Algorithms and Computability

Lecture 11: Time Complexity

Christian Schilling (christianms@cs.aau.dk)

slides courtesy of Martin Zimmermann

Last Time in Algorithms and Computability

We have seen

- Mapping reductions
- Non-computable problems and how to prove them non-computable via reductions
- Rice's theorem: everything "interesting" about Turing machines is not computable
- Consequence: everything "interesting" about programs is not computable

Agenda

1. **Motivation**
2. Big-O Notation
3. Complexity Theory
4. Time Complexity

Remember the Entscheidungsproblem?

Lecture 8

The "Entscheidungsproblem" (Hilbert and Ackermann, 1928)

> *Is there an algorithm that, given a statement in some logical language (typically predicate logic), answers "Yes" or "No" according to whether the statement is universally valid?*

Remember the Entscheidungsproblem?

Lecture 8

The “Entscheidungsproblem” (Hilbert and Ackermann, 1928)

> *Is there an algorithm that, given a statement in some logical language (typically predicate logic), answers “Yes” or “No” according to whether the statement is universally valid?*

In our view: Is $\{w \mid w \text{ encodes a valid statement}\}$ computable?

Lecture 10

The Entscheidungsproblem is not computable **[Church, Turing '36]**

Peano Arithmetic

- Peano arithmetic (named after Giuseppe Peano) is a logical system for reasoning about arithmetic of natural numbers with addition and multiplication, given by a finite set of axioms:
  - $\forall x.\ \neg(x + 1 = 0)$
  - $\forall x, y.\ (x + 1 = y + 1) \rightarrow x = y$
  - $\forall x, y, z.\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$
  - ...

Peano Arithmetic

- Peano arithmetic (named after Giuseppe Peano) is a logical system for reasoning about arithmetic of natural numbers with addition and multiplication, given by a finite set of axioms:
  - $\forall x.\ \neg(x + 1 = 0)$
  - $\forall x, y.\ (x + 1 = y + 1) \rightarrow x = y$
  - $\forall x, y, z.\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$
  - ...
- The provable statements of Peano arithmetic do not form a computable language **[Gödel '31]**

  In other words, it cannot be determined algorithmically whether a given statement over the natural numbers holds

A Computable Problem

- Presburger arithmetic (named after Mojżesz Presburger) is a logical system for reasoning about arithmetic of natural numbers with only addition, given by the following axioms:
  - $\forall x, y.\ x + y = y + x$
  - $\forall x.\ x + 0 = x$
  - $\forall x, y.\ x + (y + 1) = (x + y) + 1$
  - $[P(0) \wedge (\forall x.\ P(x) \rightarrow P(x + 1))] \rightarrow \forall x.\ P(x)$ for all Presburger formulas P (the scheme of induction)

A Computable Problem

- Presburger arithmetic (named after Mojżesz Presburger) is a logical system for reasoning about arithmetic of natural numbers with only addition, given by the following axioms:
  - $\forall x, y.\ x + y = y + x$
  - $\forall x.\ x + 0 = x$
  - $\forall x, y.\ x + (y + 1) = (x + y) + 1$
  - $[P(0) \wedge (\forall x.\ P(x) \rightarrow P(x + 1))] \rightarrow \forall x.\ P(x)$ for all Presburger formulas P (the scheme of induction)
- With these axioms, one can prove, e.g.,

$$\forall x \exists y.\ (x = y + y) \vee (x = y + y + 1)$$

A Computable Problem

- Presburger arithmetic (named after Mojżesz Presburger) is a logical system for reasoning about arithmetic of natural numbers with only addition, given by the following axioms:
  - $\forall x, y.\ x + y = y + x$
  - $\forall x.\ x + 0 = x$
  - $\forall x, y.\ x + (y + 1) = (x + y) + 1$
  - $[P(0) \wedge (\forall x.\ P(x) \rightarrow P(x + 1))] \rightarrow \forall x.\ P(x)$ for all Presburger formulas P (the scheme of induction)
- The provable statements of Presburger arithmetic form a computable language **[Presburger '29]**
- But: Any halting DTM for that language has to move its head at least $2^{2^{cn}}$ times on some inputs of length n (for some constant c) **[Fischer, Rabin '74]**

