15-451 Algorithms, Fall 2014

Homework # 2 **Due: September 9, 2014**

Your homework should be submitted as a single PDF file using `autolab`. You are allowed to work in groups of 2-3, but you must not share written work, and must write your solutions yourself. Please cite your sources, and your collaborators; mention explicitly if you did not discuss problems with others. Solve problems #1 and #2.

Problem #H is a "honors" problem. It is optional, and you won't get hints or assistance from the course staff. Nor will you get any points if you solve it. You will, however, get your name on the class honor roll on the course page. It's all for the fame and glory, and of course the joy of solving cool problems...

(60 pts) 1. **Max Stacks and Quacks**

(a) (Queues from Stacks) Suppose we had code lying around for implementing a stack, and we now wanted to implement a *queue*. (A stack supports *push* and *pop*, with the usual semantics. A queue supports *insert* and *remove* with the usual semantics.) One way to do this is to use *two* stacks $S1$ and $S2$. To insert into our queue, we push into stack $S1$. To remove from our queue we first check if $S2$ is empty and if so we dump $S1$ into $S2$ (we repeatedly pop the top of $S1$ and push it onto $S2$, until $S1$ is empty). Then we pop from $S2$. Here is pseudocode:

- insert(x): $S1$.push(x)
- remove(): If $S2$ is empty then dump($S1, S2$). Return $S2$.pop().
- dump($S1, S2$): while $S1$ is not empty do $S2$.push($S1$.pop()).

We're interested in analyzing the cost of a queue under various scenarios, where the cost of a push is 1 and the cost of a pop is 1, and performing a dump when $S1$ has n elements costs $2n$ (since we do n pushes and n pops).

For example, if we start with an empty queue, insert 3 elements, then remove 1 of them, the cost would be 10 (3 for the inserts, 6 for the dump, 1 for the remove).

Prove that starting with an empty queue, a sequence of n operations (inserts and removes) costs at most $3n$. Use the banker's method or the physicist's method (potential).

(b) We want to now maintain a max-queue, which supports the following operations:

- *insert*(number x): adds x to the back of the queue
- *remove*: removes the element at the front of the queue
- *return-max*: returns the maximum number among the elements still in the queue. (Does not enqueue or dequeue anything.)

Show how to implement a max-queue, which takes $O(1)$ amortized time per operation. That is, a sequence of n operations (consisting of some number of *insert*, *remove* and *return-max* operations in any order) should take total time $O(n)$.

Hint: one way is via first implementing a max-stack.

(40 pts) 2. **You Be the Adversary**

In HW#1, you showed how to find both the maximum element and the minimum element of an arbitrary set of n distinct numbers using $\frac{3}{2}n - 2$ comparisons, where n was even. You'll now prove a matching lower bound: $\frac{3}{2}n - 2$ comparisons are necessary. More formally, you'll prove the following theorem:

Theorem: Consider a deterministic comparison based-algorithm $\mathcal{A}$ which does the following: Given a set S of n numbers as input, $\mathcal{A}$ returns the largest and the smallest element of S. Prove that there is an input on which $\mathcal{A}$ must perform at least $\frac{3}{2}n - 2$ comparisons.

Hints:

> Call an element *top* if it has been involved in at least one comparison and it has never lost a comparison.
>
> Call an element *bottom* if it has been involved in at least one comparison and never won a comparison.
>
> Call an element *free* if it has been involved in zero comparisons.
>
> Call an element *middle* if it has won at least one comparion and lost at least one comparison.
>
> Every element falls into exactly one of these categories, and this classification of elements evolves over time. Initially all elements are free. You know that at the end of a run of any correct algorithm, there are ______free ones, ______middle ones, ______top ones and ______bottom ones.
>
> Define a potential function $\Phi()$ that is a linear function in the number of each type of element. You, the adversary, have to decide the outcome of each comparison to give to algorithm $\mathcal{A}$ when it asks you to compare two elements. By making the "right" choice of the outcome of a comparison, you should ensure that the potential decreases by at most 1 for each comparison. Using this fact, along with the initial value and final value of the potential, prove that algorithm $\mathcal{A}$ must do at least $\frac{3}{2}n - 2$ comparisons.

(0 pts) H. **Balls and Bins**

There are n balls and an infinite number of bins. A bin can have 0 or more balls in it. A move consists taking all the balls of some bin and putting them into distinct bins. The cost of a move is the number of balls moved. Consider a sequence S of states $(s_0, s_1, \ldots s_m)$, where each state is obtained from the previous one by making a move. Let $C(S)$ denote the cost of the sequence.

(a) Find a function $U(n)$ and for this function prove that for all sequences S:

$$C(S) \leq n + mU(n)$$

(b) Find a function $L(n)$ and prove that there exists a valid infinite sequence of states $(s_0, s_1, \ldots)$ such that each move costs at least $L(n)$.

Ideally $U(n) = L(n)$ (at least for an infinite number of values of n). If not, it should be the case that $U(n) = \Theta(L(n))$.