## Warm-up

**Problem 1.** Suppose we implement a stack using a singly linked list. What would be the complexity of the push and pop operations? Try to be as efficient as possible.

**Problem 2.** Suppose we implement a queue using a singly linked list. What would be the complexity of the enqueue and dequeue operations? Try to be as efficient as possible.

## Problem solving

**Problem 3.** Given a singly linked list, we would like to traverse the elements of the list in reverse order.

a) Design an algorithm that uses $O(1)$ extra space. What is the best time complexity you can get?

b) Design an algorithm that uses $O(\sqrt{n})$ extra space. What is the best time complexity you can get?

You are not allowed to modify the list, but you are allowed to use position/cursors to move around the list.

**Problem 4.** Consider the problem of given an integer $n$, generating all possible permutations of the numbers $\{1, 2, \ldots, n\}$. Provide a recursive algorithm for this problem.

**Problem 5.** Consider the problem of given an integer $n$, generating all possible permutations of the numbers $\{1, 2, \ldots, n\}$. Provide a non-recursive algorithm for this problem using a stack.

**Problem 6.** Using only two stacks, provide an implementation of a queue. Analyze the time complexity of enqueue and dequeue operations.

**Problem 7.** We want to extend the queue that we saw during the lectures with an operation GETAVERAGE() that returns the average value of all elements stored in the queue. This operation should run in $O(1)$ time and the running time of the other queue operations should remain the same as those of a regular queue.

a) Design the GETAVERAGE() operation. Also describe any changes you make to the other operations, if any.

b) Briefly argue the correctness of your operation(s).

c) Analyse the running time of your operation(s).

## Advanced problem solving

**Problem 8.** Suppose that we use a dynamic array implementation of a stack where each time we run out of space we increase the capacity of the array by $\lceil\sqrt{N}\rceil$. In other words, if the array has capacity $N$ and becomes full, we replace it with another one of capacity $N + \lceil\sqrt{N}\rceil$.

Show that performing $n$ push operations takes $\Theta(n^{3/2})$ time.

**Problem 9.** In Problem 6 we looked at implementing a queue using two stacks. Give a better implementation of a queue using only two stacks such that both operations take $O(1)$ amortized time.

**Problem 10.** Given a singly linked list, we would like to traverse the elements of the list in reverse order. You are not allowed to modify the list and you are constrained on how many positions you can keep track of:

a) Design an algorithm that uses one position variable to go over the elements of the list. What is the best time complexity you can get?

b) Design an algorithm that uses two position variables to go over the elements of the list. What is the best time complexity you can get?

In both case you are allowed to use $O(1)$ extra space for counters and whatnot, but you are **not** allowed to use this space for extra positions.

$\star$ **Problem 11.** Given a singly linked list, we would like to traverse the elements of the list in reverse order. You are only allowed to use $O(1)$ extra space, but this time you are allowed to modify the list you are traversing. Give an $O(n)$ time algorithm.