Diskrete Algorithmische Mathematik

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Betreuungsteam

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vorlesung: (Do. 11:40 bis 13:20)
  PD Dr. Ulf Lorenz
  Email: lorenz@mathematik.tu-darmstadt.de
  Raum 37, Dolivostr. 15

- Tutorien:
  Dipl. Math. Franziska Kartzow
  Email:fkartzow@mathematik.tu-darmstadt.de

- Übungen:
  Axel Lukassen, N.N., N.N.

Do.	14:25 bis 16:05	woch	ab 15.04.2010	S103/113
Do.	14:25 bis 16:05	woch	ab 15.04.2010	S103/110
Do.	16:15 bis 17:55	woch	ab 15.04.2010	S103/164
Do.	16:15 bis 17:55	woch	ab 15.04.2010	S103/110
Do.	16:15 bis 17:55	woch	ab 15.04.2010	S103/111

Modulbeschreibung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Literatur:
  - M. Aigner: Diskrete Mathematik, Vieweg
  - T.H. Cormen, C.E. Leiserson, R.L. Rivest: Introduction to Algorithms
    Es gibt Begleitmaterial (teilweise im voraus, im Netz), erleichtert das Mitschreiben, **ersetzt aber kein Lehrbuch!**
- Prüfung: Klausur, mindestens 60 Minuten
- 4,5 ECTS, 2V + 1Ü, optional: Tutorium
- **Allgemeine Konzepte**: doppeltes Abzählen, Inklusion-Exklusion, Wachstum von Funktionen und asymptotische Analyse.
  Graphentheorie: Eulersche Graphen, aufspannende Bäume, planare Graphen, kürzeste Wege, Travelling-Salesman-Problem.
  Suchprobleme: Sortieren, Entscheidungsbäume.
  Codierung/Kryptographie: Huffman-Codierung, RSA-Algorithmus.
  Weitere Themen (in Auswahl): Matchings in bipartiten Graphen, Flussalgorithmen.

Organisatorisches

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Übungsgruppen:** Listeneintragung im EVS
  Beginn der Übungen: ab Do dem 22.4.2010
  Web-Seite der Vorlesung:
  https://www3.mathematik.tu-darmstadt.de/index.php?id=84&evsid=23&evsver=851

- **Übungsaufgaben/Übungsbetrieb:**
  Ausgabe der Übungen: Do 19:00 Uhr (im Netz)
  Abgabe: in der Regel jeweils nächsten Do in den Übungen

- **Leistungsbewertung:** Klausurnote kann durch aktive Teilnahme in Übungen und „viele“ gelöste Aufgaben verbessert werden. Ein Teil der Klausur wird auf Multiple-Choice bzgl. der Hausaufgaben hinauslaufen.
- **Kleingruppenarbeit** ist sinnvoll
  (Gruppen von höchstens 4 Personen können auch gemeinsam arbeiten!)

Organisatorisches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Pro Blatt 3 bis 4 Aufgaben. In unregelmäßigen Abständen:
  Programmieraufgaben mit längerer Bearbeitungszeit.
  Zusätzlich: Präsenzaufgaben

- In den **Übungen** wird gegebenenfalls die Vorlesung nachbereitet, sowie Präsenzaufgaben in Kleingruppen bearbeitet.

! Sie können nur durch „Selbermachen“ lernen,
nur lesen und zuhören reicht nicht!

In eigener Sache

TECHNISCHE UNIVERSITÄT DARMSTADT

- Suche Studentische Hilfskräfte (HiWi)
  - Rechnerbetreuung in der Dolivostr.
  - SFB 805: Optimierung unter Unsicherheit

Inhalt

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einführung
- Komplexitätstheorie
  - Datenstrukturen und Kodierungsschemata
  - Algorithmen
  - Asymptotische Notation, untere und obere Schranken
  - Klassen P, NP, NP-vollständig
- Algorithmen auf Graphen
  - DFS Algorithmus
  - Greedy-Algorithmen (z.B.: Spannbäume berechnen)
  - Dijkstra, Moore-Bellmann (kürzeste Wege in Graphen)
  - Ford-Fulkerson (maximale Flüsse in Netzwerken, Matchings)
- Abstrakte Datentypen (Stack, Queue, Heap) und nochmal DFS, BFS und Dijkstra
- Sortieren in Arrays
  - Mergesort, Quicksort, Heapsort
  - Divide-and-Conquer
  - Untere Komplexitätsschranken für Sortieren mit Vergleichsoperatoren

Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Algorithmische Probleme:

- Wie findet ein Navigationssystem „gute“ Verbindungen?
- Wie findet man in einem Gasnetzwerk die optimalen Routen?
- Wie optimiert beispielsweise die Lufthansa den kosteneffizienten Betrieb ihrer Airline?

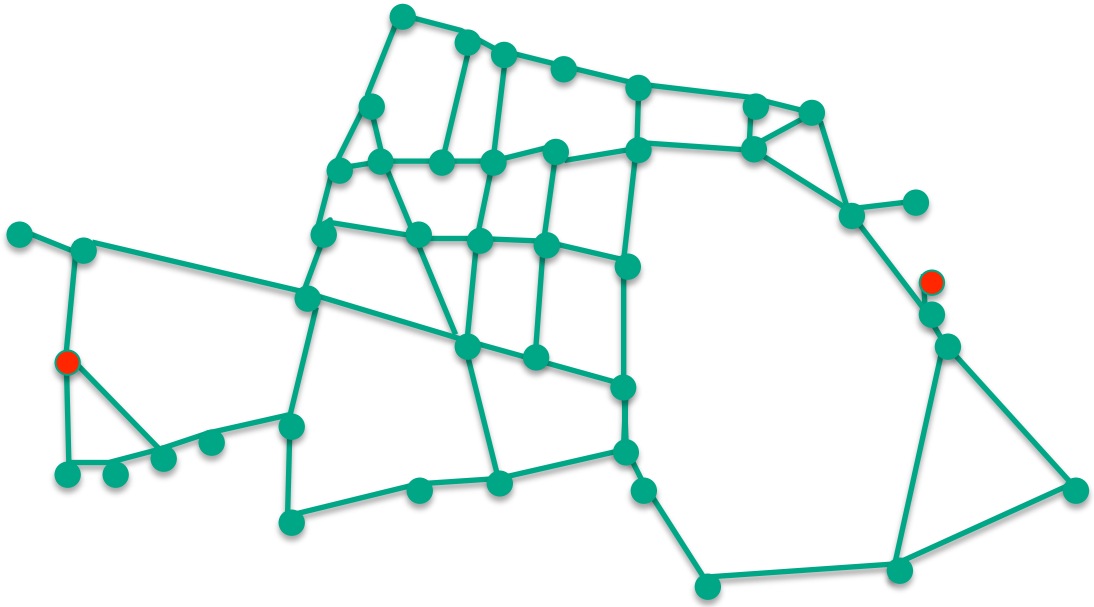
Einführung, “gute” Verbindungen

Einführung, “gute” Verbindungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Einführung, “gute” Verbindungen

TECHNISCHE UNIVERSITÄT DARMSTADT



- Straßenkarte → Graph; Problem: finde kürzesten Weg zwischen Markierungen
- Der Graph könnte auch ein Gasnetzwerk darstellen; Problem dann z.B.: Wie verteilt man am günstigsten das Gas von Einspeiseknoten an Verbraucher?