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$
- $2^{2^1} = 4$

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$
- $2^{2^1} = 4$
- $2^{2^2} = 16$

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$
- $2^{2^1} = 4$
- $2^{2^2} = 16$
- $2^{2^3} = 256$

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$
- $2^{2^1} = 4$
- $2^{2^2} = 16$
- $2^{2^3} = 256$
- $2^{2^4} = 65536$

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$
- $2^{2^1} = 4$
- $2^{2^2} = 16$
- $2^{2^3} = 256$
- $2^{2^4} = 65536$
- $2^{2^5} = 4294967296$

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$
- $2^{2^1} = 4$
- $2^{2^2} = 16$
- $2^{2^3} = 256$
- $2^{2^4} = 65536$
- $2^{2^5} = 4294967296$
- $2^{2^6} = 18446744073709551616$

Doubly-Exponential Growth

Consider $2^{2^{cn}}$ for $c = 1$:

- $2^{2^0} = 2$
- $2^{2^1} = 4$
- $2^{2^2} = 16$
- $2^{2^3} = 256$
- $2^{2^4} = 65536$
- $2^{2^5} = 4294967296$
- $2^{2^6} = 18446744073709551616$
- $2^{2^7} = 340282366920938463463374607431768211456$
- $\cdots$

A Change in Perspective

We studied the following question:

Which problems can be solved by a computer?

- The computable problems are those that can be solved by a computer
- The computably-enumerable problems are those where the "yes"-answer can be verified by a computer
- Mapping reductions can be used to find relations between problems

A Change in Perspective

We will study the following question:

Which problems can be solved efficiently by a computer?

- The complexity class P contains the problems that can be solved efficiently by a computer
- The complexity class NP contains the problems where the "yes"-answer can be verified efficiently by a computer
- Polynomial-time reductions can be used to find relations between problems

Agenda

1. Motivation
2. **Big-O Notation**
3. Complexity Theory
4. Time Complexity

Reminder

Definition

Let $f, g \colon \mathbb{N} \to \mathbb{R}_{>0}$ be functions. We write $f(n) = \mathcal{O}(g(n))$ if

- there are positive integers c, n_0 such that
- $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

Definition

Let $f, g \colon \mathbb{N} \to \mathbb{R}_{>0}$ be functions. We write $f(n) = \mathcal{O}(g(n))$ if

- there are positive integers c, n_0 such that
- $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

Examples

- $2n^2 + 5 = \mathcal{O}(n^2)$ (e.g., with $c = 3$ and $n_0 = 3$)
- $7n^5 + 82n^4 + n^2 + 213 = \mathcal{O}(n^5)$
- $\log_2(n^7) = \mathcal{O}(\log_2 n)$

Reminder

Definition

Let $f, g \colon \mathbb{N} \to \mathbb{R}_{>0}$ be functions. We write $f(n) = \mathcal{O}(g(n))$ if

- there are positive integers c, n_0 such that
- $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

Examples

- $2n^2 + 5 = \mathcal{O}(n^2)$ (e.g., with $c = 3$ and $n_0 = 3$)
- $7n^5 + 82n^4 + n^2 + 213 = \mathcal{O}(n^5)$
- $\log_2(n^7) = \mathcal{O}(\log_2 n)$

Reading

CLRS section 3.1 and 3.2 on Big-O notation

Why Do We Care about Growth Rates?

