

# Algorithmen 1

## Teile und Herrsche Karatsubas Algorithmus und Master-Theorem



# Multiplikation: Idee → Code

## Wie geht das nochmal?

$$\begin{array}{r}
 \overbrace{5\ 6\ 7\ 8}^a \cdot \overbrace{1\ 2\ 3\ 4}^b \\
 \underline{5\ 6\ 7\ 8} \\
 \underline{1\ 1\ 3\ 5\ 6} \\
 \underline{1\ 7\ 0\ 3\ 4} \\
 + \underline{2\ 2\ 7\ 1\ 2} \\
 \hline
 7\ 0\ 0\ 6\ 6\ 5\ 2
 \end{array}$$

- berechne  $a \cdot b_i$  für jede Ziffer  $b_i$  von  $b = (b_{n-1} \dots b_0)$
- verschiebe Ergebnis um  $i$  Stellen nach links
- addiere Ergebnisse

## Hohe Abstraktionsebene

- leicht zu verstehen
- schwer zu implementieren

## Konkretisierung mittels Pseudocode

```

mult( $a, b$ )                                     //  $O(n^2)$ 
|
| total := 0                                       //  $O(1)$ 
| for  $i$  from 0 to  $n - 1$  do                       //  $O(n^2)$ 
| |
| | prod :=  $a \cdot b_i \cdot 10^i$                  //  $O(n)$ 
| | total := total + prod                          //  $O(n)$ 
| return total                                    //  $O(n)$ 

```

```

 $a \cdot b_i$                                        //  $O(n)$ 
|
| carry := 0                                       //  $O(1)$ 
| for  $j$  from 0 to  $n - 1$  do                       //  $O(n)$ 
| |
| | ( $carry, c_j$ ) :=  $a_j \cdot b_i + carry$          //  $O(1)$ 
| |  $c_n := carry$                                   //  $O(1)$ 
| return ( $c_n, \dots, c_0$ )                       //  $O(n)$ 

```

# Einschub: Pseudocode

## Nutzen

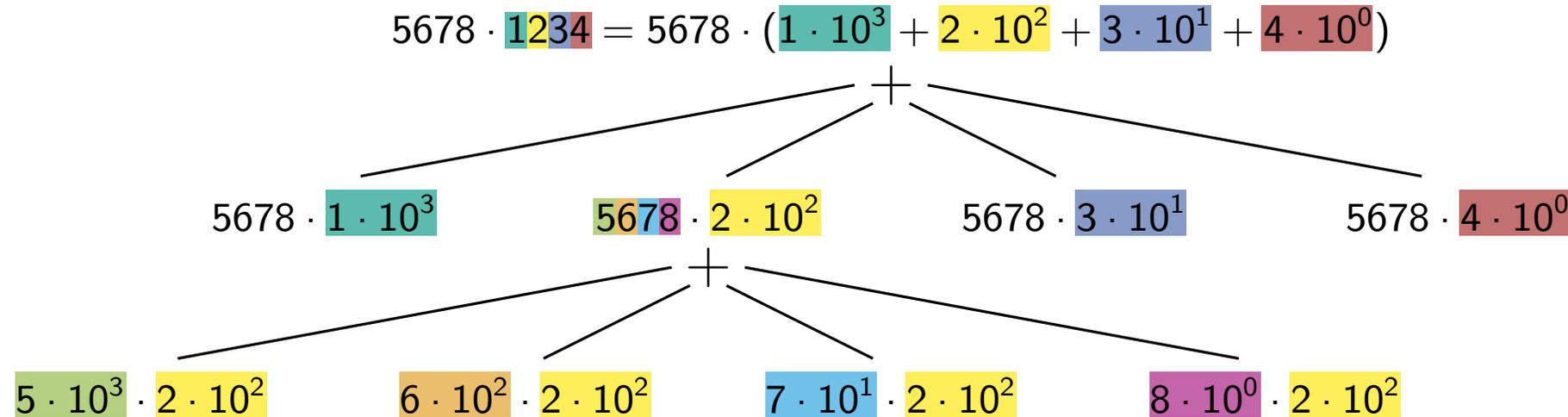
- interpoliert zwischen hoher und niedriger Abstraktionsebene
- viele mächtige Operationen → hohe Abstraktionsebene → Ideenvermittlung
- nur primitive Operationen → tiefe Abstraktionsebene → tatsächliche Implementierung
- kann helfen von der high-level Idee zu einer Implementierung zu kommen
- kann bei der Analyse (Laufzeit und Korrektheit) helfen

