EECS 311: Data Structures and Data Management Homework 2
Assigned: 10/03/08 Due: 10/10/08

1 Reading.

Chapter 3.

2 Problems.

1. Prove by induction that $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$.

2. Write C++ code to implement an array-based queue data structure. Your data structure should implement the *amortized constant time* dynamic resizing method discussed in lecture. You must implement a constructor, destructor, *enqueue*, and *dequeue*. You may optionally implement *head* and *is-empty*. Use an *assert statement* to ensure that *dequeue* and *head* are never invoked on an empty queue. Your code should complile and run correctly; however, only turn in the source code for your queue, a test program that tests that your queue operations work, and a print-out of the output of your test program. Your queue should hold `int`s; however, you may optionally make it a template holding type `Object`. You may not use the *Standard Template Library*.

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

   (c) How does this compare to the counter implementation given in class using the redundant binary representation?

   (d) Suppose we also wish to support the *decrement* operation. Show that starting from creation, there is a sequence of m increment and decrement operations with $T(m) = \Theta(m \log m)$.

   (e) How does this compare to the counter implementation given in class using the redundant binary representation?

4. Implement a queue using two stacks and a constant amount of auxillary data. Describe clearly the purpose of any auxillary data you need. You may only access the stacks through the standard stack ADT which supports operations *create*, *push*, *pop*, *top*, and *is-empty*. Give the *enqueue* and *dequeue* operations in pseudo code. Creation followed by any m queue operations should take $T(m) = \Theta(m)$ time, i.e., your queue should be amortized constant time. What is the worst-case runtime of *dequeue* and *enqueue*?