Yanwen Gong
CS 261, Spring 2020
Homework 1
Due Monday, April 6

NOTE: Homeworks for this class are provided both as a pdf file and as a LaTeX template. You may use the template to produce your answers as a pdf file (for instance by using a local LaTeX installation or by copying the template onto overleaf.com) or you may produce the answers as a pdf file in any other way (including writing them by hand and then scanning them, as long as your handwriting is legible).

1. The lecture includes the description of an implementation of stacks using a linked list, with a value and a "next" pointer in each list node, and with a "top" pointer from the stack to the first node in the list. If we use the same node structure and also store an additional "bottom" pointer to the last node in the list, it is possible to implement a queue, whose dequeue operation is the same as the pop operation of a stack, but with a different enqueue operation.

Describe (in English or pseudocode) how to perform an enqueue operation in constant time with this modified structure. Be sure to properly handle the case when the queue is empty (its top pointer is a null pointer).

```
if top = none then
    top = bottom = newNode
else
    bottom.next = newNode
    bottom = bottom.next
```

■

2. Suppose we modify the binary counter data structure described in the lecture to include an additional "decrement" operation, that subtracts one from the binary counter. Its implementation is almost the same as the "increment" operation: change low-order bits from 0 to 1 until finding the lowest-order 1 and changing it to 0, then stop.

Describe how to find, for any n, a sequence of operations that causes this operation to make at least $\Omega(n \log n)$ changes to the bits that it stores. (This is big-Omega notation, used just like O-notation but for lower bounds.) Use your sequence to argue that it is not possible for this structure to have constant amortized time per operation.

For any n, we can take certain number of operations (plus and/or minus) to change n to one number in the form of 2^m, eg $32 = 2^5$. Then we perform 2^m minus 1 and plus 1 consecutively. Each operation changes O($logm$) bits, and we will perform this operation lots of time. Since the amortized time per operation is O($logm$), which depends on m, it is not constant.

■

3. The dynamic array analysis from the lecture (in its most basic form, with the allocated array increasing in size by a factor of two each time and never decreasing) used the potential function

$$\Phi = |2 \times \text{length} - \text{available}|.$$

Explain why the amortized analysis doesn't work if we forget the absolute value signs and try to use the function

$$\Psi = 2 \times \text{length} - \text{available}$$

instead.

The length can be changed. Suppose we continue decreasing the length, when the length is smaller than half of the available, the potential ϕ will be negative, which does not satisfy the requirement to have ϕ bigger or equal for all states. Thus the absolute values sign is needed. ∎

4. Suppose that we need to implement a circular list of items (each item has an item clockwise from it in the list and another item counterclockwise of it, and these clockwise and counterclockwise links connect the items into a single cycle), with one item designated as the current item, and the following four operations:

- Move the current item one position clockwise and return its new value
- Move the current item one position counterclockwise and return its new value
- Insert a new item clockwise of the current item and move the current item to it
- Remove the current item and move to the item that was clockwise from it

You have available a library that implements a deque data structure, which maintains a linear (not circular) sequence of items and lets you add or remove items at either the left or right end of the sequence. Describe how to represent the circular list as a deque, in such a way that all four circular list operations can be performed using a constant number of deque operations. Give the details of how to implement the move-clockwise circular list operation using deque operations. (You do not need to describe the details of the other three circular list operations, but your representation should allow them all to be performed using a constant number of deque operations.)

I will use the deque to store the circular list. The current item is represented by the last item in the deque. The item clockwise from the current is the head of the deque, and the item counterclockwise of it is the second to last one in the deque. When adding a new item, it will be added to the tail of this deque as well, and represent current.

To implement move clockwise circular list operation using deque operations:
deque.addLast(deque.pollFirst())
deque.peekLast() ∎