1. (a) Prove that in any intermixed sequence of PUSH and PULLMANY operations, each operation runs in $O(1)$ amortized time.

Solution: Charge the time to PULL each element off the queue, plus the overhead of one iteration of the loop, to the corresponding PUSH operation. Each PUSH operation received O(1) charge, so the amortized cost of a PUSH is still O(1). The amortized cost of PULLMANY is actually zero. ■

Solution: Consider a mixed sequence of n PUSHes and m PULLMANYs. Clearly $m < n$, because we can't pull more than we push. Each of the n items that ever appear in the queue is pushed onto the queue once and pulled off the queue at most once. Thus, the total number of elementary queue operations is at most $2n$, so the total time for all operations is $O(n)$. We conclude that the amortized cost of each operation is $O(1)$. ■

Solution: Define the potential Φ of the queue to be the number of elements in the queue. Each PUSH operation increases the potential by 1, so its amortized cost is $O(1)$. Each operation PULLMANY(k) requires k individual PULL operations and decreases the potential by k, so its amortized cost is actually zero. ■

(b) Prove that for any integers ℓ and n, there is a sequence of ℓ PUSHCLONES and PULLMANY operations that requires $\Omega(n\ell)$ time, where n is the maximum number of items in the queue at any time. Such a sequence implies that the amortized time for each operation is $\Omega(n)$.

Solution: Consider the sequence of operations executed by the following algorithm. (Here $\bullet$ is an arbitrary value that doesn't matter.)

```
MANYMANYBAD(ℓ, n):
  for i ← 1 to ℓ
      PUSHCLONES(•, n)
      PULLMANY(n)
```

Each call to PUSHCLONES and PULLMANY takes $\Theta(n)$ time, so the total running time is $\Theta(n\ell)$, as required. ■

(c) Describe a data structure that supports arbitrary intermixed sequences of PUSHCLONES and PULLMANY operations in $O(1)$ amortized time per operation. Like a standard queue, your data structure should use only $O(1)$ space per item.

Solution: We maintain a standard FIFO queue PQ of *ordered pairs*, where each ordered pair $(item, num)$ represents num copies of $item$. However, because our default queue interface does not allow us to modify pairs inside PQ, we maintain the pair that should be at the front of PQ (that is, the pair that would be PULLed next) in a separate field $head$. (The abstract queue of pairs is empty if and only if $head.num = 0$.)

We implement PUSHCLONES and PULLMANY as follows:

```
PUSHCLONES(x, k):
  if head.num = 0
      head ← (x, k)
  else
      PUSH(PQ, (x, k))
```

```
PULLMANY(k):
  while k > head.num
      k ← k − head.num
      head ← PULL(PQ)
  x ← head.item
  if k = head.num
      head ← PULL(PQ)
  else
      head.num ← head.num − k
  return x
```

The actual running time of PUSHCLONES is $O(1)$, and the actual running time of PULLMANY is $O(1 + \ell)$, where ℓ is the number of iterations of the while loop. As in part (a), we can charge each of the ℓ calls to PULL (and the rest of that iteration of the while loop) to the corresponding call to PUSHCLONES. Thus, each operation takes $O(1)$ amortized time. ■