Advanced Algorithms, Fall 2012

Prof. Bernard Moret

Homework Assignment #1

due Sunday night, Sept. 30

Write your solutions in LaTeX using the template provided on the Moodle and web sites and upload your PDF file through Moodle by 4:00 am Monday morning, Oct. 1.

Question 1.
Implement a queue using two ordinary stacks and analyze the amortized cost of each Insert-Queue and Delete-Queue operation. (Assume the cost of each Pop or Push operation is 1).

Now implement a *double-ended* queue with two ordinary stacks and give a counterexample to show that the amortized cost of its four operations (Insert-Back, Insert-Front, Delete-Back, and Delete-Front) is no longer $O(1)$. A counterexample is a sequence of operations that takes time growing faster than the number of operations in the sequence. Indicate which single operation(s) are such that their removal makes the the amortized cost of the remaining three operations $O(1)$.

Question 2.
Suppose that, instead of using sums of powers of two, we represent integers as sums of Fibonacci numbers. In other words, instead of an array of 0/1 bits, we keep an array of 0/1 "fits", where the i^{th} least significant "fit" indicates whether the sum includes the i^{th} Fibonacci number F_i. For example, the "fitstring" 101110_F represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$.

Verify that a number need not have a unique fitstring representation (in contrast to bitstrings) and describe an algorithm to implement `increase` and `decrease` operations on a single fitstring in constant amortized time.

Question 3.
Using arrays, as in a simple queue or a binary heap, creates a problem: when the needs grow beyond the allocated array size, there is no support for simply increasing the array. Therefore consider the following solution.

- When the next insertion encounters a full array, the array is copied into one twice its size (and the old array returned to free storage) and the insertion then proceeds.
- When the next deletion reduces the number of elements below one quarter of the allocated size, the array is copied into one half its size (and the old array returned to free storage) and the deletion then proceeds.

Assume that returning an array to free storage takes constant time, but that creating a new array takes time proportional to its size.

- Prove that, in the context of a data structure that grows or shrinks one element at a time, the amortized cost per operation of array doubling and array halving is constant.
- Why not halve the array as soon as the number of elements falls below one half of the allocated size?