

# INF 4130 Exercise set 2, 05/09-2013

---

We start softly with a few short exercises on algorithm running times, and running time analysis. As you know we usually use  $O$ -notation (more correctly, asymptotic notation) for running times. A short note on the course web page describes four variants of asymptotic notation:  $O$ ,  $\Theta$ ,  $\Omega$  and  $o$ .

## Exercise 1

- Show that  $n+1$  is  $O(n)$ .
- Show that  $n \log n$  is  $O(n^2)$ .
- Is  $2^{n+1} = O(2^n)$ ?
- Is  $\frac{10n + 16n^3}{2} = O(n^2)$ ?

## Exercise 2

- What do we know about the running time of an algorithm if it is  $O(n!)$ ?
- What do we know about the running time of an algorithm if it is  $\Omega(n)$ ?
- What do we know about the running time of an algorithm if it is  $\Theta(2^n)$ ?
- What do we know about the running time of an algorithm if it is  $O(n^2)$ ?
- The statement "This algorithm has a running time of at least  $O(n^2)$ ." may seem odd. Does it make sense?

We now continue with a few exercises on string search, partially from the textbook. Spend some time repeating/discussing why/how the different shift strategies of Knuth-Morris-Pratt and simplified Boyer-Moore (Horspool) work.

## Exercise 3 (Exercise 20.3 in Berman & Paul)

Simulate CreateNext page 637-8, use the pattern "abracadabra".

## Exercise 4

Calculate the array Shift[a:z] for the pattern P = "announce" - simulate CreateShift spage 639.

## Exercise 5

Draw uncompressed suffix trees for the strings "BABBAGE" and "BAGLADY". And check if "BAG" is a common substring. Can you make do with only one tree?

## Extra

**NB:** some background knowledge on regular languages and NFAs and DFA is needed.

As a general problem setting we may want to search for a string matching a given regular expression  $R$  in a longer string  $S$ . We may then reformulate the problems as searching for a string matching the regular expression  $.*R$  at the start of  $S$ . Here  $.*$  Means any symbol in our alphabet, and the asterisk means that what comes before it may be repeated zero or more times. So  $.*$  just means that anything can come before the string we really want (expressed by  $R$ ), including the empty string.

We may solve the problem as follows: first construct an NFA (non-deterministic finite automaton) corresponding to  $.*R$ . This can be done intuitively, or by a so-called Thompson-construction. Then we transform this non-deterministic machine into a DFA (deterministic finite automaton) in the standard way.

This DFA is easily transformed into a normal computer program that reads  $S$  in linear time, and every time we arrive in a final state for the DFA, we know that we have read something that matches with  $R$ .

QUESTION: Why is this method not as fast as it might seem? What limits it's running time? When will it be fast?

[end]