## Regeln

- in der Vorlesung: Syntax wird immer on-the-fly miterklärt
- flexibler als bei richtiger Programmiersprache (z.B. mathematische Formeln)
- man kann auch etwas kreativ sein, sollte es aber konsistent halten
- Hauptziele:
  - gut lesbar (für Menschen)
  - richtige Abstraktionsebene

# Teile und Herrsche

## Schriftliche Multiplikation

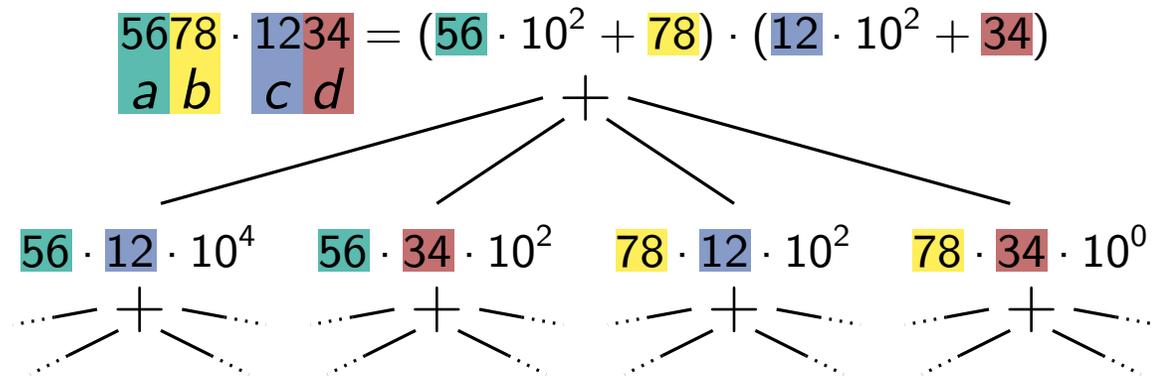


### Idee im Kern

- spalte schwieriges Problem in mehrere leichtere Probleme
- wiederhole das, bis die Instanz trivial ist
- hier: Problem selbst wird substantziell leichter (Zahl  $\rightarrow$  Ziffer)
- allgemein: Problem bleibt gleich, aber Instanz wird leichter (z.B. kleineres  $n$ )

# Teile und Herrsche

## Multiplikation mit anderer Zerlegung



## Größe des Rekursionsbaums

- jeder Knoten hat 4 Kinder  $\rightarrow$  Lage  $i$  hat  $4^i$  Knoten
- $n$  wird mit jeder Lage halbiert  $\rightarrow \log_2(n)$  Lagen

- Knoten insgesamt:  $\sum_{i=0}^{\log_2(n)} 4^i \in \Theta(4^{\log_2(n)}) = \Theta(n^2)$

- Achtung: das heißt noch nicht, dass die Laufzeit auch  $\Theta(n^2)$  ist (stimmt in diesem Fall aber)

