EECS 311: Data Structures and Data Management Homework 2
Assigned: 9/30/10 Due: 10/7/10

1 Reading.

Chapter 3.

2 Problems.

1. Prove that, for any constant c, if we resize an array based data structure from capacity n to capacity $n+c$ whenever it is full then the amortized runtime of each insert operation is $\Theta(n)$.

2. Prove by induction that $\sum_{i=0}^{k} c^i \leq c^{k+1}$ for any $c \geq 2$.

3. Write C++ or Java code to implement a *deque* (a double-ended-queue, pronounced as "deck") that supports the following operations. The two ends of a deque are the *top* and a *bottom*.

   - `create()`: creates an empty deque.
   - `push(Object x)`: inserts `x` on the top end of the deque.
   - `pop()`: removes and returns top element of deque.
   - `inject(Object x)`: inserts `x` on the bottom of the deque.
   - `eject()`: removes and returns bottom element of deque.

   All operations must be worst-case or amortized $\Theta(1)$ time and your deque must be able to accommodate any number of elements. You may not use any classes from standard libraries.

4. Consider implementing a counter that supports *create*, *increment*, and *print* operations using the standard binary representation (i.e., in a binary array). Recall that in worst-case that increment of an n bit number takes $T(n) = \Theta(n)$.

   (a) How many bits are needed to represent the counter m increment operations after creation?

   (b) Prove that, starting from creation, m increment operations take $T(m) = \Theta(m)$.

   (c) Compare the increment runtime (using big-oh) of this binary counter to the counter implementation given in class using the redundant binary representation.

   (d) Suppose we also wish to support the *decrement* operation. Show that the worst-case amortized runtime of the increment and decrement operations is $\omega(1)$, i.e., they are not constant time.