Optimierung in der Flugindustrie

TECHNISCHE UNIVERSITÄT DARMSTADT

Market Modeling

Fleet Assignment

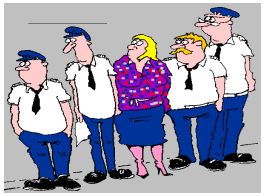
Crew Pairing

Network Design

Operation Control

Revenue Management

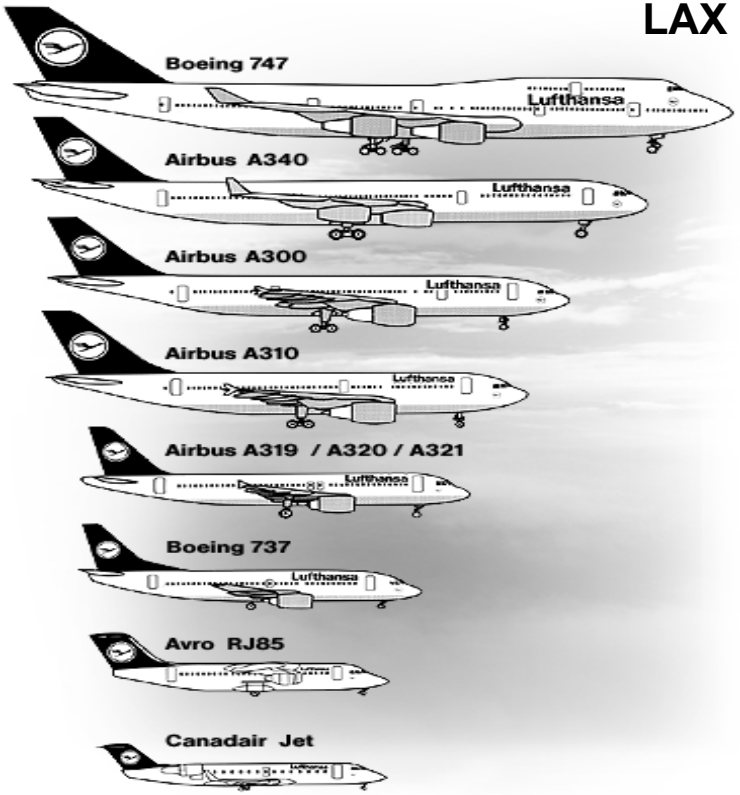
Aircraft Rotation

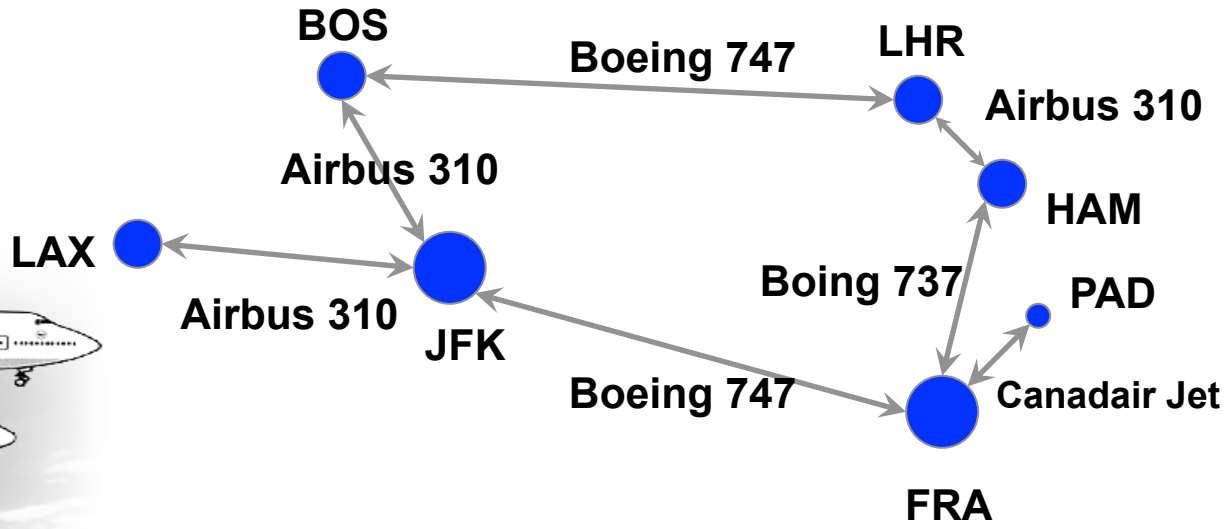


Crew Rostering

Fleet Assignment

TECHNISCHE UNIVERSITÄT DARMSTADT





Zuweisung der Flugzeugtypen zu den Flügen

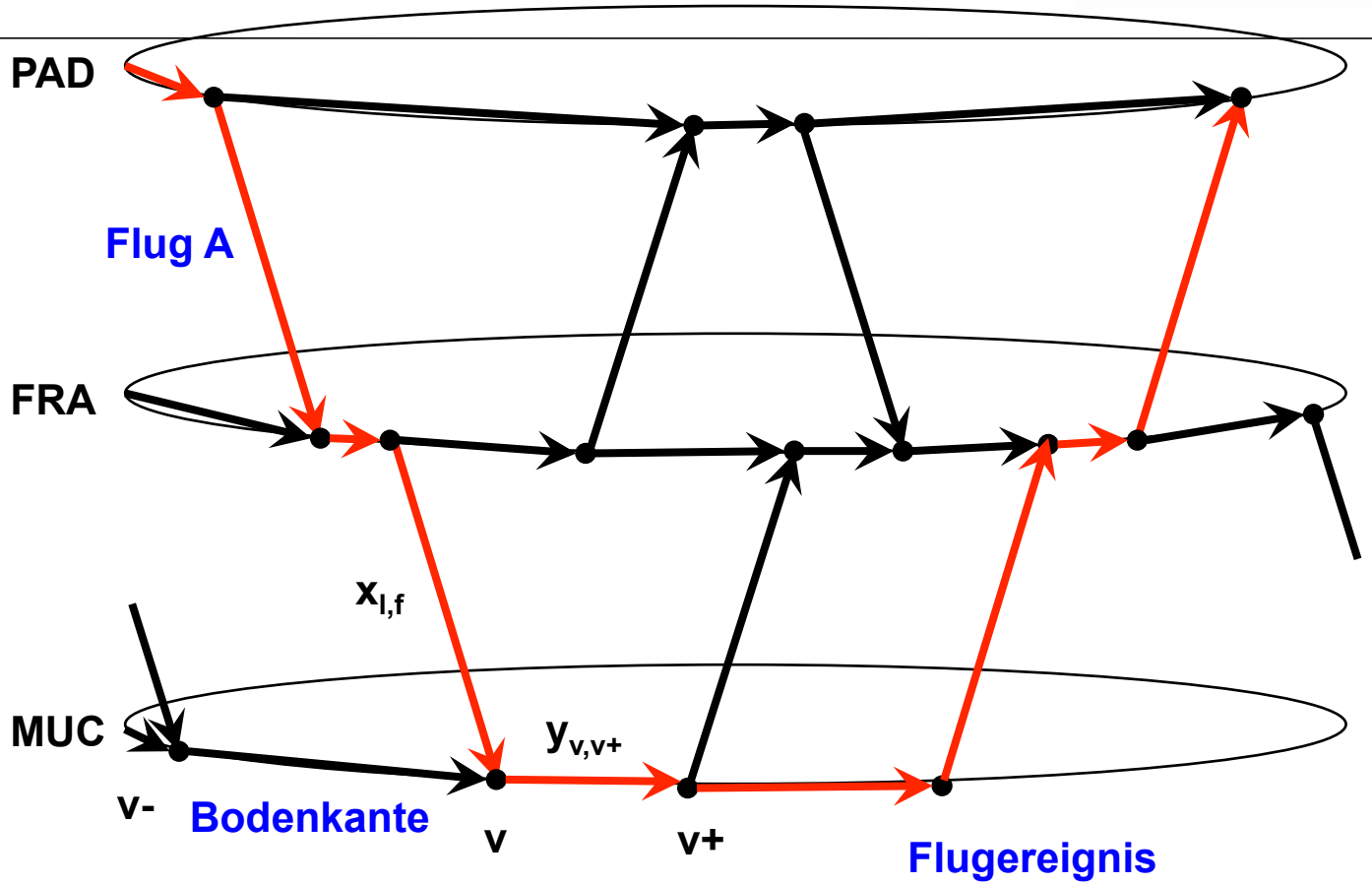
- **Maximierung des Gewinns**

Restriktionen:

- **Beschränkte Anzahl der Flugzeuge**
- **Balancierter Flugplan**
- **Zahlreiche operative Restriktionen**

Time-Space Network

TECHNISCHE UNIVERSITÄT DARMSTADT



Wochenplanung mit bis zu 10.000 Flügen, 10-23 Flugzeugtypen

Lineares Programm für Fleet Assignment

TECHNISCHE UNIVERSITÄT DARMSTADT

($x_{l,f}$ = 1) ⇔ (Flug l wird mit der Flotte f geflogen)

