Rekurrenzen

Wie finden wir einen geschlossenen Ausdruck für die Laufzeit?

Dafür müssen wir die Rekursionsgleichung lösen.

Rekurrenzen

Wie finden wir einen geschlossenen Ausdruck für die Laufzeit?

Dafür müssen wir die Rekursionsgleichung lösen.

Methoden

1. **Raten+Induktion**
   Man rät die richtige Lösung und beweist die Korrektheit mittels vollständiger Induktion. Man benötigt Erfahrung um richtig zu raten...
2. **Mastertheorem**
   Für die meisten Rekurrenzen gibt es ein allgemeines Theorem, dass die asymptotisch korrekte Laufzeit für die jeweilige Rekurrenz angibt.

Raten+Induktion

Zuerst müssen wir die $\mathcal{O}$-Notation entfernen:

$$T(n) \leq \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + c_1 n & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Raten+Induktion

Zuerst müssen wir die $\mathcal{O}$-Notation entfernen:

$$T(n) \leq \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + c_1 n & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

für $n = 2^k$:

Für diesen Fall können wir stattdessen die folgende Rekursionsgleichung betrachten:

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 n & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Raten+Induktion

Zuerst müssen wir die $\mathcal{O}$-Notation entfernen:

$$T(n) \leq \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + c_1 n & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

für $n = 2^k$:

Für diesen Fall können wir stattdessen die folgende Rekursionsgleichung betrachten:

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 n & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Man rät die richtige Lösung und beweist, dass durch Einsetzen, dass diese Lösung korrekt ist.

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$:

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

  Angenommen Aussage wahr für $n' \in \{1, \ldots, n-1\}$, und $n > 1$. Wir beweisen sie für n:

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

  Angenommen Aussage wahr für $n' \in \{1, \ldots, n-1\}$, und $n > 1$. Wir beweisen sie für n:

$$T(n) \leq T\left(\frac{n}{2}\right) + c_1$$

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

  Angenommen Aussage wahr für $n' \in \{1, \ldots, n-1\}$, und $n > 1$. Wir beweisen sie für n:

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2}\right) + c_1 \\ &\leq \left(a \log \frac{n}{2} + b\right) + c_1 \end{aligned}$$

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

  Angenommen Aussage wahr für $n' \in \{1, \ldots, n-1\}$, und $n > 1$. Wir beweisen sie für n:

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2}\right) + c_1 \\ &\leq \left(a \log \frac{n}{2} + b\right) + c_1 \\ &= a(\log n - 1) + b + c_1 \end{aligned}$$

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

  Angenommen Aussage wahr für $n' \in \{1, \ldots, n-1\}$, und $n > 1$. Wir beweisen sie für n:

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2}\right) + c_1 \\ &\leq \left(a \log \frac{n}{2} + b\right) + c_1 \\ &= a(\log n - 1) + b + c_1 \\ &= a \log n + (c_1 - a) + b \end{aligned}$$

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

  Angenommen Aussage wahr für $n' \in \{1, \ldots, n-1\}$, und $n > 1$. Wir beweisen sie für n:

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2}\right) + c_1 \\ &\leq \left(a \log \frac{n}{2} + b\right) + c_1 \\ &= a(\log n - 1) + b + c_1 \\ &= a \log n + (c_1 - a) + b \\ &\leq a \log n + b \end{aligned}$$

Raten+Induktion

$$T(n) \leq \begin{cases} T(\frac{n}{2}) + c_1 & n > 1 \\ c_2 & \text{sonst} \end{cases}$$

Ansatz: $T(n) \leq a \log n + b$.

Beweis. (durch Induktion)

- **Anfang** $(n = 1)$: wahr falls $b \geq c_2$.
- **Induktionsschritt** $1, \ldots, n-1 \to n$:

  Angenommen Aussage wahr für $n' \in \{1, \ldots, n-1\}$, und $n > 1$. Wir beweisen sie für n:

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2}\right) + c_1 \\ &\leq \left(a \log \frac{n}{2} + b\right) + c_1 \\ &= a(\log n - 1) + b + c_1 \\ &= a \log n + (c_1 - a) + b \\ &\leq a \log n + b \end{aligned}$$

  Gilt falls $a \geq c_1$.

