Algorithmic Discrete Mathematics

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Service team

- Lecture: PD Dr. Ulf Lorenz
  Tuesday 11:30 bis 15:10
  Email: lorenz@mathematik.tu-darmstadt.de
  Office 35, Dolivostr. 15

- Exercise chief instructor: Dipl. Math. David Meffert
  Email:meffert@mathematik.tu-darmstadt.de

- Exercises:

Thursday	14:25 to 16:05	every two weeks	start 19.04.2012	S103/110
Thursday	14:25 to 16:05	every two weeks	start 19.04.2012	S103/113
Thursday	16:15 to 17:55	every two weeks	start 19.04.2012	S103/113
Thursday	16:15 to 17:55	every two weeks	start 19.04.2012	S103/164
Thursday	16:15 to 17:55	every two weeks	start 19.04.2012	S103/175
Wednesday	8:00 to 9:40	every two weeks	start 18.04.2012	S103/164

Module description

- References:
  - M. Aigner: Diskrete Mathematik, Vieweg
  - **T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein: Introduction to Algorithms**
  - **Lecture notes of 2010 (in german)**
  - Further material (partially in advance, in the web), simplifies taking notes, **cannot compensate for a teaching book!**
- Written exam: 60 minutes
- 4,5 (5,0) ECTS, 2V + 1Ü
- **General concepts**: Growth of functions and asymptotic complexity.
  Graph theory: Euler graphs, spanning trees, shortest paths, Travelling-Salesman-Problem.
  Search problems: Sorting, decision trees.
  Coding/Cryptography: Huffman-encoding, RSA-algorithm.
  Further topics (examples): Matchings in bipartite graphs, flow algorithmens.

Organisation

- **exercise groups:** registration via TUCaN
  begin of exercises: Wednesday 18.4.2012

- **exercises, procedure:**
  release dates: Tuesday 19:00 (in the Web, deadline on exercise sheet)
  delivery dates: usually next Tuesday after the reading

- **assessment:** The grade of the written exam can be upgraded via active participation in exercises and with the help of solving ‚many' of the exercise items. Partially, the exam will contain multiple-choice items about the homework.
- **Working in small groups is recommended**
  (groups with up to 4 students are allowed to work together!)

Organization

• 3 or 4 items per sheet. In irregular intervals:
programing exercises with longer lasting handling time.
Additionally: simplified items in the exercises

• If necessery, the exercises will deal more extensively with the lecture,
otherwise small tasks in small groups.

! You can only learn by “do it yourself“,
only “read + listen“ is not enough!

Contents

- Introduction
- Complexity theory
  - Data structures and encoding schemes
  - Algorithms
  - Asymptotic notation, upper and lower bounds
  - Complexity classes P, NP; NP-complete problems
- Algorithms for graphs
  - DFS algorithm
  - Greedy-algorithms (e.g.: computing spanning trees)
  - Dijkstra, Moore-Bellmann (shortest paths on graphs)
  - Ford-Fulkerson (maximum flows in networks, matchings)
- Abstract data types (stack, queue, heap) and again DFS, BFS and Dijkstra
- Sorting on arrays
  - Mergesort, Quicksort, Heapsort
  - Divide-and-Conquer
  - Lower complexity bounds for sorting by comparison

Introduction

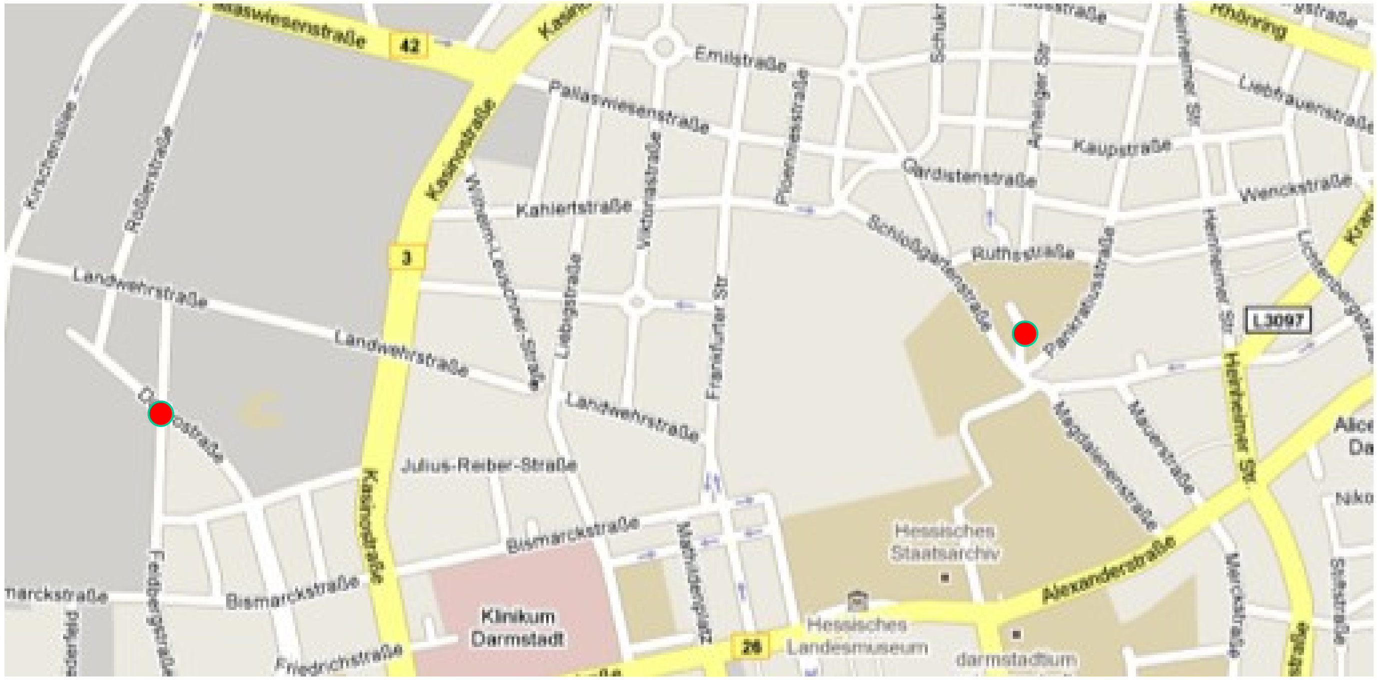
Algorithmic every day problems:

- How does a navigation system find “good“ connections?
- How can we find optimal paths for gas in a gas distribution system?
- How does e.g. Lufthansa optimize its aircraft assignments?

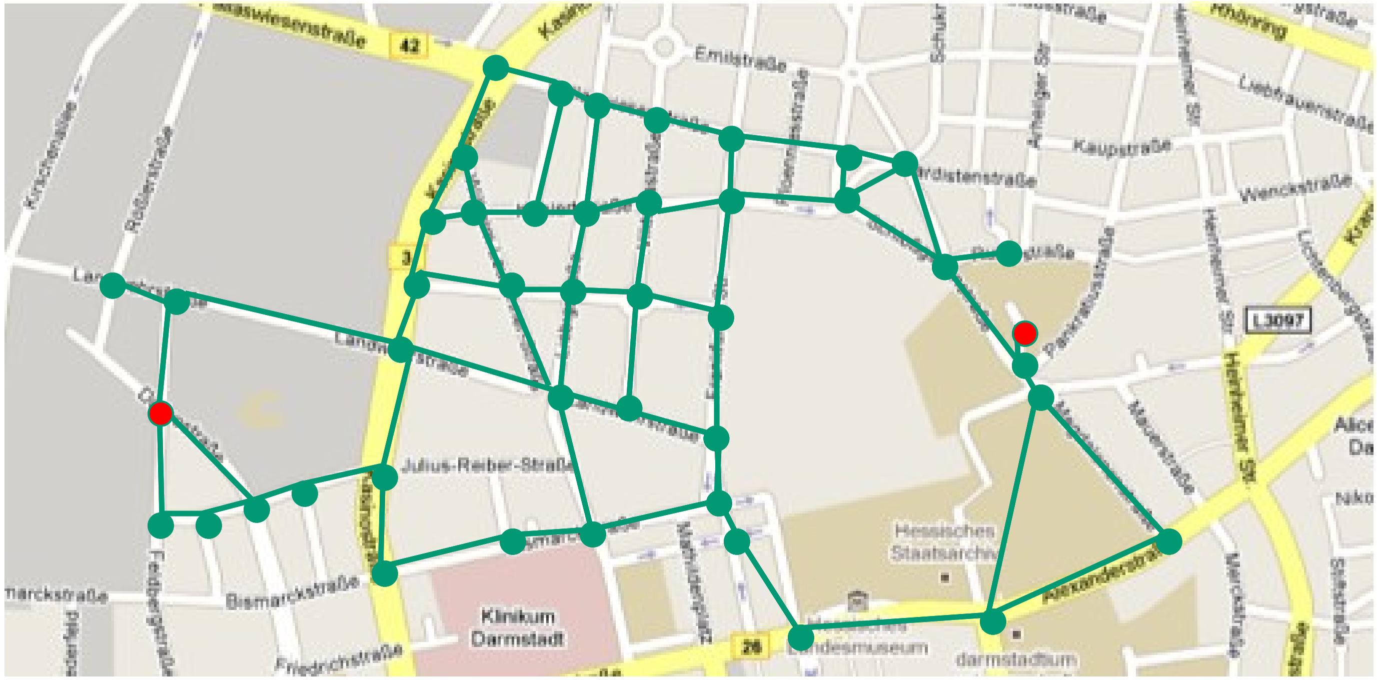
Algorithm, informal explanation:

A sequence of easy understandable actions to perform,
on the right level of abstraction.

