11 Divide-and-Conquer und Rekursionsgleichungen

Der Ansatz des *Divide-and-Conquer* lässt sich auf Probleme anwenden, die sich auf die folgende Weise lösen lassen:

1. Das Problem in Teilprobleme aufteilen. (*divide*)
2. Die einzelnen Teilprobleme (in der Regel rekursiv) unabhängig voneinander lösen.
3. Die erhaltenen Lösungen der Teilprobleme zu einer Gesamtlösung zusammensetzen. (*conquer*)

Typische Beispiele sind *Binäre Suche*, *Meregesort* und *Quicksort*.

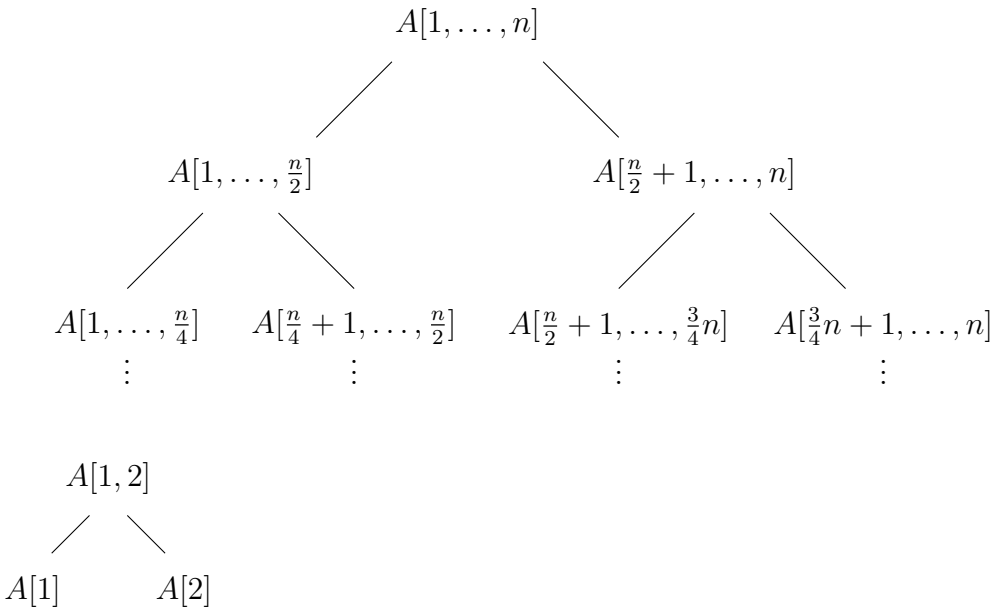
11.1 Mergesort

Algorithmus 20: Mergesort($A[1, \ldots, n]$)

Input : Array $A[1..n]$ mit n Elementen
Output : Das Array A sortiert.

```
/* Ein- bzw. nullelementige Folge ist sortiert. */
```
1 **if** $|A| == 1$ *oder* $|A| == 0$ **then**
2 | **return** A;
3 **end**
```
/* Zwei Teilfelder der Größe n/2 bilden und diese sortieren. */
```
4 B_1 =Mergesort($A[1, \ldots, \lfloor \frac{n}{2} \rfloor]$);
5 B_2 =Mergesort($A[\lfloor \frac{n}{2} \rfloor + 1, \ldots, n]$);
```
/* Neues, sortiertes, Feld aus B1 und B2 bilden. Das ist
   einfach, da die beiden Teilfelder bereits sortiert sind. */
```
6 **return** *„Mischung“* von B_1 und B_2;

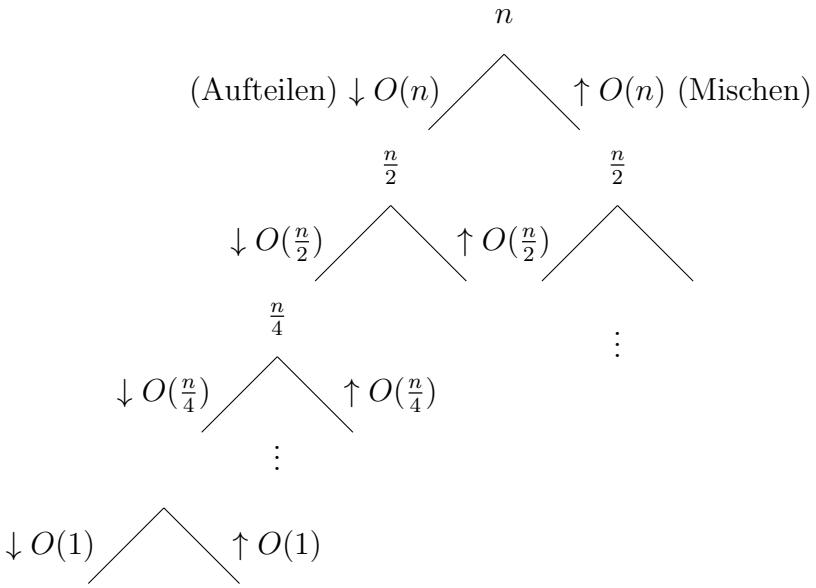
Betrachten wir den Aufrufbaum von *Mergesort* der Einfachheit halber für $n = 2^k$, einer Zweierpotenz:



Die Tiefe des Aufrufbaumes ist hier genau $\log_2 n$. Betrachten wir die Laufzeit:

- Blätter: $O(n)$
- Aufteilen: bei m Elementen $O(m)$
- Zusammensetzen von zweimal $\frac{m}{2}$ Elementen: $O(m)$

Damit erhalten wir insgesamt den folgenden Aufwand:



Für die Blätter: $n \cdot O(1) = O(n)$

Für das Aufteilen: $c \cdot n + c \cdot \frac{n}{2} + c \cdot \frac{n}{4} + \ldots + c \cdot 1 + c \cdot \frac{n}{2} + \ldots$

Wo soll das enden? Wir addieren einmal in einer anderen Reihenfolge:

$$
\begin{array}{ll}
\text{Stufe:} & \\
1 & c \cdot n + \\
2 & \quad + c \cdot \frac{n}{2} + c \cdot \frac{n}{2} + \\
3 & \quad\quad + c \cdot (\frac{n}{4} + \frac{n}{4} + \frac{n}{4} + \frac{n}{4}) + \\
4 & \quad\quad\quad + c \cdot (\underbrace{\frac{n}{8} + \cdots + \frac{n}{8}}_{8\times}) + \\
\vdots & \\
\log_2 n & \quad\quad\quad + c \cdot (\underbrace{1 + \cdots + 1}_{n\times}) \\
& = \log_2 n \cdot c \cdot n = O(n \cdot \log n)
\end{array}
$$

Ebenso für das Mischen: $O(n \cdot \log n)$

Insgesamt haben wir dann $O(n \cdot \log n) + O(n) = O(n \cdot \log n)$.