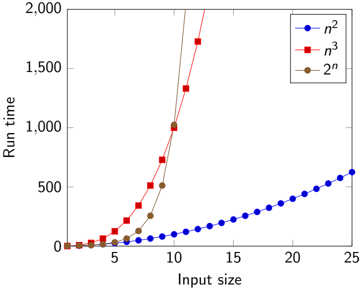
- Assume your computer performs 1 billion steps per second
- The table below shows the CPU time for inputs of size n

n	$f(n) = n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.01 microsec	0.1 microsec	1 microsec	1 microsec
20	0.02 microsec	0.4 microsec	8 microsec	1 millisec
50	0.05 microsec	2.5 microsec	125 microsec	3 days
100	0.1 microsec	10 microsec	1 millisec	$4 \cdot 10^{13}$ years

Why Do We Care about Growth Rates?

- Assume your computer performs 1 billion steps per second
- The table below shows the CPU time for inputs of size n

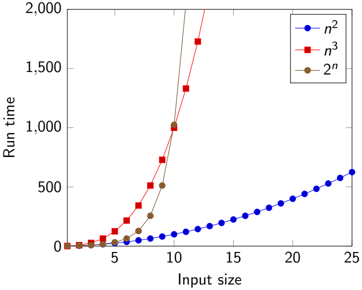
n	$f(n) = n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.01 microsec	0.1 microsec	1 microsec	1 microsec
20	0.02 microsec	0.4 microsec	8 microsec	1 millisec
50	0.05 microsec	2.5 microsec	125 microsec	3 days
100	0.1 microsec	10 microsec	1 millisec	$4 \cdot 10^{13}$ years

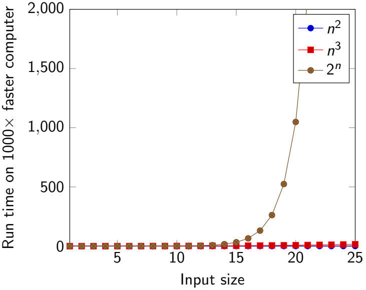


Why Do We Care about Growth Rates?

- Assume your computer performs 1 billion steps per second
- The table below shows the CPU time for inputs of size n

n	$f(n) = n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.01 microsec	0.1 microsec	1 microsec	1 microsec
20	0.02 microsec	0.4 microsec	8 microsec	1 millisec
50	0.05 microsec	2.5 microsec	125 microsec	3 days
100	0.1 microsec	10 microsec	1 millisec	$4 \cdot 10^{13}$ years





Agenda

1. Motivation
2. Big-O Notation
3. **Complexity Theory**
4. Time Complexity

Intuition

Consider the following halting DTM accepting $\{0^m 1^m \mid m \geq 1\}$:

On input w:

1. Pass once over the tape content and reject if it is not of the form 0^+1^+
2. Pass once over the tape content and replace first 0 by X and first 1 by X (if not possible, reject)
3. Pass once over the tape content. If there is still a 0 but no 1 or no 0 but still a 1, then reject. If no 0 and no 1 are left, then accept
4. Go to 2

How many steps does the Turing machine take on an input of length n at most?

Intuition

Consider the following halting DTM accepting $\{0^m 1^m \mid m \geq 1\}$:

On input w:

1. Pass once over the tape content and reject if it is not of the form 0^+1^+ n steps
2. Pass once over the tape content and replace first 0 by X and first 1 by X (if not possible, reject)
3. Pass once over the tape content. If there is still a 0 but no 1 or no 0 but still a 1, then reject. If no 0 and no 1 are left, then accept
4. Go to 2

How many steps does the Turing machine take on an input of length n at most?

Intuition

Consider the following halting DTM accepting $\{0^m 1^m \mid m \geq 1\}$:

On input w:

1. Pass once over the tape content and reject if it is not of the form 0^+1^+ n steps
2. Pass once over the tape content and replace first 0 by X and first 1 by X (if not possible, reject) $2n$ steps
3. Pass once over the tape content. If there is still a 0 but no 1 or no 0 but still a 1, then reject. If no 0 and no 1 are left, then accept
4. Go to 2

How many steps does the Turing machine take on an input of length n at most?

