

# Datenstrukturen und Algorithmen

Vorlesung am D-Math (CSE) der ETH Zürich

Felix Friedrich

FS 2017

# Willkommen!

## Course homepage

<http://lec.inf.ethz.ch/DA/2017>

## The team:

Assistenten

Alexander Pilz

Daniel Hupp

Lukas Humbel

Dozent

Felix Friedrich

# 1. Introduction

Algorithms and Data Structures, Three Examples

# Goals of the course

- Understand the design and analysis of fundamental algorithms and data structures.
- An advanced insight into a modern programming model (with C++).
- Knowledge about chances, problems and limits of the parallel and concurrent computing.

# Goals of the course

On the one hand

- Essential basic knowledge from computer science.

Andererseits

- Preparation for your further course of studies and practical considerations.

# Contents

---

## data structures / algorithms

The notion invariant, cost model, Landau notation

algorithms design, induction

searching, selection and sorting

dynamic programming

dictionaries: hashing and search trees

sorting networks, parallel algorithms

Randomized algorithms (Gibbs/SA), multiscale approach

geometric algorithms, high performance LA

graphs, shortest paths, backtracking, flow

---

## programming with C++

RAII, Move Konstruktion, Smart Pointers, Constexpr, user defined literals

Templates and generic programming

Exceptions

functors and lambdas

promises and futures

threads, mutex and monitors

---

## parallel programming

parallelism vs. concurrency, speedup (Amdahl/-Gustavson), races, memory reordering, atomir registers, RMW (CAS,TAS), deadlock/starvation

# literature

**Algorithmen und Datenstrukturen**, *T. Ottmann, P. Widmayer*, Spektrum-Verlag, 5. Auflage, 2011

**Algorithmen - Eine Einführung**, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*, Oldenbourg, 2010

**Introduction to Algorithms**, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*, 3rd ed., MIT Press, 2009

**The C++ Programming Language**, *B. Stroustrup*, 4th ed., Addison-Wesley, 2013.

**The Art of Multiprocessor Programming**, *M. Herlihy, N. Shavit*, Elsevier, 2012.

## 1.2 Algorithms

[Cormen et al, Kap. 1; Ottman/Widmayer, Kap. 1.1]



# Algorithm

Algorithm: well defined computing procedure to compute *output* data from *input* data

# example problem

**Input :** A sequence of  $n$  numbers  $(a_1, a_2, \dots, a_n)$

**Output :** Permutation  $(a'_1, a'_2, \dots, a'_n)$  of the sequence  $(a_i)_{1 \leq i \leq n}$ , such that  
 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

## Possible input

$(1, 7, 3), (15, 13, 12, -0.5), (1) \dots$

Every example represents a *problem instance*

# Examples for algorithmic problems

- routing: shortest path
- cryptography / digital signatures
- time table / working plans: linear programming
- DNA matching: dynamic programming
- fabrication pipeline: topological sort
- geometric problems, e.g. convex hull

# Characteristics

- Extremely large number of potential solutions
- Practical applicability

# Data Structures

- Organisation of the data tailored towards the algorithms that operate on the data.
- Programs = algorithms + data structures.

# Very hard problems.

- NP-complete problems: no known efficient solution (but the non-existence of such a solution is not proven yet!)
- Example: travelling salesman problem

# A dream

- If computers were infinitely fast and had an infinite amount of memory ...
- ... then we would still need the theory of algorithms (only) for statements about correctness (and termination).

# The reality

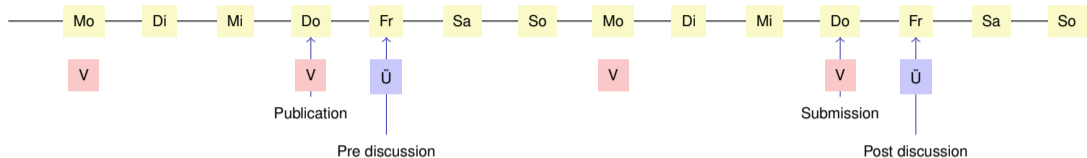
Resources are bounded and not free:

- Computing time → Efficiency
- Storage space → Efficiency



## 1.3 Organisation

# The exercise process



- Exercise publication each Thursday
- Preliminary discussion on Friday
- Latest submission Thursday one week later
- Debriefing of the exercise on follong Friday. Feedback to your submissions within a week after debriefing.

