Practice problems for amortized analysis

1. A sequence of stack operations is performed on 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 stack operations, including copying the stack, is O(n) using the accounting method and potential method.

   Solution Sketch: Include the cost of the copy in Push. So for accounting method, start with amortized cost of Push = 3, Pop = 0. For a push, of the 3, one is for the push itself, one pays for popping it if done, and one pays for copying it to backup if it is not popped. Solve potential method with the same logic.

2. Will the O(1) amortized cost if increments in a k-bit binary counter remain valid if you also allow decrement operations (i.e, in any sequence of intermixed increment and decrement operations)? If yes, show the amortized analysis. If not, justify why not.

   Solution Sketch: No. Just consider a sequence of n operations of the form increment/decrement/increment/decrement/…..starting from the value $2^k - 1$ for a k-bit counter, changes all bits each time, so O(nk) total time. Note that starting from 0 does not change this as n can be arbitrary, so we can have the first $2^k - 1$ to reach upto the value $2^k - 1$, and then follow the sequence considered.

3. Consider the implementation of a queue using two stacks A and B (I am sure you all know this implementation). Find the amortized cost of a sequence of n enqueue and dequeue operation using each of aggregate, accounting, and potential method.

   Solution Sketch: The implementation was discussed in Monday's class, but here it is again. Keep 2 stacks, IN and OUT. A push simply pushes in the IN stack. A pop can have two cases: (i) if OUT stack is empty, pop everything from the IN stack and push it in OUT stack one by one, then pop the top element from the OUT stack, (ii) if OUT stack is not empty, just pop the top element from the OUT stack.

   Aggregate method: Every element can be (i) pushed at most twice, once in the IN stack and once from the IN to OUT stack, and (ii) popped at most twice, either from the OUT stack or for transferring from IN to OUT stack. So total time = O(n), amortized cost O(1) per operation.

   Accounting method: Break the two operations into three operations: (i) Push, (ii) Easy dequeue (when OUT is not empty), (iii) Hard dequeue (when OUT is empty). Give amortized costs Push = 3, Easy dequeue = 1, Hard dequeue = 1. Basically easy dequeue pays for itself (just a pop, does not generate or use a credit), every push stores two credits

for the hard dequeue for the move, one for popping it from IN and one for pushing it to OUT. The hard dequeue uses the credits stored with each element pushed to pop it from IN and push it to OUT, plus one final pop. So all O(1).

Potential method: Choose potential = no. of elements in IN stack. Push increases potential by 1, easy dequeue leaves it unchanged, and hard dequeuer decreases it by –k, where k is the no. of elements in IN stack when the hard dequeue is done. So Amortized cost of push = O(1) + 1 = O(1), of easy dequeuer = O(1) + 0 = O(1), of hard dequeue = O(k) – k = O(1).

P