232. Implement Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push`, `peek`, `pop`, and `empty`).

Implement the `MyQueue` class:

- `void push(int x)` Pushes element x to the back of the queue.
- `int pop()` Removes the element from the front of the queue and returns it.
- `int peek()` Returns the element at the front of the queue.
- `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

Example 1:

```
Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 1, 1, false]

Explanation
MyQueue myQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false
```

Notes:

- You must use **only** standard operations of a stack, which means only `push to top`, `peek/pop from top`, `size`, and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Queue → FIFO Stack → LIFO

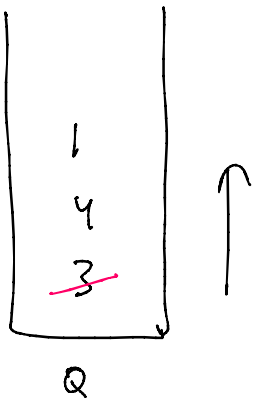
Queue

p(3)
p(4)
p(1)

top() ⇒ 3

pop() // remove 3

top() ⇒ 4



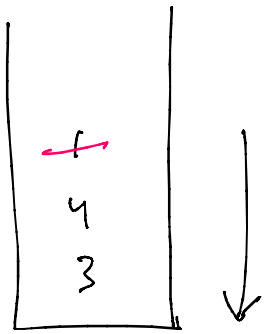
Stack

p(3)
p(4)
p(1)

top() ⇒ 1

pop() // remove 1

top() ⇒ 4

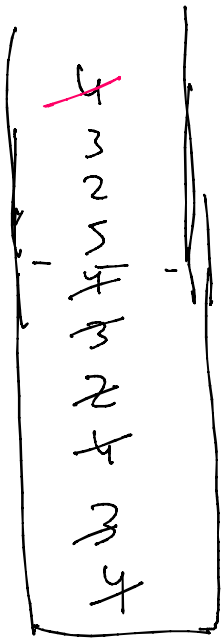


St

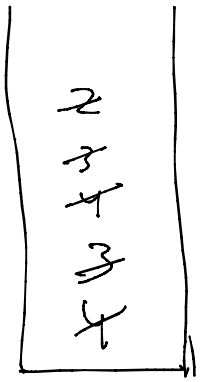
Implement Queue using Stack

1st Approach

push(4)
push(3)
push(2)
push(5)
top() → 4
pop() // 4
top() → 3



S1



S2

push(x)

(·) S1 → S2
(·) x → S1
(·) S2 → S1

pop()

S1.pop()

follow this

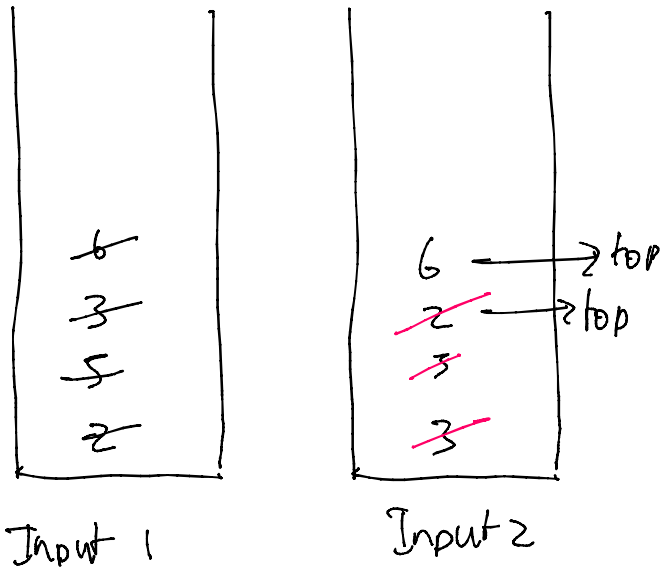
T.C ⇒ O(N)

S.C ⇒ O(2N)

2nd Approach

2nd Approach

push(2)
push(5)
push(3)
top() // 2
pop() // 2
push(6)
pop() // 5
pop() // 3
top() // 6




```
push(x)
    (·) add x → input

pop()
    if (output not empty
        output pop )
    else
        input → output
        output pop()

top()
    if (output not
        empty)
        return output top
    else
        input → output
        output.top()
```

T.C ⇒ push ⇒ O(1)

pop / top ⇒ O(1) sometimes O(N)

So O(1) amortised.

S.C ⇒ O(2N)

```
class MyQueue {
    stack<int> in;
    stack<int> out;
public:
    /** Initialize your data structure here. */
    MyQueue() {

        in = stack<int>();
        out = stack<int>();

    }

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

        in.push(x);

    }

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

        if(out.empty()) {

            //taxing
            while(!in.empty()) {

                //loading the whole stack in out
                out.push(in.top());
                in.pop();

            }

        }


        int val = out.top();
        out.pop();
        return val;

    }

    /** Get the front element. */
    int peek() {

        if(out.empty()) {

            //taxing
            while(!in.empty()) {

                //loading the whole stack in out
                out.push(in.top());
                in.pop();

            }

        return out.top();

    }

    /** Returns whether the queue is empty. */
    bool empty() {

        return out.empty() && in.empty();

    }
};
```