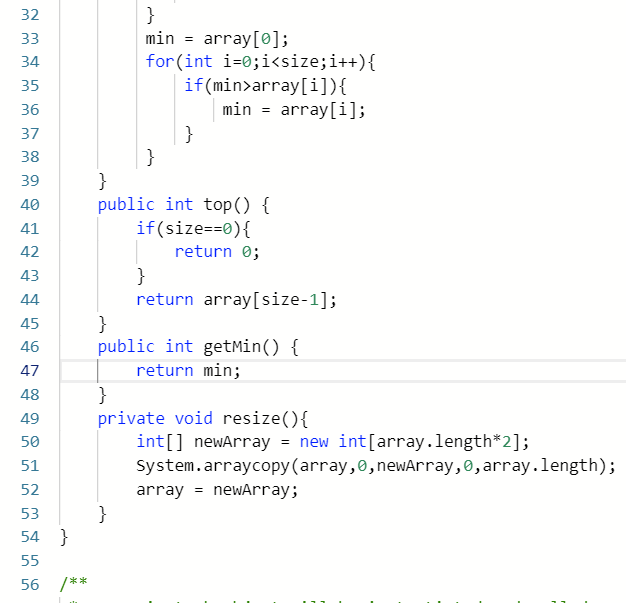


1. 155

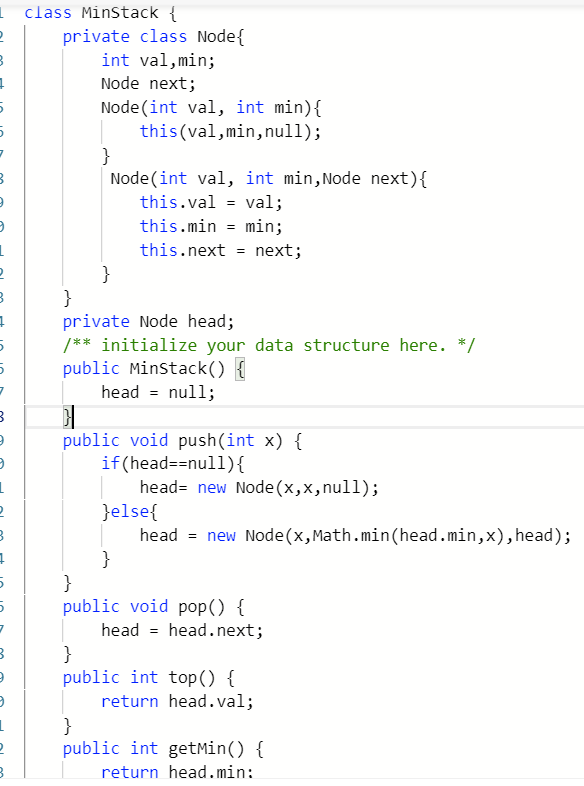


解法1：用数组实现，用min标记最小值，push把元素放到数组的末尾，pop把数组的最后一个元素remove也就是stack的peek（array满了要扩容）