$y_{v,v+}$: Anzahl wartender Flugzeuge zwischen zwei Flugereignissen

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

subject to

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

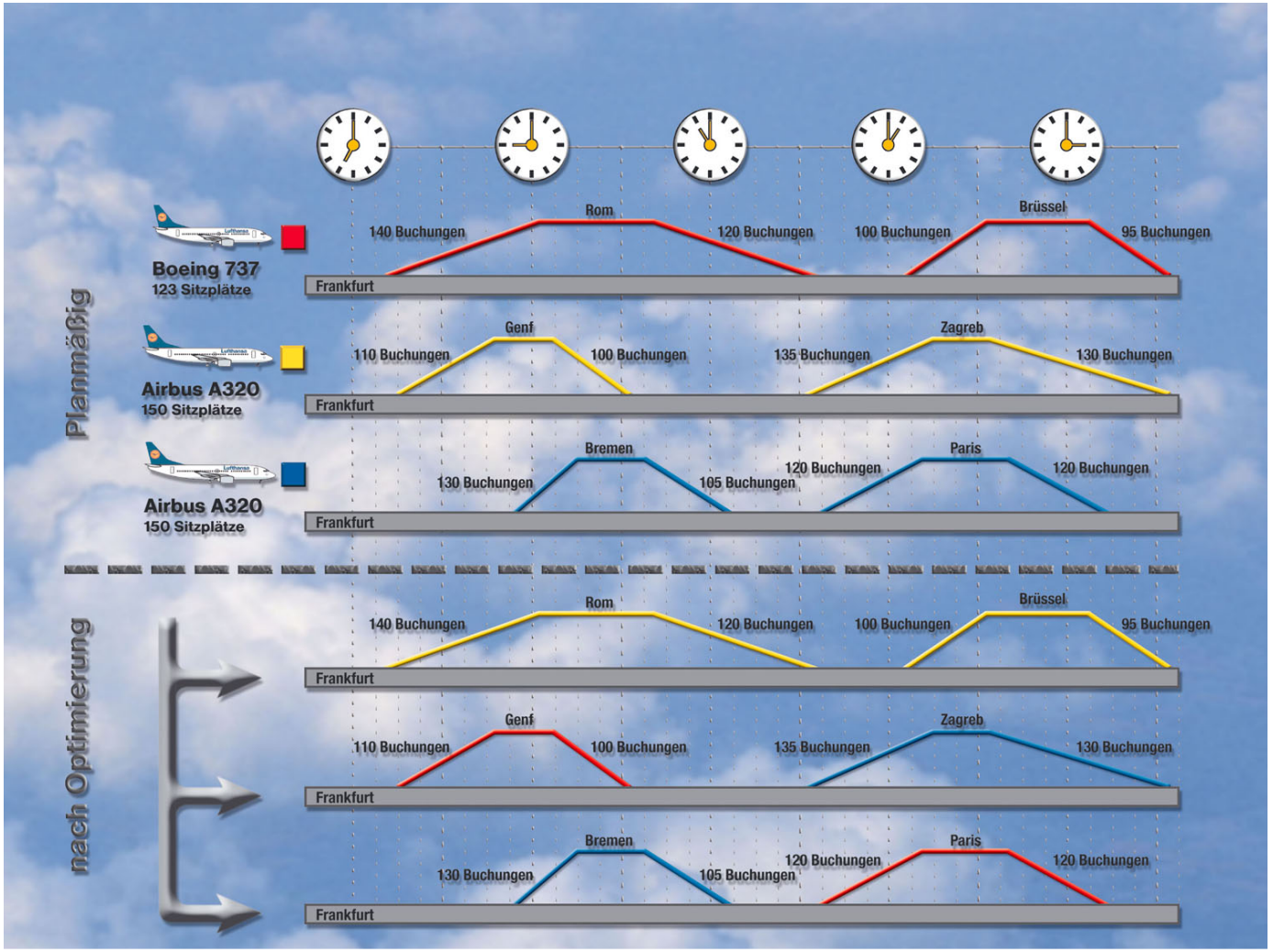
- **Mathematisches Modell, Lineares Programm → Optimierung I - III**
- **Modellgröße:**

  |Flüge|*|Flotten| ganzzahlige Variablen, 2*|Flüge|*|Flotten| Restriktionen

  also ca. 230000 Variablen, 500000 Nebenbedingungen
- **Lösungszeiten mit exakten Verfahren waren damals (um 2000) zu lang**

Lokale Suche, heuristischer Verbesserungsschritt für Fleet Assignment

TECHNISCHE UNIVERSITÄT DARMSTADT



Lokale Suche, heuristisches Optimierungsverfahren

TECHNISCHE UNIVERSITÄT DARMSTADT

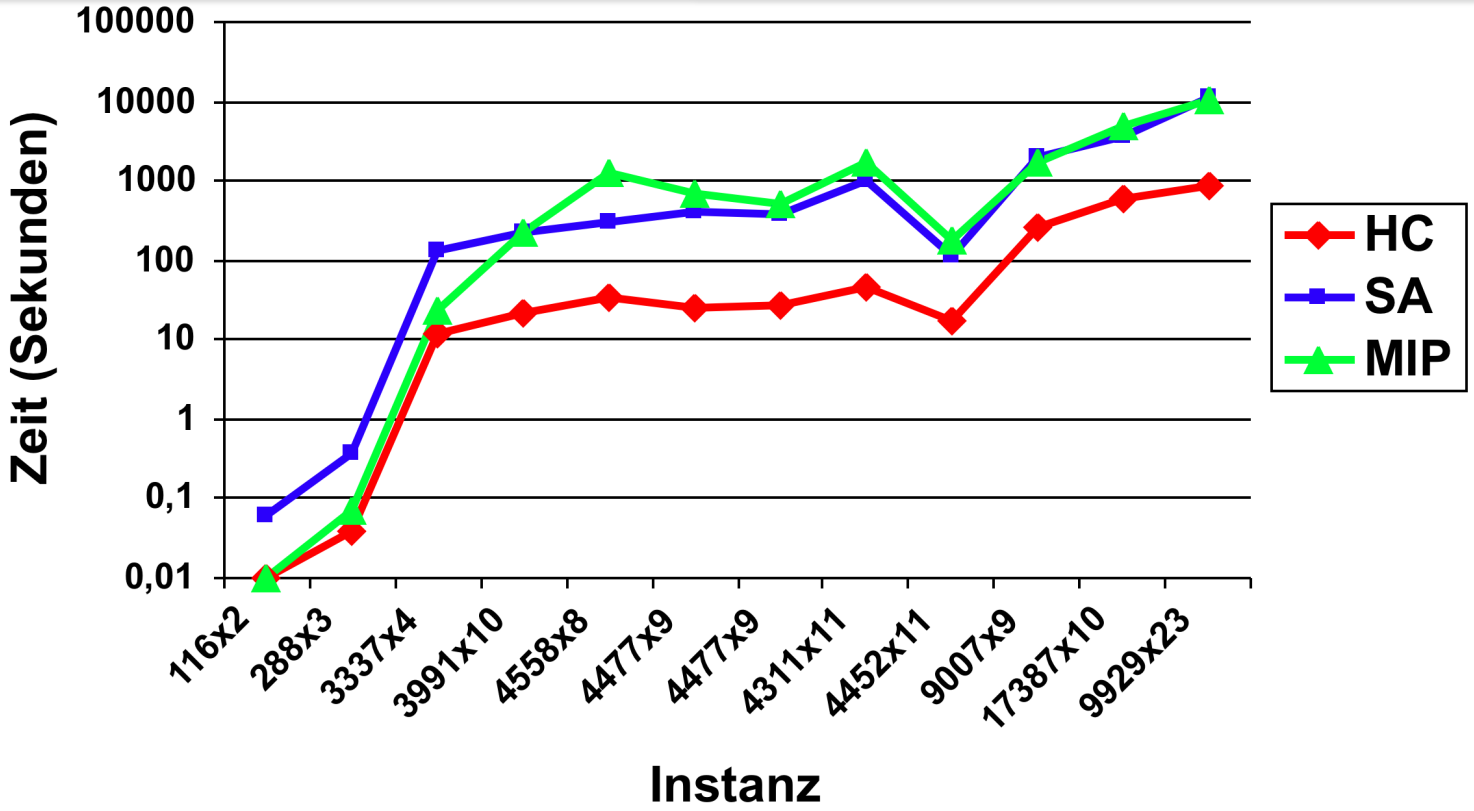
Simulated Annealing:

heuristisches Verfahren basierend auf lokaler Suche

Generiere initiale Lösung
Wahrscheinlichkeit P = P_0
do
  do
    Generiere eine lokal benachbarte Lösung
    Akzeptiere eine Verschlechterung mit Wahrscheinlichkeit P
  Until Abbruchkriterium
  P = Update(P)
until Frozen

