Names of this Lecture and its People

- 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.

Webpage

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.

Hard Problems

- 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**: $C_{\text{worst}}(n)$ – maximum over inputs of size $n$
- **Best case**: $C_{\text{best}}(n)$ – minimum over inputs of size $n$
- **Average case**: $C_{\text{avg}}(n)$ – “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, avg is expected under uniform distribution.

### 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., $C(n) = 0.5n^2$

- **Formula indicating order of growth with unknown multiplicative constant**
  
  e.g., $C(n) \approx cn^2$

### 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?

### Values of Some Functions for Various $n$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\log_2 n$</th>
<th>$\log n$</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</td>
<td>100</td>
<td>1000</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>$10^2$</td>
<td>10</td>
<td>$10^2$</td>
<td>$10^4$</td>
<td>$10^6$</td>
<td>$10^6$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>13</td>
<td>$10^3$</td>
<td>$10^6$</td>
<td>$10^9$</td>
<td>$10^9$</td>
<td>$10^6$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>17</td>
<td>$10^4$</td>
<td>$10^8$</td>
<td>$10^{12}$</td>
<td>$10^{12}$</td>
<td>$10^8$</td>
</tr>
<tr>
<td>$10^5$</td>
<td>20</td>
<td>$10^5$</td>
<td>$10^{10}$</td>
<td>$10^{18}$</td>
<td>$10^{18}$</td>
<td>$10^{10}$</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** $g(n)$
- $\Omega(g(n))$: class of functions $f(n)$ that grow **at same rate** as $g(n)$
- $\Theta(g(n))$: class of functions $f(n)$ that grow **at least as fast** as $g(n)$
Useful summation and formula rules

\[ \sum_{i=1}^{n} 1 = 1 + 1 + \ldots + 1 = n \cdot 1 + 1 \]
In particular, \( \sum_{i=1}^{n} 1 = n \cdot 1 + 1 = n \in \Theta(n) \)

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

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

In particular, \( \sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6} \in \Theta(n^3) \)

\[ \sum_{i=1}^{n} a_i = 1 + a + \ldots + a^n = (a^{n+1} - 1)/(a - 1) \text{ for any } a \neq 1 \]
In particular, \( \sum_{i=1}^{n} 2^2 + 2^3 + \ldots + 2^n = 2^{n+1} - 1 \in \Theta(2^n) \)

\[ \sum(a_i \pm b_i) = \sum a_i \pm \sum b_i \quad \sum c a_i = c \sum a_i \]
\[ \sum_{i=1}^{n} a_i = \sum_{i=1}^{m} a_i + \sum_{m+1}^{u} a_i \]