# Codeboard

*Codeboard* is an online-IDE: programming in the browser

- Examples can be tried without any tool installation.
- Used for the exercises.

Jetzt mit C++14

```
#include "tests.h" // remove slashes at beginning of
#include <iostream>

int main()
{
    Averager avg;
    int n;
    std::cin >> n;
    for (int i = 0; i < n; ++i) {
        double input;
        std::cin >> input;
        avg.add_value(input);
        std::cout << avg.average_value() << " ";
    }
    std::cout << "\n";
    return 0;
}
```

Task 2a: Averager

Not submitted yet

Solve the task and hand in your solution using the green "Submit" button.

Task Description

Write a class **Averager** that computes averages of given values of type **double**. It shall offer at least the members **add\_value** (which provides another value of type **double** to the class) and **average\_value** (which returns the average of the values provided so far as a **double**).

[based on: Exam Summer 2012, ex.6]

This will display the output.

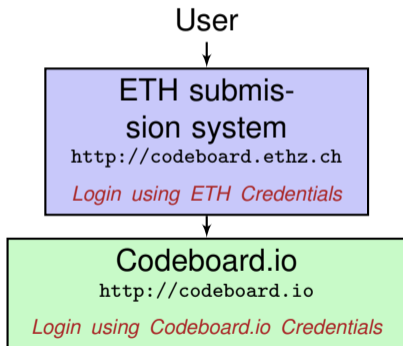
Input to your program (press Enter to send)

User: anonymous [sign in](#) to save your progress | Role: Project user | Info: Submissions are forwarded to external platform | [codeboard.io](#)

# Codeboard @ETH

Codeboard consists of two independent communicating systems:

- **The ETH submission system** Allows us to correct your submissions
- **The online IDE** The programming environment.



# Codeboard

## Codeboard.io registration

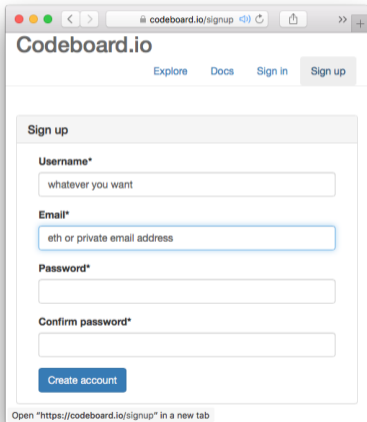
Go to <http://codeboard.io> and create an account, best is to stay logged in

## Register for the recitation sessions

Go to <http://codeboard.ethz.ch/da> and register for a recitation session there.

# Codeboard.io registration

Should you not yet have a **Codeboard.io** account ...



The image shows a browser window with the URL `codeboard.io/signup`. The page title is "Codeboard.io" and it has navigation links for "Explore", "Docs", "Sign in", and "Sign up". The "Sign up" form contains the following fields:

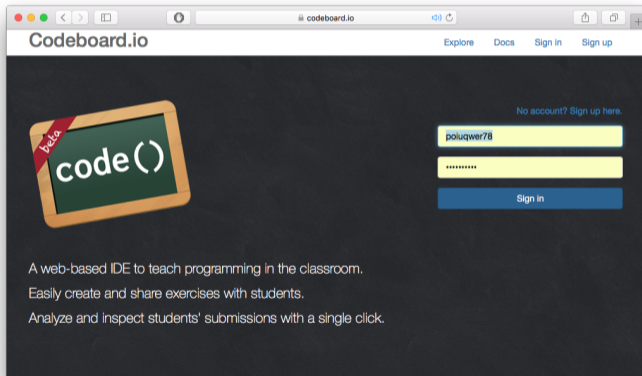
- Username\***: A text input field containing the placeholder text "whatever you want".
- Email\***: A text input field containing the placeholder text "eth or private email address".
- Password\***: A text input field.
- Confirm password\***: A text input field.

At the bottom of the form is a blue button labeled "Create account". Below the form, a status bar indicates "Open 'https://codeboard.io/signup' in a new tab".