Ergebnisse fürFleet Assignment

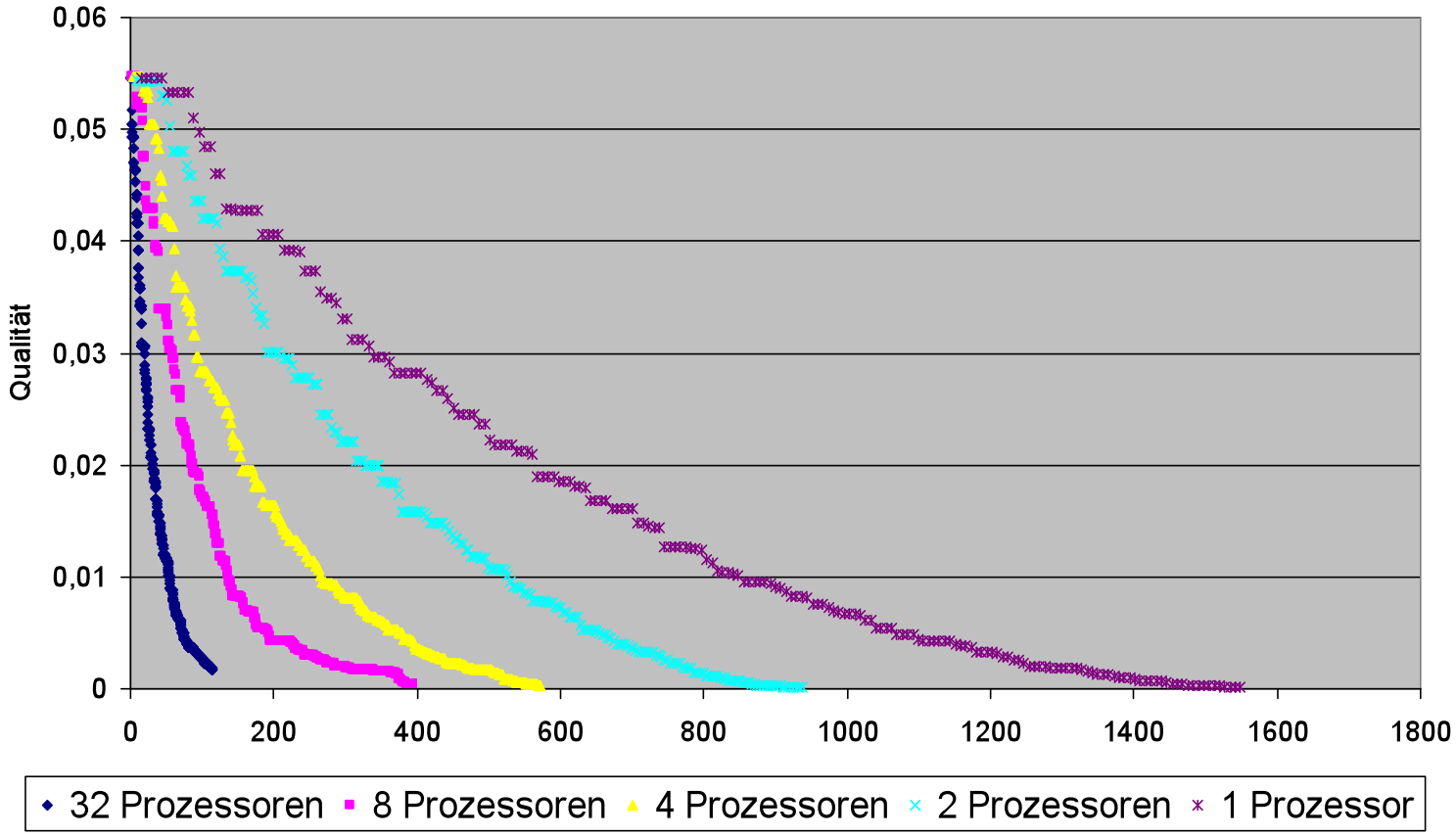
TECHNISCHE UNIVERSITÄT DARMSTADT



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

Paralleles Simulated Annealing

TECHNISCHE UNIVERSITÄT DARMSTADT



Industrieeinsatz

TECHNISCHE UNIVERSITÄT DARMSTADT

Fazit: Geschwindigkeit ist ein wichtiges Kriterium für Konkurrenzfähigkeit

Lufthansa

AIR CANADA

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Maxsummenproblem

  Eingabe: Folge $a_1,...,a_n$ ganzer Zahlen. Sei $f(i,j) := a_i+a_{i+1}+...+a_j$,
  für $1 \le i \le j \le n$.
  Gesucht: Das maximale $f(i,j)$.

  Zur Effizienzanalyse zählen wir an dieser Stelle nur
  - die Anzahl der benötigten Vergleiche V(n) und
  - die Anzahl der benötigten arithmetischen Operationen.
  - Als Laufzeit definieren wir T(n) = A(n) + V(n).

Im folgenden werden 4 Algorithmen vorgestellt und detailliert analysiert.

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Algorithmus 1 (naiver Algorithmus)**
  1. Berechne hintereinander alle f(i,j)
  2. Berechne das Maximum der f(i,j)

Beispiel: geg. (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

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Algorithmus 1 (naiver Algorithmus)**
  1. Berechne hintereinander alle f(i,j), // also *f(1,1), f(1,2), f(1,3)... f(1,n), f(2,1)...*
  2. Berechne das Maximum der f(i,j)

Analyse: Zur Berechnung von *f(i,j)* werden *j-i* Additionen benötigt.

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

Um das Maximum von L Zahlen zu bestimmen, brauchen wir L-1 Vergleiche. Hier ist L = ½ (n • (n-1)) + n. (warum?) Also ist

$$V(n) = \binom{n}{2} + \binom{n}{1} - 1 = -1 + \sum_{1}^{n} i = \frac{1}{2}(n(n-1)) + n - 1 = \frac{1}{2} n^2 + \frac{1}{2} n - 1$$

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

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Algorithmus 2 (normaler Algorithmus)**
  f(i,j) kann man effizienter berechnen, indem man etwa *f(i,j+1)* nicht neu, sondern als ***f(i,j)+a$_{j+1}$*** berechnet, wobei *f(i,j)* schon berechnet wurde.
  Wir berechnen die *f(i,j)* in folgender Reihenfolge:

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

und berechnen dann das Maximum. Nun ist V(n) wie im ersten Algorithmus, aber wie benötigen für jedes f(i,j), j > i, nur eine Addition:

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

$$T(n) = V(n) + A(n) = n^2 - 1$$

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Algorithmus 3 (Divide & Conquer Algorithmus)**

  Um besser zu werden: rechne gar nicht alle *f(i,j)* aus!

  Allgemeine Entwurfsmethode „Divide & Conquer“:

  Zerlege das Problem in mehrere Teilprobleme gleichen Typs („divide“) löse die Teilprobleme (rekursiv) und setze dann die Lösung aus den Teillösungen zusammen („Conquer“).

Im folgenden nehmen wir zur Vereinfachung der Analyse an, dass n eine Zweierpotenz $n=2^k$ ist.

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für $1 \le a \le b \le n$ definieren wir:

$$\sigma(l,r) := \sigma(a_l,\ldots,a_r) := \max\{f(i,j), l \le i \le j \le r\},$$

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

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

Dann gilt für $\sigma(1,n)$:

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

Bsp:

(-10,5,2,-7, 3,6,-9,11)
$\sigma(1,4)=7$ $\sigma(5,8)=11$
$s_1=0$ $s_2=11$

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*

(-10,5,2,-7, 3,6,-9,11)
s_1=? s_2=?
σ_1 = ? σ_2=?

