完整代码如下。注意：大家把这个代码提交给 LeetCode 需要将 MyStack3 修改成 MyStack：

class MyQueue3 {

```
private Stack<Integer> stack1;
private Stack<Integer> stack2;
int front;

/** Initialize your data structure here. */
public MyQueue3() {
    stack1 = new Stack<>();
    stack2 = new Stack<>();
}

/** Push element x to the back of queue. */
public void push(int x) {

    if(stack1.isEmpty())
        front = x;
    stack1.push(x);
}

/** Removes the element from in front of queue and returns that element. */
public int pop() {

    // 如果 stack2 不为空，直接返回 stack2 的栈首元素
    if(!stack2.isEmpty())
        return stack2.pop();

    while(stack1.size() > 1)
        stack2.push(stack1.pop());

    return stack1.pop();
}

/** Get the front element. */
public int peek() {
    if(!stack2.isEmpty())
        return stack2.peek();
    return front;
}

/** Returns whether the queue is empty. */
public boolean empty() {
    return stack1.isEmpty() && stack2.isEmpty();
}
```

}

这个代码，整体复杂度没有变，pop 的最差时间复杂度依然是 O(n) 的。

但是，同学们可以想象一下，一旦调用 pop 或者直接使用 O(1) 的复杂从 stack2 中拿到结果，或者 stack2 为空，将现在 stack 中的所有元素放到 stack2 中。

相当于，平均对于每一个元素来说，都只有一次机会进 stack1，也只有一次机会进 stack2。所以，这样实现，pop 的均摊复杂度，变成了 O(1) 的。

大家还记得均摊复杂度吗？回忆一下，我们再介绍动态数组的 resize 的时候，学习了均摊复杂度。如果需要，再复习一下？

怎么样，是不是很酷？

大家加油！：）