**mult**( $x, y$ )

// assumption:  $n = |x| = |y| = 2^k$  for  $k \in \mathbb{N}$

**if**  $n = 1$  **return**  $x \cdot y$

$a, b$  := first and second half of  $x$

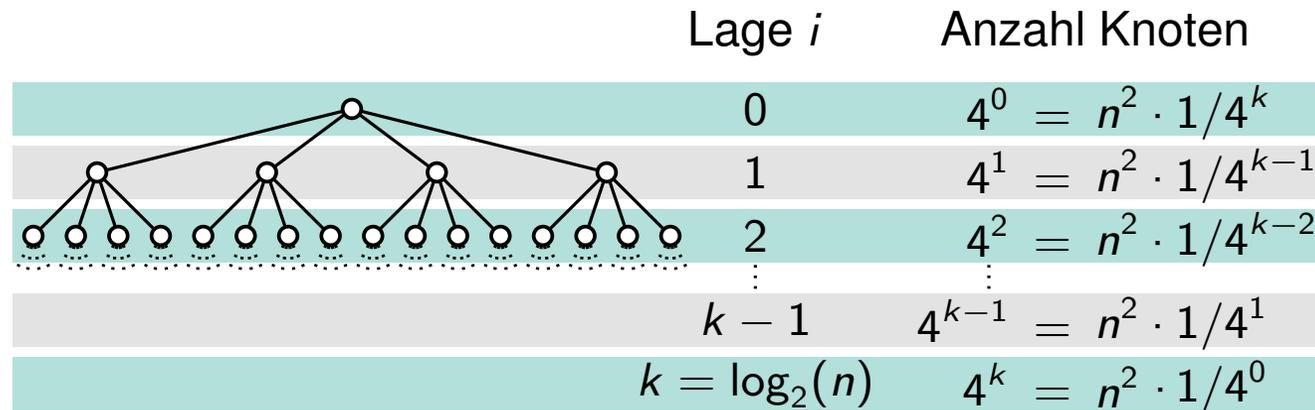
$c, d$  := first and second half of  $y$

**return** **mult**( $a, c$ )  $\cdot 10^n$  + **mult**( $b, d$ )

+ (**mult**( $a, d$ ) + **mult**( $c, b$ ))  $\cdot 10^{n/2}$

# Einschub: Exponentielle Summen & geometrische Reihen

## Exponentiell wachsende Lagen



## Summe der Knoten

klein  $\rightarrow$  groß

groß  $\rightarrow$  klein

$$\sum_{i=0}^k 4^i$$

$$\sum_{i=0}^k n^2 \cdot \left(\frac{1}{4}\right)^i = n^2 \cdot \sum_{i=0}^k \left(\frac{1}{4}\right)^i$$

geometrische Summe  $\in \Theta(1)$

$\in \Theta(n^2)$

## Mentaler Shortcut: Summen, die exponentiell schrumpfen/wachsen

- $\sum_i c^i$  wird asymptotisch vom größten Summanden dominiert

- also:  $\sum_{i=a}^b c^i \in \Theta(c^a + c^b)$

( $c \in (0, 1) \cup (1, \infty)$ ) konstant

( $a, b > 0$  können von  $n$  abhängen)

# Ziel: weniger rekursive Aufrufe

## Bisher

- $xy = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)$   
 $= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd$
- vier rekursive Multiplikationen

## Idee zur Verbesserung

- ignoriere zunächst die 10er-Potenzen
- $(a + b) \cdot (c + d) = ac + ad + bc + bd$
- wenn wir  $(a + b) \cdot (c + d)$ ,  $ac$  und  $bd$  kennen, können wir daraus  $ad + bc$  erhalten
- berechne  $(a + b) \cdot (c + d)$ ,  $ac$  und  $bd$
- berechne dann  $(ad + bc) = (a + b) \cdot (c + d) - ac - bd$
- nur drei rekursive Multiplikationen
- dafür etwas mehr Additionen

## mult(x, y)

// assumption:  $n = |x| = |y| = 2^k$  for  $k \in \mathbb{N}$

if  $n = 1$  return  $x \cdot y$

$a, b$  := first and second half of  $x$

$c, d$  := first and second half of  $y$

return  $\text{mult}(a, c) \cdot 10^n + \text{mult}(b, d)$

$+ (\text{mult}(a, d) + \text{mult}(c, b)) \cdot 10^{n/2}$

# Karatsubas Algorithmus

**mult**( $x, y$ )

// assumption:  $n = |x| = |y| = 2^k$  for  $k \in \mathbb{N}$

**if**  $n = 1$  **return**  $x \cdot y$

$a, b$  := first and second half of  $x$

$c, d$  := first and second half of  $y$

$ac$  := **mult**( $a, c$ )