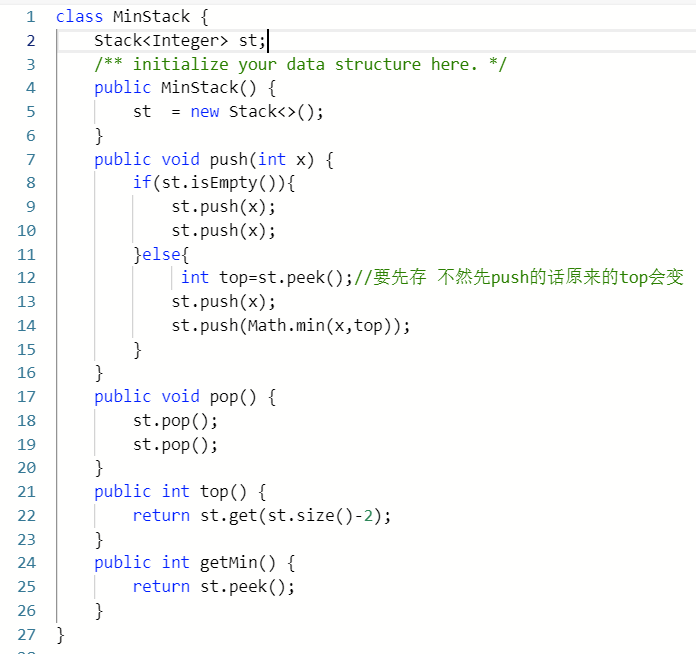


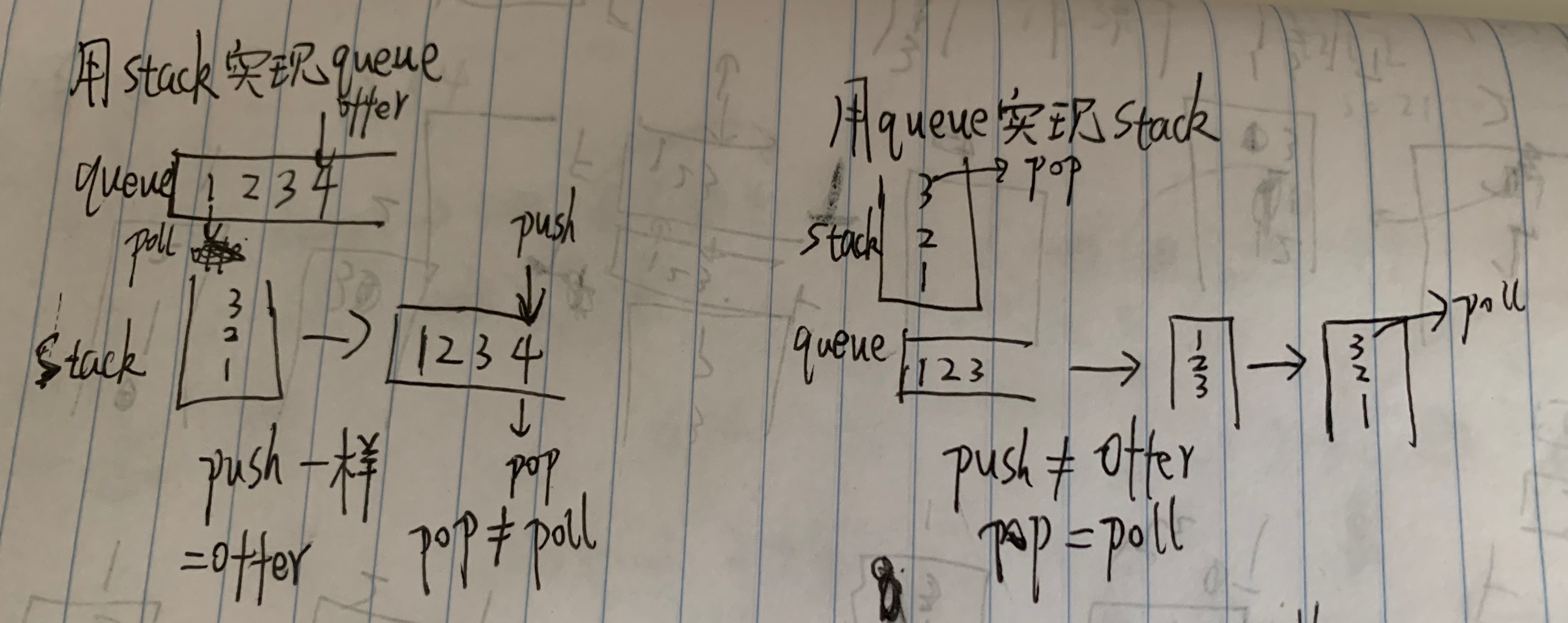
解法2：用链表实现，创建一个Node类(val,min,next)，head存储本身和head.next的min两者的最小值，所以链表最后的min是34456667，但val与这无关。这样getMin就可以是O(1)



