Introduction to Mathematical Software 7^{th} Exercise Sheet

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Mathematics
PD Dr. Ulf Lorenz
Christian Brandenburg

winter term 2009/2010
18/01/2010

Exercise 7.1 Maple Worksheet

Obtain the Maple worksheet `IMS_2009_2010_5.mw` from the course webpage (it is located under the *Lecture Slides* tab. Open the worksheet in Maple and work through it line by line to understand what it does. Consider the Maple help if you are unsure what a particular command is good for or how it works.

Exercise 6.2 The O-Notation and Complexity of Algorithms

In the analysis of algorithms, one very important issue is their efficiency for arbitrarily large inputs, i.e. we are interested in their *order of growth* and hence their *complexity*. One might argue that with the speed-up of modern computers the complexity of algorithms becomes less important. However, experience shows that the problems people try to solve grow much faster than the available computing power.

O-Notation

In this exercise we introduce the so called *O-notation* that is used to compare the efficiency of algorithms, in particular for arbitrarily large inputs.

Definition 1 *Let $g : \mathbb{N} \to \mathbb{N}$ be a function. The set $O(g)$ is defined as*

$$O(g) = \{f : \mathbb{N} \to \mathbb{N} | \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n)\}.$$

If $f \in O(g)$ (alternatively, we write $f = O(g)$), we say that "f is Big O of g", or that g is an asymptotic upper bound *for f.*

This means that f does not grow faster than g for $n \to \infty$. Note in particular that f is asymptotically positive, i.e. $f(n) > 0$ for $n \geq n_0$.

Definition 2 *Let $g : \mathbb{N} \to \mathbb{N}$ be a function. The set $\Omega(g)$ is defined as*

$$\Omega(g) = \{f : \mathbb{N} \to \mathbb{N} | \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : 0 \leq cg(n) \leq f(n)\}.$$

For $f \in \Omega(g)$, g is called an asymptotic lower bound.

This means that f grows at least as fast as g for $n \to \infty$.

a) Show that $f = O(g)$ if and only if $g = \Omega(f)$.

Definition 3 *Let $g : \mathbb{N} \to \mathbb{N}$ be a function. The set $\Theta(g)$ is defined as*

$$\Theta(g) = \{f : \mathbb{N} \to \mathbb{N} | \exists c_1, c_2 > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}.$$

For $f \in \Theta(g)$, f and g are called asymptotically equivalent.

Prove the following propositions:

b) If $c > 0$ is constant, then $cf = \Theta(f)$.

c) $f = \Theta(g)$ if and only if $g = \Theta(f)$.

d) If $h(n) = O(g)$ and $f(n) \leq h(n)$ for n large enough, then $f = O(g)$.

e) Let $h_1 = \Omega(g)$ and $h_2 = O(g)$. If $h_1(n) \leq f(n) \leq h_2(n)$ for n large enough, then $f = \Theta(g)$.

The way in which a function $f : \mathbb{N} \to \mathbb{N}$ behaves asymptotically w.r.t a function $g : \mathbb{N} \to \mathbb{N}$ can often be found by considering the limit of $n \to \infty$:

f) If $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = L$, $0 \leq L < \infty$, then $f = O(g)$.

g) If $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = L$, $0 < L \leq \infty$, then $f = \Omega(g)$.

h) If $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = L$, $0 < L < \infty$, then $f = \Theta(g)$.

Complexity of Algorithms

To characterize the complexity of an algorithm, some generic complexity classes as functions of n are used. These include powers of n (including n^0), square roots, exponentials, logarithms and factorials and multiplicative combinations of these.

a) Sort the following orders of complexity from lowest to highest:

$$O(n^2),\ O(3^n),\ O(2^n),\ O(n^2 \lg n),\ O(1),\ O(n \lg n),\ O(n^3),\ O(n!),\ O(\lg n),\ O(n).$$

The complexities of some common situations are given in the following table:

Complexity	Example
$O(1)$	Fetching the first element from a set of data; fetching a given element from an array
$O(\lg n)$	Splitting a set of data in half recursively
$O(n)$	Traversing a set of data
$O(n \lg n)$	Splitting a set of data in half recursively and traversing each half
$O(n^2)$	Matrix-vector multiplication
$O(n^3)$	Matrix inversion
$O(n!)$	Creating all possible permutations of a set of data

b) What is the time complexity of the following code fragments?

- ```
  int p = 1;
  int s = 0;
  for (int i = 0; i < n; i++)
    p = p * a[i];
  for( int i = 0; i < n; i++)
    s = s + a[i];
  ```
- ```
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      printf("%d * %d = %d\n",i, j, i*j);
  ```
- ```
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
      c[i][j] = 0;
      for (int k = 0; k < n; k++)
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
    }
  ```

c) Suppose you have two algorithms to solve the same problem. The first runs in time $T_1(n) = 400n$, the second runs in time $T_2(n) = n^2$. What are the complexities of these algorithms? For which values of n might we consider to use the algorithm with the higher complexity?
  ```