Warm-up

Problem 1. Sort the following functions in increasing order of asymptotic growth

$$n, n^3, n \log n, n^n, \frac{3^n}{n^2}, n!, \sqrt{n}, 2^n$$

Problem 2. Sort the following functions in increasing order of asymptotic growth

$$\log \log n, \log n!, 2^{\log \log n}, n^{\frac{1}{\log n}}$$

Problem 3. 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 4. 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 5. Consider the following pseudocode fragment.

```
1: function printFive(n)
2:     for i ← 1; i ≤ 5; i++ do
3:         Print a single character 'n'
```

Using the O-notation, upperbound the running time of printFive.

Problem 6. Consider the following pseudocode fragment.

```
1: function stars(n)
2:     for i ← 1; i ≤ n; i++ do
3:         Print '*' i times
```

a) Using the O-notation, upperbound the running time of stars.

b) Using the Ω-notation, lowerbound the running time of stars to show that your upperbound is in fact asymptotically tight.

Problem solving

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).

Problem 8. Using only two stacks, provide an implementation of a queue. Analyze the time complexity of enqueue and dequeue operations.

Problem 9. 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.