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. Consider the following pseudocode fragment.

```
1: function STARS($n$)
2:     for $i \leftarrow 1$; $i \leq n$; 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 4. Recall the problem we covered in lecture: Given an array A with n entries, find $0 \leq i \leq j < n$ maximizing $A[i] + \cdots + A[j]$.

Prove that the following algorithm is incorrect: Compute the array B as described in the lectures. Find i minimizing $B[i]$, find j maximizing $B[j+1]$, return (i, j).

Come up with the smallest example possible where the proposed algorithm fails.

Problem solving

Problem 5. Given an array A consisting of n integers, we want to compute the upper triangle matrix C where

$$C[i][j] = \frac{A[i] + A[i+1] + \cdots + A[j]}{j - i + 1}$$

for $0 \leq i \leq j < n$. Consider the following algorithm for computing C:

```
1: function SUMMING_UP($A$)
2:     $C \leftarrow$ new matrix of size(A) by size(A)
3:     for $i \leftarrow 0$; $i < n$; i++ do
4:         for $j \leftarrow i$; $i < n$; j++ do
5:             Compute average of entries $A[i : j]$
6:             Store result in $C[i, j]$
7:     return $C$
```

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

b) Using the Ω-notation, lowerbound the running time of SUMMING-UP.

Problem 6. Come up with a more efficient algorithm for computing the above matrix $C[i][j] = \frac{A[i]+A[i+1]+\cdots+A[j]}{j-i+1}$ for $0 \leq i \leq j < n$. Your algorithm should run in $O(n^2)$ time.

Problem 7. Give a formal proof of the transitivity of the O-notation. That is, for function f, g, and h show that if $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.

Problem 8. Using O-notation, show that the follow snippet of code runs in $O(n^2)$ time. As an extra challenge, it can be shown that it actually runs in $O(n)$ time.

```
1: function ALGORITHM(n)
2:     $j \leftarrow 0$
3:     for $i \leftarrow 0$; $i < n$; $i$++ do
4:         while $j \geq 1$ and [condition that can be checked in $O(1)$ time] do
5:             $j \leftarrow j - 1$
6:         $j \leftarrow j + 1$
7:     return $j$
```

Problem 9. Given a string (which you can see as an array of characters) of length n, determine whether the string contains k identical consecutive characters. You can assume that $2 \leq k \leq n$. For example, when we take $k = 3$, the string "abaaab" contains three consecutive a's and thus we should return true, while the string "abaaba" doesn't contain three consecutive characters that are the same.

Your task is to design an algorithm that always correctly returns whether the input string contains k identical consecutive characters. Your solution should run in $O(n)$ time. In particular, k isn't a constant and your running time shouldn't depend on it.

Problem 10. Given an array with n integer values, we would like to know if there are any duplicates in the array. Design an algorithm for this task and analyze its time complexity.