Intuition

Consider the following halting DTM accepting $\{0^m 1^m \mid m \geq 1\}$:

On input w:

1. Pass once over the tape content and reject if it is not of the form 0^+1^+ n steps
2. Pass once over the tape content and replace first 0 by X and first 1 by X (if not possible, reject) $2n$ steps
3. Pass once over the tape content. If there is still a 0 but no 1 or no 0 but still a 1, then reject. If no 0 and no 1 are left, then accept $2n$ steps
4. Go to 2

How many steps does the Turing machine take on an input of length n at most?

Intuition

Consider the following halting DTM accepting $\{0^m 1^m \mid m \geq 1\}$:

On input w:

1. Pass once over the tape content and reject if it is not of the form 0^+1^+ — n steps
2. Pass once over the tape content and replace first 0 by X and first 1 by X (if not possible, reject) — $2n$ steps
3. Pass once over the tape content. If there is still a 0 but no 1 or no 0 but still a 1, then reject. If no 0 and no 1 are left, then accept — $2n$ steps
4. Go to 2 — 1 step, executed at most $\frac{n}{2}$ times

How many steps does the Turing machine take on an input of length n at most?

Consider the following halting DTM accepting $\{0^m 1^m \mid m \geq 1\}$:

On input w:

1. Pass once over the tape content and reject if it is not of the form 0^+1^+ — n steps
2. Pass once over the tape content and replace first 0 by X and first 1 by X (if not possible, reject) — $2n$ steps
3. Pass once over the tape content. If there is still a 0 but no 1 or no 0 but still a 1, then reject. If no 0 and no 1 are left, then accept — $2n$ steps
4. Go to 2 — 1 step, executed at most $\frac{n}{2}$ times

How many steps does the Turing machine take on an input of length n at most? $\mathcal{O}(n^2)$ steps overall

Complexity of Algorithms

On the previous slide, we have analyzed the (worst-case) time complexity of one halting DTM (algorithm) for $\{0^m 1^m \mid m \geq 1\}$... but there are many different DTMs!

Complexity of Algorithms

On the previous slide, we have analyzed the (worst-case) time complexity of one halting DTM (algorithm) for $\{0^m1^m \mid m \geq 1\}$... but there are many different DTMs!

Example

Consider the sorting problem:

1. Insertion sort requires $\Omega(n^2)$ comparisons in the worst case
2. Merge sort requires $\Omega(n \log_2 n)$ comparisons in the worst case
3. Every comparison-based sorting algorithm requires $\Omega(n \log_2 n)$ comparisons in the worst case

Complexity of Algorithms

On the previous slide, we have analyzed the (worst-case) time complexity of one halting DTM (algorithm) for $\{0^m 1^m \mid m \geq 1\}$... but there are many different DTMs!

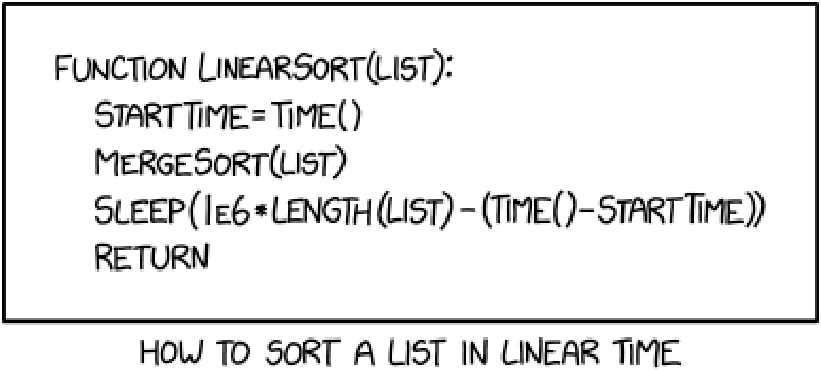
Example

Consider the sorting problem:

1. Insertion sort requires $\Omega(n^2)$ comparisons in the worst case
2. Merge sort requires $\Omega(n \log_2 n)$ comparisons in the worst case
3. Every comparison-based sorting algorithm requires $\Omega(n \log_2 n)$ comparisons in the worst case

Note

The first two statements are about specific algorithms, the third one is about all (comparison-based) algorithms