- We will be using the online IDE **Codeboard.io**
- create an account in order to be able to store your progress
- Login data can be chose arbitrarily. *Do not use your ETH password.*

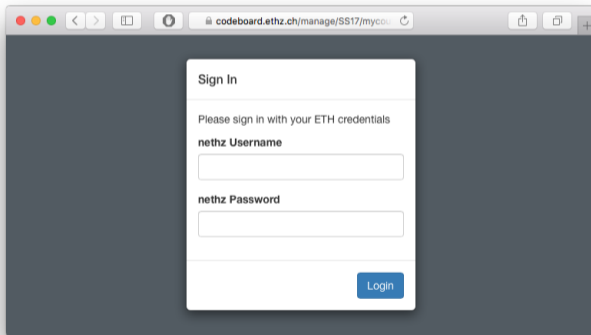
# Codeboard.io Login

If you have an account, log in:



# Recitation session registration - I

- Visit `http://codeboard.ethz.ch/da`
- Login with your ETH account



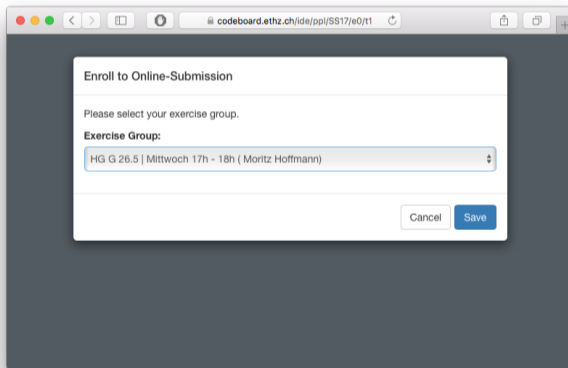
The screenshot shows a web browser window with the address bar displaying `codeboard.ethz.ch/manage/SS17/mycode`. The main content area features a white 'Sign In' form centered on a dark grey background. The form contains the following elements:

- Sign In** (Section Header)
- Please sign in with your ETH credentials
- nethz Username** (Label) followed by an empty text input field.
- nethz Password** (Label) followed by an empty password input field.
- Login** (Button) located at the bottom right of the form.



# Recitation session registration - II

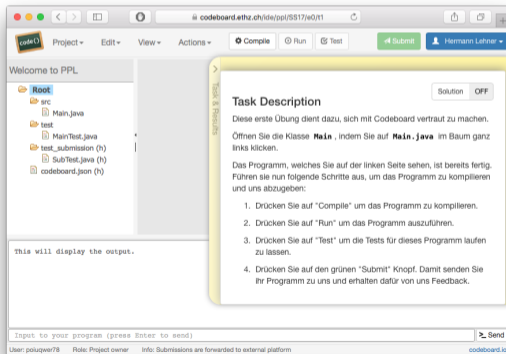
Register using the dialog with a recitation session.



The image shows a browser window with the URL `codeboard.ethz.ch/ide/pp/SS17/e0/t1`. A modal dialog titled "Enroll to Online-Submission" is displayed. The dialog contains the text "Please select your exercise group." followed by a label "Exercise Group:" and a dropdown menu. The dropdown menu is open, showing the selected option: "HG G 26.5 | Mittwoch 17h - 18h ( Moritz Hoffmann)". At the bottom right of the dialog, there are two buttons: "Cancel" and "Save".

# The first exercise

You are now registered and the first exercise is loaded. Follow the guidelines in the yellow box. The exercise sheet on the course homepage contains further instructions and explanations.



The screenshot shows the Codeboard IDE interface. The browser address bar displays `codeboard.ethz.ch/ide/pp1/SS17/w0/t1`. The top navigation bar includes 'Project', 'Edit', 'View', 'Actions', 'Compile', 'Run', 'Test', 'Submit', and 'Hermann Lehner'. The left sidebar shows a file tree with 'Root' containing 'src' (Main.java), 'test' (MainTest.java, test\_submission(h), SubTest.java(h)), and 'codeboard.json'. The main area features a 'Task Description' panel with a 'Solution OFF' toggle. The task description text is as follows:

**Task Description** Solution OFF

Diese erste Übung dient dazu, sich mit Codeboard vertraut zu machen.

Öffnen Sie die Klasse **Main**, indem Sie auf **Main.java** im Baum ganz links klicken.