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

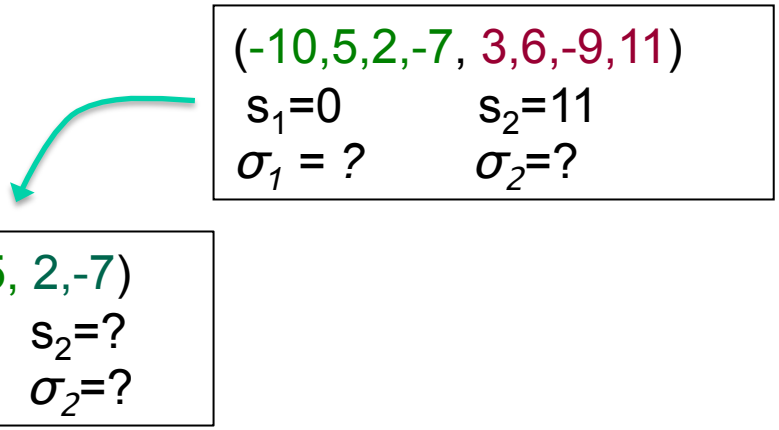
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*

(-10,5,2,-7, 3,6,-9,11)	
$s_1=0$	$s_2=11$
σ_1 = ?	σ_2=?

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

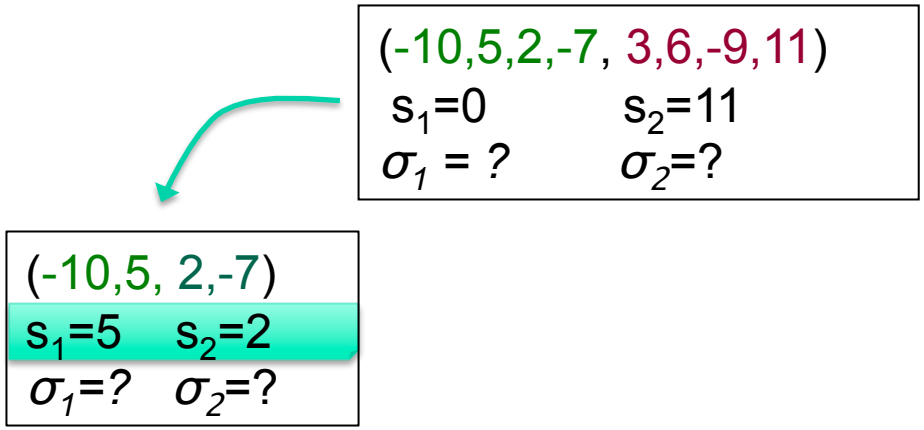
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

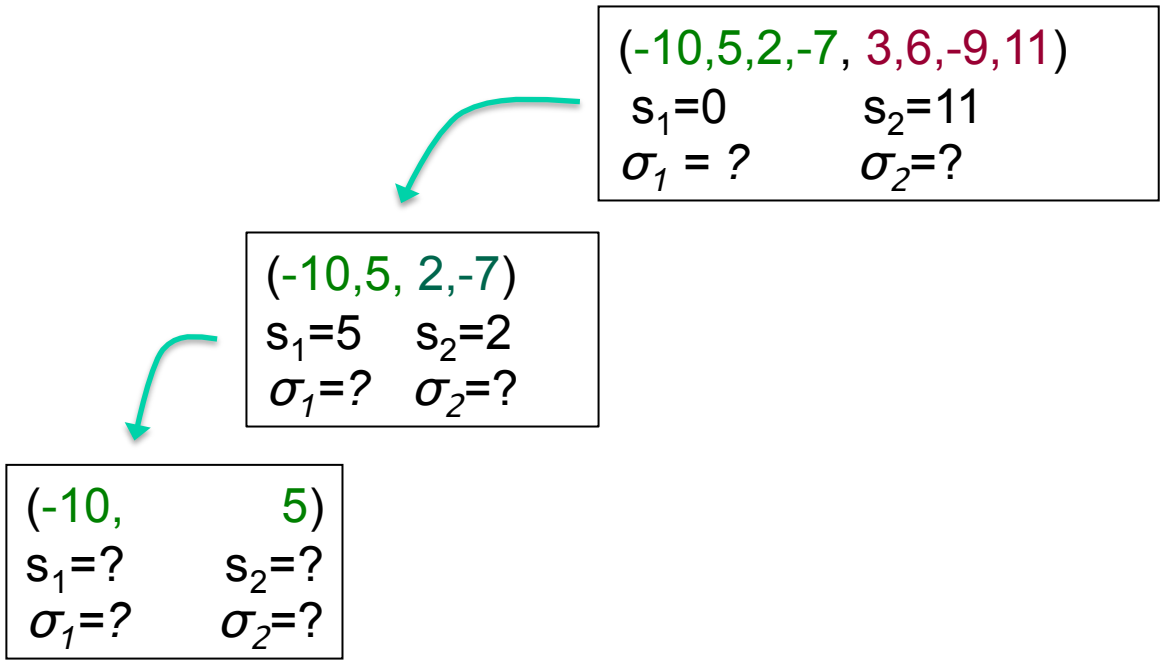
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

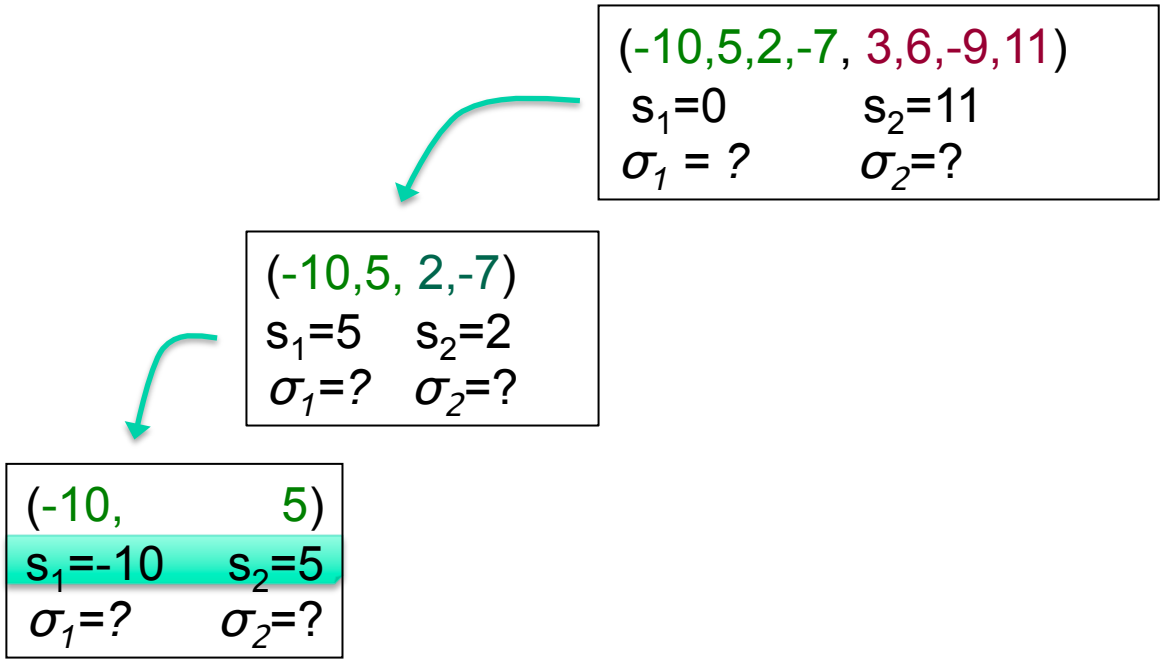
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

