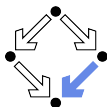
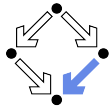


Basics of Complexity

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
<http://www.risc.jku.at>





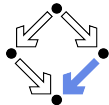
1. Complexity of Computations

2. Asymptotic Complexity

3. Working with Asymptotic Complexity

4. Complexity Classes

Complexity of Computations

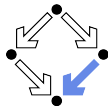


We want to determine the resource consumption of computations.

- Determine the amount of resources consumed by a computation:
 - Time
 - Space (memory)
- Determine the resource consumption for classes of inputs:
 - The **maximum** complexity for all inputs of the class.
 - The **average** complexity for these inputs.

We are going to make these notions precise.

Resource Consumption

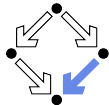


Turing machine M with input alphabet Σ that halts for every input.

- **Input set** $I = \Sigma^*$:
 - The set of input words.
- **Input size** $|\cdot| : I \rightarrow \mathbb{N}$
 - $|i|$: the length of input i .
- **Time consumption** $t : I \rightarrow \mathbb{N}$:
 - $t(i)$: the number of moves that M makes for input i until it halts.
- **Space consumption** $s : I \rightarrow \mathbb{N}$:
 - $s(i)$: the largest distance from the beginning of the tape that the tape head of M reaches for input i until M halts.

For any computational model, I , $|i|$, $t(i)$ and $s(i)$ may be defined.

Worst-case Complexity



Computational model with I , $|i|$, $t(i)$ and $s(i)$ defined.

- Worst-case time complexity $T : \mathbb{N} \rightarrow \mathbb{N}$

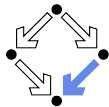
$$T(n) := \max\{t(i) \mid i \in I \wedge |i| = n\}$$

- Worst-case space complexity $S : \mathbb{N} \rightarrow \mathbb{N}$

$$S(n) := \max\{s(i) \mid i \in I \wedge |i| = n\}$$

The maximum amount of resources consumed for any input of size n by computations in a given model.

Average Complexity



■ Input distribution *Input*:

- Family of (discrete) random variables $Input_n$ that describe the distribution of inputs of each size n in I
- determined by probability function $p_I^n : I \rightarrow [0, 1]$

$p_I^n(i)$: probability that, among all inputs of size n , input i occurs.

■ Average time/space complexity $\bar{T} : \mathbb{N} \rightarrow \mathbb{N}$ and $\bar{S} : \mathbb{N} \rightarrow \mathbb{N}$

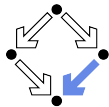
$$\bar{T}(n) := E[Time_n]$$

$$\bar{S}(n) := E[Space_n]$$

- Expected values of random variables $Time_n$ and $Space_n$
- determined by probability functions $p_T^n : \mathbb{N} \rightarrow [0, 1]$ and $p_S^n : \mathbb{N} \rightarrow [0, 1]$
 $p_T^n(t)/p_S^n(s)$: probabilities that time t /space s is consumed for input of size n assuming that inputs of size n are distributed according to $Input_n$.

The average amount of resources consumed for inputs of size n (for a given distribution of inputs) by computations in a given model.

Example



Given non-empty integer array a of size n , find minimum index j such that $a[j] = \max\{a[i] \mid 0 \leq i < n\}$.

$j := 0; m := a[j]; i := 1$	1
while $i < \text{length}(a)$	n
if $a[i] > m$ then	$n - 1$
$j := i; m := a[j]$	$N \leq n - 1$
$i := i + 1$	$n - 1$

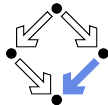
- Time: the number of lines executed.

$$T(n) = 1 + n + (n - 1) + (n - 1) + (n - 1) = 4n - 2$$

- Space: the number of variables used (including elements of a).

$$S(n) = \bar{S}(n) = n + 3$$

We are going to analyze the average time complexity $\bar{T}(n)$.



Average Time Complexity

Assume a holds n distinct values $\{1, \dots, n\}$.

- Assume all $n!$ permutations of a are equally probable.

$$p_i^n(i) := 1/n!$$

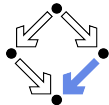
- Quantity N becomes random variable.

The number of times the corresponding line of the algorithm is executed for each permutation of a .