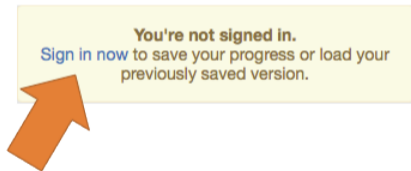
Das Programm, welches Sie auf der linken Seite sehen, ist bereits fertig. Führen sie nun folgende Schritte aus, um das Programm zu kompilieren und uns abzugeben:

1. Drücken Sie auf "Compile" um das Programm zu kompilieren.
2. Drücken Sie auf "Run" um das Programm auszuführen.
3. Drücken Sie auf "Test" um die Tests für dieses Programm laufen zu lassen.
4. Drücken Sie auf den grünen "Submit" Knopf. Damit senden Sie ihr Programm zu uns und erhalten dafür von uns Feedback.

Below the task description, there is a text area with the placeholder "This will display the output." and an input field at the bottom with the text "Input to your program (press Enter to send)" and a "Send" button. The footer shows "User: poluzwer78 Role: Project owner Info: Submissions are forwarded to external platform" and the Codeboard logo.

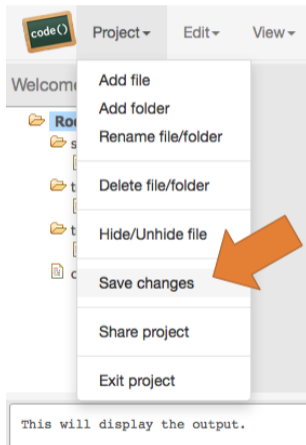
# The first exercise – Codeboard.io Login

If you see this message, click on [Sign in now](#) and log in with your **Codeboard.io** account.



# The first exercise – store progress!

*Attention!* Store your progress on a regular basis. Then you can continue somewhere else easily.



# About the exercises

- Since HS 2013 no exercise certificate required any more for exam admission
- Doing the exercises and going to the recitation sessions is optional but **highly** recommended!

# Relevant for the exam

Material for the exam comprises

- Course content (lectures, handout)
- Exercises content (exercise sheets, recitation hours)

Written exam (120 min). Examination aids: four A4 pages (or two sheets of 2 A4 pages double sided) either hand written or with font size minimally 11 pt.

# In your and our interest

Please let us know early if you see any problems, if

- the lectures are too fast, too difficult, too ...
- the exercises are not doable or not understandable ...
- you do not feel well supported ...

In short: if you have  
any issues that we can fix.



# 1.4 Ancient Egyptian Multiplication

Ancient Egyptian Multiplication



# Example 1: Ancient Egyptian Multiplication<sup>1</sup>

Compute  $11 \cdot 9$

11		9
<del>22</del>		<del>4</del>
<del>44</del>		<del>2</del>
88		1
99		—

9		11
18		5
<del>36</del>		<del>2</del>
72		1
99		—

- 1 Double left, integer division by 2 on the right
- 2 Even number on the right  $\Rightarrow$  eliminate row.
- 3 Add remaining rows on the left.

---

<sup>1</sup>Also known as russian multiplication

# Advantages

- Short description, easy to grasp
- Efficient to implement on a computer: double = left shift, divide by 2 = right shift

## Beispiel

*left shift*     $9 = 01001_2 \rightarrow 10010_2 = 18$

*right shift*     $9 = 01001_2 \rightarrow 00100_2 = 4$

# Questions

- Does this always work (negative numbers)?
- If not, when does it work?
- How do you prove correctness?
- Is it better than the school method?
- What does “good” mean at all?
- How to write this down precisely?

# Observation

If  $b > 1$ ,  $a \in \mathbb{Z}$ , then:

$$a \cdot b = \begin{cases} 2a \cdot \frac{b}{2} & \text{falls } b \text{ gerade,} \\ a + 2a \cdot \frac{b-1}{2} & \text{falls } b \text{ ungerade.} \end{cases}$$

# Termination

$$a \cdot b = \begin{cases} a & \text{falls } b = 1, \\ 2a \cdot \frac{b}{2} & \text{falls } b \text{ gerade,} \\ a + 2a \cdot \frac{b-1}{2} & \text{falls } b \text{ ungerade.} \end{cases}$$

# Recursively, Functional

$$f(a, b) = \begin{cases} a & \text{falls } b = 1, \\ f(2a, \frac{b}{2}) & \text{falls } b \text{ gerade,} \\ a + f(2a, \frac{b-1}{2}) & \text{falls } b \text{ ungerade.} \end{cases}$$

# Implemented

```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    else if (b%2 == 0)
        return f(2*a, b/2);
    else
        return a + f(2*a, (b-1)/2);
}
```

# Correctnes

$$f(a, b) = \begin{cases} a & \text{if } b = 1, \\ f(2a, \frac{b}{2}) & \text{if } b \text{ even,} \\ a + f(2a \cdot \frac{b-1}{2}) & \text{if } b \text{ odd.} \end{cases}$$

Remaining to show:  $f(a, b) = a \cdot b$  for  $a \in \mathbb{Z}$ ,  $b \in \mathbb{N}^+$ .