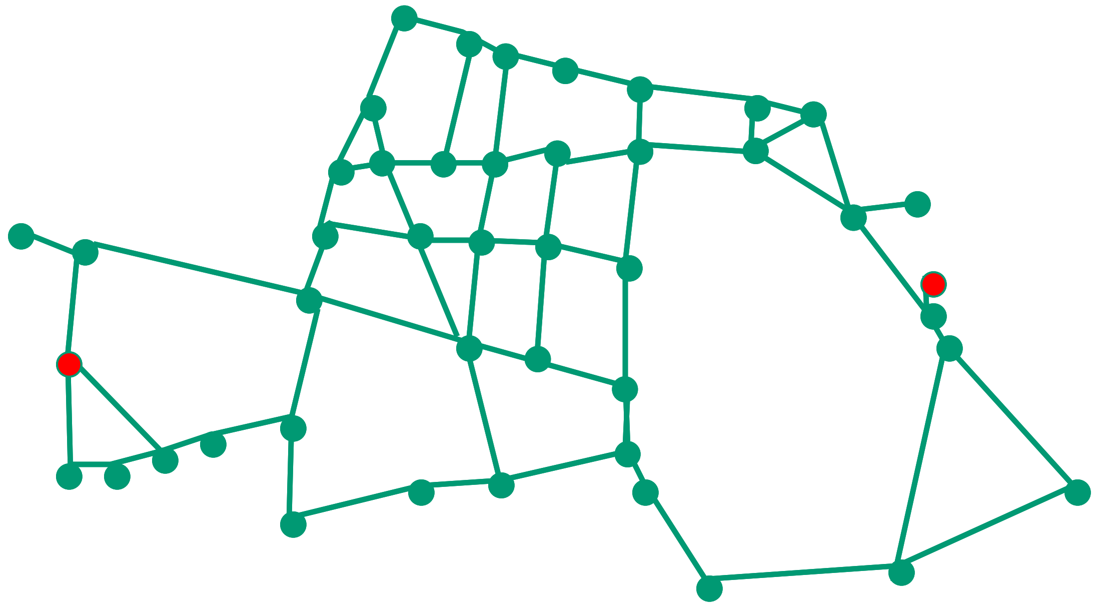
Introduction, “good” connections



Introduction, “good” connections

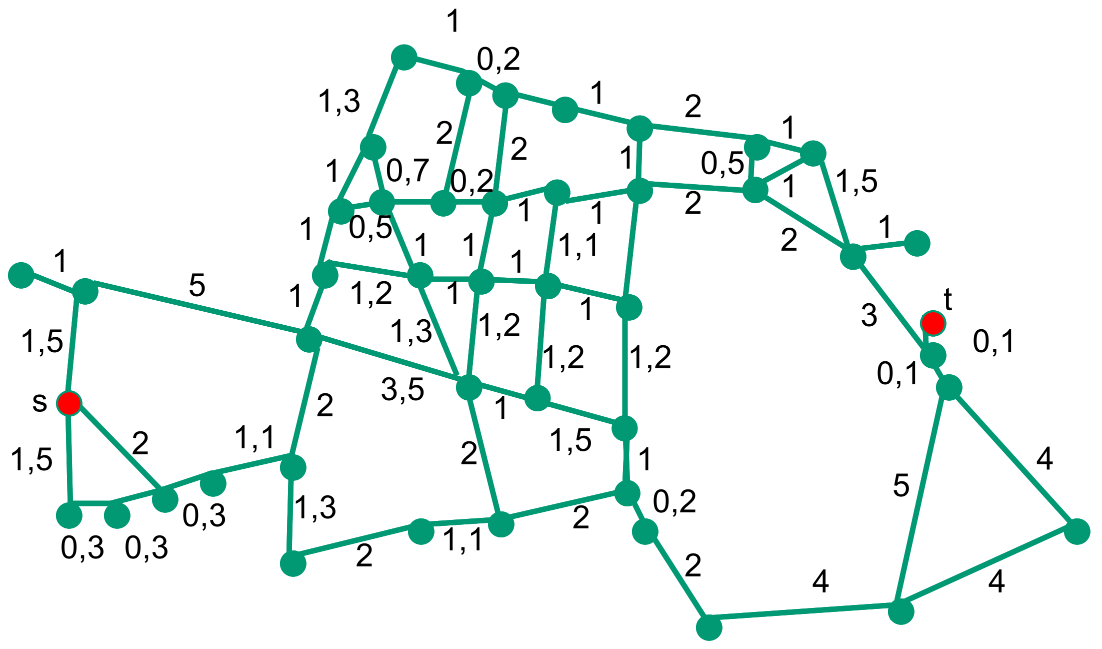


Introduction, “good” connections



• street map→ graph; problem: find shortest path between start and end points
• This graph might describe a gas network, as well; problem then e.g.: what is the best opportunity to distribute the gas to some consumers?

Introduction, “good” connections



- street map→ graph; problem: find shortest path between start and end points
- This graph might describe a gas network, as well; problem then e.g.: what is the best opportunity to distribute the gas to some consumers?