Source: `https://xkcd.com/3026`

Complexity Theory

Focus of algorithm complexity:

- Study concrete algorithms and their complexity

Focus of complexity theory:

- Study problems (i.e., languages) instead of algorithms
- Goal: classify problems according to how easy/hard they are to solve

Complexity Theory

Focus of algorithm complexity:

- Study concrete algorithms and their complexity

Focus of complexity theory:

- Study problems (i.e., languages) instead of algorithms
- Goal: classify problems according to how easy/hard they are to solve

From now on:

- Today, we only consider halting DTMs
- Later, we will also consider halting NTMs
- We analyze the (worst-case) time complexity of DTMs

  Why Turing machines?

Extended Church–Turing thesis

Everything that can be efficiently computed can be computed efficiently by a probabilistic (!) Turing machine

Extended Church–Turing thesis

Everything that can be efficiently computed can be computed efficiently by a probabilistic (!) Turing machine

- Under (unproven but widely assumed) complexity-theoretic assumptions, the word "probabilistic" can be dropped

Extended Church–Turing thesis

Everything that can be efficiently computed can be computed efficiently by a probabilistic (!) Turing machine

- Under (unproven but widely assumed) complexity-theoretic assumptions, the word “probabilistic” can be dropped
- Under (unproven but widely assumed) complexity-theoretic assumptions, the extended Church-Turing thesis is false! Quantum Turing machines cannot be simulated efficiently by (even probabilistic) Turing machines

Extended Church–Turing thesis

Everything that can be efficiently computed can be computed efficiently by a probabilistic (!) Turing machine

- Under (unproven but widely assumed) complexity-theoretic assumptions, the word "probabilistic" can be dropped
- Under (unproven but widely assumed) complexity-theoretic assumptions, the extended Church-Turing thesis is false! Quantum Turing machines cannot be simulated efficiently by (even probabilistic) Turing machines

Nevertheless, Turing machines are a robust model for computation and can efficiently simulate most other models of computation

Agenda

1. Motivation

2. Big-O Notation

3. Complexity Theory

4. **Time Complexity**

Definition

Let M be a halting DTM

- Let $\mathit{time}_M(w)$ denote the number of configurations in the unique run of M on input w
- Let $T\colon \mathbb{N} \to \mathbb{R}_{>0}$. We say that M runs within time T if $\mathit{time}_M(w) \leq T(|w|)$ for all inputs w

Run Time of a Deterministic Turing Machine

Definition

Let M be a halting DTM

- Let $\mathit{time}_M(w)$ denote the number of configurations in the unique run of M on input w
- Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$. We say that M runs within time T if $\mathit{time}_M(w) \le T(|w|)$ for all inputs w

We are interested in classifying problems (languages) according to their asymptotic time complexity

Definition

Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$ be a function. The complexity class $\textsc{Time}(T)$ is defined as

$$\textsc{Time}(T) = \{L(M) \mid M \text{ is a halting DTM that runs within time } \mathcal{O}(T)\}$$

Quiz 1

What kind of objects does $\textsc{Time}(T(n))$ contain?

Quiz 1

What kind of objects does $\textsc{Time}(T(n))$ contain?

Languages (i.e., problems)

Time Hierarchy

- A consequence of a more general theorem:

$$\textsc{Time}(n) \subsetneq \textsc{Time}(n^2) \subsetneq \textsc{Time}(n^3) \subsetneq \textsc{Time}(n^4) \subseteq \cdots \subsetneq \textsc{Time}(2^n)$$

- Intuitively: more time allows you to compute more languages

Proof: via diagonalization, construct a language that is different from every language in $\textsc{Time}(n^k)$, but that is in $\textsc{Time}(n^{k+1})$

Robustness

For one-tape halting DTMs:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \text{TIME}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \text{TIME}(n^2)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \text{TIME}(n^2)$

Robustness

For one-tape halting DTMs:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n^2)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n^2)$

For two-tape halting DTMs:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n)$