# Proof by induction

Base clause:  $b = 1 \Rightarrow f(a, b) = a = a \cdot 1$ .

Hypothesis:  $f(a, b') = a \cdot b'$  für  $0 < b' \leq b$

Step:  $f(a, b + 1) \stackrel{!}{=} a \cdot (b + 1)$

$$f(a, b + 1) = \begin{cases} f(2a, \overbrace{\frac{b+1}{2}}^{\leq b}) = a \cdot (b + 1) & \text{if } b \text{ odd,} \\ a + f(2a, \underbrace{\frac{b}{2}}_{\leq b}) = a + a \cdot b & \text{if } b \text{ even.} \end{cases}$$



# End Recursion

The recursion can be written as *end recursion*

```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    else if (b%2 == 0)
        return f(2*a, b/2);
    else
        return a + f(2*a, (b-1)/2);
}
```



```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    int z=0;
    if (b%2 != 0){
        --b;
        z=a;
    }
    return z + f(2*a, b/2);
}
```

# End-Recursion $\Rightarrow$ Iteration

```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    int z=0;
    if (b%2 != 0){
        --b;
        z=a;
    }
    return z + f(2*a, b/2);
}
```



```
int f(int a, int b) {
    int res = 0;
    while (b != 1) {
        int z = 0;
        if (b % 2 != 0){
            --b;
            z = a;
        }
        res += z;
        a *= 2; // neues a
        b /= 2; // neues b
    }
    res += a; // Basisfall b=1
    return res;
}
```

# Simplify

```
int f(int a, int b) {  
    int res = 0;  
    while (b != 1) {  
        int z = 0;  
        if (b % 2 != 0){  
            --b;  $\longrightarrow$  Teil der Division  
            z = a;  $\longrightarrow$  Direkt in res  
        }  
        res += z;  
        a *= 2;  
        b /= 2;  
    }  
    res += a;  $\longrightarrow$  in den Loop  
    return res;  
}
```



```
// pre: b>0  
// post: return a*b  
int f(int a, int b) {  
    int res = 0;  
    while (b > 0) {  
        if (b % 2 != 0)  
            res += a;  
        a *= 2;  
        b /= 2;  
    }  
    return res;  
}
```

# Invariants!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    -----
    while (b > 0) {
        if (b % 2 != 0){
            -----
            res += a;
            --b;
            -----
        }
        a *= 2;
        b /= 2;
        -----
    }
    return res;
}
```

Sei  $x = a \cdot b$ .

here:  $x = a \cdot b + res$

if here  $x = a \cdot b + res \dots$

$\dots$  then also here  $x = a \cdot b + res$   
 $b$  even

here:  $x = a \cdot b + res$

here:  $x = a \cdot b + res$  und  $b = 0$

Also  $res = x$ .

# Conclusion

The expression  $a \cdot b + res$  is an *invariant*

- Values of  $a$ ,  $b$ ,  $res$  change but the invariant remains basically unchanged
- The invariant is only temporarily discarded by some statement but then re-established
- If such short statement sequences are considered atomic, the value remains indeed invariant
- In particular the loop contains an invariant, called *loop invariant* and operates there like the induction step in induction proofs.
- Invariants are obviously powerful tools for proofs!

# Further simplification

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    while (b > 0) {
        if (b % 2 != 0)
            res += a;
        a *= 2;
        b /= 2;
    }
    return res;
}
```



```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    while (b > 0) {
        res += a * (b%2);
        a *= 2;
        b /= 2;
    }
    return res;
}
```