Optimization in airline industries

Market Modeling

Fleet Assignment

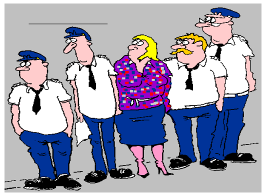
Crew Pairing

Network Design

Operation Control

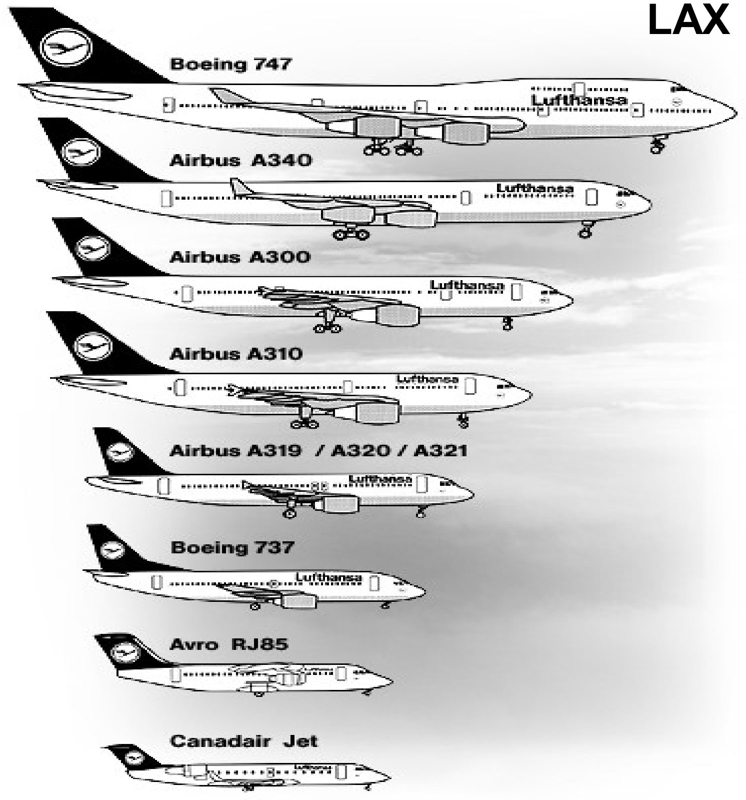
Revenue Management

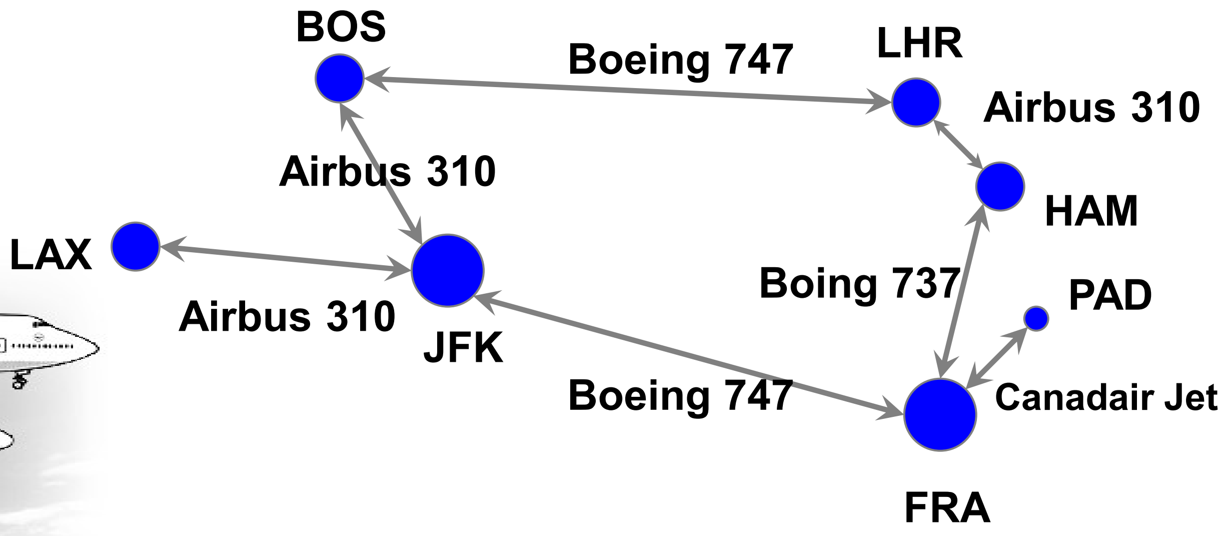
Aircraft Rotation



Crew Rostering

Fleet Assignment





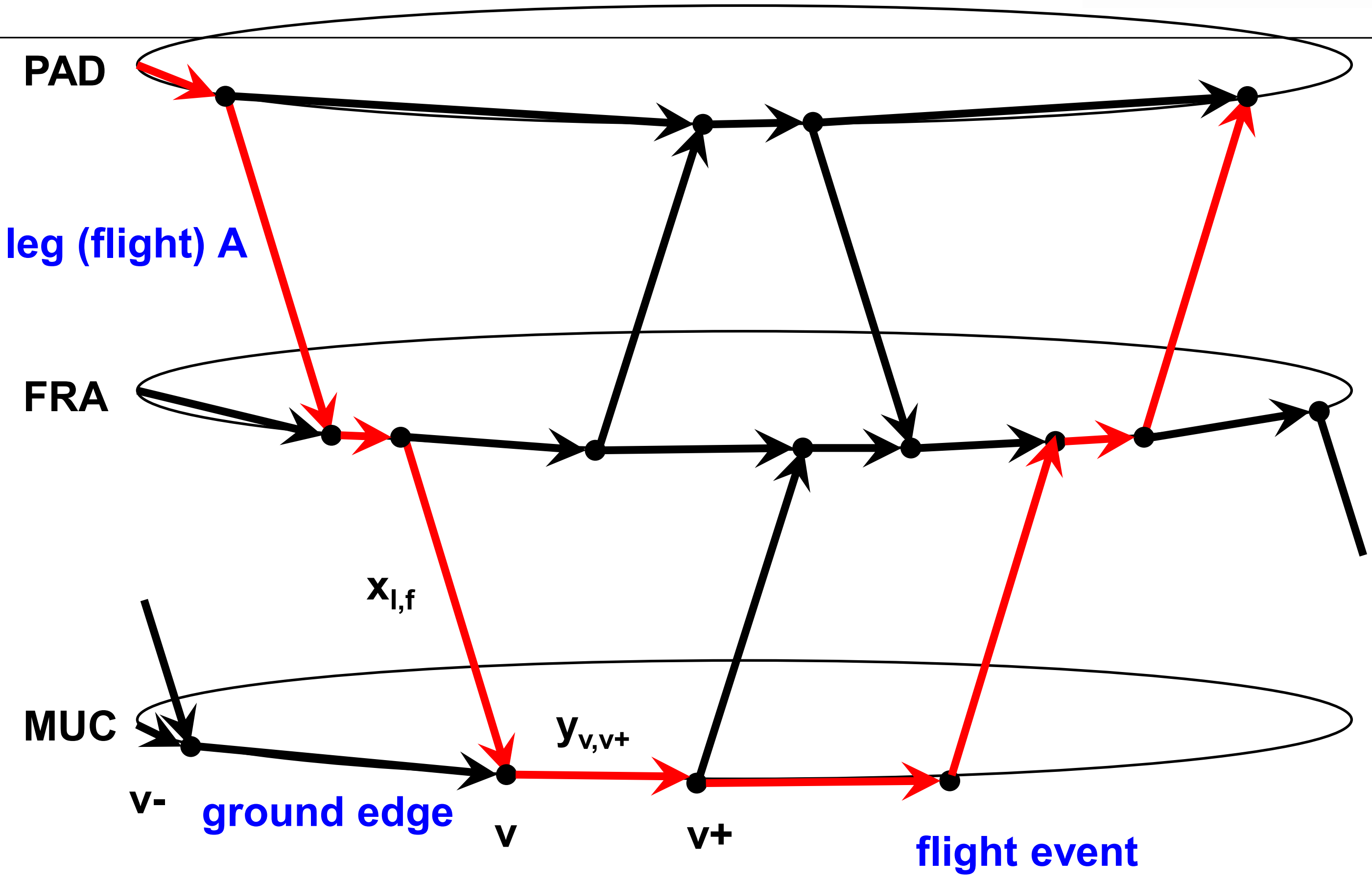
Assignment of aircraft types to so called legs

- Maximize the earnings

Restrictions:

- limited amount of aircrafts
- balanced flight plan
- lots of operative restrictions

Time-Space Network



Weekly planning with up to 10.000 flight events, 10-23 aircraft types

Linear Program for Fleet Assignment

($x_{l,f}$ = 1) ⇔ (leg l is operated with fleet f)

$y_{v,v+}$: number of waiting aircrafts bewteen two flight events

$$\text{maximize} \quad \sum_{l\in\mathcal{L}} \sum_{f\in\mathcal{F}} p_{l,f} \cdot x_{l,f}$$

subject to

$$\begin{array}{rcll}
\sum_{f\in\mathcal{F}} x_{l,f} & = & 1 & \forall\, l \in \mathcal{L} \\
\sum_{l_f^{arr}=v} x_{l,f} - \sum_{l_f^{dep}=v} x_{l,f} + y_{v^-,v} - y_{v,v^+} & = & 0 & \forall\, v \in V \\
\sum_{l_f\in E_{F0}^f} x_{l,f} + \sum_{(v,v^+)\in E_{G0}^f} y_{v,v^+} & \leq & size_f & \forall f \in \mathcal{F} \\
x_{l,f} & \in & \{0,1\} & \forall\, l \in \mathcal{L}; \forall\, f \in \mathcal{F} \\
y_{v,v^+} & \geq & 0 & \forall\, (v,v^+) \in E_G
\end{array}$$

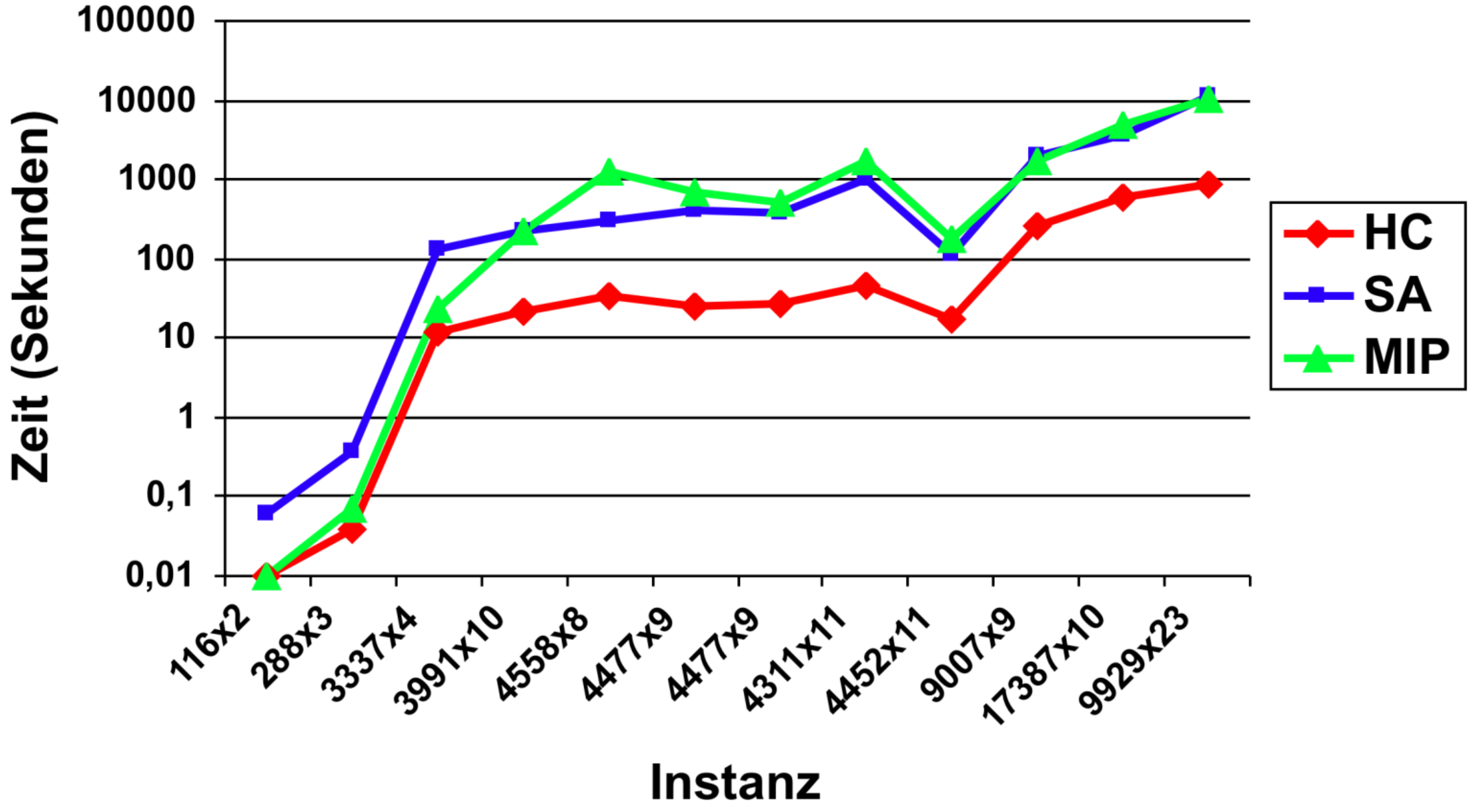
- Mathematical Model, Linear Programm → Optimization I - III
- Model size:

  |legs|*|fleets| integer variables, 2*|legs|*|fleets| constraints

  thus about 230,000 variables, 500,000 constraints
- solution times for gettimg exact solutions took too long at that time (ca. year 2k)