$bd$  := **mult**( $b, d$ )

sum := **mult**(( $a + b$ ), ( $c + d$ )) -  $ac$  -  $bd$

**return**  $ac \cdot 10^n + \text{sum} \cdot 10^{n/2} + bd$

## Erinnerung

- $xy = ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd$
- $(ad + bc) = (a + b) \cdot (c + d) - ac - bd$

## Anzahl Knoten im Rekursionsbaum

- 3 Kinder  $\rightarrow 3^i$  Knoten auf Lage  $i$
- $n$  wird immer halbiert  $\rightarrow \log_2(n)$  Lagen

- $$\sum_{i=0}^{\log_2(n)} 3^i \in \Theta \left( 3^{\log_2(n)} \right) = \Theta \left( n^{\log_2(3)} \right)$$

$$\subseteq O \left( n^{1,585} \right)$$

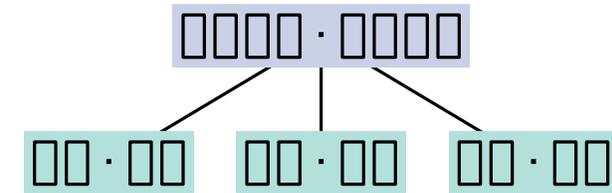
## Offene Fragen

- Wie viel Arbeit macht man in jedem Knoten? Sind die zusätzlichen Additionen schlimm?
- Können wir annehmen, dass  $n$  eine Zweierpotenz ist? Könnte nicht  $|a + b| > \frac{n}{2}$  sein?

# Karatsubas Algorithmus – Laufzeitanalyse

## Arbeit pro Knoten

- linear in der Größe der Zahlen
- also:  $\Theta\left(n \cdot \frac{1}{2^i}\right)$  für einen Knoten auf Lage  $i$



## Arbeit pro Lage

- $3^i$  Knoten auf Lage  $i$
- Gesamtarbeit für Lage  $i$ :  $3^i \cdot \Theta\left(n \cdot \frac{1}{2^i}\right) = \Theta\left(n \cdot \left(\frac{3}{2}\right)^i\right)$

## Arbeit insgesamt

- exponentielle Summe  $\rightarrow$  dominiert durch letzte Lage  $i = \log_2(n)$
- also:  $\Theta\left(n \cdot \left(\frac{3}{2}\right)^{\log_2(n)}\right) = \Theta\left(n \cdot \left(\frac{n^{\log_2(3)}}{n}\right)\right) = \Theta\left(n^{\log_2(3)}\right)$
- Sanity Check: letzte Lage hat  $n^{\log_2(3)}$  Knoten und konstanten Aufwand pro Knoten

# Rundungsfehler?

**Annahme:  $n = |x| = |y|$  ist Zweierpotenz**

- Problem:  $a + b$  hat ggf.  $\frac{n}{2} + 1$  Ziffern
- also: keine Zweierpotenz mehr im rekursiven Aufruf

**Umgang mit ungeradem  $n$**

- $n$  ungerade  $\Rightarrow |a| = \lceil \frac{n}{2} \rceil = \frac{n}{2} + \frac{1}{2}$
- damit hat  $a + b$  bis zu  $\frac{n}{2} + \frac{3}{2}$  Ziffern

**Was ändert sich bei der Analyse?**

- $n_i = \# \text{Ziffern auf Lage } i \text{ (} n_0 = n \text{)}$
- bisher:  $n_i = \frac{n}{2^i}$
- jetzt:  $n_{i+1} \leq \frac{n_i}{2} + \frac{3}{2} \leq \frac{n}{2^i} + 3$
- kein asymptotischer Unterschied  $\rightarrow$  Laufzeit  $\Theta(n^{\log_2(3)})$

Achtung: Abbruch der Rekursion müsste man hier nochmal genauer anschauen.  
 Möglichkeit 1: Zeige, dass es für konstant viele Ziffern nach konstanter Zeit terminiert.  
 Möglichkeit 2: Brich die Rekursion nicht erst bei  $n = 1$ , sondern etwas früher ab.

