Design and Analysis of Algorithms (Fall 2009)

1. Practical arrangements
   - www.cs.oku.fi/~kilpelai/ASA09
2. Some motivation
3. Overview of the course
4. Computational problems and solutions
5. Algorithm correctness
6. Algorithm notation and Time complexity

Some motivation

- Position within Computer Science

R.E. Tarjan (CACM, March 1987):

The quest for efficiency in computational methods yields not only fast algorithms, but also insights that lead to elegant, simple, and general problem-solving methods.

Computational problems

Realize a specified mapping from inputs to outputs:

Set of inputs (= problem instances, ongelman tapaukset) is often unlimited

Examples:

Multiplication $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}; f(x, y) = x \ast y$

Halting problem

of programs $P$ with inputs $x$:

$$f(P, x) = \begin{cases} 
  \text{true} & \text{if } P(x) \text{ terminates} \\
  \text{false} & \text{otherwise}
\end{cases}$$

Sorting ("lajittelu" eli järjestäminen)

of sequences of an ordered type $T$:

$$f : T^* \to T^*; f(a_1, \ldots, a_n) = (b_1, \ldots, b_n)$$

where $(b_1, \ldots, b_n)$ is a permutation of $(a_1, \ldots, a_n)$ with $b_1 \leq b_2 \leq \cdots \leq b_n$
Solution of a computational problem

an algorithm, a method/procedure A s.t.
\[ A(x) = f(x) \]
for all problem instances \( x \)

- "mechanical" (deterministic)
- terminates in finite time

(Extensions: non-deterministic, randomized, ... alghs)

Example Insertion sort (lisayslajittelul)

procedure insSort (modifies A:
array [1..n] of T)
1. for \( j := 2 \) to \( n \) do
2. \( x := A[j]; \)
3. \( i := j - 1; \)
4. while \( i > 0 \) and \( A[i] > x \) do
5. \( A[i + 1] := A[i]; \)
6. \( i := i - 1; \)
7. \( A[i + 1] := x; \)

How to see algorithms correct?

Main method: Mathematical induction

If (i) \( P(k) \) holds for some \( k \in \mathbb{N} \), and
(ii) \( P(k), P(k + 1), \ldots, P(n) \Rightarrow P(n + 1), \)
then \( P(n) \) holds for all \( n \geq k \)

Example (Lemma 1.1)
Let \( S(n) = \sum_{i=1}^{n} i \)
Then \( S(n) = n(n + 1)/2 \) for all \( n \in \mathbb{N} \)

Proof:

Example (Lemma 1.2)
The height of a binary tree with \( n \) leaves is at least \( \log_2 n \)

Proof \( P(n) : h \geq \log n \iff n \leq 2^h \)
Induction on the height \( h \)
insSort(A[1..n]) is correct for all \( n \geq 1 \)

**Proof (sketch):**
(i) \( P(1) \): \( A[1..1] \) is OK
(ii) \( n \geq 2 \): Assume \( P(n-1) \)

\[
\text{insSort}(A[1..n]) \sim \text{insSort}(A[1..(n-1)]), \text{ followed by}
\]

\[
x := A[n]; i := n - 1;
\textbf{while} \; i > 0 \land A[i] > x \; \textbf{do}
A[i + 1] := A[i]; i := i - 1;
A[i + 1] := x;
\]

**Algorithm notation**

Basically, any unambiguous description using realistic constant-time operations will do, like

- natural language (OK, if not too vague)
- flow-chart diagrams (OK for simple processes)
- programming languages (OK for implementation; otherwise often too specific/detailed)
- random-access and Turing machines (later, as formal basis; otherwise often too detailed)

We'll mainly use **pseudocode**, for example as follows:

**Basic constructs and their time**

**Assignment statement** \( \text{Var} := \text{Expr}; \)

Time \( \approx \) \( \text{Time(Expr)} \), often constant

(For example, for

\[
A := B[1..n];
\]

distinguish renaming/address assignment and full copying)

**Conditionals**

if Cond then Statements
[elseif Cond then Statements ...]
[else Statements];

Time \( = \) time to evaluate \( \text{Conds} \) up to the first true one, and the Statements of the selected branch (if any)

**Iterations**

\[
\textbf{while} \; \text{Cond} \; \textbf{do} \; \text{Body}; \; // \; \text{N} \geq 0 \; \text{iterations}
\]

\[
\text{Time} = N \times \text{Time(\text{Body})} + (N - 1) \times \text{Time(\text{Cond})}
\]

\[
\textbf{repeat} \; \text{Body} \; \textbf{until} \; \text{Cond}; \; // \; \text{N} \geq 1 \; \text{iterations}
\]

\[
\text{Time} = N \times \left( \text{Time(\text{Body})} + \text{Time(\text{Cond})} \right)
\]

\[
\textbf{for} \; \text{var} := a \; \textbf{to} \; b \; \textbf{do} \; \text{Body(var)};
\]

Time \( \approx \sum_{i=a}^{b} \text{Time(\text{Body(i)})} \) (+ constant)

with downto,

Time \( \approx \sum_{i=a}^{b} \text{Time(\text{Body(i)})} \) (+ constant)
Procedures and Functions

procedure procName(Params) [returns T]
    Body;

Declaration, no execution time

Invocation: procName(ArgExprs);

Time: time to evaluate arguments, and the Body using them

return Expr;  // immediately to the call

Time: time to evaluate Expr

Input and output

read var;

Time ~ the length of the value; often constant, with large ints $n \sim \log n$

write Expr;

Time: time to evaluate Expr

Also higher-level expressions can be used, as long as we recognize their time.
For example:

$x \leftarrow \max A[1..n];  // in time $Θ(n)$

"Sort $A[1..n]$ in increasing order";
doable in time $Θ(n \log n)$
(unless studying some specific sorting algorithm, as often on this course)

Time complexity

How does the execution time, or # of basic ops, depend from input size $n$?

$T(n) =$  

Rather detailed analysis of insSort($A[1..n]$)
(Less detailed asymptotic analyses in the sequel)

1. for $j := 2$ to $n$ do
2.   $x := A[j];$
3.   $i := j - 1;$
4.   while $i > 0$ and $A[i] > x$ do
6.      $i := i - 1;$
7.      $A[i+1] := x;$

\[ T(n) \leq c_1 n + (n - 1)(c_{2,3} + c_7) + \sum_{j=2}^{n} [c_4 j + c_{5,6}(j - 1)] \]
\[ = (c_{1,3} + c_7)n - (c_{2,3} + c_7) + \]
\[ c_4 \sum_{j=2}^{n-1} j + c_{5,6} \sum_{j=1}^{n-1} j \]
\[ = (c_{1,3} + c_7)n - (c_{2,3} + c_7) + \]
\[ c_4 \frac{n(n + 1)}{2} - 1 + (c_{5,6} \frac{n(n-1)n}{2}) \]
\[ = \frac{c_{4,6} n^2}{2} + \frac{c_{1,3} + c_4 - c_{5,6} + c_7}{2} + (c_{2,4} + c_7) \]
\[ = an^2 + bn + c \]