Syllabus

Syllabus:
• An asymptotical notation
• (Binary trees,) AVL trees, Red-Black trees
• B-trees
• Hashing
• Graph alg: searching, topological sorting, strongly connected components
• Minimal spaning tree (d.s. Union-Find)
• Divide et Impera method
• Sorting: a lower bound on the complexity of sorting, average case of Quicksort, randomization of Quicksort, linear sorting alg.
• Algebraic alg. (LUP decomposition)
• Literature

• Organization
  – Lecture
  – Exercises
Comparing algorithms

- Measures:
  - A time complexity, in (elementary) steps
  - A space complexity, in words/cells
  - A communication complexity, in packets/bytes
  - (in practice: money ~ programmer/human time)

- How it is measured:
  - A worst case, an average case (wrt a probability distribution)
  - Usually an approximation: an upper bound

- Using functions depending on a size of input data
  - We abstract from particular data to a data size, |D|
  - We compare functions
Size of data

• Q: How to measure a size of (input) data?
• Formally: a number of bits of data
• Ex: Inputs are natural numbers $a_1, \ldots, a_n \in \mathbb{N}$, a size $D$ of input data is $|D| = \sum_{i=1}^{n} \lfloor \log_2 a_i \rfloor$
• A time complexity: a function $f$: $\mathbb{N} \rightarrow \mathbb{N}$, such that $f(|D|)$ gives a number of algorithm steps depending on data of the size $|D|$.
• Intuitively: An asymptotical behaviour: an exact graph of a function $f$ does not matter (ignoring additive and multiplicative constants), a class of $f$ matters (linear, quadratic, exponential)
A step of an algorithm

- In theory: Based on an abstract machine: Random Access Machine (RAM), Turing m.
  - Informally: an algorithm step = an operation executable in a constant time (independent of data size)
- RAM, operations:
  - Arithmetical: +, -, *, mod, <<, && …
  - A comparison of two numbers
  - An assignment of basic data types (not for arrays)
    - Numbers have a fixed maximal size
- Ex: sorting of $n$ numbers: $|D| = n$
- (Counter)Ex: a test for a zero vector
Why to measure a time complexity

- Why sometimes a faster machine doesn't help
- Time of $f(n)$ for data of size $n$, $10^6$ ops per second

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>20µs</td>
<td>40µs</td>
<td>60µs</td>
<td>80µs</td>
<td>100µs</td>
<td>500µs</td>
<td>1ms</td>
</tr>
<tr>
<td>$n \log n$</td>
<td>86µs</td>
<td>0.2ms</td>
<td>0.35ms</td>
<td>0.5ms</td>
<td>0.7ms</td>
<td>4.5ms</td>
<td>10ms</td>
</tr>
<tr>
<td>$n^2$</td>
<td>0.4ms</td>
<td>1.6ms</td>
<td>3.6ms</td>
<td>6.4ms</td>
<td>10ms</td>
<td>0.25s</td>
<td>1s</td>
</tr>
<tr>
<td>$n^3$</td>
<td>8ms</td>
<td>64ms</td>
<td>0.22s</td>
<td>0.5s</td>
<td>1s</td>
<td>125s</td>
<td>17min</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1s</td>
<td>11.7days</td>
<td>36ky</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n!$</td>
<td>77ky</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why to measure a time complexity 2

- A difference between polynomial and slower algs.
- How a speed-up of a computation enables to increase a size of a „workable“ data; the current size is $x$

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>speed-up</th>
<th>10 times</th>
<th>100 times</th>
<th>1000 times</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td>10x</td>
<td>100x</td>
<td>1000x</td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>$10^n$</td>
<td>$100^n$</td>
<td>$1000^n$</td>
<td></td>
</tr>
<tr>
<td>$n \log n$</td>
<td>$7.02^n$</td>
<td>$53.56^n$</td>
<td>$431.5^n$</td>
<td></td>
</tr>
<tr>
<td>$n^2$</td>
<td>$3.16^n$</td>
<td>$10^n$</td>
<td>$31.62^n$</td>
<td></td>
</tr>
<tr>
<td>$n^3$</td>
<td>$2.15^n$</td>
<td>$4.64^n$</td>
<td>$10^n$</td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td>$x+3$</td>
<td>$x+6$</td>
<td>$x+9$</td>
<td></td>
</tr>
</tbody>
</table>
Asymptotical complexity

• measures a behaviour of the algorithm on „big“ data
  – Ignores a finite number of exceptions

• supresses additive and multiplicative constants
  – Abstracts from a processor, a language, (the Moore law)

• classifies algs to categories: linear, quadratic, logarithmic, exponential, constant …
  – Compares functions
Asymptotical („Big“) O notation

• f(n) is asymptotically less or equal g(n), notation \( f(n) \in O(g(n)) \), „big O“
  iff \( \exists c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n) \)

• f(n) is asymptotically greater or equal g(n), notation \( f(n) \in \Omega(g(n)) \), „big Omega“
  iff \( \exists c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n) \)

• f(n) is asymptotically equal g(n), notation \( f(n) \in \Theta(g(n)) \), „big Theta“
  iff \( \exists c_1, c_2 > 0 \exists n_0 \forall n \geq n_0 : 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \)
O-notation, def. II

• $f(n)$ is *asymptotically strictly less* than $g(n)$, notation $f(n) \in o(g(n))$, „small o“
  \[ \forall c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n) \]

• $f(n)$ is *asymptotically strictly greater* than $g(n)$, notation $f(n) \in \omega(g(n))$, „small omega“
  \[ \forall c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n) \]

• Examples of classes: $O(1)$, $\log \log n$, $\log n$, $n$, $n \log n$, $n^2$, $n^3$, $2^n$, $2^2n$, $n!$, $n^n$, $2^{(2^n)}$, ...

• Some functions are incomparable
Exercises

• Notation $f \leq O(g)$ is sometimes used
• To prove: $\max(f,g) \in \Theta(f+g)$
• To prove: if $c,d>0$, $g(n)=c.f(n)+d$ then $g \in O(f)$
• Ex.: if $f \in O(h)$, $g \in O(h)$ then $(f+g) \in O(h)$
  – Application: A bound to a sequence of commands
• Compare $n+100$ to $n^2$ ; $2^{10}n$ to $n^2$
  – Simple algorithms (with a low overhead) are sometimes better for small data