- We are interested in the expected value $E[N]$.
- Average time complexity $\bar{T}(n)$:

$$\bar{T}(n) = 1 + n + (n-1) + E[N] + (n-1) = 3n - 1 + E[N]$$

Our goal is to determine the expected value $E[N]$.



Average Time Complexity (Contd)

- p_{nk} : probability that $N = k$ for array of size n .

$$p_{n0} + p_{n1} + p_{n2} + \dots + p_{n,n-1} = \sum_{k=0}^{n-1} p_{nk} = 1$$

- $p_{nk} = 0$ for $k \geq n$:

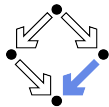
$$p_{n0} + p_{n1} + p_{n2} + \dots = \sum_k p_{nk} = 1$$

- $E[N]$ is sum of products of probability of $N = k$ and value k :

$$E[N] = p_{n0} \cdot 0 + p_{n1} \cdot 1 + p_{n2} \cdot 2 + \dots = \sum_k p_{nk} \cdot k$$

Our goal is to determine the value of sum $\sum_k p_{nk} \cdot k$.

Average Time Complexity (Contd)



We apply the technique of “generating functions”.

- $G_n(z)$: power series with coefficients p_{n0}, p_{n1}, \dots

$$G_n(z) := p_{n0} \cdot z^0 + p_{n1} \cdot z^1 + p_{n2} \cdot z^2 + \dots = \sum_k p_{nk} \cdot z^k$$

- $G'_n(z)$: derivative of $G_n(z)$

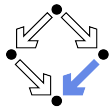
$$G'_n(z) = p_{n0} \cdot 0 \cdot z^{-1} + p_{n1} \cdot 1 \cdot z^0 + p_{n2} \cdot 2 \cdot z^1 + \dots = \sum_k p_{nk} \cdot k \cdot z^{k-1}$$

- $G'_n(1)$:

$$G'_n(1) = p_{n0} \cdot 0 + p_{n1} \cdot 1 + p_{n2} \cdot 2 + \dots = \sum_k p_{nk} \cdot k$$

Our goal is to determine $G'_n(1)$.

Average Time Complexity (Contd)



We derive a recurrence relation for $G'_n(1)$.

- $n = 1$: $p_{10} = 1$ and $p_{1k} = 0$ for all $k \geq 1$

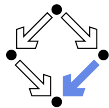
$$G'_1(1) = 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 2 + \dots = 0$$

- $n > 1$: if the loop has already found the maximum of the first $n - 1$ array elements, the last iteration
 - will either increment N (if the last element is the largest one)
 - Probability $1/n$.
 - N becomes k for n , if N was $k - 1$ for $n - 1$.
 - or will leave N as it is (if the last element is not the largest one).
 - Probability $(n - 1)/n$.
 - N becomes k for n , if N was k for $n - 1$.

$$p_{nk} = \frac{1}{n} \cdot p_{n-1, k-1} + \frac{n-1}{n} \cdot p_{n-1, k}$$

Our goal is to determine $G'_n(1)$ for $n > 1$.

Average Time Complexity (Contd)



- Determine $G_n(z)$ from p_{nk} :

$$p_{nk} = \frac{1}{n} \cdot p_{n-1,k-1} + \frac{n-1}{n} \cdot p_{n-1,k}$$

$$G_n(z) = \frac{1}{n} \cdot z \cdot G_{n-1}(z) + \frac{n-1}{n} \cdot G_{n-1}(z) = \frac{z+n-1}{n} \cdot G_{n-1}(z)$$

- Compute $G'_n(z)$ by derivation of $G_n(z)$:

$$G'_n(z) = \frac{1}{n} \cdot G_{n-1}(z) + \frac{z+n-1}{n} \cdot G'_{n-1}(z)$$

- Compute $G'_n(1)$:

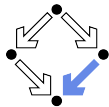
$$G'_n(1) = \frac{1}{n} \cdot G_{n-1}(z) + \frac{z+n-1}{n} \cdot G'_{n-1}(z)$$

$$\stackrel{(*)}{=} \frac{1}{n} \cdot 1 + \frac{1+n-1}{n} \cdot G'_{n-1}(1)$$

$$= \frac{1}{n} + G'_{n-1}(1)$$

$$(*) \quad G_n(1) = p_{n0} + p_{n1} + p_{n2} + \dots = \sum_k p_{nk} = 1$$

Average Time Complexity (Contd)