Mastertheorem

Lemma 1

Seien $a \geq 1, b \geq 1$ und $\epsilon > 0$ Konstanten. Betrachte die Rekurrenz

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \ .$$

1. Fall:

Falls $f(n) = \mathcal{O}(n^{\log_b(a)-\epsilon})$ gilt $T(n) = \Theta(n^{\log_b a})$.

2. Fall:

Falls $f(n) = \Theta(n^{\log_b(a)} \log^k n)$ gilt $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$, $k \geq 0$.

Mastertheorem

Wir beweisen das Mastertheorem für den Fall $n = b^\ell$, und nehmen an, dass der nichtrekursive Fall für Problemgröße 1 Kosten 1 verursacht.

Der Rekursionsbaum

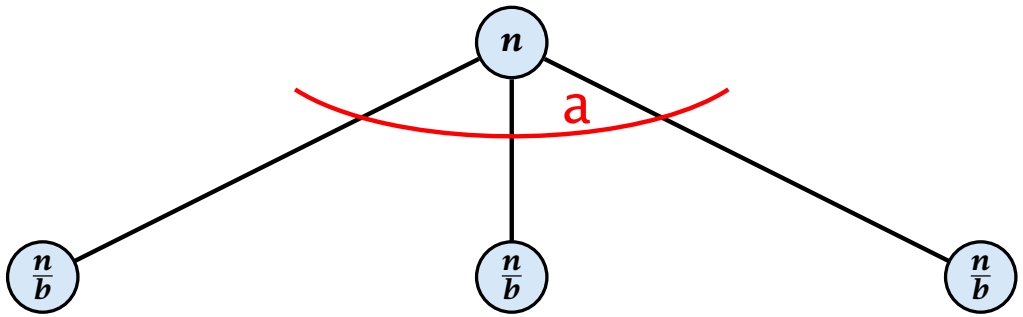
Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:

Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:

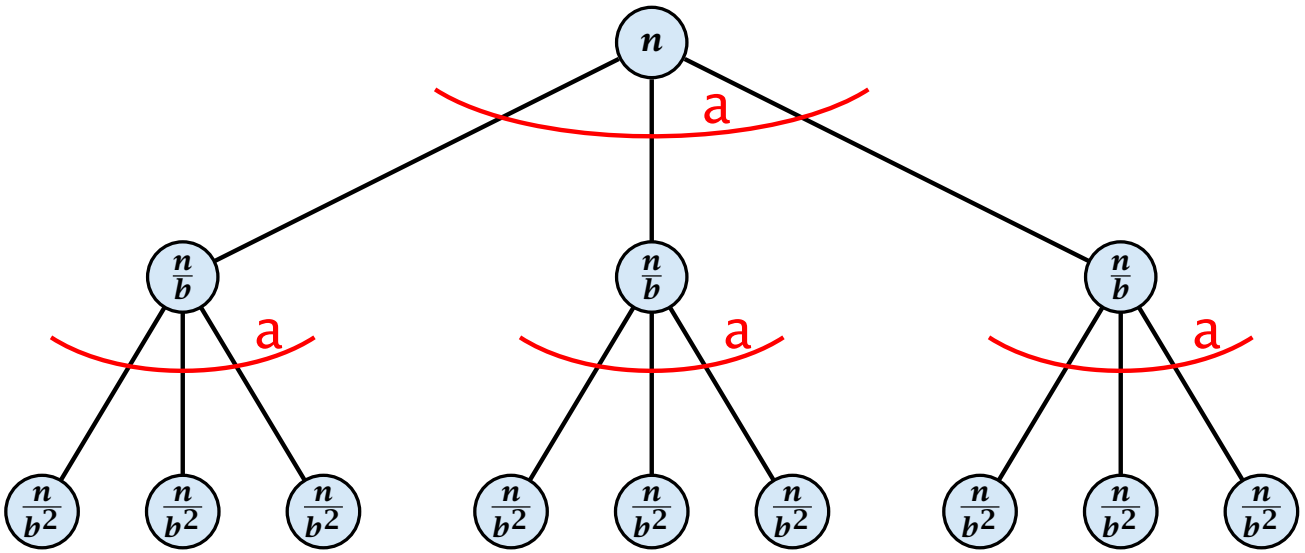
Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