- *Wichtig:* Richtige Reihenfolge beim Zusammenaddieren. Bei Bäumen oft stufenweise.
- Die eigentliche Arbeit beim divide-and-conquer geschieht im Aufteilen *und* im Zusammenfügen. Der Rest ist rekusiv.
- Mergesort lässt sich auch einfach bottom-up ohne Rekursion lösen, da der Aufrufbaum unabhängig von der Eingabe *immer* die gleiche Struktur hat.

  1. $A[1,2], A[3,4], \ldots, A[n-1,n]$ Sortieren, dann
  2. $A[1,\ldots,4], A[5,\ldots,8], \ldots$ Sortieren
  3. $\ldots$

  Auch $O(n \cdot \log n)$.

- Heapsort sortiert ebenfalls in $O(n \cdot \log n)$ und
- Bubblesort, Insertion Sort, Selection Sort in $O(n^2)$. Das sind aber keine divide-and-conquer Algorithmen.

11.2 Quicksort

Eingabe: Array $A[1, \ldots, n]$ of „geordneter Datentyp“

Algorithmus 21: Quicksort($A[1, \ldots n]$)

Input : Array $A[1..n]$ mit n Elementen
Output : Das Array A sortiert.

```
1  if |A| == 1 oder |A| == 0 then
2  |  return A;
3  end
4  a = A[1];                 /* Erstes Element aus A als Pivotelement. */
5  B_1 = [ ]; B_2 = [ ];          /* Zwei leere Felder der Größe n − 1. */
6  for j = 2 to n do
7  |  if A[j] ≤ a then
8  |  |  A[j] als nächstes Element zu B_1;
9  |  else /* A[j] > a */
10 |  |  A[j] als nächstes Element zu B_2;
11 |  end
12 end
   /* Elemente ≤ a und > a getrennt sortieren. */
13 B_1 = Quicksort(B_1);
14 B_2 = Quicksort(B_2);
   /* Die sortierten Teilfelder zum Gesamtfeld zusammensetzen. */
15 return B_1.a.B_2;
```

Beachte: Die Prozedur Partition erlaubt es, mit dem Array A alleine auszukommen. Das heisst es muß nichts kopiert werden und an *Quicksort* werden nur die *Grenzen* der Teilarrays übergeben. Dann sieht das so aus:

Algorithmus 22: Quicksort(A, l, r) mit Partition

Input : Array A per Referenz; l, r linke und rechte Grenze.
Output : $A[l, \ldots, r]$ sortiert.

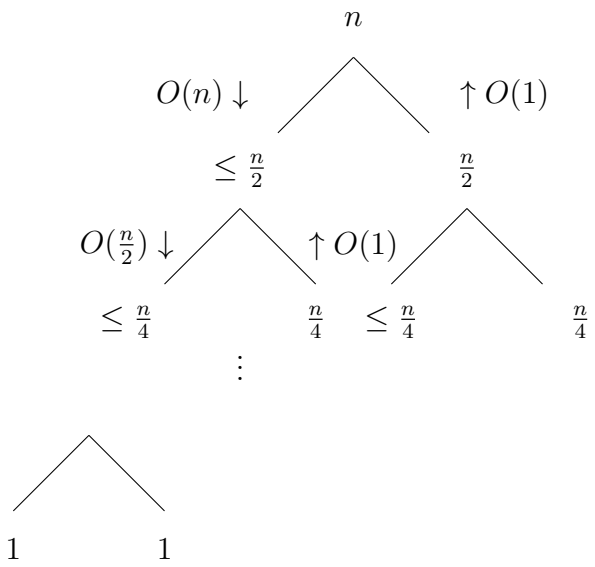
```
1  if l < r then
2  |  j = Partition(A, l, r);
3  |  Quicksort(A, l, j − 1);
4  |  Quicksort(A, j + 1, r);
5  end
```

Prozedur Partition($A[1,\dots n], l, r$)

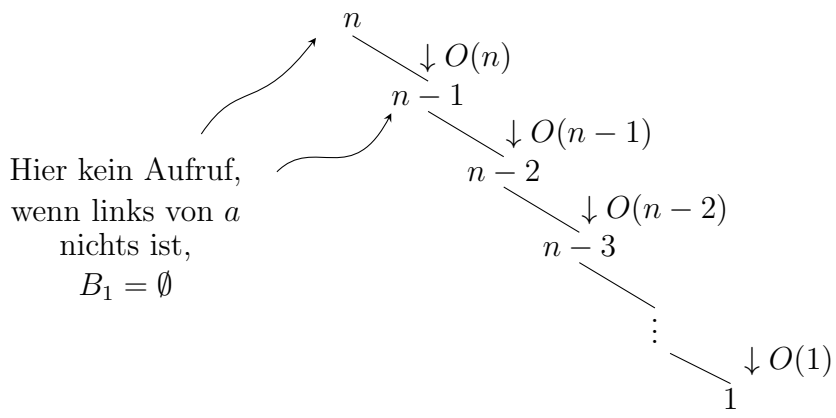
Input : Array A als *Referenz*, es wird direkt auf A gearbeitet. Indizes l, r linke und rechte Grenze des Bereichs, der partitioniert werden soll.
Output : Index j des letzten Elementes von A, das $\leq$ dem Pivotelement ist.

```
1 pivot = A[l]; i = l; j = r;
2 while i < j do
  |  /* Alle, die bezüglich des Pivotelementes richtig stehen,
  |     übergehen. */
3 |  while A[i] < pivot do  i = i + 1;
4 |  while A[j] > pivot do  j = j − 1;
5 |  if i < j then
6 |  |  t = A[i]; A[i] = A[j]; A[j] = t;   /* A[i] und A[j] vertauschen. */
7 |  end
8 end
9 return j;
```

Sehen wir uns einige mögliche Prozedurbäume an:



Hier ist die Laufzeit $O(n \cdot \log n) = O(n) + 2 \cdot O(\frac{n}{2}) + 4 \cdot O(\frac{n}{4}) + \ldots + n \cdot O(1) + \underbrace{O(n)}_{\text{Blätter}}$.
Das stellt praktisch den Idealfall dar, den wir uns normalerweise wünschen. Es ist aber auch folgendes möglich, wenn die Eingabe „ungünstig“ ist.