- Recurrence relation for $G'_n(1)$:

$$G'_1(1) = 0$$

$$G'_n(1) = \frac{1}{n} + G'_{n-1}(1), \text{ if } n > 0$$

- Solution of $G'_n(1)$:

$$G'_n(1) = \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \sum_{k=2}^n \frac{1}{k} = H_n - 1$$

- $H(n) = \sum_{k=1}^n \frac{1}{k}$: n -th harmonic number

- $H(n) = \ln n + \gamma + \varepsilon_n$, $\gamma \approx 0.577$, $0 < \varepsilon_n < 1/(2n)$.

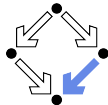
- Solution of $E[n]$:

$$E[N] = \ln n + \gamma + \varepsilon_n - 1$$

- Average time complexity $\bar{T}(n)$:

$$\bar{T}(n) = 3n - 1 + E[N] = 3n + \ln n + \varepsilon_n + \gamma - 2$$

Analysis of average complexity is more difficult than that of the worst-case.



Complexity Approximations

Typically, we are only interested to capture the “overall behavior” of a complexity function for large inputs.

- Exact analysis:

$$\overline{T}(n) = 3n + \ln n + \varepsilon_n + \gamma - 2$$

- Approximation:

“ $\overline{T}(n)$ is of the order $3n + \ln n$ ”

- Coarser approximation:

“ $\overline{T}(n)$ is of the order $3n$ ”.

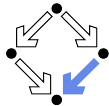
- Even coarser approximation:

“ $\overline{T}(n)$ is linear”

- Formalism:

$$\overline{T}(n) = O(n)$$

We are going to formalize such complexity approximations.



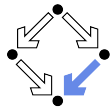
1. Complexity of Computations

2. Asymptotic Complexity

3. Working with Asymptotic Complexity

4. Complexity Classes

The Landau Symbols



Take $g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ from the natural numbers to the non-negative reals.

- $O(g)$: the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ such that

$$\exists c \in \mathbb{R}_{> 0}, N \in \mathbb{N} : \forall n \geq N : f(n) \leq c \cdot g(n)$$

- $f(n) = O(g(n))$: $f \in O(g)$.

f is bounded from above by g .

- $\Omega(g)$: the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ such that

$$\exists c \in \mathbb{R}_{> 0}, N \in \mathbb{N} : \forall n \geq N : f(n) \geq c \cdot g(n)$$

- $f(n) = \Omega(g(n))$: $f \in \Omega(g)$.

f is bounded from below by g .

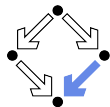
- $\Theta(g)$: the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ such that

$$f \in O(g) \wedge f \in \Omega(g)$$

- $f(n) = \Theta(g(n))$: $f \in \Theta(g)$.

f is bounded from above and below by g .

Common Practice of the Landau Symbols



We need to understand the historically developed usage of the symbols.

- Most wide spread: $f(n) = O(g(n))$.
 - Often used when actually $f(n) = \Theta(g(n))$ is meant, i.e.,
 - when $g(n)$ is not only a bound from above but also from below.
- Abuse of notation: $f(n) = O(g(n))$
 - $=$ does not denote equality but set inclusion.
 - Notation has nevertheless been universally adopted.
- Ambiguous notation: $f(n) = O(g(n))$
 - Terms $f(n)$ and $g(n)$ with implicit free variable n .
 - To derive $f \in O(g)$, we have to identify the free variable.

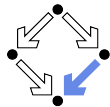
“Let $c > 1$. Then $x^c = O(c^x)$.”