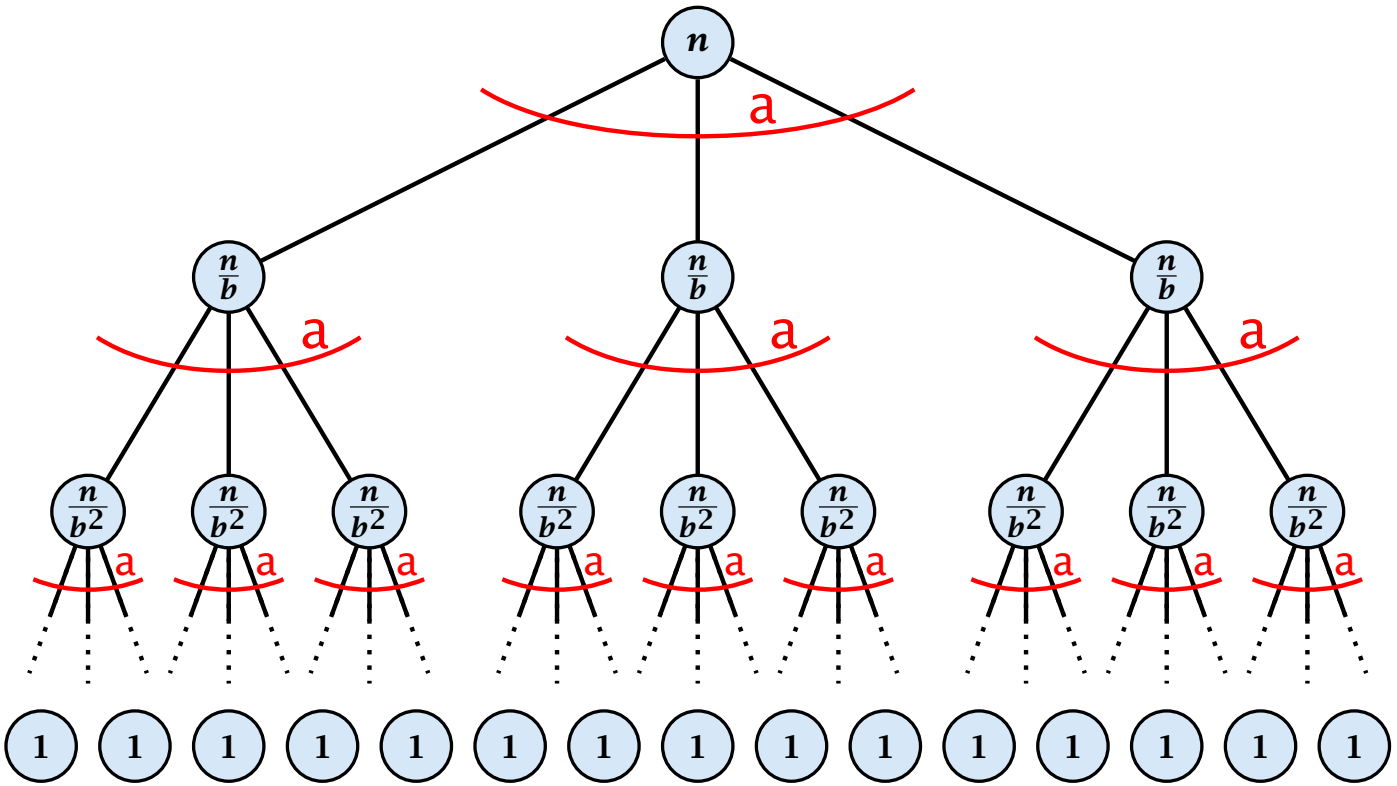
Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