**mult**( $x, y$ )

...

$a, b :=$  first, second half of  $x$

...

sum := **mult**(( $a + b$ ), (...)) ...

## Nebenrechnung

$$n_0 = n$$

$$n_1 = \frac{n}{2} + \frac{3}{2} \quad \left. \begin{array}{l} \nearrow \\ \cdot \frac{1}{2} + \frac{3}{2} \end{array} \right\}$$

$$n_2 = \frac{n}{2^2} + \frac{3}{2^2} + \frac{3}{2} \quad \left. \begin{array}{l} \nearrow \\ \cdot \frac{1}{2} + \frac{3}{2} \end{array} \right\}$$

$$n_3 = \frac{n}{2^3} + \frac{3}{2^3} + \frac{3}{2^2} + \frac{3}{2} \quad \left. \begin{array}{l} \nearrow \\ \cdot \frac{1}{2} + \frac{3}{2} \end{array} \right\}$$

$$n_i = \frac{n}{2^i} + 3 \cdot \sum_{j=1}^i \frac{1}{2^j} \leq \frac{n}{2^i} + 3$$

# Karatsubas Algorithmus

## Theorem

Karatsubas Algo. multipliziert zwei  $n$ -Ziffern Zahlen in  $\Theta(n^{\log_2(3)}) \subseteq O(n^{1,585})$  Zeit.

## Analyse via Rekursionsbaum

- Wie viele Knoten sind auf Lage  $i$ ?
- Wie groß ist das  $n$  auf Lage  $i$ ?
- Wie viel Zeit kostet ein Knoten in Lage  $i$ ?



Warum ist das so viel Arbeit?

Muss ich das für jeden rekursiven Algorithmus so analysieren?

Kann man da nicht irgendwie abkürzen?

**mult**( $x, y$ )

**if**  $n = 1$  **return**  $x \cdot y$

$a, b :=$  first and second half of  $x$

$c, d :=$  first and second half of  $y$

$ac :=$  **mult**( $a, c$ )

$bd :=$  **mult**( $b, d$ )

$sum :=$  **mult**(( $a + b$ ), ( $c + d$ ))  $- ac - bd$

**return**  $ac \cdot 10^n + sum \cdot 10^{n/2} + bd$

# Rekurrenzen

## Laufzeit als Formel

$$T(n) = \begin{cases} \Theta(1) & \text{wenn } n = 1, \\ 3 \cdot T\left(\frac{n}{2}\right) + \Theta(n) & \text{wenn } n > 1. \end{cases}$$

Anzahl rekursiver Aufrufe  
→ Anzahl Kinder im Baum

Laufzeit für den Aufruf selbst  
→ Kosten für einen Knoten

neue Instanzgrößen

→ Änderung von  $n$  zwischen Lagern

$$\Rightarrow T(n) \in \Theta\left(n^{\log_2(3)}\right)$$

## Können wir das verallgemeinern?

$$T(n) = \begin{cases} \Theta(1) & \text{wenn } n = 1, \\ a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c) & \text{wenn } n > 1. \end{cases}$$

$$\Rightarrow T(n) \in \Theta(??)$$

## mult( $x, y$ )

**if**  $n = 1$  **return**  $x \cdot y$

$a, b :=$  first and second half of  $x$

$c, d :=$  first and second half of  $y$

$ac :=$  **mult**( $a, c$ )

$bd :=$  **mult**( $b, d$ )

$sum :=$  **mult**(( $a + b$ ), ( $c + d$ ))  $- ac - bd$

**return**  $ac \cdot 10^n + sum \cdot 10^{n/2} + bd$

## Analyse via Rekursionsbaum

- Wie viele Knoten sind auf Lage  $i$ ?
- Wie groß ist das  $n$  auf Lage  $i$ ?
- Wie viel Zeit kostet ein Knoten in Lage  $i$ ?