Results for Fleet Assignment, Heuristics

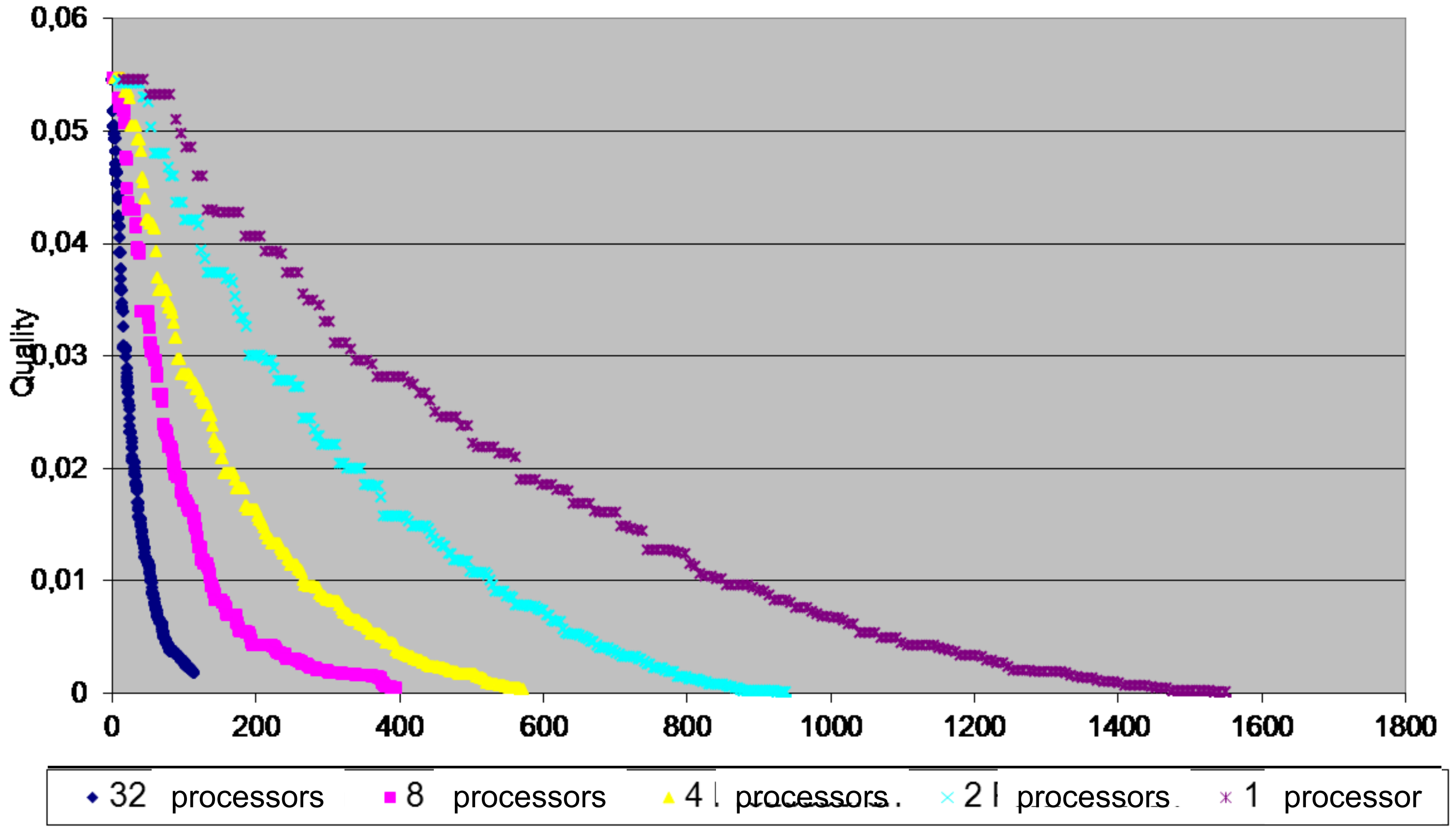
TECHNISCHE UNIVERSITÄT DARMSTADT



Verfahren	HC	SA	MIP
∅-Lösungsqualität	98,5%	99,7%	>99,9%

Parallel Simulated Annealing

TECHNISCHE UNIVERSITÄT DARMSTADT

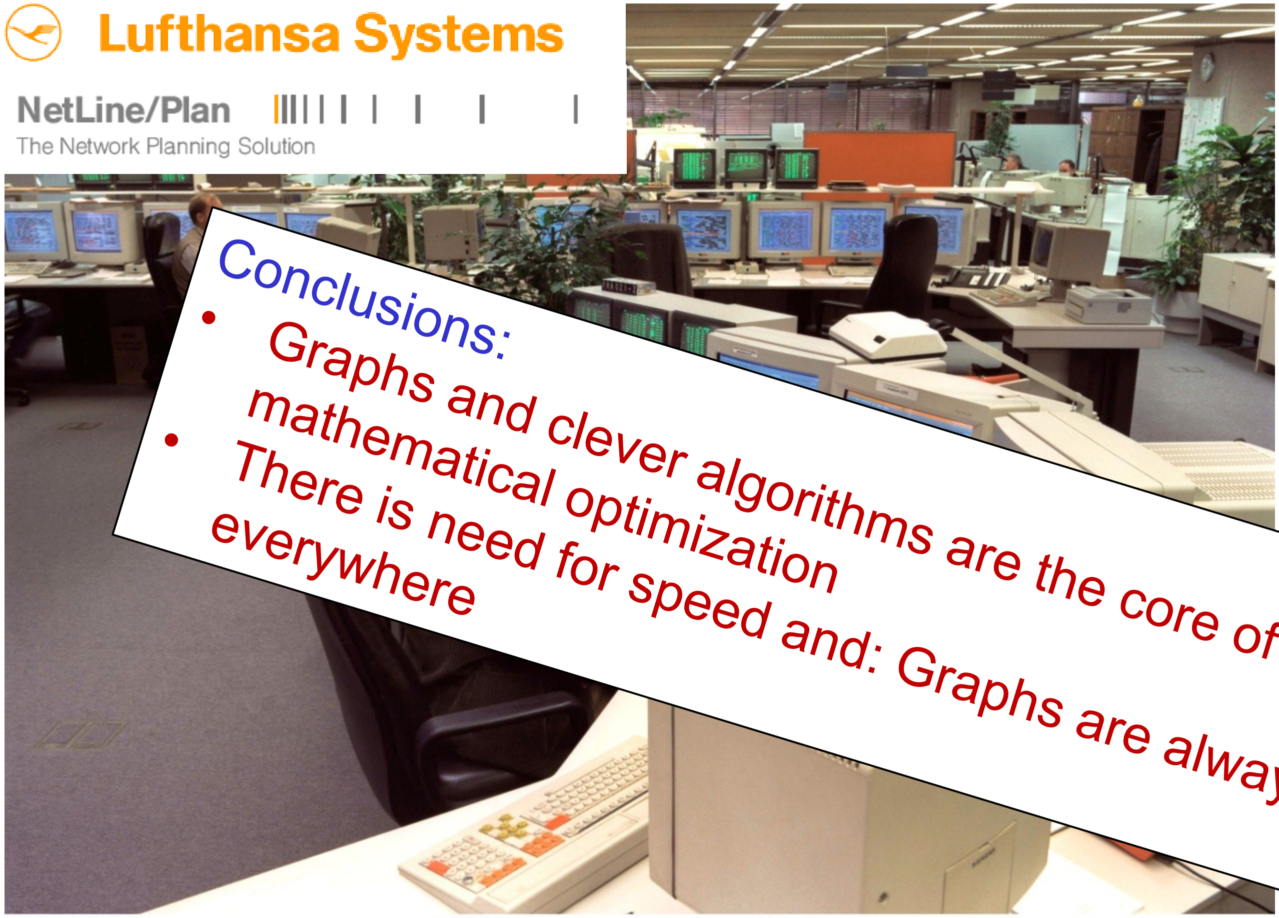


Important for practice is speed

Lufthansa Systems

NetLine/Plan

The Network Planning Solution



Lufthansa

AIR CANADA

Conclusions:

- Graphs and clever algorithms are the core of successful mathematical optimization
- There is need for speed and: Graphs are always and everywhere

First algorithmic example

TECHNISCHE UNIVERSITÄT DARMSTADT

- The max sum problem

  Input: Sequence $a_1,...,a_n$ of integer numbers. Let $f(i,j) := a_i+a_{i+1}+...+a_j$,
  for $1 \leq i \leq j \leq n$.
  Desired: maximized $f(i,j)$.

  For an efficiency analysis, we only count at this place
  – the number of used comparisons V(n) beteen numbers and
  – the number of used arithmetic operations.
  – The runtime is defined as T(n) = A(n) + V(n).

In the following, 4 different algorithms are presented and analysed in detail.