Jetzt erhalten wir für die Laufzeit:

$$\begin{aligned} n + n - 1 + \ldots + 3 + 2 + 1 = \frac{n \cdot (n-1)}{2} &= O(n^2) \quad \text{(Aufteilen)} \\ 1 + 1 + \ldots + 1 + 1 + 1 &= O(n) \quad \text{(Zusammensetzen)} \end{aligned}$$

Also: $O(n^2)$ und auch $\Omega(n^2)$.

Wieso wird Quicksort trotzdem in der Praxis häufig eingesetzt? In der Regel tritt ein „gutartiger“ Baum auf, und wir haben $O(n \cdot \log n)$. Vergleiche innerhalb eines Aufrufes finden immer mit einem festem a statt. Dieser Wert kann in einem Register gehalten werden. Damit erreicht man ein schnelleres Vergleichen. Dagegen sind beim Mischen von Mergesort öfter beide Elemente des Vergleiches neu.

Der Aufrufbaum von Quicksort ist vorher nicht zu erkennen. Dadurch ist keine nicht-rekursive Implementierung wie bei Mergesort möglich. Nur mit Hilfe eines (Rekursions-)Kellers.

11.3 Rekursionsgleichungen

Noch einmal zu Mergesort. Wir betrachten den Fall, dass n eine Zweierpotenz ist. Dann sind $\frac{n}{2}, \frac{n}{4}, \ldots, 1 = 2^0$ ebenfalls Zweierpotenzen. Sei

$$T(n) = \text{ worst-case-Zeit bei } A[1, \ldots, n],$$

dann gilt für ein geeignetes c:

$$\begin{aligned} T(1) &\leq c \\ T(n) &\leq c \cdot n + 2 \cdot T\left(\frac{n}{2}\right) \end{aligned}$$

- Einmaliges Teilen $O(n)$
- Mischen $O(n)$
- Aufrufverwaltung hier $O(n)$

Betrachten wir nun den Fall

$$\begin{aligned} T(1) &= c \\ T(n) &= c \cdot n + 2 \cdot T\left(\frac{n}{2}\right). \end{aligned}$$

Dann erhalten wir durch Abwickeln der rekursiven Gleichung:

$$\begin{aligned} T(n) &= c \cdot n + 2 \cdot T\left(\frac{n}{2}\right) \\ &= c \cdot n + 2 \cdot c \cdot \frac{n}{2} + 2 \cdot 2 \cdot T\left(\frac{n}{4}\right) \\ &= c \cdot n + c \cdot n + 4 \cdot c \cdot \frac{n}{2} + 2 \cdot 2 \cdot 2 \cdot T\left(\frac{n}{8}\right) \\ &\vdots \\ &= c \cdot n \cdot \log n + 2 \cdot 2 \cdot \cdots \cdot 2 \cdot T(1) \\ &= c \cdot n \cdot \log n + c \cdot n \\ &= O(c \cdot n \log n) = O(n \cdot \log n) \end{aligned}$$

Schließlich noch ein strenger Induktionsbeweis, dass $T(n) \leq d \cdot n \cdot \log n = O(n \cdot \log n)$ ist.

Induktionsanfang:

$$\begin{array}{rcll} T(1) & = & c \neq O(1 \cdot \log 1) = 0, \text{ das stimmt so nicht. Also fangen wir bei } T(2) \text{ an.} \\ T(2) & = & c \cdot 2 + 2 \cdot T(1) = 4 \cdot c & \\ & \leq & d \cdot 2 \cdot \log 2 \ = 2 \cdot d & \text{(Gilt für } d \geq 2 \cdot c\text{.)} \\ & = & O(2 \cdot \log 2) \quad \checkmark & \end{array}$$

Die Behauptung gilt also für $n = 2$.

Induktionsschluss:

$$\begin{array}{rcll} T(2n) & = & c \cdot 2n + 2 \cdot T(n) & \text{(nach Definition)} \\ & \leq & c \cdot 2n + 2 \cdot d \cdot n \cdot \log n & \text{(nach Induktionsvoraussetzung)} \\ & \leq & d \cdot n + 2 \cdot d \cdot n \cdot \log n & \text{(mit } d \geq 2 \cdot c \text{ von oben)} \\ & = & d \cdot 2n - d \cdot n + 2 \cdot d \cdot n \cdot \log n & \\ & = & d \cdot 2n \cdot (1 + \log n) - d \cdot n & \\ & \leq & d \cdot 2n \cdot \log 2n & \\ & = & O(2n \cdot \log 2n) \quad \square & \end{array}$$

Wie sieht das bei Quicksort aus? Wir betrachten

$$T(n) = \text{ worst-case-Zeit von Quicksort auf } A[1, \ldots, n].$$

Dann gelten die folgenden Ungleichungen. Damit sind alle Möglichkeiten beschrieben, wie das Feld aufgeteilt werden kann. Einer dieser Fälle muß der worst-case sein.

$$\begin{array}{rcl} T(n) & \leq & c \cdot n + T(n-1) \\ T(n) & \leq & c \cdot n + T(1) + T(n-2) \\ T(n) & \leq & c \cdot n + T(2) + T(n-3) \\ & \vdots & \\ T(n) & \leq & c \cdot n + T(n-2) + T(1) \\ T(n) & \leq & c \cdot n + T(n-1) \end{array}$$

Also haben wir

$$\begin{array}{rcl} T(1) & \leq & c \\ T(n) & \leq & c \cdot n + \max\Big\{\big\{T(n-1)\big\} \cup \big\{T(i) + T(n-i-1) \mid 1 \leq i \leq n-2\big\}\Big\}. \end{array}$$

Dann gilt $T(n) \leq d \cdot n^2$ für ein geeignetes d.

