Sanity Check

- Where are //@requires, //@loop_invariant, //@ensures, and //@assert checked?
- What steps are needed to prove that a loop invariant is correct?
- What does it mean for a program with runtime $T(n)$ to be $O(f(n))$?

Data Structures Review

Interface vs. Implementation
We have been implementing stacks and queues as using linked lists. Let’s revisit queues and think about the different levels of abstraction involved in implementing and using them.

**INTERFACE**

```c
queue Q = queue_new();

enq(Q, 5);

enq(Q, 6);

int x = deq(Q);
```

**IMPLEMENTATION**

```c
queue Q = alloc(struct queue_header);
list* p = alloc(struct list_node);
Q->front = p;
Q->back = p;

enq(Q, 5);

enq(Q, 6);

int x = Q->front->data;
Q->front = Q->front->next;
```
Recursion

We can also reason about the correctness of recursive functions. First we prove partial correctness, then termination. Once we have both, we know that the function is correct. Let’s prove the correctness of a recursive factorial function.

```c
int factIter(int n) {
    // Assume factIter(n) = n! for n >= 0
    return 1;
}
```

```c
int factRec(int n) {
    // Assumes n >= 0
    if (n == 0)
        return 1;
    else
        return n * factRec(n-1);
}
```

**PARTIAL CORRECTNESS:**

**Base case:** \( n = 0 \) — We return 1 in this case, and since \( 0! = 1 \), the function is correct for \( n=0 \).

**Induction Hypothesis:** Assume \( \text{factRec}(n) = n! \) for some \( n \geq 0 \)

**Induction Step:** Consider \( \text{factRec}(n+1) = (n+1) \cdot \text{factRec}(n) \).

**TERMINATION:**

**Base Case:** \( n = 0 \) — The function terminates immediately.

**Induction Hypothesis:** Assume \( \text{factRec}(n) \) terminates for some \( n \geq 0 \).

**Induction Step:** Consider \( \text{factRec}(n+1) \). It returns \( n \cdot \text{factRec}(n) \), and since we know \( \text{factRec}(n) \) terminates by the IH, and multiplication is a terminating operation, we know the function terminates.