Self-study session 1, Discrete mathematics

First year mathematics for the technology and science programmes
Aalborg University

In this self-study session we are working with time complexity. Space complexity will not be considered. We will be using the math (CAS) software called Maple. Maple can be downloaded from `http://www.ekstranet.its.aau.dk/software` Screencast 1 from `http://first.math.aau.dk/dan/software/maple/` explains (in Danish) how to install Maple on your pc. Screencast 2 (og 3) shows how to use Maple in some examples (also in Danish).

Loops and complexity

Consider the following toy algorithm. You can copy it into Maple:

```
forfour:=proc(n::integer)
local i,j,k,l;
local a;
a:=0;
for i from 1 to n do
 for j from 1 to n do
  for k from 1 to n do
   for l from 1 to n do
    a:=a+1;
   od;
  od;
 od;
od;
return a;
end proc;
```

You can now use the procedure forfour on an arbitraty integer input. For instance,

```
forfour(0)
```

returns 0, and

```
forfour(7)
```

returns 2401.

Exercise 1:
Consider the algorithm forfour.

- *Prove that the worst-case complexity is* $\mathcal{O}(n^4)$.
- *Prove that the average-case complexity is* $\mathcal{O}(n^4)$.

If you want to know how much time Maple uses on a given calculation, you can use the command "time()".

Exercise 2:

- *Enter in Maple the command time(forfour(20)). Maple returns the time spent. Repeat this computation until you have 10 executions. Then compute the average time spent.*
- *Do the same for time(forfour(40))*
- *Finally, do the same for time(forfour(80))*
- *Show that doubling the input causes the execution time to be multiplied by approximately* 16 *(e.g. when you let the input grow as* $20 \to 40 \to 80$*).*
- *Explain how this compares with the estimates of complexity.*

We now modify the algorithm so that some part of it is not always executed.

```
forfourrand:=proc(n::integer)
local i,j,k,l,dice;
local a;
global b,c;
a:=0;
dice:=rand(1..10);
b:=dice();
c:=dice();
for i from 1 to n do
 if not b=2 then
  for j from 1 to n do
   for k from 1 to n do
    if not c=2 then
     for l from 1 to n do
      a:=a+1;
     od;
     else a:=7;
    fi;
   od;
  od;
 fi;
od;
return a;
end proc;
```

The line

```
dice:=rand(1..10);
```

defines a random number generator, returning integers in $1, 2, \ldots, 10$. The procedure calss

```
b:=dice();
c:=dice();
```

assigns random numbers between 1 and 10 to the variables b and c. We see that the algorithm runs faster if b is assigned the value 2. It is also faster if b is assigned an another value, but c is assigned the value 2.

Exercise 3:

- *Determine the worst-case complexity of "forfourrand".*
- *Determine the average-case complexity of "forfourrand".*
- *Test your knowledge about complexity using the command "time()".*

Finally, we modify "forfour" in a different way:

```
forfourvild:=proc(n::integer)
local i,j,k,l;
local a;
a:=1;
for i from 1 to n do
 for j from 1 to n do
  for k from 1 to n do
   for l from 1 to n do
    a:=2*a;
   od;
  od;
 od;
od;
return a;
end proc;
```

Exercise 4:
In this exercise we test the algorithm "forfourwild". Computation of complexity does not make much sense in this case, as we will see.

- *Show that the computation of complexity in exercise 1 still holds.*
- *Perform tests as in exercise 2, but with much smaller inputs. Maybe you can stop Maple by clicking the "stop"-button if necessary.*
- *Realize that complexity calculations (i.e., estimating number multiplications, addition, etc.) does say how much time is used if numbers are very big. After all, it is faster to multiply 56 by 2 than to multiply 2000009045 by 2.*

You can refine the complexity calculations so that it counts binary operations rather than operations in $\mathbb{Z}$. We will not do that in this self-study session.

Computation of determinant

In the remaining part og the self-study, we will work with determinants of $n \times n$ matrices

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}.$$

In the linear algebra course you have learned two ways to compute a determinant. In the following, they are called Method 1 and Method 2. **Metode 1:** Let A be an $n \times n$ matrix.