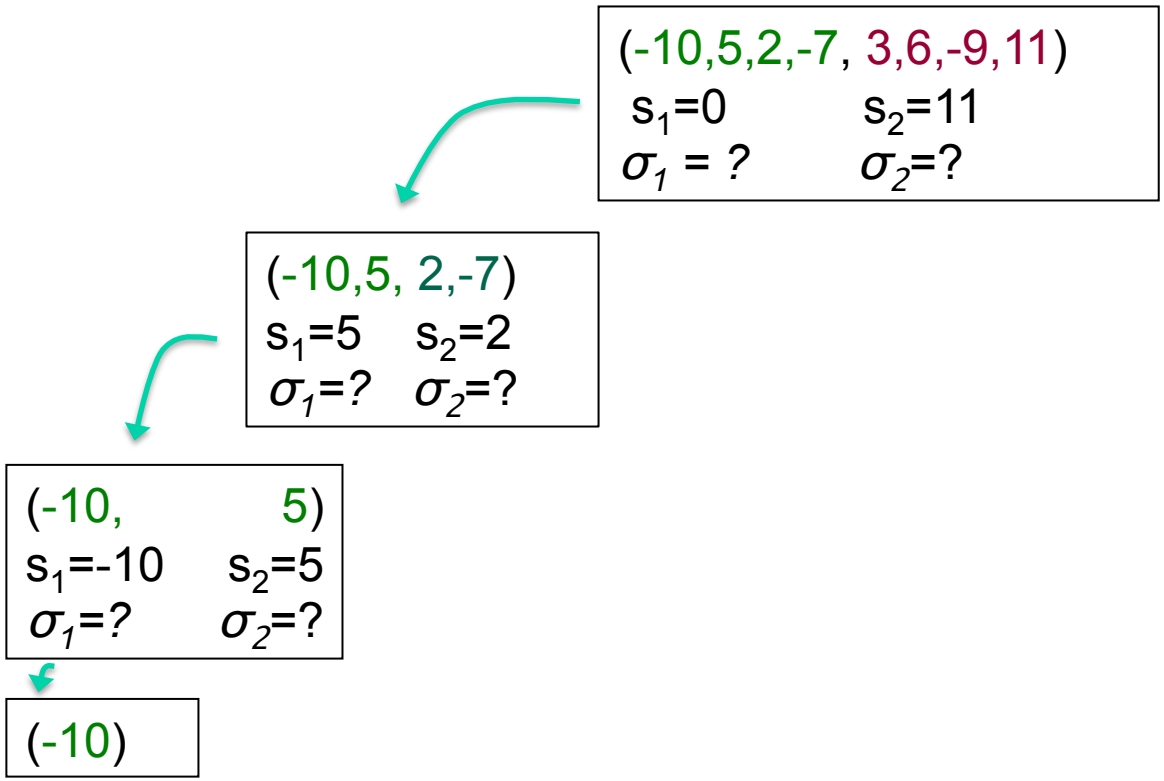
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

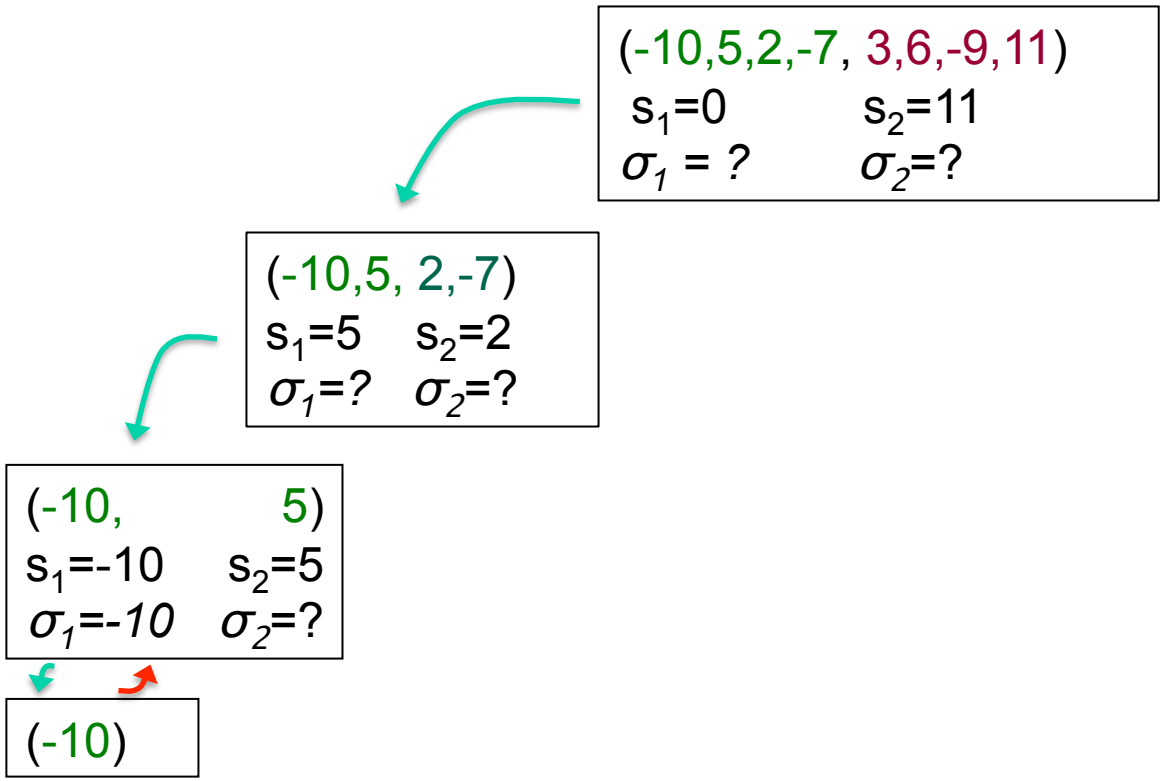
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

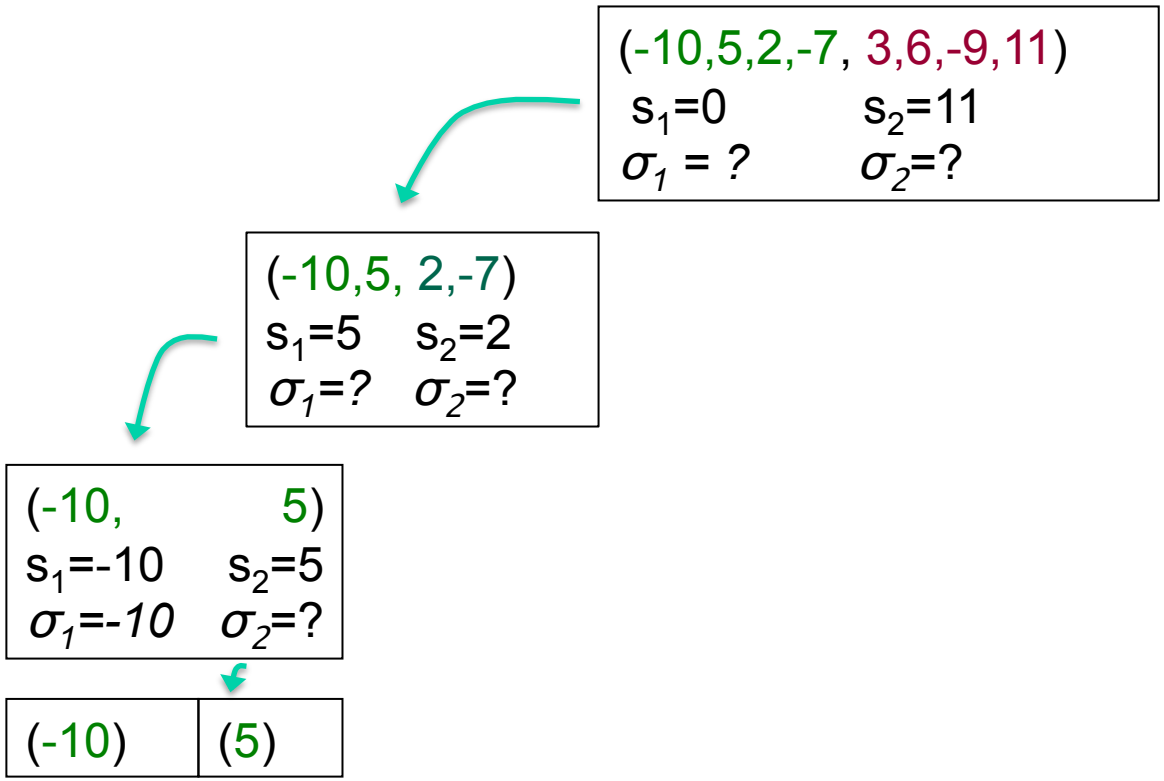
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