First algorithmic example

- **Algorithm 1 (naive algorithm)**
  1. compute all f(i,j), one after the other
  2. choose the one with largest f(i,j)

Example: given. (3,-2,4,-5)

f(1,1) = 3	f(1,2) = 3-2	f(1,3) = 3-2+4	f(1,4) = 3-2+4-5
f(2,2) = -2	f(2,3) = -2+4	f(2,4) = -2+4-5	
f(3,3) = 4	f(3,4) = 4-5		
f(4,4) = -5			

First algorithmic example

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Algorithm 1 (naive algorithm)**
  1. compute all f(i,j), *// f(1,1), f(1,2), f(1,3)... f(1,n), f(2,1)...*
  2. choose the largest f(i,j)

Analysis: We need j-i sumations in order to compute *f(i,j)*.

$$A(n) = \sum_{1\le i\le n}\ \sum_{i\le j\le n} \quad -i) = \sum_{1\le i\le n}\ \sum_{0\le k\le n-i} = \text{L} = \frac{1}{6}n^3 - \frac{1}{6}n$$

In order to determine the maximum of L numbers, L-1 comparisons are needed. Here, it is L = [redacted]. (why?) Therefore,

$$V(n) = \left(\quad\right) + \left(\quad\right) - 1 = -1 + \sum_{1} = \text{[redacted]} - 1 = \frac{1}{2}n^2 + \frac{1}{2}n - 1$$

$$T(n) = V(n) + A(n) = \frac{1}{6}n^3 + \frac{1}{2}n^2 + \frac{1}{3}n - 1$$

First algorithmic example

- **Algorithm 2 (normal algorithm)**
  f(i,j) can be computed more efficiently: do not compute *f(i,j+1)* completely new, but instead use that f(i,j+1) = ***f(i,j)+a$_{j+1}$*** and f(i,j) is already known.
  We compute *f(i,j)* in the following order:

$$\begin{array}{cccccc}
(a_1 = f(1,1) & f(1,2) & f(1,3) & \cdots & f(1,n) \\
 & (a_2 = f(2,2) & f(2,3) & \cdots & f(2,n) \\
 & & (a_3 = f(3,3) & \cdots & f(3,n) \\
 & & & \ddots & \vdots \\
 & & & & (a_n = f(n,n)
\end{array}$$

and thereafter compute the maximum. Now, V(n) is the same as in the naive algorithm. However, we consume only one addition in order to compute f(i,j), j > i:

$$A(n) = \sum_{i=}^{n} i - \;) = -n + \sum_{i=}^{n} i \; : \frac{1}{2} n^2 - \frac{1}{2} n$$

$$T(n) = V(n) + A(n) = n^2 -$$

First algorithmic example

- **Algorithm 3 (Divide & Conquer algorithm)**

  In order to become better: do not compute all *f(i,j)* !

  General methodology „Divide & Conquer“:

  Partition the given problem into several subproblems of the same type („divide“), solve the subproblems (recursively) and construct a solution of the original problem with the help of the two partial solutions („Conquer“).

In the following, we assume for simplicity that *n* is of the form
$n=2^k$ for some natural number k.

First algorithmic example

- For *1 ≤ a ≤ b ≤ n* we define:

$$\sigma(l,r) := \sigma(a_l,\dots,a_r) := \max\left\{ \quad (i,j), l \le i \le j \le r \right\},$$

$$s_1 := \max\left\{ \quad (i,\frac{n}{2}), \quad 1 \le i \le \frac{n}{2} \right\},$$

$$s_2 := \max\left\{ \quad (\frac{n}{2}+1,j), \quad \frac{n}{2}+1 \le j \le n \right\}.$$

Then it is valid for *σ(1,n)*:

$$\sigma(1,n) := \max\left\{ \quad (1,\frac{n}{2}), \sigma(\frac{n}{2}+1,n), s_1 + s_2 \right\}.$$

Example:

(-10,5,2,-7,	3,6,-9,11)
σ(1,4)=7	*σ(5,8)*=11
s_1=0	s_2=11

First algorithmic example

- **function $\sigma(a_1,\ldots,a_n)$**

```
    **if** $n=1$, **return** $a_n$
    **if** $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
    **return** $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```

(-10,5,2,-7, 3,6,-9,11)

s_1=? s_2=?

$\sigma_1 = ?$ σ_2=?

First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

```
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
    return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```

(-10,5,2,-7, 3,6,-9,11)	
$s_1=0$	$s_2=11$
$\sigma_1 = ?$	$\sigma_2=?$

First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

```
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
    return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```

(-10,5,2,-7, 3,6,-9,11)
$s_1=0$ $s_2=11$
$\sigma_1 = ?$ $\sigma_2=?$

(-10,5, 2,-7)
$s_1=?$ $s_2=?$
$\sigma_1=?$ $\sigma_2=?$

First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

```
  if $n=1$, return $a_n$
  if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
  return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```

(-10,5,2,-7, 3,6,-9,11)

$s_1=0$ $s_2=11$

$\sigma_1 = ?$ $\sigma_2=?$

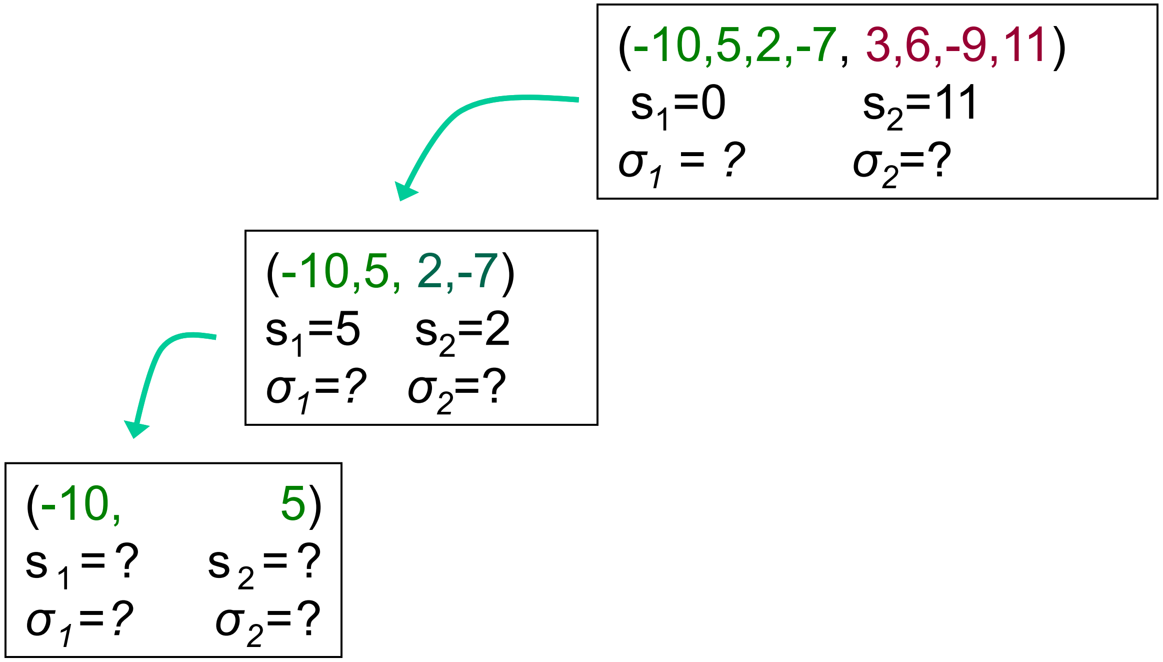
(-10,5, 2,-7)

$s_1=5$ $s_2=2$

$\sigma_1=?$ $\sigma_2=?$

First algorithmic example

```
function $\sigma(a_1,\dots,a_n)$
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\dots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\dots,a_n)$.
    return $\max\{\sigma(a_1,\dots,a_{n/2}), \sigma(a_{n/2+1},\dots,a_n), s_1+s_2\}$.
```



(-10,5,2,-7, 3,6,-9,11)
$s_1=0$ $s_2=11$
$\sigma_1 = ?$ $\sigma_2=?$

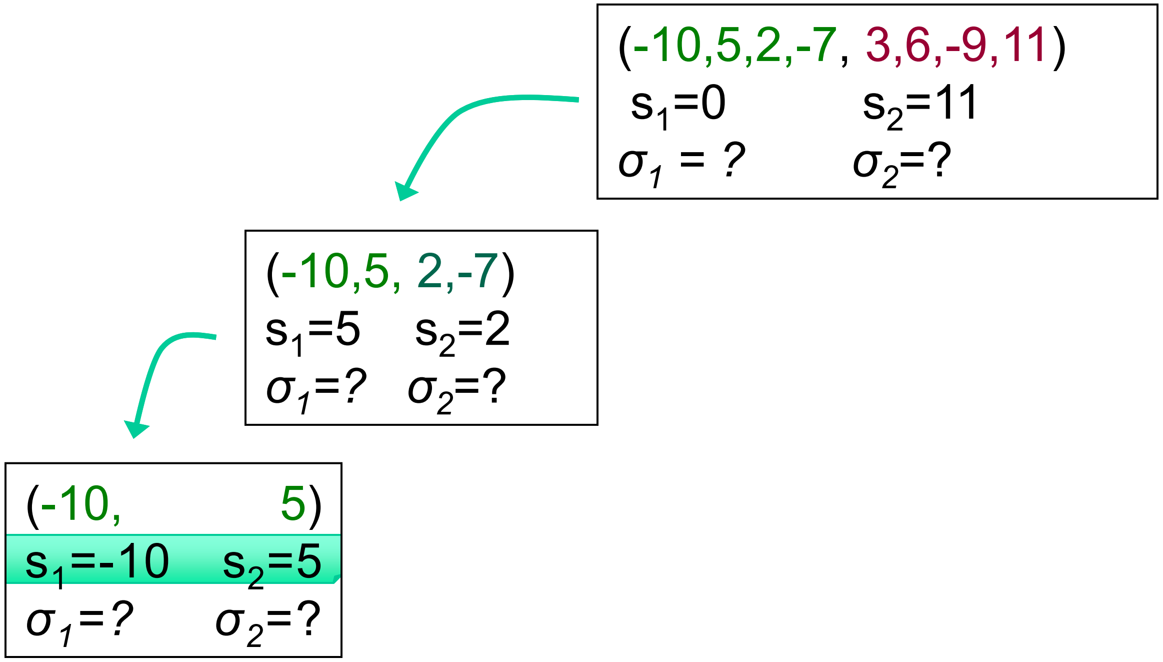
(-10,5, 2,-7)
$s_1=5$ $s_2=2$
$\sigma_1=?$ $\sigma_2=?$