Induktionsanfang:

$$\begin{array}{lcll} T(1) & \leq & c \cdot 1 \leq d \cdot 1^2 & \text{gilt für } d \geq c. \checkmark \\ T(2) & \leq & c \cdot 2 + T(1) & \\ & \leq & 3c \leq d \cdot 2^2 & \text{gilt für } d \geq \frac{3}{4}c. \checkmark \end{array}$$

Wir wählen $d \geq c$, dann gilt die Behauptung für alle $n \leq 2$.

Indukstionsschluss:

$$\begin{array}{rcl} & & \text{(nach Definition)} \\ T(n+1) & \leq & c(n+1) + \max\Big\{\big\{T(n)\big\} \cup \big\{T(i) + T((n+1) - i - 1) \mid 1 \leq i \leq (n+1) - 2\big\}\Big\} \\ & = & c(n+1) + \max\Big\{\big\{T(n)\big\} \cup \big\{T(i) + T(n-i) \mid 1 \leq i \leq n-1\big\}\Big\} \\ & & \\ & & \text{(nach Induktionsvoraussetzung)} \\ & \leq & c(n+1) + \max\Big\{\big\{dn^2\big\} \cup \big\{di^2 + d(n-i)^2 \mid 1 \leq i \leq n-1\big\}\Big\} \\ & = & c(n+1) + \max\Big\{\big\{dn^2\big\} \cup \underbrace{\big\{ dn^2 + 2di\underbrace{(i-n)}_{<0} \mid 1 \leq i \leq n-1\big\}}_{<dn^2}\Big\} \\ & \leq & c(n+1) + dn^2 \\ & & \\ & & \text{(mit } d \geq c\text{)} \\ & \leq & d(n^2+n+1) \leq d(n+1)^2 \qquad \square \end{array}$$

Aufgabe: Stellen Sie die Rekursionsgleichung für eine rekursive Version der binären Suche auf und schätzen sie diese bestmöglich ab.

11.4 Multiplikation großer Zahlen

Im Folgenden behandeln wir das Problem der Multiplikation großer Zahlen. Bisher haben wir angenommen:

$$\text{Zahlen passen in ein Speicherwort} \Longrightarrow \text{arithmetische Ausdrücke in } O(1).$$

Man spricht vom uniformen Kostenmaß. Bei größeren Zahlen kommt es auf den Multiplikationsalgorithmus an. Man misst die Laufzeit in Abhängigkeit von der #Bits, die der Rechner verarbeiten muss. Das ist bei einer Zahl n etwa $\log_2 n$.

Genauer: Für $n \geq 1$ werden $\lfloor \log_2 n \rfloor + 1$ Bits benötigt. Man misst nicht in der Zahl selbst!

Die normale Methode, zwei Zahlen zu *addieren*, lässt sich in $O(n)$ Bitoperationen implementieren, wenn die Zahlen die Länge n haben.

$$\begin{array}{rccccc} & a_1 & \ldots & & a_n \\ + & b_1 & \ldots & & b_n \\ \hline & & \ldots & a_n + b_n & \end{array}$$

Multiplikation in $O(n^2)$:

$$\begin{array}{rll} (a_1 \ldots a_n) \cdot (b_1 \ldots b_n) = & (a_1 \ldots a_n) \cdot (b_1\ 0\ \ldots\ 0) & O(n) \\ + & (a_1 \ldots a_n) \cdot (b_2 0 \ldots\ 0) & O(n) \\ & \ddots & \\ + & (a_1 \ldots a_n) \cdot (b_n) & O(n) \\ \hline = & \textit{(Summenbildung)} & \end{array}$$

Zur Berechnung des Ergebnissen sind noch $n-1$ Additionen mit Zahlen der Länge $\leq 2n$ notwendig. Insgesammt haben wir dann $O(n^2)$ viele Operationen.

Mit divide-and-conquer geht es besser! Nehmen wir zunächst einmal an, n ist eine Zweierpotenz. Dann können wir die Zahlen auch so schreiben:

$$\begin{array}{ccccc} \overbrace{a_1 \ldots a_n}^{a:=} & = & \overbrace{a_1 \ldots a_{\frac{n}{2}}}^{a':=} & \overbrace{a_{\frac{n}{2}+1} \ldots a_n}^{a'':=} \\ \underbrace{b_1 \ldots b_n}_{b:=} & = & \underbrace{b_1 \ldots b_{\frac{n}{2}}}_{b':=} & \underbrace{b_{\frac{n}{2}+1} \ldots b_n}_{b'':=} \end{array}$$

Nun schreiben wir $a \cdot b$ als:

$$\begin{aligned} a \cdot b &= \underbrace{(a' \cdot 2^{\frac{n}{2}} + a'')}_{=a} \cdot \underbrace{(b' \cdot 2^{\frac{n}{2}} + b'')}_{=b} \\ &= (a' \cdot b') \cdot 2^n + (a' \cdot b'' + a'' \cdot b') \cdot 2^{\frac{n}{2}} + (a'' \cdot b'') \end{aligned}$$

Mit den vier Produkten, bei denen die Faktoren nur noch die Länge $\frac{n}{2}$ haben, machen wir rekursiv weiter. Das Programm sieht in etwa so aus:

Algorithmus 23: Multiplikation großer Zahlen

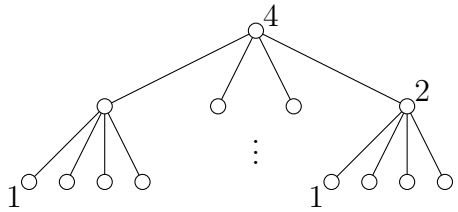
Input : a, b zwei Zahlen mit n Bits
Output : Produkt $a \cdot b$

1 **if** $n == 1$ **then return** $a \cdot b$;

`/* Divide */`
2 $a' = a_1 \dots a_{\frac{n}{2}}$; $a'' = a_{\frac{n}{2}+1} \dots a_n$;
3 $b' = b_1 \dots b_{\frac{n}{2}}$; $b'' = b_{\frac{n}{2}+1} \dots b_n$;
4 $m_1 = \text{Mult}(a', b')$; $m_2 = \text{Mult}(a', b'')$;
5 $m_3 = \text{Mult}(a'', b')$; $m_4 = \text{Mult}(a'', b'')$;

`/* Conquer */`
6 **return** $m_1 \cdot 2^n + (m_2 + m_3) \cdot 2^{n/2} + m_4$;

Der Aufrufbaum für $n = 4$ sieht dann wie unten aus. Die Tiefe ist $\log_2 4 = 2$.