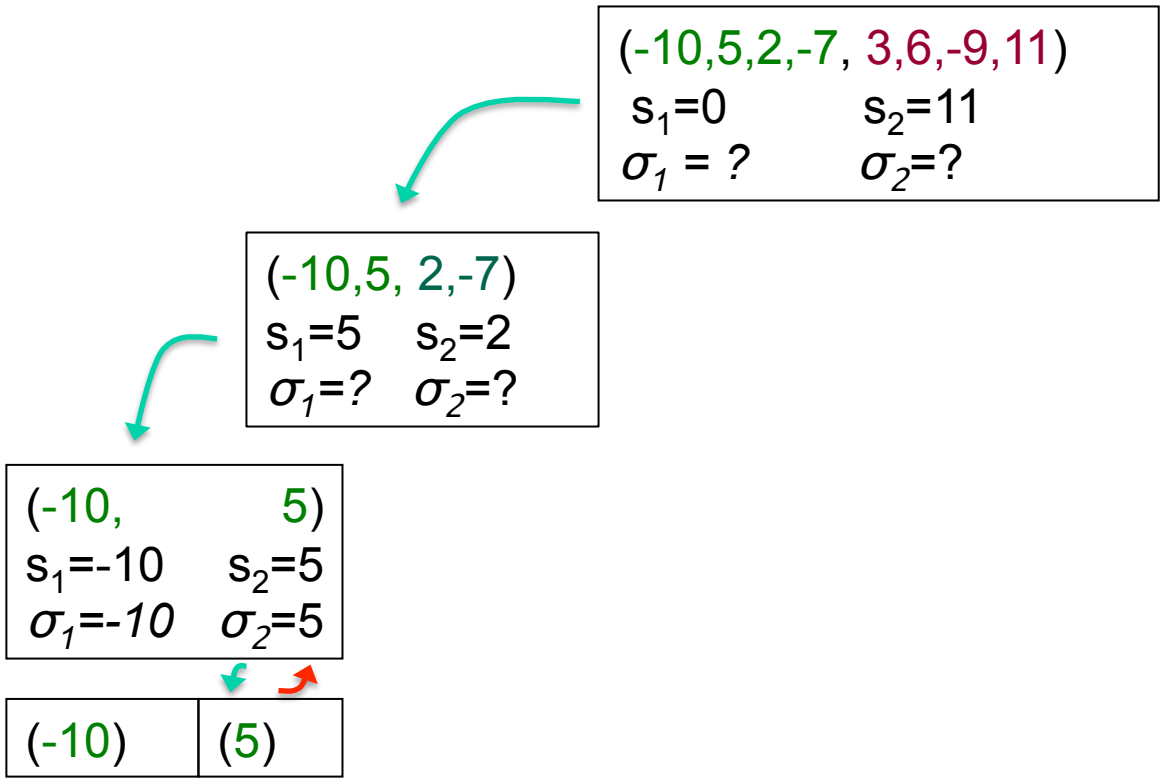
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

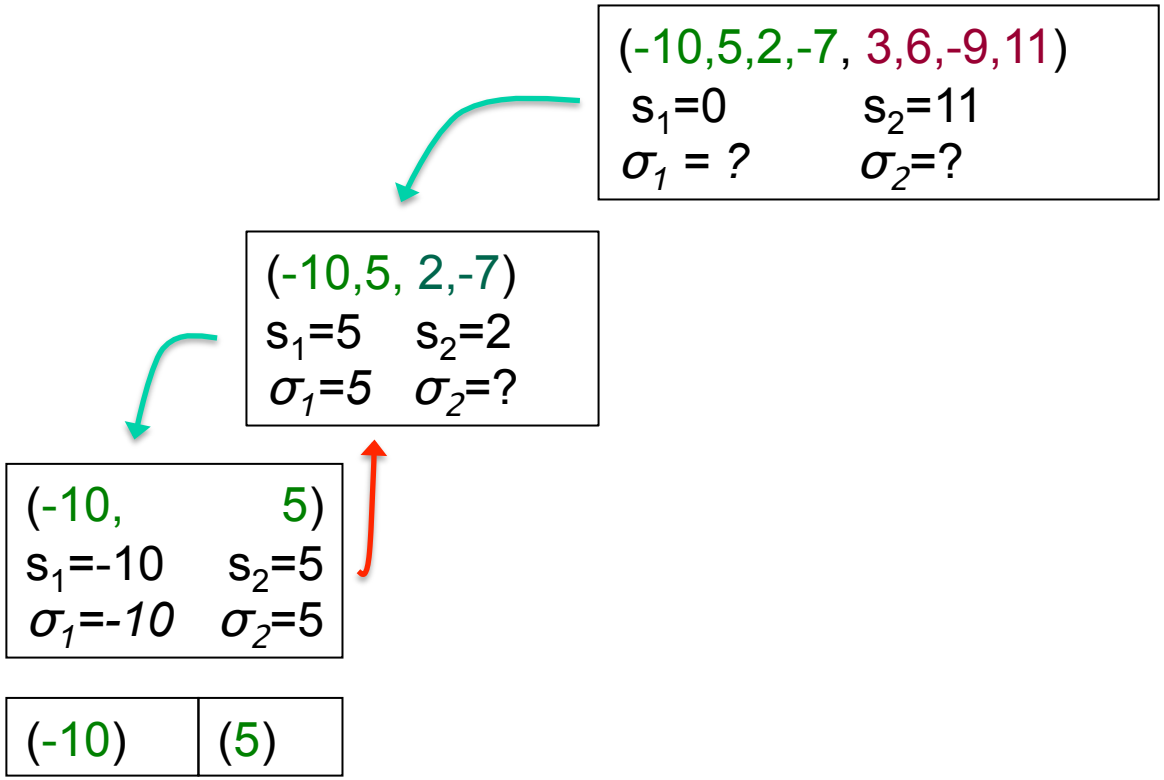
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

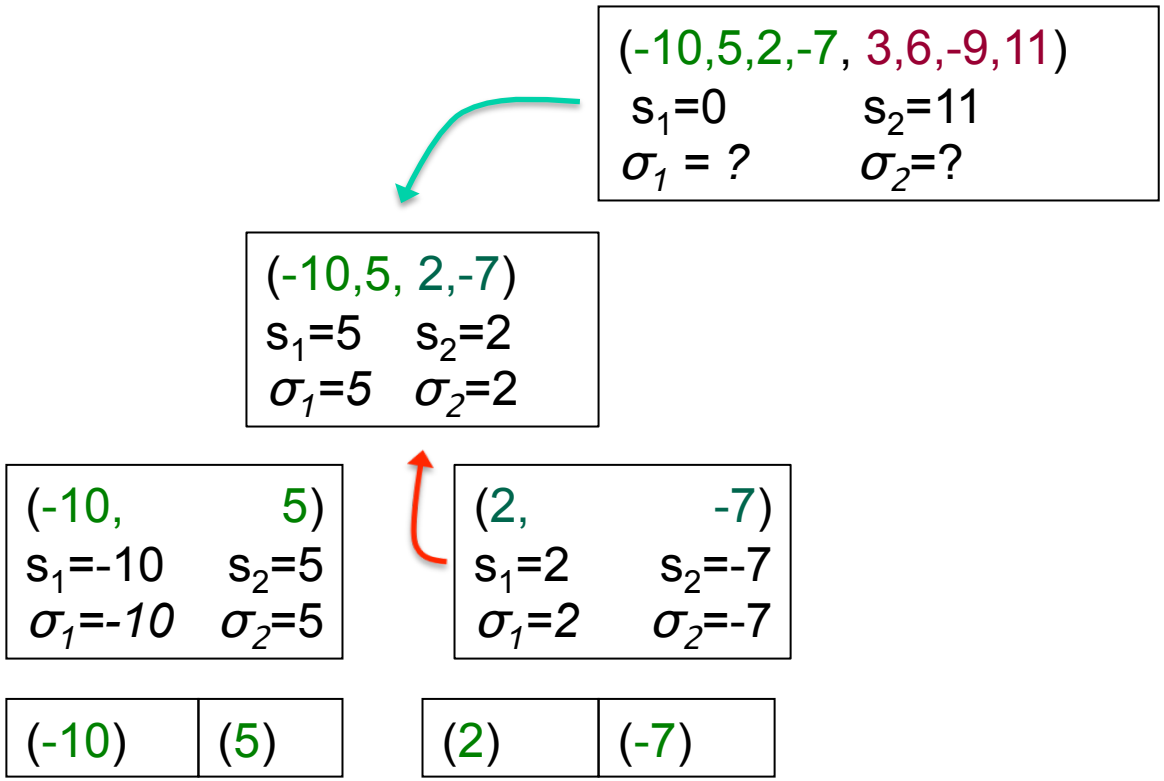
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

