AD20-03

Motiv: ein Problem — viele Algorithmen

"choose wisely"

"offensichtlicher" Algorithmus oft schlechte Wahl

Beispiel:

Kursveränderungen einer Aktie:

7 −11 18 10 −23 −3 27 −1

Wann kaufen? Wann wieder verkaufen?

Gewinn: 18 + 10 − 23 − 3 + 27 = 29

Problem: Maximum Subarray

Eingabe: $a_1, \dots, a_n \in \mathbb{Z}$ (ganze Zahlen), $n \geq 1$

Ausgabe: grösstmöglich Teilsumme

$S^* = a_i + \dots + a_j \quad (i, j \in \{1, \dots, n\}, \; i \leq j)$

$S^* = 0$ falls alle Zahlen negativ

naiver Algorithmus: berechne alle Teilsummen

$$S_{i,j} := a_i + \cdots + a_j$$

elementare Operationen: Addition, Vergleichen, ...

Anzahl Additionen: $T(n) := \sum_{i=1}^{n} \sum_{j=i}^{n} (j-i)$

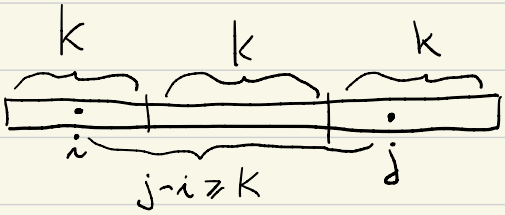
$(j-i)$: Anzahl Additionen für $S_{i,j}$

Obere Schranke:

$$T(n) = \sum_{i=1}^{n} \sum_{j=i}^{n} (j-i) \leq \sum_{i=1}^{n} \sum_{j=1}^{n} n = n^3$$

Untere Schranke: $(n = 3k \in \mathbb{N})$

$$T(n) = \sum_{i=1}^{n} \sum_{j=i}^{n} (j-i) \geq \sum_{i=1}^{k} \sum_{j=2k+1}^{n} \underbrace{(j-i)}_{\geq k} \geq \sum_{i=1}^{k} \sum_{j=2k+1}^{n} k = k^3 = \frac{n^3}{27}$$



Also: $T(n) \leq O(n^3)$ und $n^3 \leq O(T(n))$

Schreibweise dafür: $T(n) = \Theta(n^3)$ "gleiche Ordnung"

Geht es besser?

Idee: Teilsummen nicht unabhängig

z.B.: $S_{3,7} = S_{3,6} + a_6$

Pseudo code:

```
For i = 1..n :
    S_ii ← a_i
    For j = i+1 .. n :
        S_ij ← S_i,j-1 + a_j
```

} 2 nested loops

Anzahl Additionen:

i	1	2	3	...	n-1	n	Total
Additionen	n-1	n-2	n-3	...	1	0	$\frac{n\cdot(n-1)}{2} \leq n^2$

geht es besser?

müssen wir alle $\Theta(n^2)$ Teilsummen berechnen?

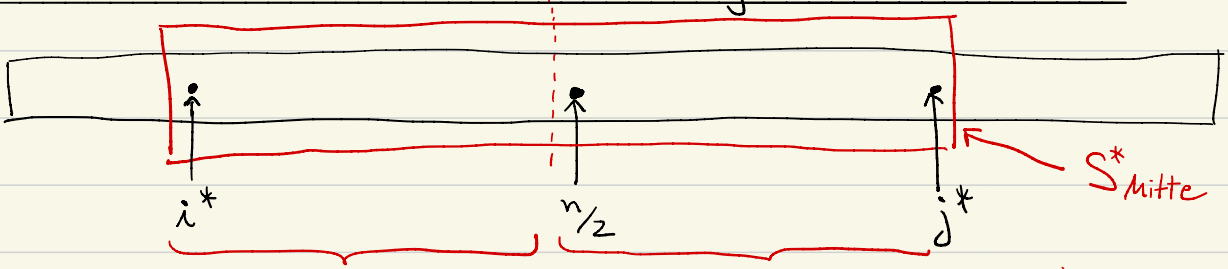
einfacheres Problem: (n gerade)

finde grösstmögliche Teilsumme, die $a_{n/2}$ enthält

$$S^*_{\text{mitte}} := \max_{i \le n/2 \le j} S_{i,j}$$

Anzahl Teilsummen: $\frac{n}{2} \cdot (\frac{n}{2}+1) \geq n^2/4$

Berechne S^*_{mitte} mit weniger Teilsummen:



$S_{i^*,\frac{n}{2}-1}$ $\quad$ $S_{\frac{n}{2},j^*}$ $\quad$ $S^*_{\text{mitte}} = S_{i^*,\frac{n}{2}-1} + S_{\frac{n}{2},j^*}$

also: $$S^*_{\text{mitte}} = \left(\max_{i \le \frac{n}{2}} S_{i,(\frac{n}{2}-1)}\right) + \left(\max_{j \ge \frac{n}{2}} S_{\frac{n}{2},j}\right)$$

Anzahl Teilsummen: $\underbrace{\frac{n}{2}}_{i} + \underbrace{(\frac{n}{2}+1)}_{j} = n+1$

wobei $S_{\frac{n}{2},\frac{n}{2}-1} := 0$ "leere Teilsumme"

Algorithmus:

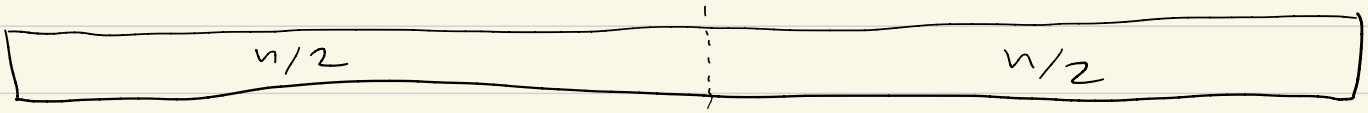
$S_{\frac{n}{2},\frac{n}{2}-1} \leftarrow 0$

For $j = \frac{n}{2} .. n$: $S_{\frac{n}{2},j} \leftarrow S_{\frac{n}{2},j-1} + a_j$

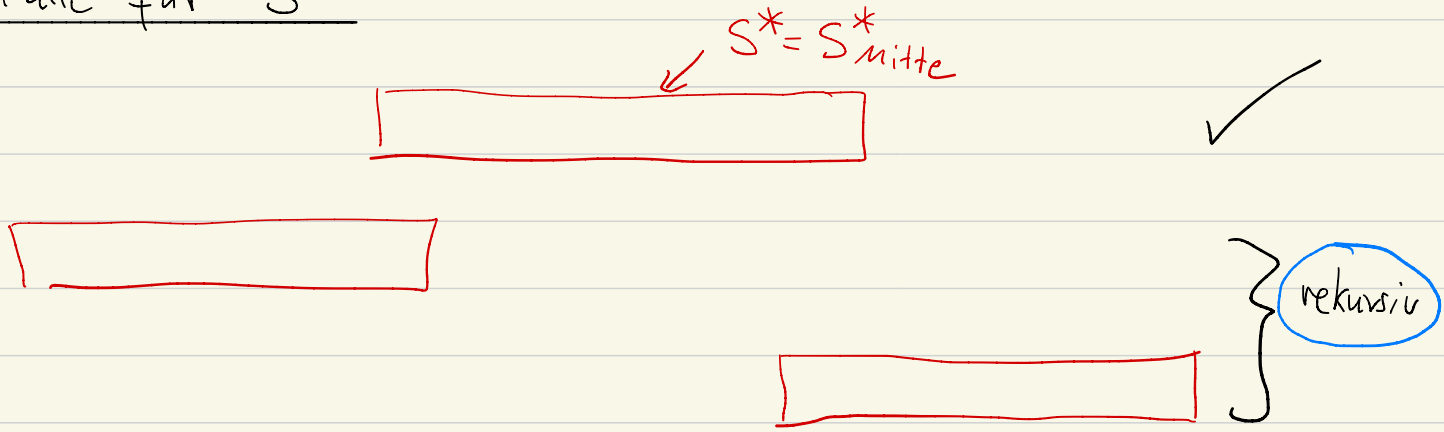
For $i = \frac{n}{2}-1 .. 1$: $S_{i,\frac{n}{2}-1} \leftarrow S_{i+1,\frac{n}{2}-1} + a_i$

Anzahl Additionen: n

rekursiver Ansatz für Maximum Subarray:



3 Fälle für S^*:

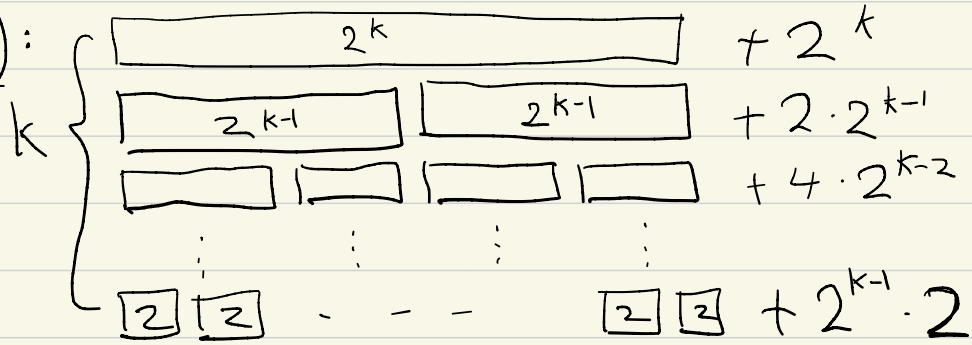


Algorithmus:

1. löse beide Hälften rekursiv
2. berechne S^*_{mitte} (n Additionen)
3. Ausgabe: beste der 3 Teillösungen

Rekurrenz: $T(n) \leq 2 \cdot T(n/2) + n$, $T(1) = 0$

Bildlich ($n = 2^k$):



$k \cdot 2^k$

$= n \cdot \log_2 n$

Also: $T(n) \leq n \cdot \log_2(n)$

geht es besser?

Rekursion: neuer Versuch

Idee: mehr in Rekursion berechnen damit Gesamtlösung schneller kombinierbar

Randmaxima: $R_j := \max_{i \le j} S_{ij}$

- berechne $R_1, \dots, R_{n-1}$ rekursiv
- $R_n \leftarrow \begin{cases} R_{n-1} + a_n & \text{falls } R_{n-1} \ge 0 \\ a_n & \text{sonst} \end{cases}$
- Ausgabe: $S^* \leftarrow \max\{R_1, \dots, R_n\}$

Rekurrenz: $T(n) \le T(n-1) + 1$, $T(1) = 0$

Wiederholtes Einsetzen:

$$\begin{aligned} T(n) &\le T(n-1) + 1 \\ &\le T(n-2) + 1 + 1 \\ &\vdots \\ &\le \underbrace{T(1)}_{=0} + \underbrace{1 + \dots + 1}_{n-1} = n-1 \end{aligned}$$

nicht besprochen

geht es besser?

für manche Instanzen, ja:

alle a_i negativ:

$S^* = 0$, keine Additionen

geht es immer besser?

für manche Instanzen, nein:

alle a_i positiv:

$S^* = a_1 + \dots + a_n$, $n-1$ Additionen

(egal welche Klammerung; informell)

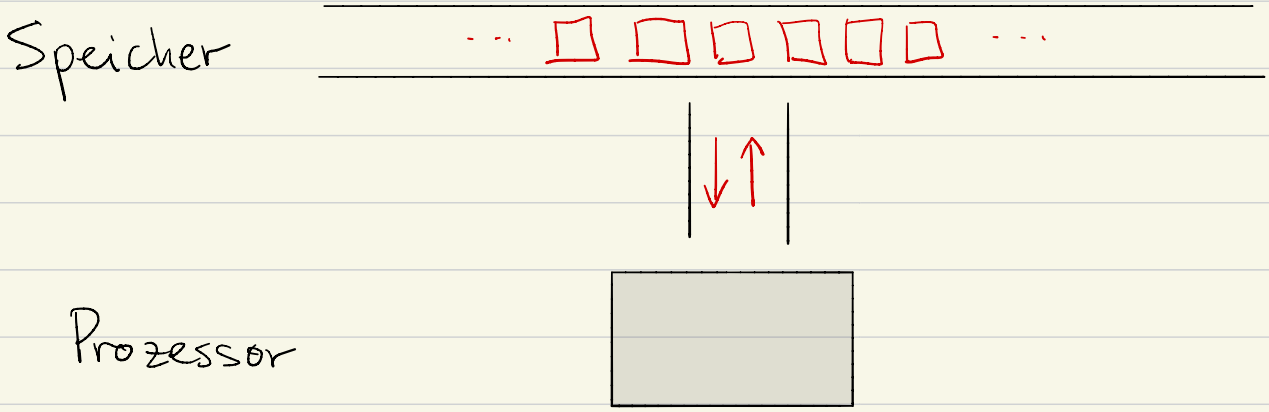
wir sagen: im worst-case sind

$n-1$ Additionen notwendig;

unser Algorithmus ist worst

case optimal

Standard Rechnermodell



Speicher: Liste von Speicherzellen

- leer oder enthält Zahl ($\leq n^{100}$)
- frei addressierbar

Prozessor: führt elementare Operationen aus

- lesen / schreiben von Speicherzellen
- vergleichen ($=, >, <$)
- rechnen ($+, -, \cdot, \div$)

wenn nicht anders angegeben:

Laufzeit = Anzahl elementarer Operation in diesem Modell

weiterhin:

- worst-case analysis
- asymptotische Notation