Problem Session 1

Problem 1-1. Asymptotic behavior of functions

For each of the following sets of five functions, order them so that if f_a appears before f_b in your sequence, then $f_a = O(f_b)$. If $f_a = O(f_b)$ and $f_b = O(f_a)$ (meaning f_a and f_b could appear in either order), indicate this by enclosing f_a and f_b in a set with curly braces. For example, if the functions are:

$$f_1 = n, \qquad f_2 = \sqrt{n}, \qquad f_3 = n + \sqrt{n},$$

the correct answers are $(f_2, \{f_1, f_3\})$ or $(f_2, \{f_3, f_1\})$.

a)	**d)**
$f_1 = (\log n)^{2019}$	$f_1 = 2^n$
$f_2 = n^2 \log(n^{2019})$	$f_2 = n^3$
$f_3 = n^3$	$f_3 = \binom{n}{n/2}$
$f_4 = 2.019^n$	$f_4 = n!$
$f_5 = n \log n$	$f_5 = \binom{n}{3}$

Problem 1-2. Given a data structure `D` supporting the four first/last sequence operations:
`D.insert_first(x)`, `D.delete_first()`, `D.insert_last(x)`, `D.delete_last()`,
each in $O(1)$ time, describe algorithms to implement the following higher-level operations in terms of the lower-level operations. Recall that `delete` operations return the deleted item.

(a) `swap_ends(D)`: Swap the first and last items in the sequence in `D` in $O(1)$ time.

(b) `shift_left(D, k)`: Move the first k items in order to the end of the sequence n `D` in $O(k)$ time. (After, the k^{th} item should be last and the $(k+1)^{\text{st}}$ item should be first.)

Problem 1-3. Double-Ended Sequence Operations

A dynamic array can implement a Sequence interface supporting worst-case $O(1)$-time indexing as well as insertion and removal of items at the back of the array in amortized constant time. However, insertion and deletion at the front of a dynamic array are not efficient as every entry must be shifted to maintain the sequential order of entries, taking linear time.

On the other hand, a linked-list data structure can be made to support insertion and removal operations at both ends in worst-case $O(1)$ time (see PS1), but at the expense of linear-time indexing.

LL lookups take linear time

Show that we can have the best of both worlds: design a data structure to store a sequence of items that supports **worst-case** $O(1)$-time index lookup, as well as **amortized** $O(1)$-time insertion and removal at both ends. Your data structure should use $O(n)$ space to store n items.

Problem 1-4. Jen & Berry's

Jen drives her ice cream truck to her local elementary school at recess. All the kids rush to line up in front of her truck. Jen is overwhelmed with the number of students (there are $2n$ of them), so she calls up her associate, Berry, to bring his ice cream truck to help her out. Berry soon arrives and parks at the other end of the line of students. He offers to sell to the last student in line, but the other students revolt in protest: "The last student was last! This is unfair!"

The students decide that the fairest way to remedy the situation would be to have the back half of the line (the n kids furthest from Jen) reverse their order and queue up at Berry's truck, so that the last kid in the original line becomes the last kid in Berry's line, with the $(n+1)^{\text{st}}$ kid in the original line becoming Berry's first customer.

(a) Given a linked list containing the names of the $2n$ kids, in order of the original line formed in front of Jen's truck (where the first node contains the name of the first kid in line), describe an $O(n)$-time algorithm to modify the linked list to reverse the order of the last half of the list. Your algorithm should not make any new linked list nodes or instantiate any new non-constant-sized data structures during its operation.

(b) Write a Python function `reorder_students(L)` that implements your algorithm.

```
def reorder_students(L):
    '''
    Input:  L    | linked list with head L.head and size L.size
    Output: None |
    This function should modify list L to reverse its last half.
    Your solution should NOT instantiate:
        - any additional linked list nodes
        - any other non-constant-sized data structures
    '''
    ##################
    # YOUR CODE HERE #
    ##################
    return
```

6.006 Introduction to Algorithms

Spring 2020