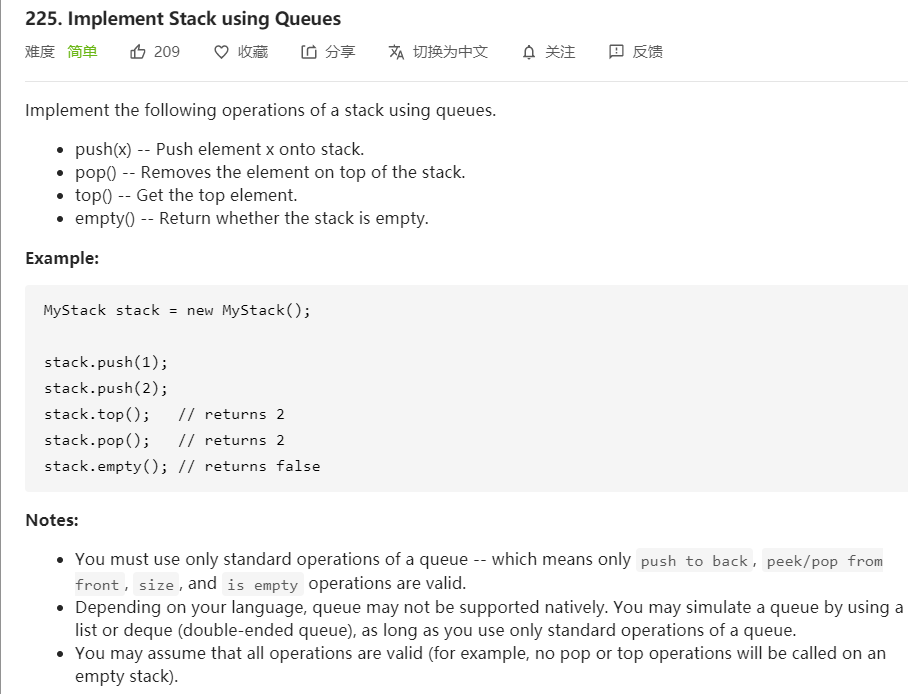
解法3：一个stack，每次放一个元素，都再顶端放一个stack里最小的元素

放2则是2-2 再放3则是2-2-3-2 再放1则是2-2-3-2-1-1所以pop要把顶端两个元素都pop掉，top则是倒数第二个元素，也就是最后加的元素

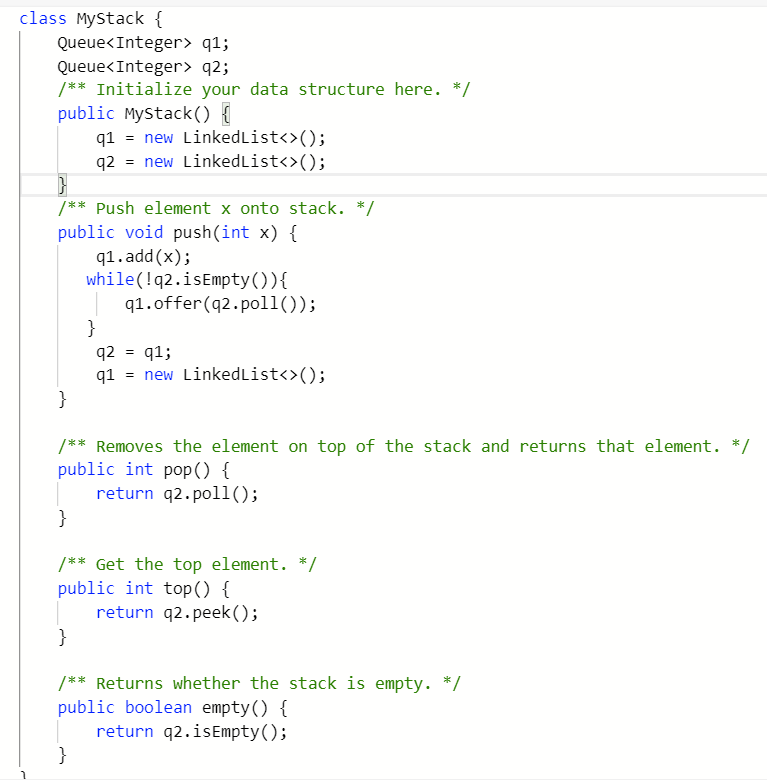




