AlgoDat 03

Thema:

viele verschiedene Algorithmen möglich für dasselbe Problem

Konkretes Beispiel:

Kurs veränderungen einer Aktie:

7 -11 18 10 -23 -3 27 -1

Bester Kauf- und Verkaufzeitpunkt?

Gewinn: 18 + 10 - 23 - 3 + 27 = 29

Problem: Maximum Subarray

gegeben: $a_1, \dots, a_n \in \mathbb{Z}$

finde: größt mögliche Teilsumme

$$S^* = a_i + \dots + a_j$$

$$(i, j \in \{1, \dots, n\}, i \leq j)$$

$S^* = 0$ falls alle Zahlen negativ

naiver Algorithmus: berechne alle Teilsummen

$$S_{ij} := a_i + \dots + a_j$$

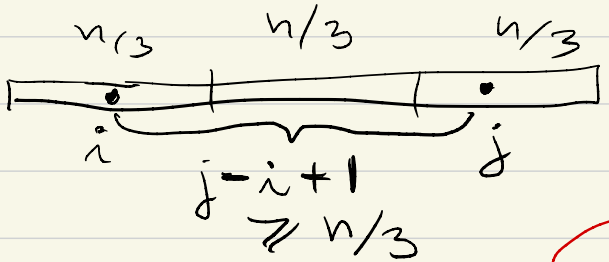
```
For i = 1 ... n :
    For j = i ... n :
        S_ij ← 0
        For k = i ... j :
            S_ij ← S_ij + a_k
```

} 3 verschachtelte Schleifen (nested loops)

Elementare Operation: Addition zweier Zahlen

Anzahl: $T := \sum_{i=1}^{n} \sum_{j=i}^{n} \underbrace{(j-i+1)}_{\leq n}$

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



(hier: 3 teilt n)

$$T \approx \sum_{i=1}^{n/3} \sum_{j=n/2+1}^{n} n/3 = \frac{n^3}{27}$$

Also: $O(T) = O(n^3)$ gibt es besser?

Idee: Teilsummen sind nicht unabhängig

z.B. gilt $S_{i,j} = S_{i,j-1} + a_j \quad (i<j)$

For $i = 1 .. n$:

$S_{i,i} \leftarrow a_i$

For $j = i+1 .. n$:

$S_{i,j} \leftarrow S_{i,j-1} + a_j$

} 2 nested loops

$\leq n^2$ Additionen

geht es besser?

$\geq n^2/2$ Teilsummen!

müssen wir alle berechnen?

einfacheres Problem:

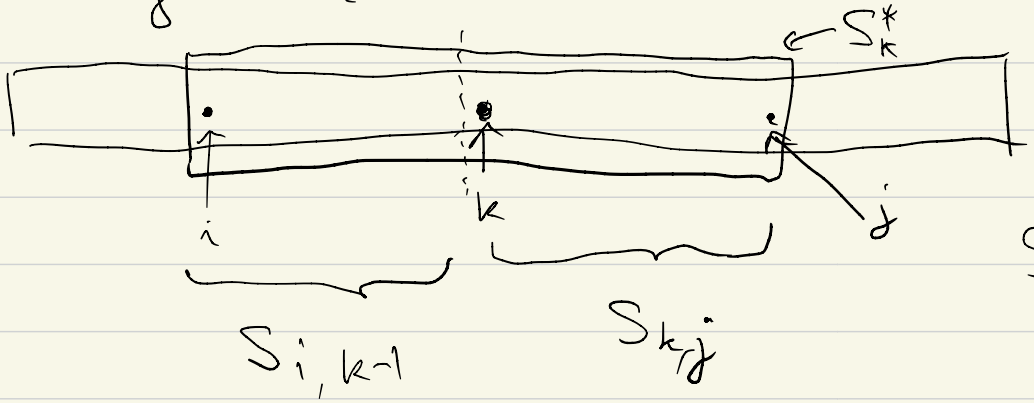
für $k \in \{1, \dots, n\}$, betrachte nur Teilsummen, die a_k enthalten

$$S_k^* := \max_{i \le k \le j} S_{ij}$$

$k \cdot (n-k+1)$ Teilsummen

$\geq n^2/4$ für $k = n/2$

weniger Teilsummen berechnen?



$$S_k^* = S_{i,k-1} + S_{kj}$$

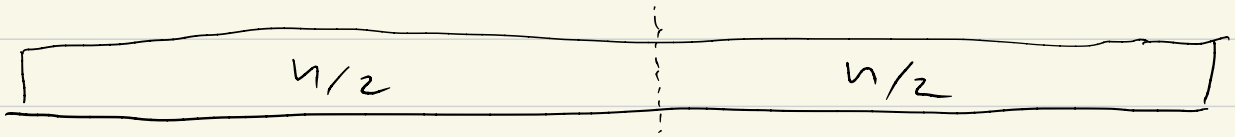
also: $S_k^* = \left(\max_{i \le k} S_{i,k-1} \right) + \left(\max_{j \ge k} S_{kj} \right)$

