1 Introduction

Summer Term 2011

Jan-Georg Smaus
This lecture is entitled “Computer Science Theory I”, in German “Informatik Theorie I”, in short “Theory I”.

The course was given in 2010 by Robert Elsässer and in previous years by Georg Lausen, Thomas Ottmann, Peter Thiemann, Fang Wei and others.

This year the lecturer is Jan-Georg Smaus, the assistant is Alexander Schimpf and the tutor is Ahmed Mahdi.
http://www.informatik.uni-freiburg.de/~ki/teaching/ss11/theoryI/
Prerequisite of Theory I

- Programming language, such as C++
- Basic knowledge on data structures and algorithms, mathematics
- Sufficient knowledge on English for reading research papers
Course Goal

- Get an in-depth knowledge on
  - Data structures and algorithms
  - Programming languages, logic, and software engineering
  - Database systems

- Improve your problem solving ability by doing the exercises

- Practicing skills in conducting research work:
  - Reading papers efficiently
  - Writing reviews and surveys
Course load

- 11 exercises
  - exercises posted one week ahead
  - exercise classes take place on each Wednesday (one hour)
    - hand in your solution before the exercise class starts!
- 5-6 Reading assignments (submit reviews on the assigned papers)
- Final exam
Topics

- Data structures and algorithms
  - Trees, balanced trees
  - Hashing, dynamic tables, randomization
  - Text search
- Logic, programming languages and software engineering
  - Logic and relations
  - Abstract datatypes
- Database systems
Algorithm

- An algorithm is a sequence of computational steps that transform the input into the output, e.g. sorting.
- In computer science we distinguish:
  - Input/output
  - Algorithms
  - Programs
  - Processes
Properties of Algorithms

- An algorithm should meet the following requirements:
  - Effectiveness
  - Determinacy
  - Finiteness
  - Termination
  - Generality
  - Precision

- Correctness of algorithms is a concern in this course.
There are some problems for which no efficient solution is known.

Why are NP-complete problems interesting?

- although no efficient algorithm for an NP-complete problem has ever been found, it is unknown whether or not efficient algorithms exist for NP-complete problems.

- the set of NP-complete problems has the remarkable property that if an efficient algorithm exists for any one of them, then efficient algorithms exist for all of them.

- if a problem is proven NP-complete, one can instead spend your time developing an efficient algorithm that gives a good, but not the best possible, solution.
Analysis of Algorithms

- Approaches:
  - theoretical analysis
  - empirical analysis

- Issues:
  - correctness
  - time efficiency
  - space efficiency
  - optimality
For most algorithms, the time and space requirements depend on the size and form of the input:

- **Worst case**: $C_{\text{worst}}(n) - \text{maximum over inputs of size } n$

- **Best case**: $C_{\text{best}}(n) - \text{minimum over inputs of size } n$

- **Average case**: $C_{\text{avg}}(n) - \text{“average” over inputs of size } n$
  - Number of times the basic operation will be executed on typical input
  - NOT the average of worst and best case
  - Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs. So, $\text{avg} = \text{expected under uniform distribution.}$
**Example: Sequential Search**

**Algorithm**  

$\text{SequentialSearch}(A[0..n - 1], K)$

//Searches for a given value in a given array by sequential search  
//Input: An array $A[0..n - 1]$ and a search key $K$  
//Output: The index of the first element of $A$ that matches $K$  
// or $-1$ if there are no matching elements

$i \leftarrow 0$

while $i < n$ and $A[i] \neq K$ do

$i \leftarrow i + 1$

if $i < n$ return $i$

else return $-1$

- Worst case
- Best case
- Average case
Type of Formula for Basic Operations’ Count

- Exact formula
  e.g., \( C(n) = \frac{n(n-1)}{2} \)

- Formula indicating order of growth with specific multiplicative constant
  e.g., \( C(n) \approx 0.5 \, n^2 \)

- Formula indicating order of growth with unknown multiplicative constant
  e.g., \( C(n) \approx c\, n^2 \)
Order of Growth