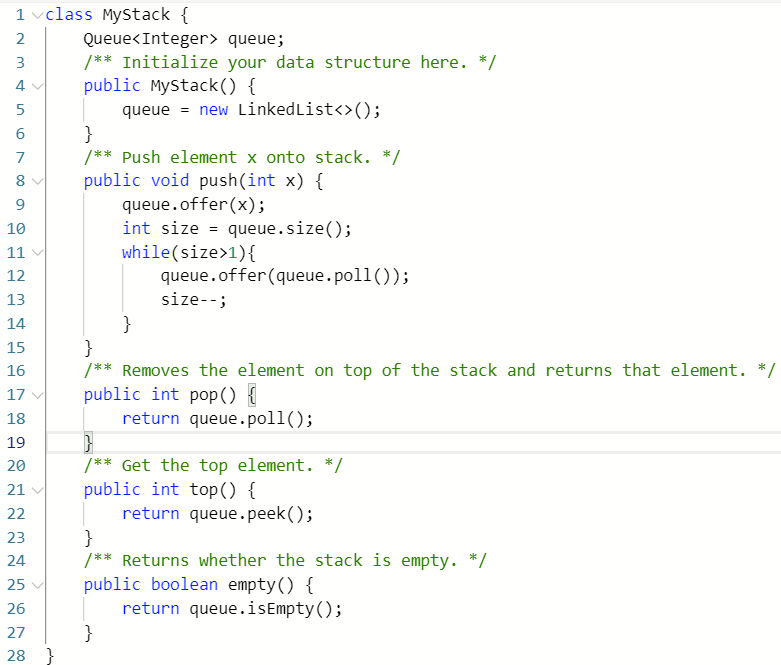
3.表明front和tail不然会弄错



解法1：用两个Queue<Integer> a = new LinkedList<>()，其中一个每次push时候都是空的，直接放入push的元素再把另一个queue的元素放入a，之后交换元素 然后a又是空的，b则是所有正确顺序的元素



