Algorithms
Homework 7

Saad Mneimneh
Computer Science
Hunter College of CUNY

This homework covers amortized analysis. It might be a little harder than usual because it requires some careful thought about how to achieve the amortized time. I provided hints, but still... Next homework will be about graphs.

Problem 1: A bulk stack
Consider a stack that stores items that are collections of m elements. So one could only push and pop a collection at a time. For this bulk stack, we will call the operations $Bush$ and Bop. Using the bulk stack as an underlying structure, we want to implement a regular stack that can $Push$ and Pop individual elements.

Given a collection C that is on the top of the stack, to $Push$ an element we can $Bop\ C$, add the element to it, and $Bush$ it back on the stack. Similarly, to Pop an element, we can $Bop\ C$, delete it's top element, and $Bush$ it back on the stack. We assume:

- we can check if the bulk stack is empty
- a collection C itself acts like a stack, so we can add and remove to/from its "top" ($Push$ and Pop) in $O(1)$ time
- we can determine the number of elements in a collection C (which is at least 0 and at most m)
- $Bush$ and Bop take $O(m)$ time

(a) One can keep a copy of the top collection "active" while performing $Push$ and Pop operations and refraining from using $Bush$ and Bop until $Push$ sees a full collection or Pop sees an empty collection. Describe a sequence of $Push$ and Pop operations that still has an average of $O(m)$ time per operation.

(b) Describe a mechanism to achieve $O(1)$ amortized time per $Push/Pop$ operation by keeping two active collections. Do the analysis using the aggregate method, the accounting method, and the potential method.

Hint: For the potential method, consider the distance from the top of the stack to either the bottom of the top collection or the top of the bottom collection, whichever is smaller. So this distance is zero if the top of the stack is the bottom of the top collection or the top of the bottom collection. The distance is at most $m - 1$.

Problem 2: Queue with IncreaseKey and Maximum
Consider a FIFO queue Q that supports the following operations:

- $Insert(Q, x)$: adds x to the queue
- $Extract(Q)$: removes the earliest inserted item
- $IncreaseKey(Q, k)$: increases $key[x]$ by k for every $x \in Q$

(a) Describe a way to implement all these operations in $O(1)$ time each.

(b) Assume now we want to add the following operation:

- $Maximum(Q)$: simply returns the object x with largest key in Q.

Describe a way to implement all four operations in $O(1)$ amortized time. *Hint*: Maintain in another structure a chronological list of non-increasing keys. These keys act as successive maximums.