Laufzeit? Bei $\text{Mult}(a, b)$ braucht der divide-Schritt und die Zeit für die Aufrufe selbst zusammen $O(n)$. Der conquer-Schritt erfolgt auch in $O(n)$. Zählen wir wieder pro Stufe:

1. Stufe:	$d \cdot n$	(von $O(n)$)
2. Stufe:	$4 \cdot d \cdot \frac{n}{2}$	
3. Stufe:	$4 \cdot 4 \cdot d \cdot \frac{n}{4}$	
$\vdots$		
$\log_2 n$-te Stufe:	$4^{\log_2 n - 1} \cdot d \cdot \frac{n}{2^{\log_2 n - 1}}$	
#Blätter:	$4^{\log_2 n} \cdot d$	

Das gibt:

$$\begin{aligned} T(n) &\leq \sum_{i=0}^{\log_2 n} 4^i \cdot d \cdot \frac{n}{2^i} \\ &= d \cdot n \cdot \sum_{i=0}^{log_2 n} 2^i \\ &= d \cdot n \cdot \frac{2^{\log_2 n+1} - 1}{2-1} \\ &= O(n^2) \qquad \text{mit } \sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1} \quad \text{für alle } x \neq 1, \textit{ geometrische Reihe.} \end{aligned}$$

Betrachten wir die induzierte Rekursionsgleichung. Annahme: $n = 2^k$ Zweierpotenz. (*Beachte:* $2^{\lfloor \log_2 n \rfloor} \leq n \leq 2^{\lfloor \log_2 n \rfloor + 1}$, das heißt zwischen n und $2n$ existiert eine Zweierpotenz.)

$$\begin{aligned} T(1) &= d \\ T(n) &= dn + 4 \cdot T\left(\frac{n}{2}\right) \end{aligned}$$

Versuchen $T(n) = O(n^2)$ durch Induktion zu zeigen.

1. Versuch: $T(n) \leq d \cdot n^2$

Induktionsanfang: ✓

Induktionsschluss:

$$\begin{aligned} T(n) &= dn + 4 \cdot T\left(\frac{n}{2}\right) \\ &\leq dn + dn^2 \quad > dn^2 \qquad \text{Induktion geht nicht!} \end{aligned}$$

Wir müssen irgendwie das dn unterbringen.

2. Versuch: $T(n) \leq 2dn^2$ gibt im Induktionsschluss:

$$T(n) \leq dn + 2dn^2 > 2dn^2$$

3. Versuch: $T(n) \leq 2dn^2 - dn$ „Überraschenderweise“ ist das ganau das, was wir uns oben für die Laufzeit überlegt haben.

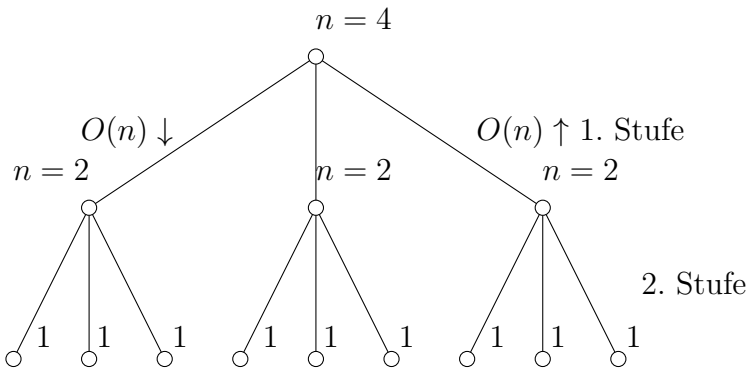
Induktionsanfang: $T(1) \leq 2d - d = d$ ✓

Induktionsschluss:

$$\begin{aligned} T(n) &= dn + 4 \cdot T\left(\frac{n}{2}\right) \\ &\leq dn + 4\left(2d\left(\frac{n}{2}\right)^2 - d\frac{n}{2}\right) \\ &= dn + 2dn^2 - 2dn \\ &= 2dn^2 - dn \qquad \square \end{aligned}$$

Vergleiche auch Seite 161, wo $\log_2 \frac{n}{2} = \log_2 n - 1$ wichtig ist.

Bisher haben wir mit dem *divide-and-conquer*-Ansatz also noch nichts gegenüber der einfachen Methode gewonnen. Unser nächstes Ziel ist die Verbesserung der Multiplikation durch nur noch *drei rekursive Aufrufe* mit jeweils $\frac{n}{2}$ vielen Bits. Analysieren wir zunächst die Laufzeit.



Wir haben wieder $\log_2 n$ viele divide-and-conquer-Schritte und die Zeit an den Blättern.

1. Stufe:	$d \cdot n$
2. Stufe:	$3 \cdot d \cdot \frac{n}{2}$
3. Stufe:	$3 \cdot 3 \cdot d \cdot \frac{n}{4}$
⋮	
$\log_2 n$-te Stufe:	$3^{\log_2 n - 1} \cdot d \cdot 2$
#Blätter:	$3^{\log_2 n} \cdot d \cdot 1$

Dann ist die Laufzeit

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\log_2 n} 3^i \cdot d \cdot \frac{n}{2^i} \\
&= d \cdot n \cdot \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i \\
&= d \cdot n \cdot \frac{(\frac{3}{2})^{\log_2 n+1} - 1}{\frac{3}{2} - 1} \\
&\leq 2 \cdot d \cdot n \cdot \frac{3}{2} \cdot \left(\frac{3}{2}\right)^{\log_2 n} \\
&= 3 \cdot d \cdot n \cdot 2^{\log_2(3/2)\cdot\log_2 n} \\
&= 3 \cdot d \cdot n \cdot n^{\overbrace{\log_2(3/2)}^{<1}} \\
&= 3 \cdot d \cdot n \cdot n^{\log_2 3 - \overbrace{\log_2 2}^{=1}} \\
&= 3 \cdot d \cdot n^{log_2 3} = O(n^{1.59}) \qquad \square
\end{aligned}$$

also tatsächlich besser als $O(n^2)$. Fassen wir zusammen:

- Zwei Aufrufe mit $\frac{n}{2}$, linearer Zusatzaufwand:

  (Auf jeder Stufe $d \cdot n +$ 2 Aufrufe mit halber Größe.)

$$\begin{aligned}
T(n) &= d \cdot n \cdot \sum_{i=0}^{\log_2 n} \underbrace{\frac{2^i}{2^i}}_{=1} \\
&= O(n \cdot \log n)
\end{aligned}$$

- Drei Aufrufe:

$$\begin{aligned}
T(n) &= d \cdot n \cdot \sum \left(\frac{3}{2}\right)^i \qquad \text{(3 Aufrufe, halbe Größe)} \\
&= O(n^{\log_2 3})
\end{aligned}$$

- Vier Aufrufe:

$$\begin{aligned}
T(n) &= d \cdot n \cdot \sum \left(\frac{4}{2}\right)^i \qquad \text{(4 Aufrufe, halbe Größe)} \\
&= d \cdot n \cdot n \\
&= O(n^2)
\end{aligned}$$

Aufgabe: Stellen Sie für den Fall der drei Aufrufe die Rekursionsgleichungen auf und beweisen Sie $T(n) = O(n^{log_2 3})$ durch eine ordnungsgemäße Induktion.

Zurück zur Multiplikation.

$$\begin{array}{rcl} \overbrace{a_1 \dots a_n}^{a:=} & = & \overbrace{a_1 \dots a_{\frac{n}{2}}}^{a':=} \overbrace{a_{\frac{n}{2}+1} \dots a_n}^{a'':=} \\ \underbrace{b_1 \dots b_n}_{b:=} & = & \underbrace{b_1 \dots b_{\frac{n}{2}}}_{b':=} \underbrace{b_{\frac{n}{2}+1} \dots b_n}_{b'':=} \end{array}$$

Wie können wir einen Aufruf einsparen? Erlauben uns zusätzliche Additionen. Wir beobachten zunächst allgemein:

$$(x-y)\cdot(u-v) = x(u-v) + y\cdot(u-v) = x\cdot u - x\cdot v + y\cdot u - y\cdot v$$

Wir haben *eine explizite* und *vier implizite Multiplikationen*, die in der Summe verborgen sind, durchgeführt.

Wir versuchen jetzt $a'b'' + a''b'$ auf einen Schlag zu ermitteln:

$$(a'-a'')\cdot(b''-b') = a'b'' - a'b' - a''b'' + a''b'$$

Die Produkte $a'b'$ und $a''b''$ berechnen wir normal. Damit bekommen wir

$$a'b'' + a''b' = (a'-a'')\cdot(b''-b') + a'b' + a''b''.$$

Die Terme $(a'-a'')$ und $(b''-b')$ können < 0 sein. Diese Fälle müssen wir in unsere Rechnung mit einbeziehen.

Algorithmus 24: Multiplikation großer Zahlen (schnell)

Input : a, b zwei Zahlen mit n Bits
Output : Produkt $a \cdot b$

1 **if** $n == 1$ **then return** $a \cdot b$;

`/* Divide */`
2 $a' = a_1 \dots a_{\frac{n}{2}}$; $a'' = a_{\frac{n}{2}+1} \dots a_n$; $b' = b_1 \dots b_{\frac{n}{2}}$; $b'' = b_{\frac{n}{2}+1} \dots b_n$;
3 $m_1 = \text{Mult}(a', b')$; $m_2 = \text{Mult}(a'', b'')$;

`/* Beachte:` $|a'' - a'|$ `und` $|b' - b''|$ `ist immer` $\leq 2^{n/2} - 1$`, es reichen also immer` $\frac{n}{2}$ `viele Bits. */`
4 **if** $(a' - a'') > 0$ *und* $(b'' - b') > 0$ **then** $m_3 = \text{Mult}(a' - a'', b'' - b')$;
5 **if** $(a' - a'') < 0$ *und* $(b'' - b') < 0$ **then** $m_3 = \text{Mult}(a'' - a', b' - b'')$;
6 **if** $(a' - a'') < 0$ *und* $(b'' - b') > 0$ **then** $m_3 = -\text{Mult}(a'' - a', b'' - b')$;
7 **if** $(a' - a'') > 0$ *und* $(b'' - b') < 0$ **then** $m_3 = -\text{Mult}(a' - a'', b' - b'')$;
8 **if** $(a' - a'') == 0$ *oder* $(b'' - b') == 0$ **then** $m_3 = 0$;

`/* Conquer */`
9 **return** $m_1 \cdot 2^n + \underbrace{(m_3 + m_1 + m_2)}_{\text{Zeit } O(n/2)} \cdot 2^{n/2} + m_2$;

Damit bekommen wir für die Laufzeit:

$$\begin{aligned} T(1) &\leq d \\ T(n) &\leq d \cdot n + 3 \cdot T\left(\frac{n}{2}\right) \quad \Rightarrow O(n^{log_2 3}). \end{aligned}$$

Die Addition zweier Zahlen geht linear in der Anzahl der Bits. Für die Multiplikation ist kein Linearzeitverfahren bekannt.

11.5 Schnelle Matrixmultiplikation

Wir gehen jetzt wieder davon aus, dass die Basisoperationen in der Zeit $O(1)$ durchführbar sind.

Erinnern wir uns an die Matrizenmultiplikation. Dabei betrachten wir zunächst nur *quadratische Matrizen*.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Im allgemeinen haben wir bei zwei $n \times n$-Matrizen:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nn} \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{n} a_{1k}b_{k1} & \dots & \sum_{k=1}^{n} a_{1k}b_{kn} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{nk}b_{k1} & \dots & \sum_{k=1}^{n} a_{nk}b_{kn} \end{pmatrix}$$

Der Eintrag c_{ij} der Ergebnismatrix berechnet sich als $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

Laufzeit:

- Pro Eintrag des Ergebnisses n Multiplikationen und $n-1$ Additionen, also $O(n)$.
- Bei n^2 Einträgen $O(n^3)$ $(= O(n^2)^{3/2})$.