解法2：用单链表，push时候先加进去，再把除了新加的元素其他元素放到新加的元素的后面

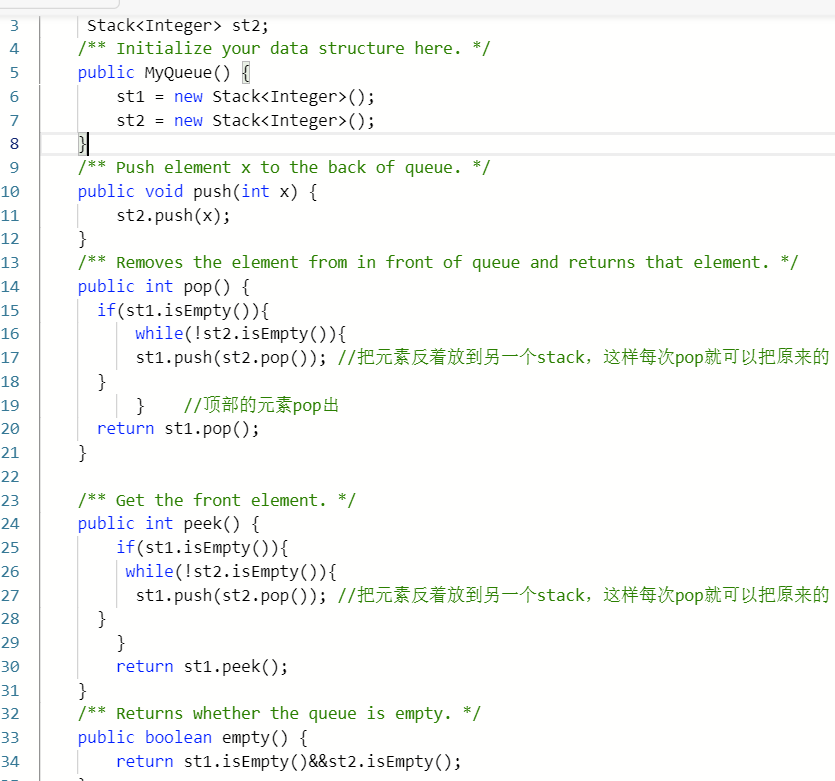


4.232

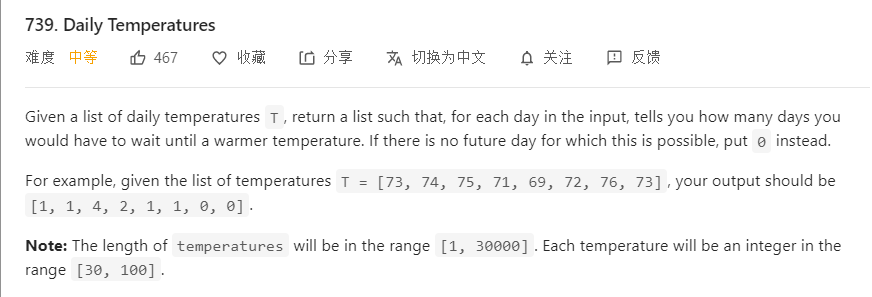
要把stack横过来看，push和pop就是在末尾增减

用stack实现queue

解法：用两个stack，一个用来加元素，另一个用来pop和peek，只有当另一个为空时候才会把第一个元素reverse地放进去，这样保证后加的在st1，先加的在st2先out



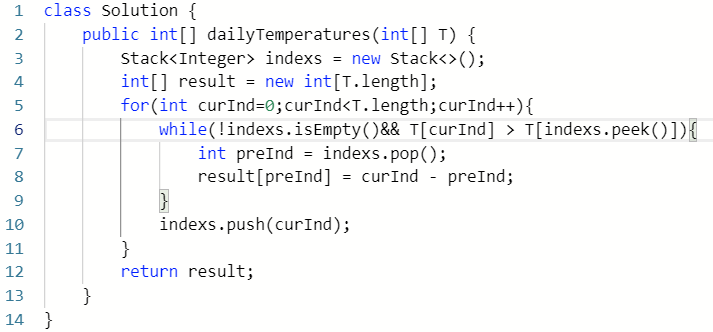
5.



解法1：在遍历数组时用栈把数组中的数存起来，如果当前遍历的数比栈顶元素来的大，说明栈顶元素的下一个比它大的数就是当前元素

每次都先比较第i和i+1个，第i个就是stack的peek，i+1就是curInd的元素，

如果i+1比i小则继续往后比较i+1和i+2，如果i+2比i+1大则把i+1这个元素(也就是stack的顶端元素)pop出，再比较i+2和i，如果大，则继续pop出，小则用i+3去和他比



解法2：根据题意，从最后一天推到第一天，这样会简单很多。因为最后一天显然不会再有升高的可能，结果直接为0。再看倒数第二天的温度，如果比倒数第一天低，那么答案显然为1，如果比倒数第一天高，又因为倒数第一天对应的结果为0，即表示之后不会再升高，所以倒数第二天的结果也应该为0。自此我们容易观察出规律，要求出第i天对应的结果，只需要知道第i+1天对应的结果就可以：

若T[i] < T[i+1]，那么res[i]=1；

若T[i] > T[i+1]：

- res[i+1]=0，那么res[i]=0;