(-10, 5)
$s_1=?$ $s_2=?$
$\sigma_1=?$ $\sigma_2=?$

First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

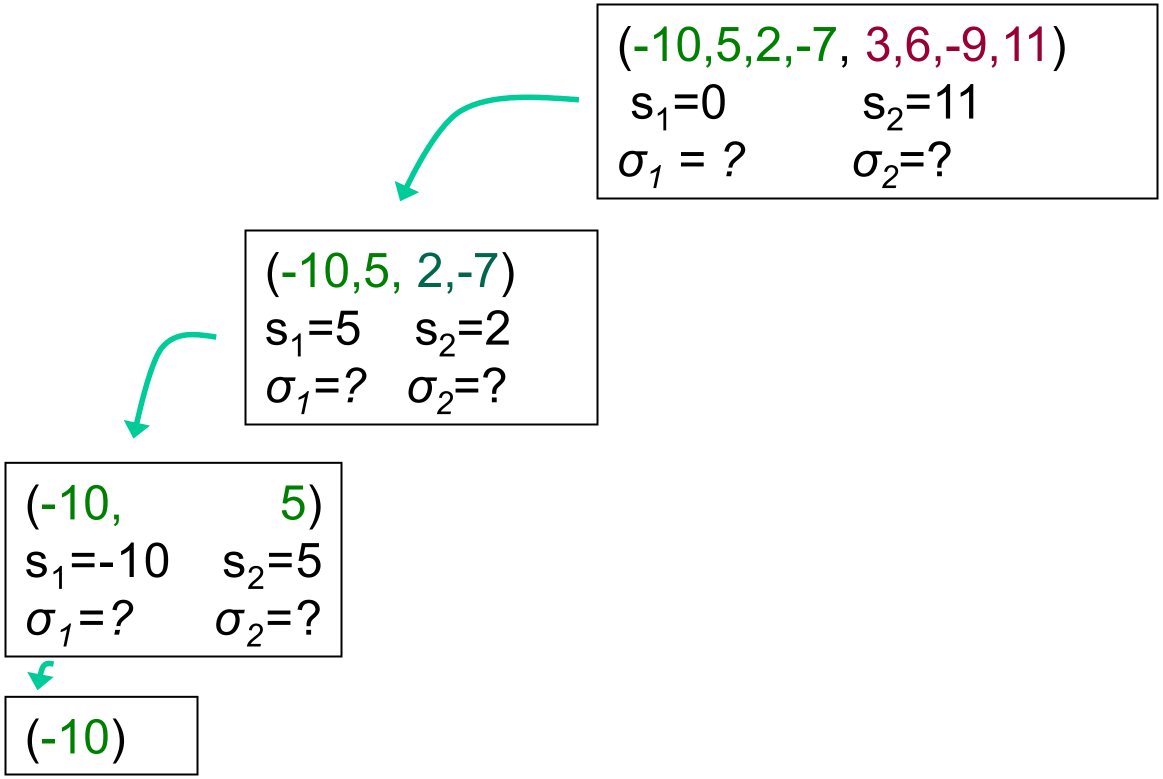
```
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
    return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```



First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

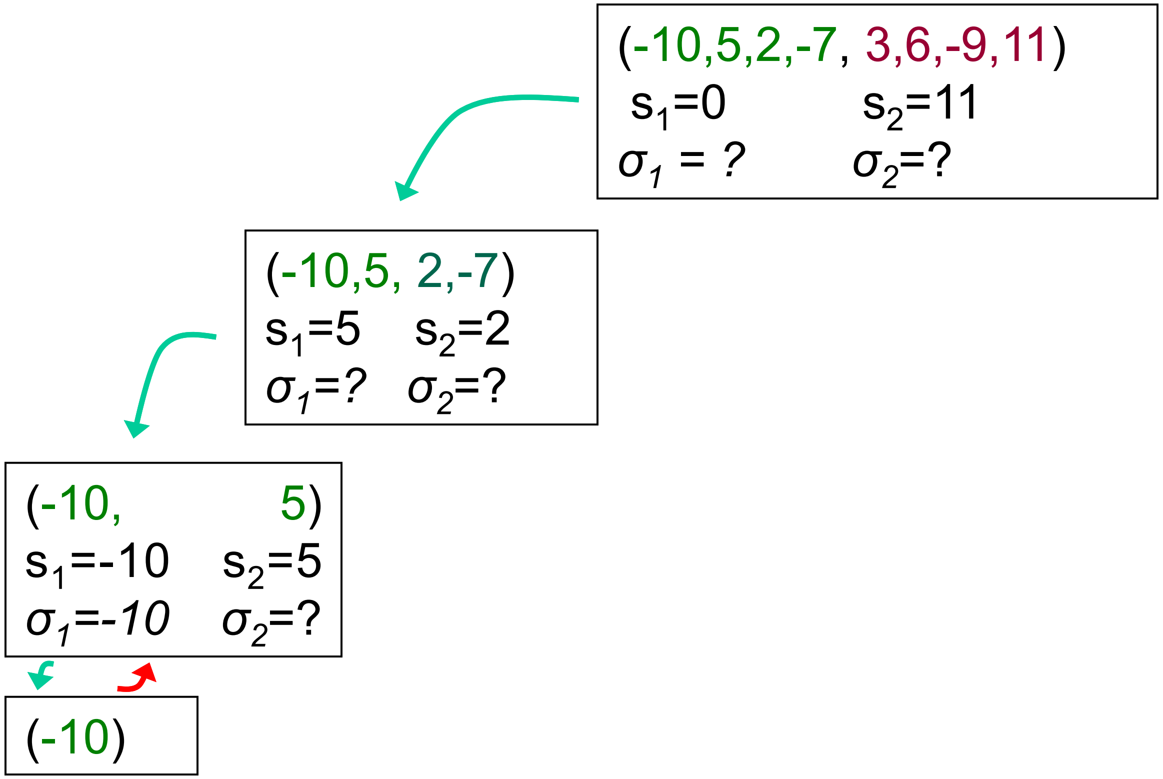
```
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
    return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```



First algorithmic example

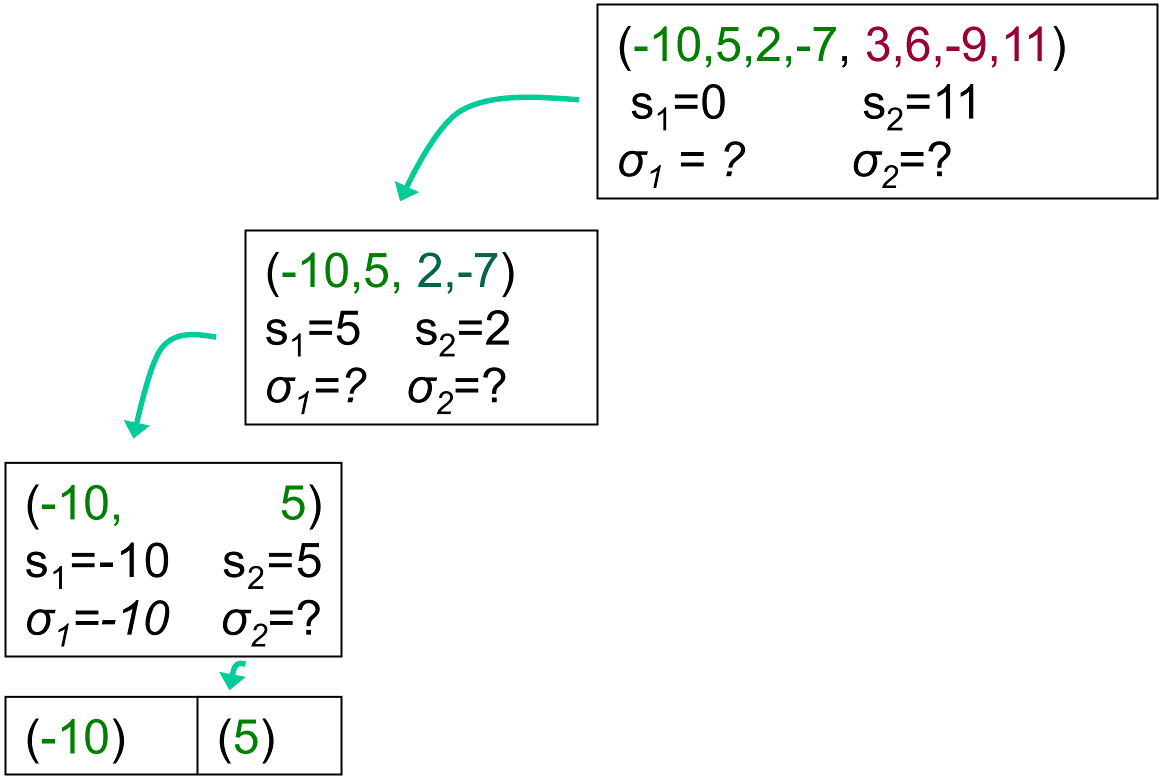
- **function** $\sigma(a_1,\dots,a_n)$

```
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\dots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\dots,a_n)$.
    return $\max\{\sigma(a_1,\dots,a_{n/2}), \sigma(a_{n/2+1},\dots,a_n), s_1+s_2\}$.
```