Überraschend war seinerzeit, dass es mit divide-and-conquer besser geht. Wie könnte ein divide-and-conquer-Ansatz funktionieren? Teilen wir die Matrizen einmal auf:

$$\left(\begin{array}{ccc|ccc} a_{1,1} & \dots & a_{1,\frac{n}{2}} & a_{1,\frac{n}{2}+1} & \dots & a_{1,n} \\ \vdots & & \ddots & & & \vdots \\ a_{\frac{n}{2},1} & \dots & a_{\frac{n}{2},\frac{n}{2}} & a_{\frac{n}{2},\frac{n}{2}+1} & \dots & a_{\frac{n}{2},n} \\ \hline a_{\frac{n}{2}+1,1} & \dots & a_{\frac{n}{2}+1,\frac{n}{2}} & a_{\frac{n}{2}+1,\frac{n}{2}+1} & \dots & a_{\frac{n}{2}+1,n} \\ \vdots & & & \ddots & & \vdots \\ a_{n,1} & \dots & a_{n,\frac{n}{2}} & a_{n,\frac{n}{2}+1} & \dots & a_{n,n} \end{array}\right) = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

Die $A_{i,j}$ sind $\frac{n}{2} \times \frac{n}{2}$-Matrizen. Wir nehmen wieder an, dass n eine Zweierpotenz ist.

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad \text{ist dann analog aufgeteilt.}$$

Das Ergebnis schreiben wir dann so:

$$A \cdot B = \begin{pmatrix} \sum_{k=1}^{n} a_{1k} b_{k1} & \dots & \sum_{k=1}^{n} a_{1k} b_{kn} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{nk} b_{k1} & \dots & \sum_{k=1}^{n} a_{nk} b_{kn} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Betrachten wir einmal die Submatrix C_{11} des Ergebnisses.

$$\begin{aligned} C_{11} &= \begin{pmatrix} \sum_{k=1}^{n} a_{1,k} b_{k,1} & \dots & \sum_{k=1}^{n} a_{1,k} b_{k,\frac{n}{2}} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{\frac{n}{2},k} b_{k,1} & \dots & \sum_{k=1}^{n} a_{\frac{n}{2},k} b_{k,\frac{n}{2}} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{k=1}^{\frac{n}{2}} a_{1,k} b_{k,1} + \sum_{k=\frac{n}{2}+1}^{n} a_{1,k} b_{k,1} & \dots & \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{\frac{n}{2}} a_{\frac{n}{2},k} b_{k,1} + \sum_{k=\frac{n}{2}+1}^{n} a_{\frac{n}{2},k} b_{k,1} & \dots & \end{pmatrix} \end{aligned}$$

Bei genauem hinsehen, ergibt sich $C_{11} = A_{11}B_{11} + A_{12}B_{21}$, denn

$$A_{11}B_{11} = \begin{pmatrix} \sum_{k=1}^{\frac{n}{2}} a_{1,k}b_{k,1} & \cdots & \sum_{k=1}^{\frac{n}{2}} a_{1,k}b_{k,\frac{n}{2}} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{\frac{n}{2}} a_{\frac{n}{2},k}b_{k,1} & \cdots & \sum_{k=1}^{\frac{n}{2}} a_{\frac{n}{2},k}b_{k,\frac{n}{2}} \end{pmatrix} \quad \text{und}$$

$$A_{12}B_{21} = \begin{pmatrix} \sum_{k=\frac{n}{2}+1}^{n} a_{1,k}b_{k,1} & \cdots & \sum_{k=\frac{n}{2}+1}^{n} a_{1,k}b_{k,\frac{n}{2}} \\ \vdots & \ddots & \vdots \\ \sum_{k=\frac{n}{2}+1}^{n} a_{\frac{n}{2},k}b_{k,1} & \cdots & \sum_{k=\frac{n}{2}+1}^{n} a_{\frac{n}{2},k}b_{k,\frac{n}{2}} \end{pmatrix}$$

Für die restlichen Submatrizen ergibt sich ganz analog

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22}, \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21}, \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

Das sieht genauso aus wie wenn die A_{ij}, B_{ij} und C_{ij} ganz normale Skalare wären.

Für die Anzahl der Operationen ergibt das die folgende Rekurrenz für ein geeignetes d.

$$\begin{aligned} T(1) &\leq d \\ T(n) &\leq \underbrace{d \cdot n^2}_{\substack{\text{linear in der} \\ \#\text{Einträge}}} + 8 \cdot \underbrace{T\left(\frac{n}{2}\right)}_{\substack{(n/2)^2 \,=\, n^2/4 \\ \text{Einträge}}} \end{aligned}$$

Das führt (analog zur Rechnung auf Seite 166) zu

$$T(n) \leq d \cdot n^2 \cdot \sum_{i=0}^{\log_2 n} \left(\frac{8}{4}\right)^i = O(n^3).$$

Angenommen, wir könnten, ähnlich wie bei der Multiplikation großer Zahlen, eine der acht Multiplikationen der $\frac{n}{2} \times \frac{n}{2}$-Matrizen einsparen. Wir bekämen dann nur noch sieben rekursive Aufrufe. Das führt auf die Rekurrenz:

$$\begin{aligned} T(1) &\leq d \\ T(n) &\leq d \cdot n^2 + 7 \cdot T\left(\frac{n}{2}\right) \end{aligned}$$

Das führt auf

$$\begin{aligned} T(n) &\leq d \cdot n^2 \cdot \sum_{i=0}^{\log_2 n} \left(\frac{7}{2^2}\right)^i \\ &= d \cdot n^2 \cdot \frac{(7/4)^{\log_2 n+1} - 1}{1 - 7/4} \\ &= \frac{7}{3} d \cdot n^{\log_2 n} - \frac{4}{3} d \cdot n^2 \\ &= O(n^{\log_2 7}), \end{aligned}$$

da $\log_2 7 < 2.81 < 3$. Die entsprechenden Rechnungen sind eine gute Übung zur Induktion.

Wir brauchen immernoch

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22}, \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21}, \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

Wir suchen jetzt sieben Produkte von $\frac{n}{2} \times \frac{n}{2}$-Matrizen, $P_1, \ldots, P_7$, so dass die C_{ij} ohne weitere Multiplikationen erzeugt werden können. Wir betrachten Produkte der Art

$$(A_{11} - A_{21})(B_{11} + B_{21})$$

und ähnlich. Dazu führen wir zur übersichtlichen Darstellung noch die folgende Notation ein.

