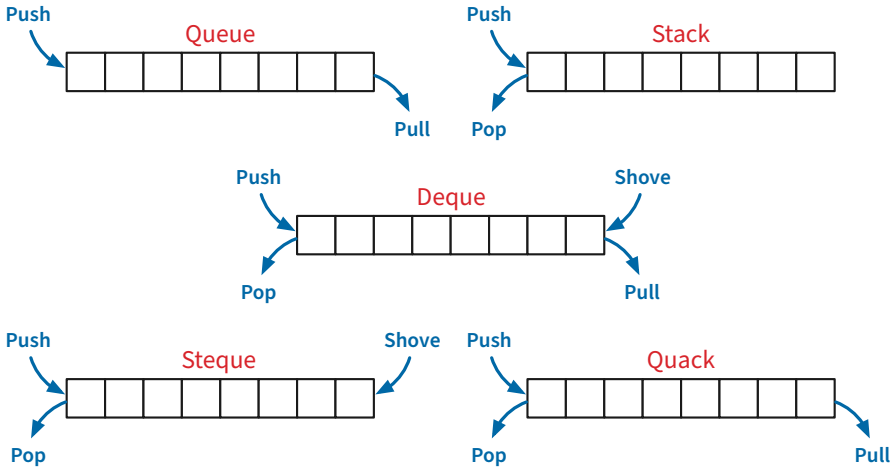
There are five canonical species of sequence data types that support insertions and deletions only at one or both ends; these are illustrated below.



In Monday's honors lecture, we discussed the following implementation of a queue using two black-box stacks, which implements each queue operation using $O(1)$ amortized stack operations. Here "black-box" means that we know *nothing* about how the stacks are implemented; we only know that they correctly support the standard stack operations PUSH, POP, and ISEMPTY.

Our queue data structure consists of a pair of black-box stacks *In* and *Out* and a counter *num*, which stores the number of items in the queue. After each queue operation, each item in the queue is stored in exactly one of the two component stacks.

```
INITQUEUE():
  q ← new queue
  q.num ← 0
  q.In ← INITSTACK()
  q.Out ← INITSTACK()
  return q
```

```
QUEUEPUSH(q, x):
  STACKPUSH(q.In, x)
```

```
QUEUEISEMPTY(q):
  return STACKISEMPTY(q.In) ∧ STACKISEMPTY(q.Out)
```

```
QUEUEPULL(q, x):
  if ISEMPTY(q.Out)
    ⟨⟨Transfer In to Out⟩⟩
    while ¬ISEMPTY(q.In)
      STACKPUSH(q.Out, STACKPOP(q.In))
  return STACKPOP(q.Out)
```

The only interesting part of these algorithms is the highlighted *transfer* loop in QUEUEPULL Let n_0 and n_1 respectively denote the number of items in the queue just after a transfer and just before the next transfer, respectively. Between these two transfers, we must perform exactly n_0 QUEUEPULLS (to empty *Out*) and exactly n_1 QUEUEPUSHES (all into *In*). The second transfer requires exactly $2n_1$ stack operations; we can charge these to the n_1 intermediate QUEUEPUSHES. Each QUEUEPUSH pays for 2 stack operations, and therefore uses 3 amortised stack operations. Because all transfer operations are charged elsewhere, each QUEUEPULL uses 1 amortized stack operation.

Alternatively, we can compute amortized costs using summation. Each item that is ever pushed into the queue participates in at most four stack operations: pushed onto *In*, popped from *In*, pushed onto *Out*, and finally popped from *Out*. Thus, any sequence of N_I pushes and N_O pulls induces at most $4N_I$ stack operations. We conclude that each QUEUEPUSH uses 4 amortized stack operations, and each QUEUEPULL uses ***zero*** amortized stack operations.

This week's exercises ask you to generalize this reduction to other ended-sequence data types.

"Quack" and "steque" (or equivalently, "two" and "three") were swapped in the original release of this homework.[1] (I forgot that ducks like peas.) Feel free to submit solutions for either problem 1 *or* problem 2(a) for feedback.

1. Describe how to implement a quack with ***three*** black-box stacks, so that each quack operation requires $O(1)$ amortized stack operations. Your solution should include both a description of the quack update algorithms and an amortized time analysis.

The remaining problems are for you play with on your own.
Discussion in office hours or on Discord is welcome, but don't submit solutions!

---

2. (a) Describe how to implement a steque with ***two*** black-box stacks, so that each steque operation requires $O(1)$ amortized stack operations.

   (b) Describe how to implement a deque with two black-box stacks and a black-box queue, so that each quack operation requires $O(1)$ amortized stack/queue operations.

   (c) Describe how to implement a deque with one black-box stack and one black-box steque, so that each deque operation requires $O(1)$ amortized stack/steque operations.

*3. Describe how to implement a stack with two black-box queues, so that each stack operation takes $O(\sqrt{n})$ amortized queue operations, where n is the current number of items in the stack. *[Hint: Why is $O(1)$ amortized queue operations impossible?]*

★4. Describe how to implement a queue with $O(1)$ black-box stacks, so that each queue operation takes $O(1)$ stack operations *in the worst case*. *[Hint: Build double- and half-size array-lists* incrementally *instead of all at once. Your implementation can store more than one copy of each item in the abstract queue.]*

[1] Jeff holds up his right hand and says, "Sometimes I can swear this is my *right* hand."