Daily Coding Problem #16

Problem

This problem was asked by Twitter.

You run an e-commerce website and want to record the last `N` order ids in a log. Implement a data structure to accomplish this, with the following API:

- record(order_id): adds the order_id to the log
- get_last(i): gets the ith last element from the log. i is guaranteed to be smaller than or equal to N.

You should be as efficient with time and space as possible.

Solution

It seems like an array would be the perfect fit for this problem. We can just initialize the array to have size N, and index it in constant time. Then, when we record any orders, we can pop off the first order and append it to the end. Getting the ith last order would then just be indexing the array at `length - i`.

```
class Log(object):
    def __init__(self, n):
        self._log = []
        self.n = n

    def record(self, order_id):
        if len(self._log) >= self.n:
            self._log.pop(0)
        self._log.append(order_id)

    def get_last(self, i):
        return self._log[-i]
```

This is one issue with this solution, however: when we have to pop off an element when the array is full, we have to move every other element down by 1. That means `record` takes O(N) time. How can we improve this?

What we can do to avoid having to moving every element down by 1 is to keep a current index and move it up each time we record something. For `get_last`, we can simply take `current - i` to get the appropriate element. Now, both `record` and `get_last` should take constant time.

```
class Log(object):
    def __init__(self, n):
        self.n = n
        self._log = []
        self._cur = 0
```

```
    def record(self, order_id):
        if len(self._log) == self.n:
            self._log[self._cur] = order_id
        else:
            self._log.append(order_id)
        self._cur = (self._cur + 1) % self.n

    def get_last(self, i):
        return self._log[self._cur - i]
```

By the way, this is called a ring buffer or circular buffer!