- Most important: Order of growth within a constant multiple as \( n \to \infty \)

- Example:
  - How much faster will algorithm run on computer that is twice as fast?
  - How much longer does it take to solve problem of double input size?
<table>
<thead>
<tr>
<th>( n )</th>
<th>( \log_2 n )</th>
<th>( n )</th>
<th>( n \log_2 n )</th>
<th>( n^2 )</th>
<th>( n^3 )</th>
<th>( 2^n )</th>
<th>( n! )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.3</td>
<td>( 10^1 )</td>
<td>( 3.3 \cdot 10^1 )</td>
<td>( 10^2 )</td>
<td>( 10^3 )</td>
<td>( 10^3 )</td>
<td>( 3.6 \cdot 10^6 )</td>
</tr>
<tr>
<td>( 10^2 )</td>
<td>6.6</td>
<td>( 10^2 )</td>
<td>( 6.6 \cdot 10^2 )</td>
<td>( 10^4 )</td>
<td>( 10^6 )</td>
<td>( 1.3 \cdot 10^{30} )</td>
<td>( 9.3 \cdot 10^{157} )</td>
</tr>
<tr>
<td>( 10^3 )</td>
<td>10</td>
<td>( 10^3 )</td>
<td>( 1.0 \cdot 10^4 )</td>
<td>( 10^6 )</td>
<td>( 10^9 )</td>
<td>( 10^9 )</td>
<td></td>
</tr>
<tr>
<td>( 10^4 )</td>
<td>13</td>
<td>( 10^4 )</td>
<td>( 1.3 \cdot 10^5 )</td>
<td>( 10^8 )</td>
<td>( 10^{12} )</td>
<td>( 10^{12} )</td>
<td></td>
</tr>
<tr>
<td>( 10^5 )</td>
<td>17</td>
<td>( 10^5 )</td>
<td>( 1.7 \cdot 10^6 )</td>
<td>( 10^{10} )</td>
<td>( 10^{15} )</td>
<td>( 10^{15} )</td>
<td></td>
</tr>
<tr>
<td>( 10^6 )</td>
<td>20</td>
<td>( 10^6 )</td>
<td>( 2.0 \cdot 10^7 )</td>
<td>( 10^{12} )</td>
<td>( 10^{18} )</td>
<td>( 10^{18} )</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.1** Values (some approximate) of several functions important for analysis of algorithms
Asymptotic Order of Growth

A way of comparing functions that ignores constant factors and small input sizes

- \( O(g(n)) \): class of functions \( f(n) \) that grow no faster than \( g(n) \)

- \( \Theta(g(n)) \): class of functions \( f(n) \) that grow at same rate as \( g(n) \)

- \( \Omega(g(n)) \): class of functions \( f(n) \) that grow at least as fast as \( g(n) \)
Useful summation and formula rules

\[ \sum_{l \leq i \leq n} 1 = 1 + 1 + \ldots + 1 = n - l + 1 \]

In particular, \( \sum_{1 \leq i \leq n} 1 = n - 1 + 1 = n \in \Theta(n) \)

\[ \sum_{1 \leq i \leq n} i = 1 + 2 + \ldots + n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2) \]

\[ \sum_{1 \leq i \leq n} i^2 = 1^2 + 2^2 + \ldots + n^2 = n(n+1)(2n+1)/6 \approx n^3/3 \in \Theta(n^3) \]

\[ \sum_{0 \leq i \leq n} a^i = 1 + a + \ldots + a^n = (a^{n+1} - 1)/(a - 1) \text{ for any } a \neq 1 \]

In particular, \( \sum_{0 \leq i \leq n} 2^i = 2^0 + 2^1 + \ldots + 2^n = 2^{n+1} - 1 \in \Theta(2^n) \)

\[ \sum (a_i \pm b_i) = \sum a_i \pm \sum b_i \quad \sum c a_i = c \sum a_i \]

\[ \sum_{l \leq i \leq u} a_i = \sum_{l \leq i \leq m} a_i + \sum_{m+1 \leq i \leq u} a_i \]