Robustness

For one-tape halting DTMs:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n^2)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n^2)$

For two-tape halting DTMs:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n)$

In general:

Theorem

Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$ be a function such that $T(n) \geq n$. Every k-tape halting DTM with time complexity $T(n)$ can be simulated by an equivalent one-tape halting DTM with time complexity $(T(n))^2$

Robustness

For one-tape halting DTMs:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n^2)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n^2)$

For two-tape halting DTMs:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n)$

In general:

Theorem

Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$ be a function such that $T(n) \geq n$. Every k-tape halting DTM with time complexity $T(n)$ can be simulated by an equivalent one-tape halting DTM with time complexity $(T(n))^2$

Proof: Simulation presented in Lecture 8 has the desired properties

Definition

The complexity class P (polynomial time) is defined as

$$\mathrm{P} = \bigcup_{k \geq 0} \textsc{Time}(n^k)$$

- Robust definition (can use other deterministic (!) models of computation, e.g., multi-tape Turing machines)
- Cobham's thesis: A problem can be efficiently computed if and only if it is in P

About Cobham's Thesis

A problem with fastest algorithm of running time...

- n^{58}: efficient!
- $n^{\log_2 \log_2 \log_2 \log_2 n}$: not efficient!

About Cobham's Thesis

A problem with fastest algorithm of running time...

- n^{58}: efficient!

  But 2^{58} is roughly the age of the universe in seconds
- $n^{\log_2 \log_2 \log_2 \log_2 n}$: not efficient!

About Cobham's Thesis

A problem with fastest algorithm of running time...

- n^{58}: efficient!

  But 2^{58} is roughly the age of the universe in seconds
- $n^{\log_2 \log_2 \log_2 \log_2 n}$: not efficient!

  But it is roughly $3n$ for $n = 1.000.000$ and roughly $60n$ for $n = 1.000.000.000$

About Cobham's Thesis

A problem with fastest algorithm of running time...

- n^{58}: efficient!

  But 2^{58} is roughly the age of the universe in seconds
- $n^{\log_2 \log_2 \log_2 \log_2 n}$: not efficient!

  But it is roughly $3n$ for $n = 1.000.000$ and roughly $60n$ for $n = 1.000.000.000$
- In practical applications: Even quadratic running time is infeasible for data-intensive problems

Problems in P

Many problems are in P:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n^2)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n^2)$

Problems in P

Many problems are in P:

- $\{0^m 1^{m'} \mid m, m' \geq 1\} \in \textsc{Time}(n)$
- $\{0^m 1^m \mid m \geq 1\} \in \textsc{Time}(n^2)$
- $\{w\#w \mid w \in \mathbb{B}^*\} \in \textsc{Time}(n^2)$
- Graph reachability
- NFA emptiness
- Primality
- Solving linear equation systems
- Linear programming
- Word problem for context-free grammars
- And many other problems

Theorem

P *is closed under union, intersection, complementation, concatenation, and iteration*

Proof: The constructions presented in Lecture 9 and Tutorial 9 yield halting DTMs with polynomial time complexity when applied to halting DTMs with polynomial time complexity

Conclusion

We have compared algorithmic complexity:

- Study of concrete algorithms and precise running times
- Difference between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ is huge
- Running times depend on model of computation

and complexity theory:

- Study of problems (languages) rather than algorithms
- Difference between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ may just depend on choice of model (but people still try to find optimal (for fixed model) algorithms)
- Results should be valid for most models of computation

Conclusion

We have compared algorithmic complexity:

- Study of concrete algorithms and precise running times
- Difference between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ is huge
- Running times depend on model of computation

and complexity theory:

- Study of problems (languages) rather than algorithms
- Difference between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ may just depend on choice of model (but people still try to find optimal (for fixed model) algorithms)
- Results should be valid for most models of computation

Note

Algorithmic complexity is still important for complexity theory: giving an algorithm with polynomial running time for a problem L implies $L \in \mathrm{P}$

Reading

Sections 3.1 and 3.2 of "Computability and Complexity"