\rightsquigarrow

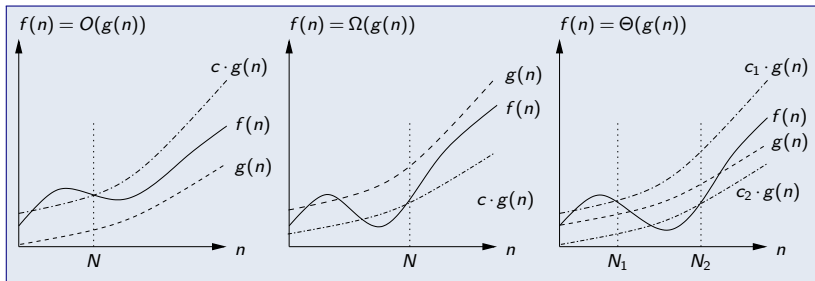
“Let $c > 1$, $f(x) := x^c$, and $g(x) := c^x$. Then $f \in O(g)$.”

We stick to the common practice.

Understanding the Landau Symbols

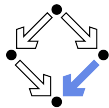


$f \in \mathcal{O}(g)$: g represents a bound for f , from above and/or below.



- It suffices, if bound holds from a certain start value N on.
 - Finitely many exceptions are allowed.
- It suffices, if the bound holds up to arbitrarily large constant c .
 - Bounds are independent of “measurement units”.

The Landau symbols talk about the *asymptotic* behavior of functions.



Duality of O and Ω

- **Theorem:** for all $f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$, we have

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

- **Proof \Rightarrow :** We assume $f(n) = O(g(n))$ and show $g(n) = \Omega(f(n))$. By the definition of Ω , we have to find constants N_1, c_1 such that

$$\forall n \geq N_1 : g(n) \geq c_1 \cdot f(n)$$

Since $f(n) = O(g(n))$, we have constants N_2, c_2 such that

$$\forall n \geq N_2 : f(n) \leq c_2 \cdot g(n)$$

Take $N_1 := N_2$ and $c_1 := 1/c_2$. Then we have, since $N_1 = N_2$, for all $n \geq N_1$,

$$c_2 \cdot g(n) \geq f(n)$$

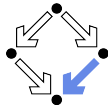
and therefore

$$g(n) \geq (1/c_2) \cdot f(n) = c_1 \cdot f(n).$$

- **Proof \Leftarrow :** analogously.

O and Ω are dual.

Example



We prove $3n^2 + 5n + 7 = O(n^2)$.

- We have to find constants c and N such that

$$\forall n \geq N : 3n^2 + 5n + 7 \leq cn^2$$

- For $n \geq 1$, we have

$$3n^2 + 5n + 7 \stackrel{1 \leq n}{\leq} 3n^2 + 5n + 7n = 3n^2 + 12n$$

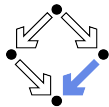
- For $n \geq 12$, we also have

$$3n^2 + 12n \stackrel{12 \leq n}{\leq} 3n^2 + n \cdot n = 4n^2$$

- We take $N := 12$ ($= \max\{1, 12\}$) and $c := 4$ and have for $n \geq N$

$$3n^2 + 5n + 7 \stackrel{1 \leq n}{\leq} 3n^2 + 5n + 7n = 3n^2 + 12n \stackrel{12 \leq n}{\leq} 3n^2 + n \cdot n = 4n^2 = cn^2$$

Demonstrates general technique for asymptotics of polynomial functions.



Asymptotic Laws

- **Theorem:** for all $a_0, \dots, a_m \in \mathbb{R}$, we have

$$a_m n^m + \dots + a_2 n^2 + a_1 n + a_0 = \Theta(n^m)$$

- Proof: analogous to previous example.
- **Theorem:** for all $a, b \in \mathbb{R}_{>0}$, we have

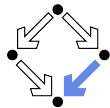
$$\log_a n = O(\log_b n)$$

- Proof: take $c := \log_a b$ and $N := 0$. Then we have for all $n \geq N$

$$\log_a n = \log_a(b^{\log_b n}) = (\log_a b) \cdot (\log_b n) = c \cdot (\log_b n)$$

Polynomials are dominated by the monomial with the highest exponent; in logarithms, bases don't matter.