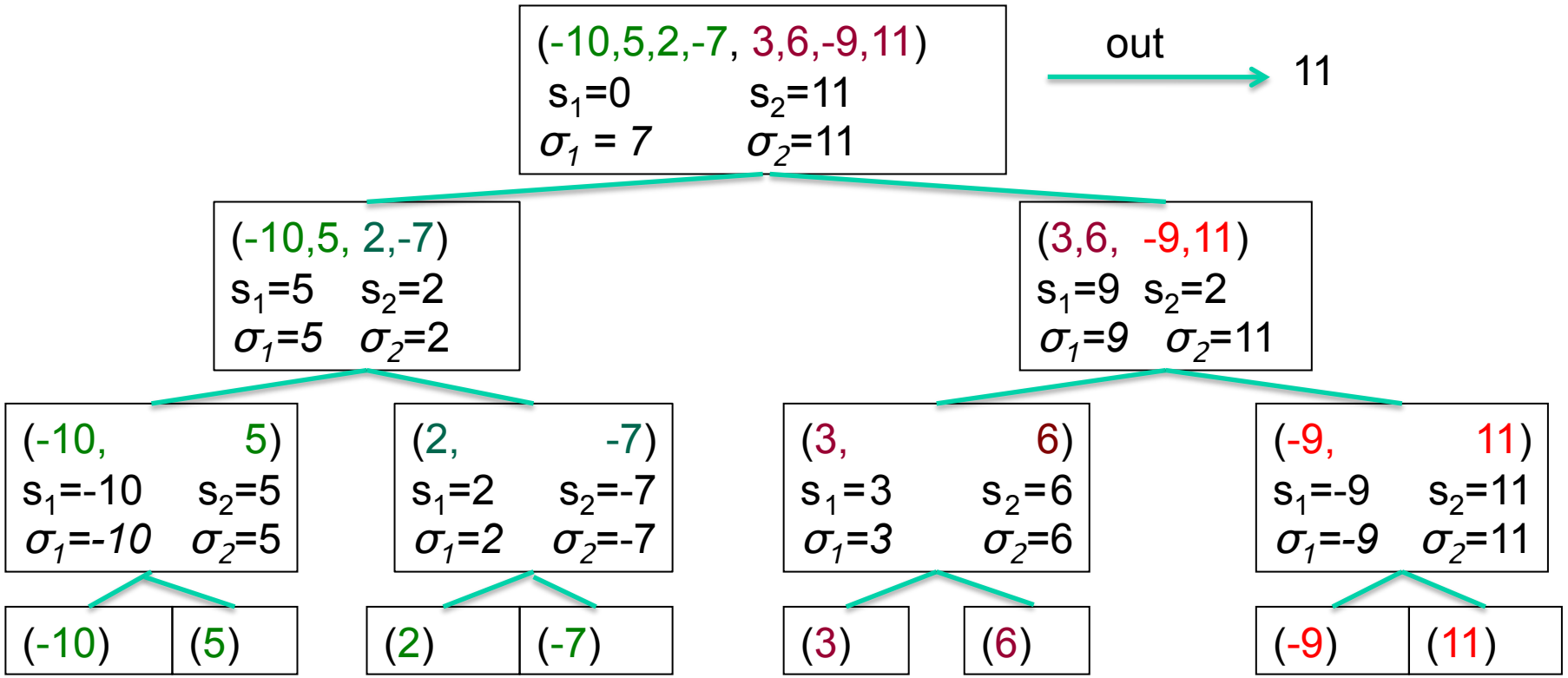
- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Funktion $\sigma(a_1,...,a_n)$**
  falls $n=1$, *gib* a_n *aus*
  falls $n>1$, berechne s_1 und s_2 sowie $\sigma_1=\sigma(a_1,...,a_{n/2})$ und $\sigma_2=\sigma(a_{n/2+1},...,a_n)$.
  gib $max\{\sigma(a_1,...,a_{n/2}), \sigma(a_{n/2+1},...,a_n), s_1+s_2\}$ *aus.*



Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Analyse:**
  sei T(n) die Anzahl der Operationen (Vergleiche + Additionen), die der D&C Algorithmus bei Eingaben der Länge n durchführt.

  Dann gilt: $T(1) = 0$, und für $n > 1$: $T(n)=2T(n/2)+2n-1$

  Begründung:
  - *um* $\sigma(a_1,...,a_{n/2})$, $\sigma(a_{n/2+1},...,a_n)$ zu berechnen, wird unser Algorithmus für Eingaben der Länge *n/2* aufgerufen
  - um s_1 und s_2 zu berechnen sind jeweils *n/2-1* Additionen und Vergleiche nötig.
  - Um $s_1 + s_2$ zu berechnen sind eine weitere Addidtion nötig
  - Um das anschließende Maximum zu berechnen zwei weitere Vergleiche.

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

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Analyse:**
  Ergebnis: so genannte Rekursionsgleichung / Rekurrenz für die Laufzeit (aus technischen Gründen substituieren wir 2k für n)

$$T(1) = 0, und\ für\ k \geq 1 : T(2^k) = 2T(2^{k-1}) + 2^{k+1} - 1$$

(typisch für rekursive Algorithmen, inbesondere für D&C)
Durch mehrfaches Einsetzen ergibt sich:

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

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Analyse:**

$$T(1) = 0, und\ \ für\ k \geq 1 : T(2^k) = 2T(2^{k-1}) + 2^{k+1} - 1$$

... und damit die Vermutung:

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

so dass für *l*=*k* gilt:

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

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

$$= 2n\log_2(n) - n + 1$$ Beweis: Übung

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Algorithmus 4 (Cleverer Algorithmus)**
  (läuft nur einmal einziges über die Eingabe)

Max := a_1; *Max** := *Max*;
For *l* = 2,...,*n*
    *Max** := max{*Max** + a_l, a_l}
    Max := max{*Max**,*Max*}
Ausgabe: Max

Beispiel:
(-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

Korrektheit:
Beh.: nach dem l-ten Schleifendurchlauf ist

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

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

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Analyse**:

  Induktion:

  l=2: nach dem ersten Schleifendurchlauf:

$$Max^* = \max\{a_1 + a_2, a_2\} = \max\{f(1,2), f(2,2)\},$$

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

  l-1 → l:

  (Max_{l-1} und Max^*_{l-1} bezeichnen Max und Max* nach dem l-2-ten Schleifendurchlauf)
  Nach Induktionsvoraussetzung gilt:

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

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

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Analyse**:

  es folgt:

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

$$\overset{IV}{=} \max\{\{f(i,l-1) + a_l \ mit\ 1 \le i \le l-1\} \cup \{a_l\}\}$$

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

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

$$\overset{IV}{=} \max\{\{\sigma(1,l-1)\} \cup \{f(i,l)\ mit\ 1 \le i \le l\}\}$$

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

Also wird mit Max = $\sigma(1,n)$ der richtige Wert ausgegeben und

- A(n) = n-1
- V(n): zwei Vergleiche je Durchlauf, also 2(n-1) Vergleiche
- **T(n) = 3n-3**

Erstes algorithmisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

n	*naiv* $1/6\ n^3+1/2\ n^2+1/3\ n-1$	*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

z.B. *clever* benötigt für $n = 1024$ etwa so lange wie *divide & conquer* für $n = 256$ oder wie *normal* für $n = 64$.

Fragen zum ersten Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Muss man Analysen immer so detailiert machen? Das führt ja schon bei 3-Zeilern zu Komplikationen.
- Wieso werden Vergleiche und Additionen gezählt? Könnte man nicht auch Multiplikationen zählen? Warum wird eine Addition behandelt, wie ein Vergleich? Ist das wirklich sinnvoll?
- Wieso ist der Parameter gerade die Anzahl der Summanden? Führt das nicht dazu, dass jeder parametrisiert, was ihm gerade einfällt? Kann man nicht bessere Vergleichbarkeit erreichen?
- Woher weiß man, ob man einen guten Algorithmus gefunden hat?