剑指33 二叉搜索树的后序遍历序列

等于：用一个后序遍历数组，创建一个二叉搜索树，能否成功。

做不出啊！！！

知道关键点在于：后序遍历的最后一个一定为根节点，且左半边元素全小于它，右半边元素全大于它。

方法一：简单分治。O(n^2)
　最后一个一定是当前的根节点，然后遍历前面所有项，满足前面一段全小于根节点，后面一段全大于根节点。然后分两半各自再递归。

方法二：利用栈。
　后序遍历的逆序。
　TODO没有细看，没理解。

```
// 法一：
class Solution {
public:

    bool judge(vector<int>& postorder, int left, int right) {
        if(left >= right) return true;

        int border = right - 1;
        while(border >= left && postorder[border] > postorder[right]) border--;
        int p = border;
        while(p >= left && postorder[p] < postorder[right])  p--;
        if (p >= left) return false;
        return judge(postorder, left, border) && judge(postorder, border + 1, right - 1);
    }

    bool verifyPostorder(vector<int>& postorder) {
        return judge(postorder, 0, postorder.size() - 1);
    }
};
```

```
// 评论区法二：
class Solution {
    public boolean verifyPostorder(int[] postorder) {
        // 单调栈使用，单调递增的单调栈
        Deque<Integer> stack = new LinkedList<>();
        int pervElem = Integer.MAX_VALUE;
        // 逆向遍历，就是翻转的先序遍历
        for (int i = postorder.length - 1;i>=0;i--){
            // 左子树元素必须要小于递增栈被peek访问的元素，否则就不是二叉搜索树
            if (postorder[i] > pervElem){
```

```
            return false;
        }
        while (!stack.isEmpty() && postorder[i] < stack.peek()){
            // 数组元素小于单调栈的元素了，表示往左子树走了，记录下上个根节点
            // 找到这个左子树对应的根节点，之前右子树全部弹出，不再记录，因为不可能在往根节点的右子树走了
            pervElem = stack.pop();
        }
        // 这个新元素入栈
        stack.push(postorder[i]);
    }
    return true;
  }
}
```