CS3230 : Design and Analysis of Algorithms (Fall 2014)

Tutorial Set #5

[For discussion during Week 7]

S-Problems are due: Thu, 02-Oct, before noon.

OUT: 27-Sep-2014 **Tutorials:** Tue-Wed, 30-Sep & 01-Oct 2014

Topics: Amortized Time Complexity

IMPORTANT: Read "Remarks about Homework" – also applies to tutorials.

Prepare your answers to all the D-Problems in every tutorial set.

When preparing to present your answers,

- Think of a CLEAR EXPLANATION
- Illustrate with a good worked example;
- Describe the main ideas,
- Can you sketch why the solution works;
- Give analysis of running time, if appropriate
- Can you think of other (perhaps simpler) solutions?

Helpful Hints Series: (On Amortized Analysis)

Read *and understand this.* Reflect on this after you finished each question in this tutorial.

When doing amortized analysis for a data structure that supports multiple operations (like INSERT, DELETE, DELETEMIN, etc), you must give an amortized analysis for *each and every* operation supported by the data structure.

In doing so, we will inevitably also analyze the worst-case time of each operation, in addition to the amortized time of each operation. It is quite OK that for some operations, the worst-case time and the amortized time are the same.

Of course, for some operations, the amortized time is *much better* than the worst-case time – because we are "paying for" the extra work using credits saved (accounting method) or saved potential (potential method).

And this is *precisely* the reason we do amortized complexity analysis.

Learn to reflect on each operation and see where do we save up credits, and for what? Often, this will help you develop the appropriate potential function.

Routine Practice Problems *-- do not turn these in -- but make sure you know how to do them.*

[**IMPORTANT:** Go back and read the Helpful Help Series for this tutorial.]

R1. Exercises 17.1-1 of [CLRS] **(Stack with MultiPUSH and MultiPOP)**
If a MULTIPUSH operation were included in the set of stack operations, would the $O(1)$ bound on the amortized cost of stack operations continues to hold?
[First read [CLRS]-17.1on "Stack operations", especially part on MULTIPOP operation.]

R2. Exercises 17.2-1 of [CLRS] **(Bounded Stack with Backup)**
A sequence of stack operations is performed on a stack whose size never exceeds k. After every k operations, a copy of the entire stack is made for backup purposes. Show that the cost of n operations, including copying the stack, is $O(n)$ by assigning suitable amortized costs to the various stack operations.

S-Problems: (To do and submit by due date given in page 1)
Solve this S-problem(s) and submit for grading.

IMPT: Write your NAME, Matric No, Tutorial Group (eg: G6) in your Answer Sheet.
It will be helpful if you use the name as it appears in your Matric Card.

T5-S1. Modified from 17.2-3 of [CLRS] (Binary Counter with Reset)
In the example of incrementing a binary counter given in the lectures, suppose we wish not only to increment a counter but also to *reset* the counter to zero (i.e., set all bits in it to 0). We want to implement a counter as a bit vector so that any sequence of n INCREMENT / RESET operations takes time O(n) on an initially zero counter.
(*Hint:* Keep a pointer to the most-significant, highest-order 1-bit. *Why do we need it*?)

(a) Give the **algorithms** for INCREMENT and RESET operations.
(b) Using the **accounting method**, show that any sequence of n INCREMENT / RESET operations take time O(n) on an initially zero counter.

D-Problems: Solve these D-problems and prepare to discuss them in tutorial class. You may be called upon to present your solution *or your best attempt at a solution*. Your solution presentation does NOT need to be fully correct, given your best attempt. The TA will help clarify and correct any issues or errors.

D1. Exercises 17.3-6 of [CLRS] **(Queue with Two Stacks)**
Show how to implement a queue with two ordinary stacks so that the amortized cost of each ENQUEUE and each DEQUEUE operation is $O(1)$.

D2. (Exercise 17.1-2 of [CLRS]) (INCREMENT and DECREMENT)
Show that if a DECREMENT operation were included in the k-bit counter example, n operations could cost as much as $\Theta(nk)$ time. (Note: Give such a sequence of operations.)

D3. Modified from Problem 17.3-3 of [CLRS]) Amortized Analysis of Binary Heap
Consider the usual binary min-heap data structure (from [CLRS]-C6), with INSERT and DELETE-MIN done in $O(\log n)$ worst-case time.
Give an amortized analysis that shows that INSERT has amortized time $O(\log n)$ and DELETE-MIN has amortized time $O(1)$.

D4* Alternative Dynamic Table
Suppose we can INSERT or DELETE an element into a dynamic table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an INSERT operation, if the table is more than ¾ fill, we allocate a new table twice as big as current table, insert everything into the new table, and then free the old table.
- After a DELETE operation, if the table is less than ¼ full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of n operations (any combination of INSERT or DELETE operations), the amortized time per operation is still a constant.

(*Note:* Do not use the potential method (it makes it much more difficult))

[* A little bit harder, but not *too much* harder. Can skip if there is not enough time.]

Advanced Problems – Try these for challenge and fun. There is no deadline for A-problems. *Turn in your attempts DIRECTLY to Prof. Leong. Do not combine it with your HW solutions.*)

A6. [*Careful* Recursive Max-and-Min]
Chapter 9.1 of [CLRS] presents an iterative algorithm for computing *both* the maximum and minimum in an array $A[1..n]$ of n numbers using at most $3\lfloor n/2 \rfloor$ comparisons in the worst case. Give a (recursive) divide and conquer algorithm to do the same thing and also using at most $3\lfloor n/2 \rfloor$ comparisons, for all values of n.
[*Hint:* Check your recursive algorithm when n=12. How many comparisons does it make?]

A7. [Average number of trailing 1's]
Let $X(k)$ be the total number of trailing 1's in the k-bit binary expansion of *all* integers $0, 1, 2, 3, \ldots, 2^k-2, 2^k-1$.

(a) Find a recurrence relation satisfied by the sequence $\{ X(k) \}$.
(b) Solve the recurrence for $X(k)$.
(c) Hence, find the *average* number of trailing 1's in a k-bit string.
(d) Why is Prof. Leong interested in this question? Where is the link to analysis of algorithms?