Wir schreiben C_{11} noch einmal etwas anders hin. Dabei betrachten wir die A_{ij}, B_{ij} jetzt einfach nur als Symbole.

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ &= (A_{11}, A_{12}, A_{21}, A_{22}) \cdot \begin{pmatrix} +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{pmatrix} \end{aligned}$$

Dafür schreiben wir jetzt:

$$\begin{array}{c} \\ A_{11} \\ A_{12} \\ A_{21} \\ A_{22} \end{array} \begin{array}{c} \begin{array}{cccc} B_{11} & B_{21} & B_{12} & B_{22} \end{array} \\ \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{array} = C_{11}$$

Wenn wir Minuszeichen eintragen, erhalten wir zum Beispiel folgendes:

$$\begin{pmatrix} - & \cdot & \cdot & \cdot \\ - & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = (-A_{11} - A_{12}) \cdot B_{11}$$

Beachte: Das Matrixprodukt ist assoziativ, da $(AB)C = A(BC)$ für alle A, B, C $n \times n$-Matrizen. Aber es ist *nicht kommutativ*, da im allgemeinen $AB \neq BA$ ist. Etwa bei

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix}.$$

In unserer Schreibweise suchen wir jetzt:

$$C_{11} = \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = \begin{pmatrix} \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \end{pmatrix} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix} = A_{21}B_{12} + A_{22}B_{22}$$

Wir versuchen, die Produkte $P_1, \ldots, P_7$ mit jeweils einer Multiplikation zu ermitteln. Wir beginnen einmal mit $C_{12} = P_1 + P_2$.

P_1 könnte $\begin{pmatrix} \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$ und P_2 könnte $\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$ sein.

Dann ist $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_2$. Damit gewinnen wir allerdings nichts. Eine andere Möglichkeit wäre

$$P_1 = \begin{pmatrix} \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = (A_{11} + A_{12})B_{22}, \quad P_2 = \begin{pmatrix} \cdot & \cdot & + & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = A_{11}(B_{12} - B_{22}).$$

Gibt es noch andere Matrizen für ein einzelnes Produkt?

$$\begin{pmatrix} \cdot & \cdot & + & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = A_{11}(B_{12} + B_{22}),$$

$$\begin{pmatrix} \cdot & + & \cdot & \cdot \\ \cdot & - & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & - & \cdot & \cdot \end{pmatrix} = (A_{11} - A_{12} + A_{21} - A_{22})B_{21}$$

Beachte: Die A_{ij} stehen immer links!

Wir nehmen jetzt die folgenden offiziellen P_1, P_2 für C_{12}.

$$C_{12} = \begin{pmatrix} \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = \underbrace{\begin{pmatrix} \cdot & \cdot & + & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}}_{:=P_1} + \underbrace{\begin{pmatrix} \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}}_{:=P_2}$$

$$\begin{aligned} P_1 &= A_{11}(B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12})B_{22} \end{aligned}$$

Ebenso P_3, P_4 für C_{21}.

$$C_{21} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \end{pmatrix} = \underbrace{\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \end{pmatrix}}_{:=P_3} + \underbrace{\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & + & \cdot & \cdot \end{pmatrix}}_{:=P_4}$$

$$\begin{aligned} P_3 &= (A_{21} + A_{22})B_{11} \\ P_4 &= A_{22}(-B_{11} + B_{21}) \end{aligned}$$

Bis jetzt haben wir vier Multiplikationen verbraucht und C_{12} und C_{21} erzielt. Das ist noch nichts neues! Wir haben noch drei Multiplikationen für

$$C_{11} = \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}, \quad C_{22} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix}$$

übrig. Beachte $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$, die Hauptdiagonale fehlt noch.

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

Versuchen wir einmal zwei Produkte auf der rechten Seite gleichzeitig zu berechnen. Wir nehmen P_5 wie unten, das ist auch ein einzelnes Produkt von Matrizen.

$$P_5 = \begin{pmatrix} + & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & + \end{pmatrix} = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_5 = \underbrace{A_{11}B_{11}}_{\text{in } C_{11}} + \overbrace{\underbrace{A_{11}B_{22}}_{\text{in } P_1,P_2} + \underbrace{A_{22}B_{11}}_{\text{in } P_3,P_4}}^{\text{Das ist zuviel.}} + \underbrace{A_{22}B_{22}}_{\text{in } C_{22}}$$

Abhilfe schafft hier zunächst einmal

$$\begin{aligned} P_5 + P_4 - P_2 &= \begin{pmatrix} + & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & + \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & + & \cdot & \cdot \end{pmatrix} - \begin{pmatrix} \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \\ &= \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & + \end{pmatrix} = A_{11}B_{11} + A_{22}B_{21} + A_{22}B_{22} - A_{12}B_{22}. \end{aligned}$$

Mit der Addition von P_6 bekommen wir dann endlich C_{11}.

$$\begin{aligned} P_6 &= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & - & \cdot & - \end{pmatrix} = (A_{12} - A_{22})(B_{21} + B_{22}) \\ &= A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22} \\ C_{11} &= (P_5 + P_4 - P_2) + P_6 = A_{11}B_{11} + A_{12}B_{21} \end{aligned}$$

Für C_{22} benutzen wir auch wieder P_5 und berechnen

$$\begin{aligned} P_5 + P_1 - P_3 &= \begin{pmatrix} + & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & + \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & + & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} - \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \end{pmatrix} \\ &= \begin{pmatrix} + & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix} = A_{11}B_{11} + A_{11}B_{12} - A_{21}B_{11} + A_{22}B_{22}. \end{aligned}$$

Mit der letzten Multiplikation berechnen wir P_7 und C_{22}.

$$\begin{aligned} P_7 &= \begin{pmatrix} + & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & - & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = (A_{11} - A_{21})(B_{11} + B_{12}) \\ &= A_{11}B_{11} + A_{11}B_{12} - A_{21}B_{11} - A_{21}B_{12} \\ C_{22} &= (P_5 + P_1 - P_3) - P_7 = A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Mit diesen $P_1, \ldots, P_7$ kommen wir dann insgesammt auf $O(n^{\log_2 7})$.