# Analysis

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    while (b > 0) {
        res += a * (b%2);
        a *= 2;
        b /= 2;
    }
    return res;
}
```

Ancient Egyptian Multiplication corresponds to the school method with radix 2.

$$\begin{array}{r} 1\ 0\ 0\ 1 \times 1\ 0\ 1\ 1 \\ \hline \phantom{1\ 0\ 0\ 1} 1\ 0\ 0\ 1 \quad (9) \\ \phantom{1\ 0\ 0\ 1} 1\ 0\ 0\ 1 \quad (18) \\ \hline \phantom{1\ 0\ 0\ 1} 1\ 1\ 0\ 1\ 1 \\ \phantom{1\ 0\ 0\ 1} 1\ 0\ 0\ 1 \quad (72) \\ \hline 1\ 1\ 0\ 0\ 0\ 1\ 1 \quad (99) \end{array}$$



# Efficiency

Question: how long does a multiplication of  $a$  and  $b$  take?

- Measure for efficiency

- Total number of fundamental operations: double, divide by 2, shift, test for “even”, addition
- In the recursive code: maximally 6 operations per call

- Essential criterion:

- Number of recursion calls or
- Number iterations (in the iterative case)

- $\frac{b}{2^n} \leq 1$  holds for  $n \geq \log_2 b$ . Consequently not more than  $6 \lceil \log_2 b \rceil$  fundamental operations.

# 1.5 Fast Integer Multiplication

[Ottman/Widmayer, Kap. 1.2.3]

## Example 2: Multiplication of large Numbers

Primary school:

<i>a</i>	<i>b</i>		<i>c</i>	<i>d</i>	
6	2	·	3	7	
			1	4	<i>d · b</i>
			4	2	<i>d · a</i>
			6		<i>c · b</i>
	1	8			<i>c · a</i>
=	2	2	9	4	

$2 \cdot 2 = 4$  single-digit multiplications.  $\Rightarrow$  Multiplication of two  $n$ -digit numbers:  $n^2$  single-digit multiplications

# Observation

$$\begin{aligned}ab \cdot cd &= (10 \cdot a + b) \cdot (10 \cdot c + d) \\&= 100 \cdot a \cdot c + 10 \cdot a \cdot c \\&\quad + 10 \cdot b \cdot d + b \cdot d \\&\quad + 10 \cdot (a - b) \cdot (d - c)\end{aligned}$$

# Improvement?

$a$	$b$		$c$	$d$	
6	2	.	3	7	
<hr/>					
			1	4	$d \cdot b$
			1	4	$d \cdot b$
			1	6	$(a - b) \cdot (d - c)$
			1	8	$c \cdot a$
	1	8			$c \cdot a$
<hr/>					
=	2	2	9	4	

→ 3 single-digit multiplications.

# Large Numbers

$$6237 \cdot 5898 = \underbrace{62}_{a'} \underbrace{37}_{b'} \cdot \underbrace{58}_{c'} \underbrace{98}_{d'}$$

Recursive / inductive application: compute  $a' \cdot c'$ ,  $a' \cdot d'$ ,  $b' \cdot c'$  and  $b' \cdot d'$  as shown above.

→  $3 \cdot 3 = 9$  instead of 16 single-digit multiplications.

# Generalization

Assumption: two numbers with  $n$  digits each,  $n = 2^k$  for some  $k$ .

$$\begin{aligned}(10^{n/2}a + b) \cdot (10^{n/2}c + d) &= 10^n \cdot a \cdot c + 10^{n/2} \cdot a \cdot c \\ &+ 10^{n/2} \cdot b \cdot d + b \cdot d \\ &+ 10^{n/2} \cdot (a - b) \cdot (d - c)\end{aligned}$$

Recursive application of this formula: algorithm by Karatsuba and Ofman (1962).

# Analysis

$M(n)$ : Number of single-digit multiplications.

Recursive application of the algorithm from above  $\Rightarrow$  recursion equality:

$$M(2^k) = \begin{cases} 1 & \text{if } k = 0, \\ 3 \cdot M(2^{k-1}) & \text{if } k > 0. \end{cases}$$



# Iterative Substitution

Iterative substitution of the recursion formula in order to guess a solution of the recursion formula:

$$\begin{aligned}M(2^k) &= 3 \cdot M(2^{k-1}) = 3 \cdot 3 \cdot M(2^{k-2}) = 3^2 \cdot M(2^{k-2}) \\ &= \dots \\ &\stackrel{!}{=} 3^k \cdot M(2^0) = 3^k.\end{aligned}$$

# Proof: induction

*Hypothesis H:*

$$M(2^k) = 3^k.$$

*Base clause ( $k = 0$ ):*

$$M(2^0) = 3^0 = 1. \quad \checkmark$$

*Induction step ( $k \rightarrow k + 1$ ):*

$$M(2^{k+1}) \stackrel{\text{def}}{=} 3 \cdot M(2^k) \stackrel{\text{H}}{=} 3 \cdot 3^k = 3^{k+1}.$$



# Comparison

Traditionally  $n^2$  single-digit multiplications.

Karatsuba/Ofman:

$$M(n) = 3^{\log_2 n} = (2^{\log_2 3})^{\log_2 n} = 2^{\log_2 3 \log_2 n} = n^{\log_2 3} \approx n^{1.58}.$$

Example: number with 1000 digits:  $1000^2/1000^{1.58} \approx 18$ .

# Best possible algorithm?

We only know the upper bound  $n^{\log_2 3}$ .

There are (for large  $n$ ) practically relevant algorithms that are faster.  
The best upper bound is not known.

Lower bound:  $n/2$  (each digit has to be considered at least once)

## 1.6 Finde den Star

# Is this constructive?

Exercise: find a faster multiplication algorithm.

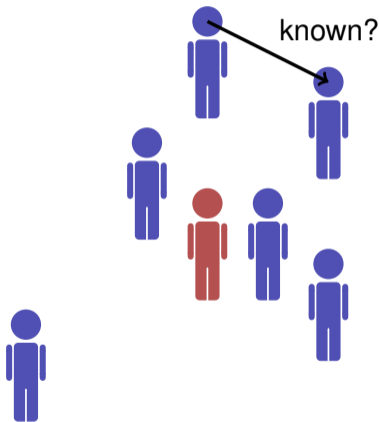
Unsystematic search for a solution  $\Rightarrow$  .

Let us consider a more constructive example.

## Example 3: find the star!

Room with  $n > 1$  people.

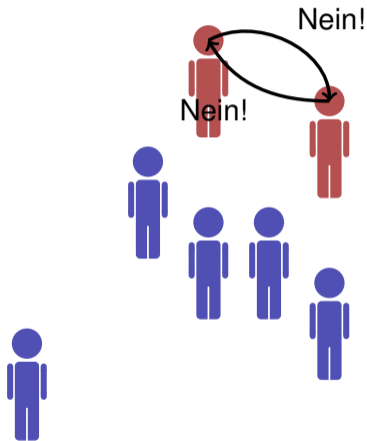
- *Star*: Person that does not know anyone but is known by everyone.
- *Fundamental operation*: Only allowed question to a person  $A$ : "Do you know  $B$ ?" ( $B \neq A$ )



# Problemeigenschaften

- Possible: no star present
- Possible: one star present
- More than one star possible?

Assumption: two stars  $S_1, S_2$ .  
 $S_1$  knows  $S_2 \Rightarrow S_1$  no star.  
 $S_1$  does not know  $S_2 \Rightarrow S_2$  no star.  $\perp$





# Naive solution

Ask everyone about everyone

Result:

	1	2	3	4
1	-	yes	no	no
2	no	-	no	no
3	yes	yes	-	no
4	yes	yes	yes	-

Star is 2.

Numer operations (questions):  $n \cdot (n - 1)$ .

# Better approach?

Induction: partition the problem into smaller pieces.

- $n = 2$ : Two questions suffice
- $n > 2$ : Send one person out. Find the star within  $n - 1$  people.  
Then check  $A$  with  $2 \cdot (n - 1)$  questions.

Overall

$$F(n) = 2(n - 1) + F(n - 1) = 2(n - 1) + 2(n - 2) + \dots + 2 = n(n - 1).$$

No benefit. 😞

# Improvement

Idea: avoid to send the star out.

- Ask an arbitrary person  $A$  if she knows  $B$ .
- If yes:  $A$  is no star.
- If no:  $B$  is no star.
- At the end 2 people remain that might contain a star. We check the potential star  $X$  with any person that is out.

# Analyse

$$F(n) = \begin{cases} 2 & \text{for } n = 2, \\ 1 + F(n-1) + 2 & \text{for } n > 2. \end{cases}$$

Iterative substitution:

$$F(n) = 3 + F(n-1) = 2 \cdot 3 + F(n-2) = \dots = 3 \cdot (n-2) + 2 = 3n - 4.$$

Proof: exercise!

# Moral

With many problems an inductive or recursive pattern can be developed that is based on the piecewise simplification of the problem. Next example in the next lecture.