Asymptotic Laws



- **Theorem:** for all $a, b \in \mathbb{R}$ with $b > 1$, we have

$$n^a = O(b^n)$$

- Proof: we know the Taylor series expansion

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Since $b^n = (e^{\ln b})^n = e^{n \ln b}$, we have for all $n \in \mathbb{N}$

$$b^n = \sum_{i=0}^{\infty} \frac{(n \ln b)^i}{i!} = 1 + (n \ln b) + \frac{(n \ln b)^2}{2!} + \frac{(n \ln b)^3}{3!} + \dots$$

Since $b > 1$, we have $\ln b > 0$; therefore we know

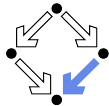
$$b^n > \frac{(n \ln b)^a}{a!} = \frac{(\ln b)^a}{a!} n^a$$

Consequently

$$n^a < \frac{a!}{(\ln b)^a} b^n$$

Thus we define $N := 0$ and $c := a! / (\ln b)^a$ and are done.

Polynomials are dominated by all exponentials with base greater one.



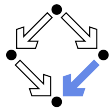
1. Complexity of Computations

2. Asymptotic Complexity

3. Working with Asymptotic Complexity

4. Complexity Classes

Asymptotic Laws



$$c \cdot f(n) = O(f(n))$$

$$n^m = O(n^{m'}), \text{ for all } m \leq m'$$

$$a_m n^m + \dots + a_2 n^2 + a_1 n + a_0 = \Theta(n^m)$$

$$\log_a n = O(\log_b n), \text{ for all } a, b > 0$$

$$n^a = O(b^n), \text{ for all } a, b \text{ with } b > 1$$

■ Reflexivity:

$$f = O(f), f = \Omega(f), f = \Theta(f).$$

■ Symmetry:

- If $f = O(g)$, then $g = \Omega(f)$.

- If $f = \Omega(g)$, then $g = O(f)$.

- If $f = \Theta(g)$, then $g = \Theta(f)$.

■ Transitivity:

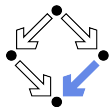
- If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.

- If $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$.

- If $f = \Theta(g)$ and $g = \Theta(h)$, then $f = \Theta(h)$.

The proof of reflexivity/symmetry/transitivity is an easy exercise.

Asymptotic Notation in Equations



A more general form of “syntactic abuse”.

■ **Equation:**

$$\dots \mathcal{O}_1(f(n)) \dots = \dots \mathcal{O}_2(g(n)) \dots$$

- (Possibly multiple) occurrences of $\mathcal{O}_1, \mathcal{O}_2 \in \{\mathcal{O}, \Omega, \Theta\}$ on the left hand respectively right hand side of the equation.

■ **Interpretation:**

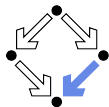
$$\forall f' \in \mathcal{O}_1(f) : \exists g' \in \mathcal{O}_2(g) :$$

$$\forall n \in \mathbb{N} : \dots f'(n) \dots = \dots g'(n) \dots$$

- Every occurrence of \mathcal{O} is replaced by a function in the corresponding asymptotic complexity class.
- Functions on the left side are universally quantified, functions on the right side are existentially quantified.

A convenient shortcut to express asymptotic relationships.

Example



- Example:

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right)$$

- There is a function $f \in O(1/n)$ such that, for all $n \in \mathbb{N}$,
 $H_n = \ln n + \gamma + f(n)$.

- Example:

$$2n^2 + 3n + 1 = O(2n^2) + O(n) = O(n^2)$$

- Equation $2n^2 + 3n + 1 = O(2n^2) + O(n)$

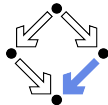
$$\exists f \in O(2n^2), g \in O(n) :$$

$$\forall n \in \mathbb{N} : 2n^2 + 3n + 1 = f(n) + g(n)$$

- Equation $O(2n^2) + O(n) = O(n^2)$

$$\forall f \in O(2n^2), g \in O(n) : \exists h \in O(n^2) :$$

$$\forall n \in \mathbb{N} : f(n) + g(n) = h(n)$$



Further Asymptotic Equations

We thus can express further asymptotic relationships.

$$O(O(f(n))) = O(f(n))$$

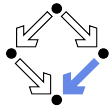
$$O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|)$$

$$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$$

$$O(f(n) \cdot g(n)) = f(n) \cdot O(g(n))$$

$$O(f(n)^m) = O(f(n))^m, \text{ for all } m \geq 0$$

The proofs are simple exercises.

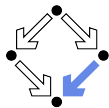


1. Complexity of Computations

2. Asymptotic Complexity

3. Working with Asymptotic Complexity

4. Complexity Classes



Further Landau Symbols

Take $g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ from the natural numbers to the non-negative reals.