- If $n = 1$, then $A = [a_{11}]$ and we define $\det A = a_{11}$.
- If $n \geq 2$, we define A_{ij} to be the matrix obtained by deleting row i and column j from A. (This is an $(n-1) \times (n-1)$ matrix). We have:

  $$\det A = (-1)^{1+1} a_{11} \det A_{11} + (-1)^{1+2} a_{12} \det A_{12} + \cdots + (-1)^{1+n} a_{1n} \det A_{1n}. \quad (1)$$

  This method is also called cofactor expansion along the first row.

Exercise 5:
Use Method 1 to determine $\det\left(\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix}\right)$ *and to determine* $\det\left(\begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix}\right)$

If Method 1 is applied to a 2×2 matrix A (that is, $n = 2$), we obtain

$$\det A = a_{11}a_{22} - a_{12}a_{21}. \quad (2)$$

If Method 1 is applied to a 3×3 matrix A (that is, $n = 3$), we get

$$\det A = a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31}. \quad (3)$$

Exercise 6.
Verify that the above calculations (equations (2) *and* (3)*) is a sum of n! terms, each being $+/-$ a product of n elements.*

Theorem
For a general $n \times n$ matrix A, Method 1 corresponds to computing a sum of $n!$ terms, where each term is $+/-$ a product of n elements.

Exercise 7:

- *Using the above theorem, show that Method 1 has worst-case complexity* $\mathcal{O}((n!)n)$.
- *Show that this implies that the worst case complexity is* $\mathcal{O}((n+1)!)$.
- *Show that the worst case complexity of Method 1 is* $\Theta((n!)n)$.

Metode 2:
Bring A to reduced echelon form (not necessarily *reduced*) by means of Gaussian elimination using only the following two elementary row operations

R1: "$\mathbf{r}_i \leftrightarrow \mathbf{r}_j$" (for $i \neq j$). Interchange rows.

R2: "$\mathbf{r}_i + c\mathbf{r}_j \rightarrow \mathbf{r}_i$" (for $i \neq j$). Add a multiple of one row to another row.

Let s the number of operations of type R1. Let B be the row echelon form of A. Then:

$$\det A = (-1)^s b_{11} b_{22} \cdots b_{nn}. \tag{4}$$

Exercise 8:
In this exercise, we estimate the complexity of Metode 2.

- *Show that the worst-case complexity of the Gaussian elimination in Method 2 is* $\mathcal{O}(n^3)$.
- *We can assume that the Gaussian elimination uses at most n row interchanges (operations of type R1). Why?*
- *Having completed the Gaussian elimination, we can compute the right-hand side of* (4). *Show that this computation uses at most* $\mathcal{O}(n)$ *operations.*
- *Show that the worst-case complexity of Method 2 is* $\mathcal{O}(n^3)$.

We now compare Method 1 and Method 2. We ignore the unknown constants hidden in the expressions $\mathcal{O}((n!)n)$ and $\mathcal{O}(n^3)$. More precisely, let us say that Method 1 uses $(n!)n$ operations and that Method 2 requires n^3 operations.

Exercise 9:
If a computer performs $1000000000 = 10^9$ *operations a second, then how long does it take to compute the determinant of an* $n \times n$ *matrix using Methods 1 and 2 for each of the following values of n.*

- $n = 20$*?*
- $n = 21$*?*
- $n = 22$*?*
- *Why does it make sense to ignore the constants in the expressions* $\mathcal{O}((n!)n)$ *and* $\mathcal{O}(n^3)$*?*

Remark:
In the analysis of the complexities of Methods 1 and 2, we have merely considered the number of multiplications and additions in $\mathbb{R}$. We have ignored that the numbers during the computation can explode when a string of numbers are multiplied. If this was to be considered, things become more involved. It is possible to show that Method 2 still has reasonable complexity, whereas the complexity of Method 1 becomes even worse.