First algorithmic example

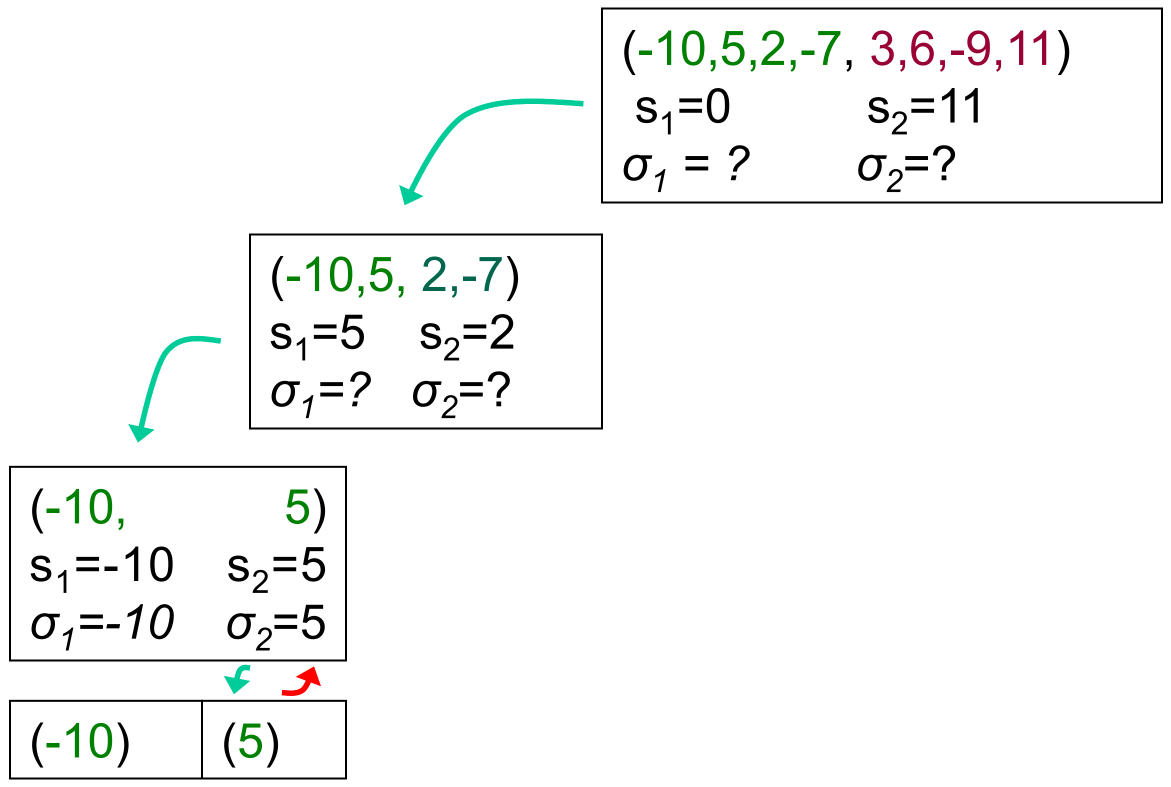
- **function** $\sigma(a_1,\ldots,a_n)$
  if $n=1$, **return** a_n
  if $n>1$, compute s_1 and s_2 as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
  return $max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.



First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

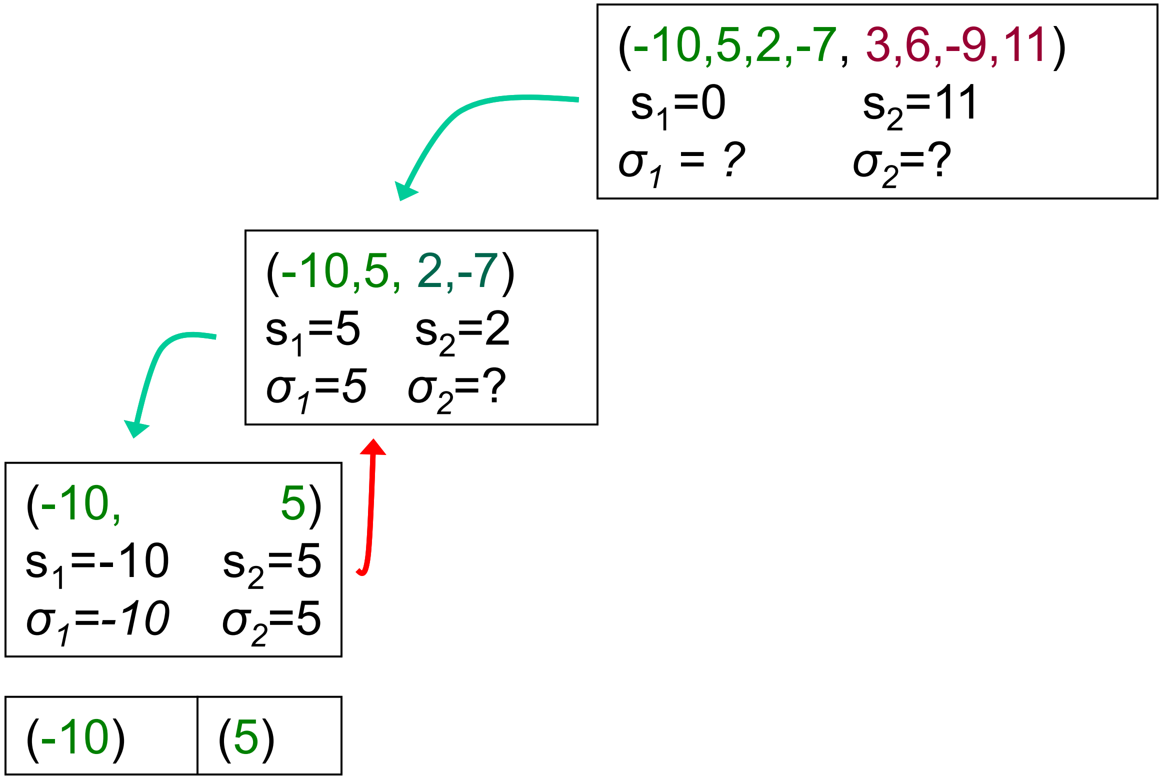
```
if $n=1$, return $a_n$
if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```



First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

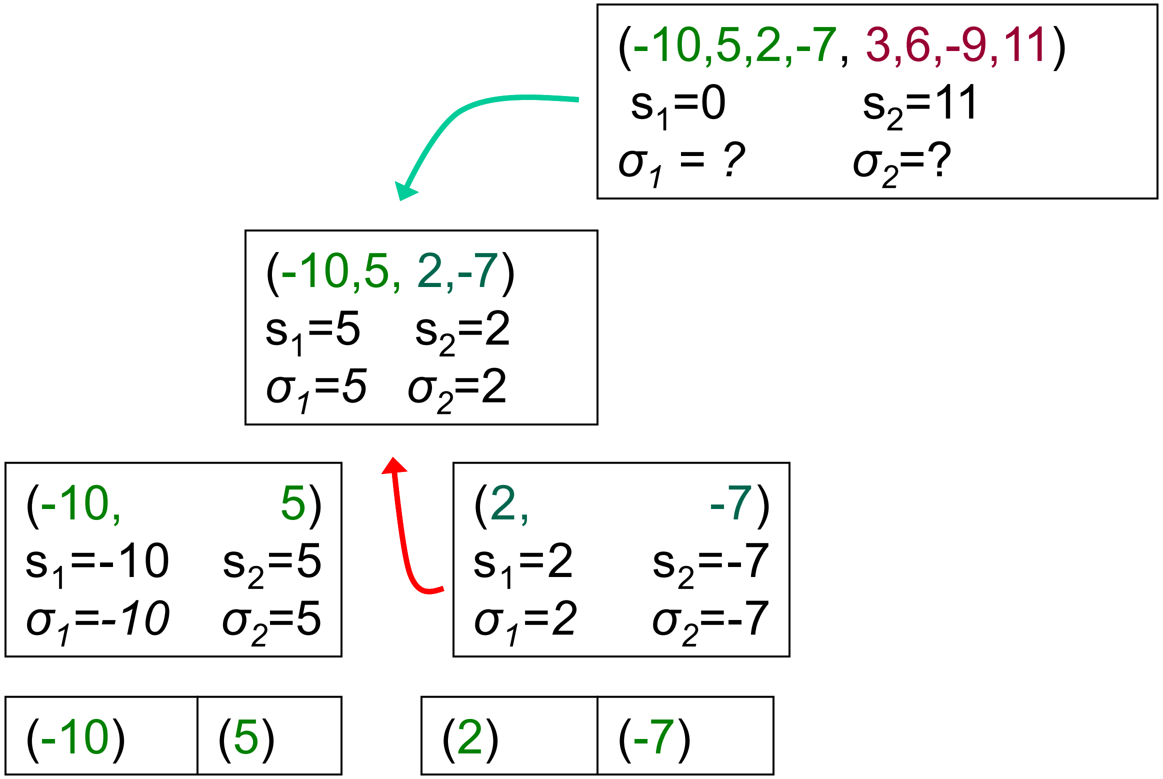
```
  if $n=1$, return $a_n$
  if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
  return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```