- res[i+1]!=0，那就比较T[i]和T[i+1+res[i+1]]（即将第i天的温度与比第i+1天大的那天的温度进行比较）

最后一个元素直接标为0，因为后面没有day了

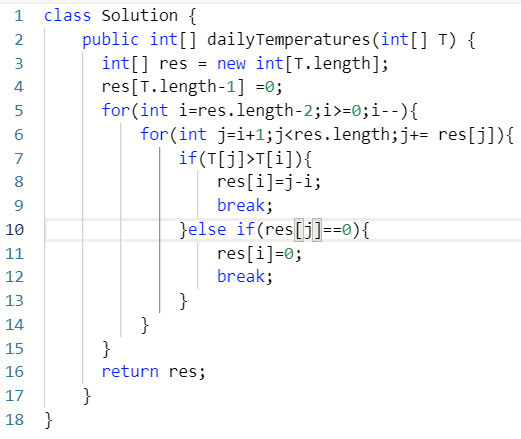
所以是从倒数第三个开始，与倒数第二个元素比较

如果前者小于后者，则直接j-i看之间差了几天

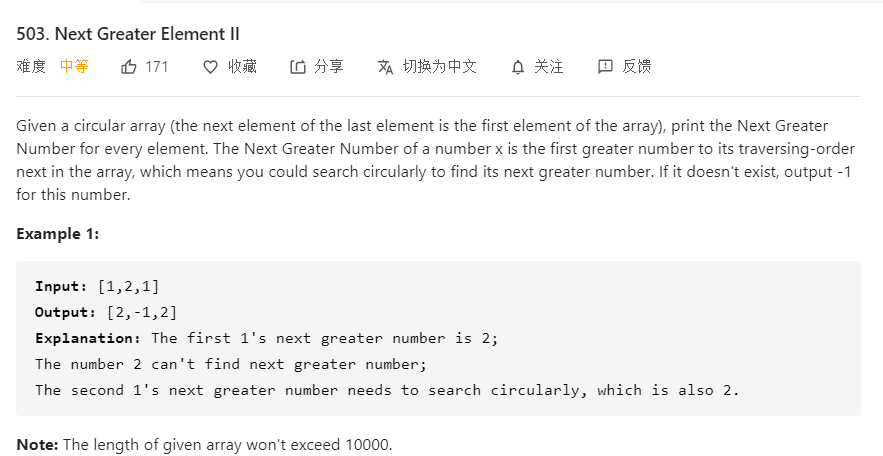
如果前者大于等于后者，而且后者res上对应位置是0，则说明后者后面的元素没有比他大的了，所以后面的元素也不会有比前者大的元素，所以前者对应的res上也是0.

如果前者大于等于后者，但后者对应的res位置上不是0(说明后面还有比后者大的元素，所以后者的位置(j)要加其对应的res上的数(res[j])来得到比后者大的那个数，再用这个数与前者比较，反复这样)

break不能少不然会一直往后面比



6.



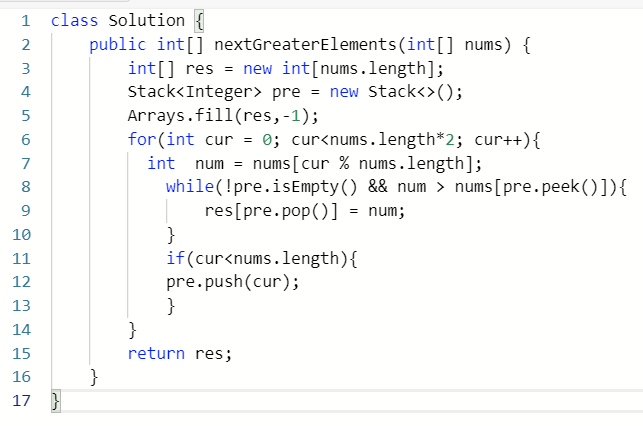
Arrays.fill(array,-1) 把array里每个元素都变为-1

解法1：先把返回的数组填满-1，对于每个index，index+1比index大则直接放

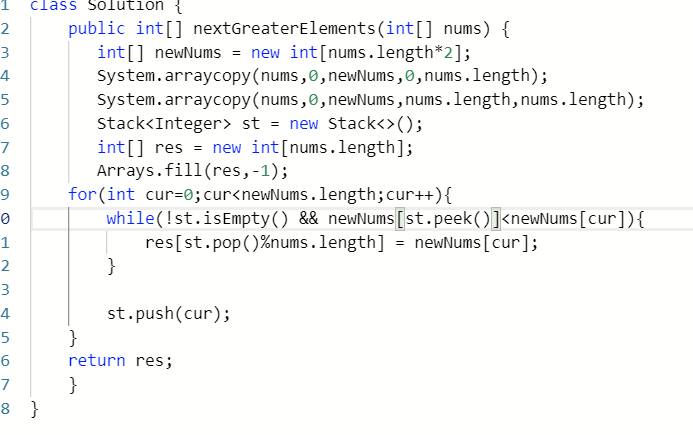
如果后面比前面的小则放stack上，之后继续比较index+1和index+2，同上，