- $o(g)$: the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ such that

$$f \in O(g) \wedge f \notin \Theta(g)$$

- $f(n) = o(g(n))$: $f \in o(g)$.

f is asymptotically smaller than g .

- $\omega(g)$: the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ such that

$$f \in \Omega(g) \wedge f \notin \Theta(g)$$

- $f(n) = \omega(g(n))$: $f \in \omega(g)$.

f is asymptotically larger than g .

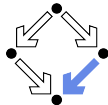
- **Theorem**: for all $f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$, we have

$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

Both notions are dual.

Useful to create a hierarchy of asymptotic growth functions.

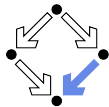
Hierarchy of Complexity Classes



Define $f \prec g : \Leftrightarrow f = o(g)$.

$$\begin{aligned} 1 &\prec \log \log \log n \prec \log \log n \prec \sqrt{\log n} \prec \log n \prec (\log n)^2 \prec (\log n)^3 \\ &\prec \sqrt[3]{n} \prec \sqrt{n} \prec n \prec n \log n \prec n\sqrt{n} \prec n^2 \prec n^3 \\ &\prec n^{\log n} \prec 2^{\sqrt{n}} \prec 2^n \prec 3^n \prec n! \prec n^n \prec 2^{n^2} \prec 2^{2^n} \\ &\prec 2^{2^{\dots^2}} \quad (n \text{ times}) \end{aligned}$$

Fundamental knowledge about complexity classes.

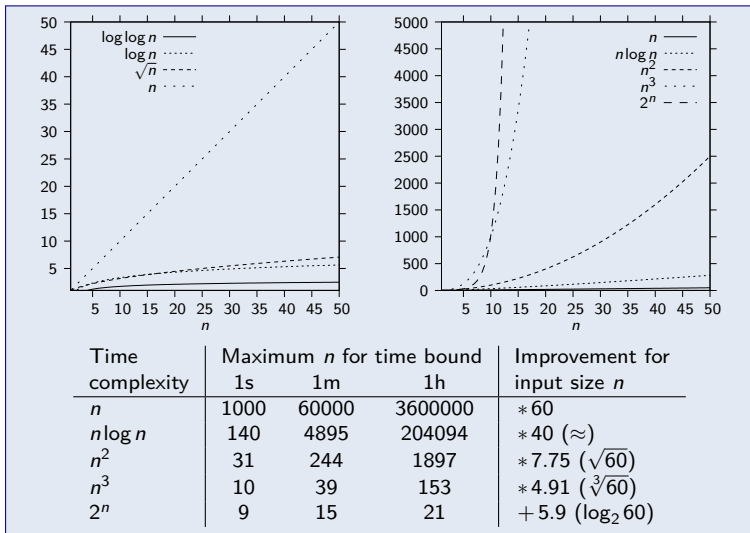
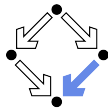


Hierarchy of Complexity Classes

- $O(1)$ (**Constant**): upper limit on function values.
 - Space complexity of algorithms that work with fixed memory size.
- $O(\log n)$ (**Logarithmic**): values grow very slowly.
 - Time complexity of binary search.
- $O(n)$ (**Linear**): values grow proportionally with argument.
 - Time complexity of linear search.
- $O(n \log n)$ (**Linear-Logarithmic**): value growth is reasonably well behaved.
 - Time complexity of fast sorting algorithms, e.g., Mergesort.
- $O(n^c)$ (**Polynomial**): values grow rapidly but with polynomial bound.
 - Executions still “feasible” for large inputs, e.g., matrix multiplication.
- $O(c^n)$ (**Exponential**): values grow extremely rapidly.
 - Executions only reasonable for small inputs; e.g, finding exact solutions to many optimization problems (“traveling salesman problem”).
- $O(c^{d^n})$ (**Double Exponential**): Function values grow overwhelmingly rapidly.
 - Decision of statements about real numbers (“quantifier elimination”), solving multivariate polynomial equations (“Buchberger’s algorithm”).

Only computations up to polynomial complexity are considered “feasible”.

Complexity Classes



Improvement in asymptotic complexity outperforms technological speedup.