First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

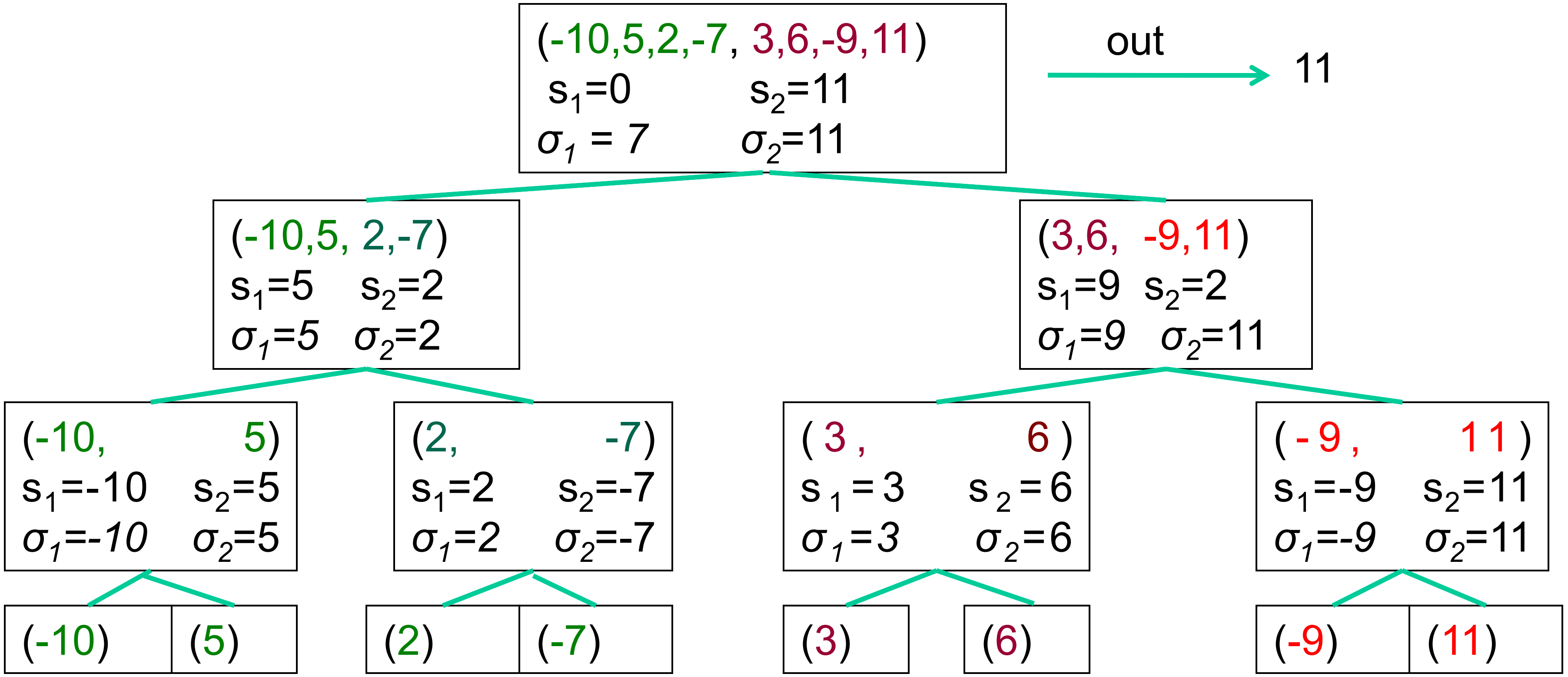
```
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
    return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```



First algorithmic example

- **function** $\sigma(a_1,\ldots,a_n)$

```
    if $n=1$, return $a_n$
    if $n>1$, compute $s_1$ and $s_2$ as well as $\sigma_1=\sigma(a_1,\ldots,a_{n/2})$ and $\sigma_2=\sigma(a_{n/2+1},\ldots,a_n)$.
    return $\max\{\sigma(a_1,\ldots,a_{n/2}), \sigma(a_{n/2+1},\ldots,a_n), s_1+s_2\}$.
```



First algorithmic example

- **Analysis:**
  let T(n) be the number of operations (comparisons + additions), consumed by the D&C algorithm on inputs of length n.

  Then: $T(1) = 0$, and for $n > 1$: $T(n)=2T(n/2)+2n-1$

  Reasoning:

  In total: $2(n/2 – 1 + n/2 -1) + 2 +1 = 2n - 1$

First algorithmic example

- **Analysis:**
  Result: a so called recursive equation / recurrence for the running time
  (for technical reasons we substitute *n* by *2^k*)

$$T(1) = \;[illegible],\ and\ for\ k \geq \;[illegible] : T(2^k) = [illegible]T(2^{k-[illegible]}) + [illegible]^{k+[illegible]} - [illegible]$$

(this is typical for recursive algorithms, especially for D&C)
Several replacements result in:

$$\begin{aligned}
T(2^k) &= 2T(2^{k-1}) + 2^{k+1} - 1 \\
&= 2(2T(2^{k-2}) + 2^k - 1) + 2^{k+1} - 1 \\
&= 4T(2^{k-2}) + (2^{k+1} - 2) + (2^{k+1} - 1) \\
&= 8T(2^{k-3}) + (2^{k+1} - 4) + (2^{k+1} - 2) + (2^{k+1} - 1)
\end{aligned}$$

First algorithmic example

- **Analysis:**

$$T(1) = \,), \; and \; for \; k \geq \; : T(2^k) = \, T(2^{k-} \,) + \; ^{k+} \; -$$

... and, as a consequence, the following conjecture:

$$T(2^k) = 2^l \, T(2^{k-l}) + \sum_{i=1}^{l} \; {}^{+1} - 2^{i-1}$$

Such that for *l*=*k* it will be valid:

$$T(2^k) = 2^k \, T(2^0) + \sum_{i=1}^{k} \; {}^{k+1} - 2^{i-1})$$

$$= 0 + k \cdot 2^{k+1} - \sum_{i=0}^{k-1} \quad = 2k \cdot 2^k - (2^k - 1)$$

$$= 2n \log_2(n) - n + 1$$

Proof: Exercise

First algorithmic example

- **Algorithm 4 (clever algorithmus)**
  (scans over the input exactly once)

```
$Max := a_1$; $Max^* := Max$;
**For** $l = 2,\dots,n$
        $Max^* := \max\{Max^* + a_l, a_l\}$
        $Max := \max\{Max^*, Max\}$
Output: Max
```

Example:

```
(-10,5,2,-7,3,6,-9,11) Max*= -10, Max = -10
(-10,5,2,-7,3,6,-9,11) Max*=   5, Max =   5
(-10,5,2,-7,3,6,-9,11) Max*=   7, Max =   7
(-10,5,2,-7,3,6,-9,11) Max*=   0, Max =   7
(-10,5,2,-7,3,6,-9,11) Max*=   3, Max =   7
(-10,5,2,-7,3,6,-9,11) Max*=   9, Max =   9
(-10,5,2,-7,3,6,-9,11) Max*=   0, Max =   9
(-10,5,2,-7,3,6,-9,11) Max*=  11, Max =  11
```

Correctness:

Claim.: after the l-th loop execution, it is

$$Max^* = \max\left\{ \quad (i,l), 1 \le i \le l \right\}$$

$$Max = \sigma(1,l) = \max\left\{ \quad (i,j), 1 \le i \le j \le l \right\}$$

First algorithmic example

Insertion, repetition: Induction proofs

Claim: A certain statement A(n) is valid for all $n \in IN$

Proof: I.) proof of the „base clause“ !

A(1) is true, is a correct statement

II.) proof of the „induction step“!

For all $n \in IN$: if A(n-1) is true, then also A(n) is true

The claim follows from I.) and II.):

A(1) is true, because of I.)

=> A(2) is true, because of II.)

=> A(3) is true, because of II.) => ... Etc.

First algorithmic example

- **Analysis**:

  Induction:

  l=2: after the first loop execution:

$$Max^* = \max\{\quad + a_2, a_2\} = \max\{\quad (1,2), f(2,2)\},$$

$$Max = \max\{\quad + a_2, a_2, a_1\} = \sigma(1,2)$$

l-1 → l:

(Max$_{l-1}$ und Max*$_{l-1}$ describe Max and Max* after the (l-2)-th execution of the loop)
Concerning the induction hypothesis, it is:

$$Max^*_{l-1} = \max\{\quad (i, l-1), 1 \leq i \leq l-1\},$$

$$Max_{l-1} = \sigma(1, l-1)$$

First algorithmic example

- **Analysis**:

  this implies: $Max^*_l = \max\{Max^*_{l-1} + a_l, a_l\}$

$$\overset{IH}{=} \max\{\{f(i,l-1) + a_l \; with \; 1 \le i \le l-1\} \cup \{a_l\}\}$$

$$= \max\{f(i,l) \; with \; 1 \le i \le l\}$$

$$Max_l = \max\{Max_{l-1}, Max^*_l\}$$

$$\overset{IH}{=} \max\{\{\sigma(1,l-1)\} \cup \{f(i,l) \; with \; 1 \le i \le l\}\}$$

$$= \sigma(1,l)$$

Thus, with Max =*σ(1,n)*, the correct value is computed and
- A(n) = n-1
- V(n): two comparisons per loop execution, thus 2(n-1) comparisons in total
- **T(n) = 3n-3**

First algorithmic example

n	*naive* $1/6\, n^3+1/2\, n^2+1/3\, n-1$	*normal* $n^2 - 1$	*divide & conquer* $2n \log (n) - n + 1$	*clever* $3n - 3$
$2^2 = 4$	19	15	13	9
$2^4 = 16$	814	255	113	45
$2^6 = 64$	45759	4095	705	189
$2^8 = 256$	2829055	65535	3841	765
$2^{10} = 1024$	179418599	1048575	19457	3069
$2^{15} = 32768$	$> 5 \cdot 10^{12}$	$\approx 10^9$	950273	98301

e.g.. *clever* needs for *n = 1024* approx. As much time as *divide & conquer* for *n = 256* or as *normal* for *n = 64*.

First algorithmic example

Questions:

- Is it really necessary to make such an effort for analyses? This is already complicated and boring for small examples with three lines.
- Why did we count comparisons and additions? Could we also count multiplications? Why did we deal with the addition in the same way as with the comparison? Is that really the rigth way?
- Why do we use just the number of sequence items as an parameter of analysis? Isn‘t the consequence that everybody counts whatever he/she likes to count? Is it possible to reach better comparability?
- How do I know whether I have discovered a good algorithm?