但是因为是circular array，所以cur表示的数可以用nums[cur%给的数组的长度](这样就可以遍历到最后再从头遍历)，只要设置成小于2倍的length就行。但只有当cur<n时候才在stack上加index，这样确保元素不会被重复加

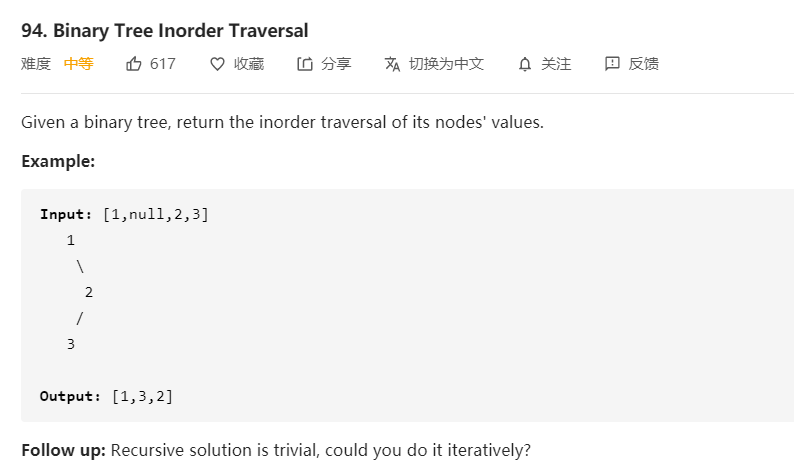
用num这个变量去记录cur的位置的值



解法2：把原数组copy两份结合就和高温天的题目一样了



5.inorder traversal



解法1：用递归 因为是中序遍历，所以是左中右

解法2：通用解法 中序是右中左(原来是左中右 也就是反一下其他同理)

public List<Integer> inorderTraversal(TreeNode root) {

      List<Integer> result = new ArrayList<>();

      Stack<TreeNode> st = new Stack<TreeNode>();

      if(root!=null){

          st.push(root);

      }

      while(!st.isEmpty()){

           TreeNode node = st.peek();

          if(node!=null){

              st.pop(); //先把节点弹出 后面再依次加 右中左

              if(node.right!=null) st.push(node.right);

                st.push(node);//添加中节点

                st.push(null);//中节点还没被处理过 标记一下

                if(node.left!=null) st.push(node.left);

          }else{

              st.pop();//把null的节点弹出

            node = st.peek();

            st.pop();

            result.add(node.val);

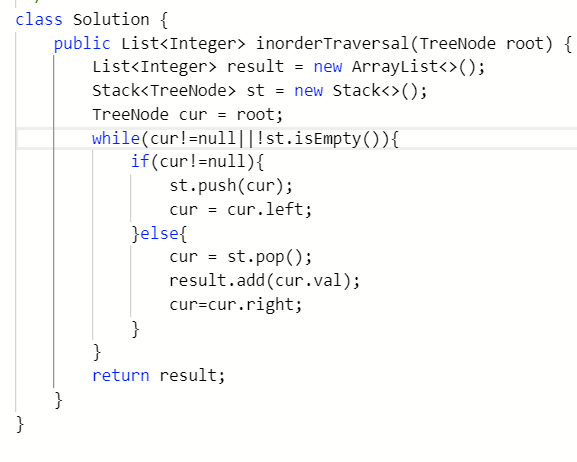
          }

      }

        return result;

}

解法3：用list去放返回的元素，用stack去遍历先往左遍历到null再从stack里把元素pop到list里 每pop一次都是current指针指向pop出来的node的右边



6.level order traversal



解法1：用helper method，里面用LinkedList，只要queue里面不为空，每次poll一个元素到要返回的list，并且在末尾offer上这个poll出来的左右儿子，这样能确保每次出来的parents并且把他的儿子们放在末尾