# Lösen allgemeiner Rekurrenzen

## Können wir das verallgemeinern?

$$T(n) = \begin{cases} \Theta(1) & \text{wenn } n = 1, \\ a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c) & \text{wenn } n > 1. \end{cases}$$

## Analyse via Rekursionsbaum

- Wie viele Knoten sind auf Lage  $i$ ?
- Wie groß ist das  $n$  auf Lage  $i$ ?
- Wie viel Zeit kostet ein Knoten in Lage  $i$ ?

### Situation für Lage $i$

- #Knoten:  $a^i$
- Instanzgröße:  $\frac{n}{b^i}$
- Kosten pro Knoten:  $\frac{n^c}{b^{ci}}$

### Gesamtkosten für Lage $i$ Gesamtkosten für den Baum

$$a^i \cdot \frac{n^c}{b^{ci}} = \left(\frac{a}{b^c}\right)^i \cdot n^c$$

$$\sum_i \left(\frac{a}{b^c}\right)^i \cdot n^c$$

### Anzahl Lagen

$$\log_b(n)$$

exponentielle Summe  
 → dominiert von größtem Summanden  
 (außer wenn  $a/b^c = 1$ )

$$a < b^c$$

$$T(n) \in \Theta(n^c)$$

(Kosten der Wurzel dominiert)

$$a = b^c$$

$$T(n) \in \Theta(n^c \log n)$$

(Kosten auf jeder der  $\log_b(n)$  Lagen gleich)

$$a > b^c$$

$$T(n) \in \Theta(n^{\log_b(a)})$$

(Kosten der  $n^{\log_b(a)}$  Blätter dominiert)

# Master-Theorem

## Theorem

Sei  $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$  mit  $f(n) \in \Theta(n^c)$  und  $T(1) \in \Theta(1)$ . Dann gilt

$$T(n) \in \begin{cases} \Theta(n^c) & \text{wenn } a < b^c, \\ \Theta(n^c \log n) & \text{wenn } a = b^c, \\ \Theta(n^{\log_b(a)}) & \text{wenn } a > b^c. \end{cases}$$

## Ein Name, viele Theoreme

- es gibt viele Varianten des Master-Theorems
- Einschränkung  $f(n) \in \Theta(n^c)$  kann man aufweichen
- asymmetrische Verzweigung: z.B.  $T(n) = T\left(\frac{n}{2}\right) + 2T\left(\frac{n}{4}\right)$
- Runden auf ganze Zahlen

aktuelles Papier zum Thema:

William Kuszmaul, Charles E. Leiserson

Floors and Ceilings in Divide-and-Conquer Recurrences

Symposium on Simplicity in Algorithms (SOSA 2021)

# Zusammenfassung

## Karatsubas Algorithmus

[Karatsuba; 1962]

- algorithmische Technik: Teile und Herrsche
- Multiplikation zweier  $n$ -Ziffern Zahlen in  $\Theta(n^{\log_2(3)}) \subseteq O(n^{1,585})$  Zeit
- noch schneller:
  - $\Theta(n^{\log_3(5)}) \subseteq O(n^{1,465})$
  - $O(n \cdot \log n \cdot \log \log n)$
  - $O(n \cdot \log n \cdot 2^{O(\log^* n)})$

[Toom, Cook; 1963–1966]

[Schönhage, Strassen; 1971]

[Fürer; 2007]

## Auflösung von Rekurrenzen

- Analyse des Rekursionsbaums + exponentielle Summen  $\rightarrow$  Master-Theorem
- Master-Theorem funktioniert nicht immer (z.B.  $T(n) = 2T(n - 1)$ )
  - weitere Techniken: branching vector, vollständige Induktion
  - nützlich: WolframAlpha



$T(n) = 3 * T(n / 2) + n$  =

---

Asymptotic bound

$T(n) \in \Theta\left(n^{\log(3)/\log(2)}\right)_{n \rightarrow \infty}$

## Pseudocode