n Teilsummen

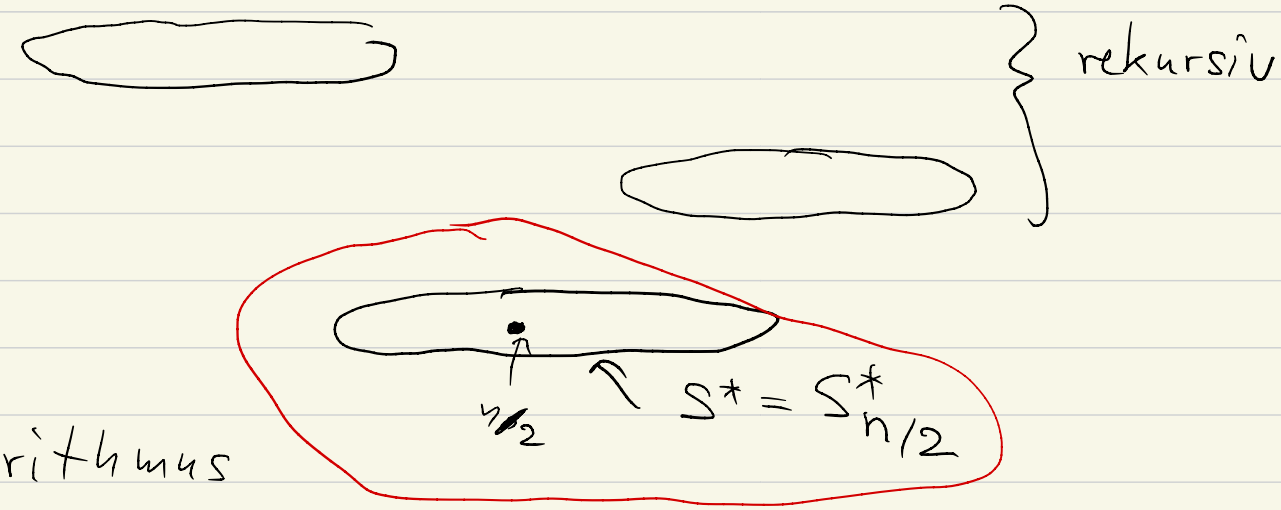
$\leq n$ Additionen

wobei $S_{k,k-1} := 0$ "leere Teilsumme"

rekursiver Ansatz:



3 Fälle für S^*:

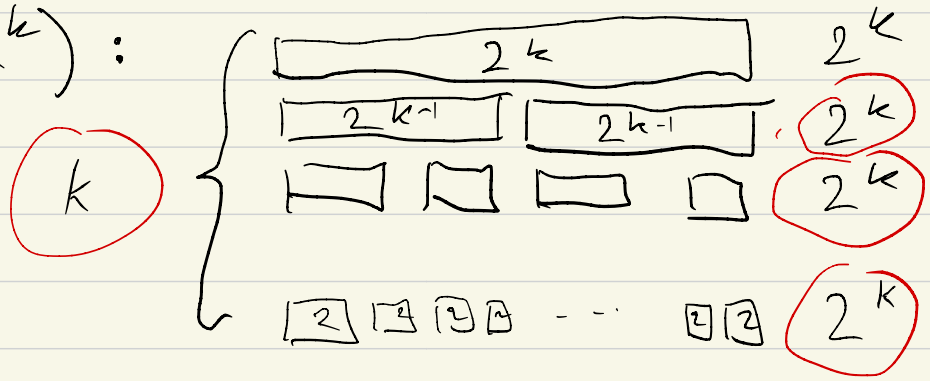


Algorithmus

1. löse beide Hälften rekursiv
2. berechne $S^*_{n/2}$ ($\leq n$ Additionen)
3. Ausgabe: maximum dieser 3 Teillösungen

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

Visuell ($n = 2^k$):



$k = \log_2 n$

$\leq k \cdot 2^k$

$\rightsquigarrow T(n) \leq n \cdot \log_2 n$

geht es besser?

Rekursion: neuer Versuch

Idee: mehr in Rekursion berechnen

⇝ Gesamtlösung schneller kombinieren

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

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

Basisfall: $R_1 \leftarrow a_1$

Rekurrenz:

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

Wiederholtes Einsetzen:

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

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

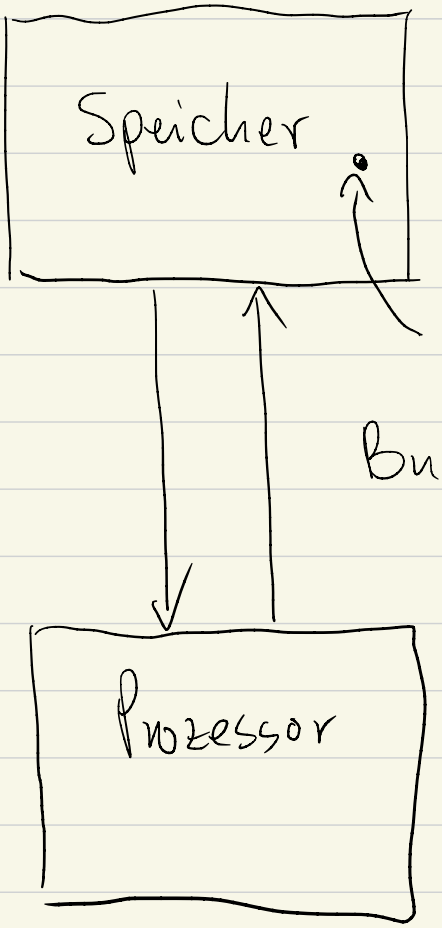
case optimal

nicht besprochen:

Standard Rechner modell:

gute Wahl von elementaren Operation

(für fast alle Berechnungs probleme)



Liste von Speicherzellen
- beliebig viele
- frei addressier bar

ite Speicherzelle

Bus (verbindet Speicher und Prozessor)

Führt elementare Operationen aus
- Lesen/Schreiben von Speicher zellen
- Vergleichen (=, >, <)
- Arithmetik (+, -, ·, /)

- Laufzeit
- worst-case Laufzeit