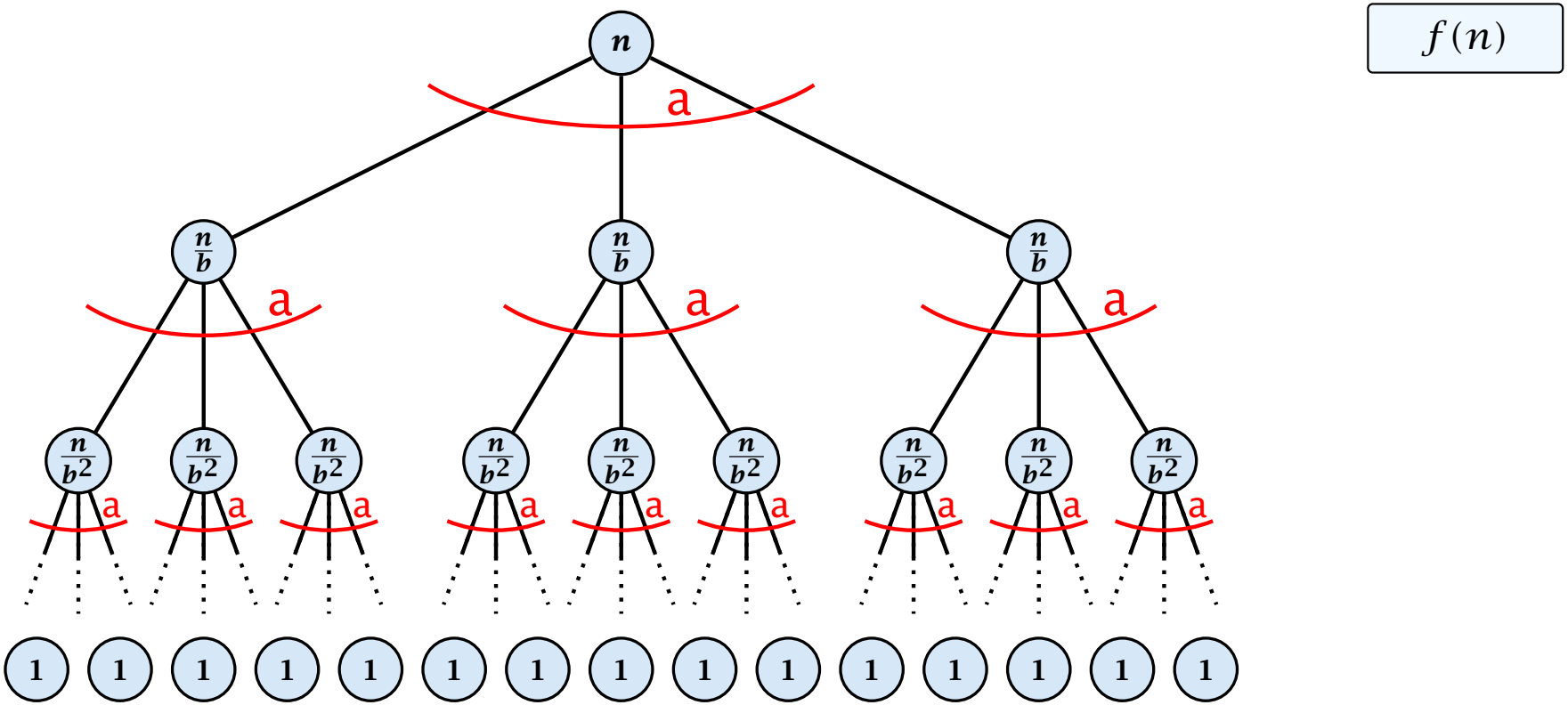
Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



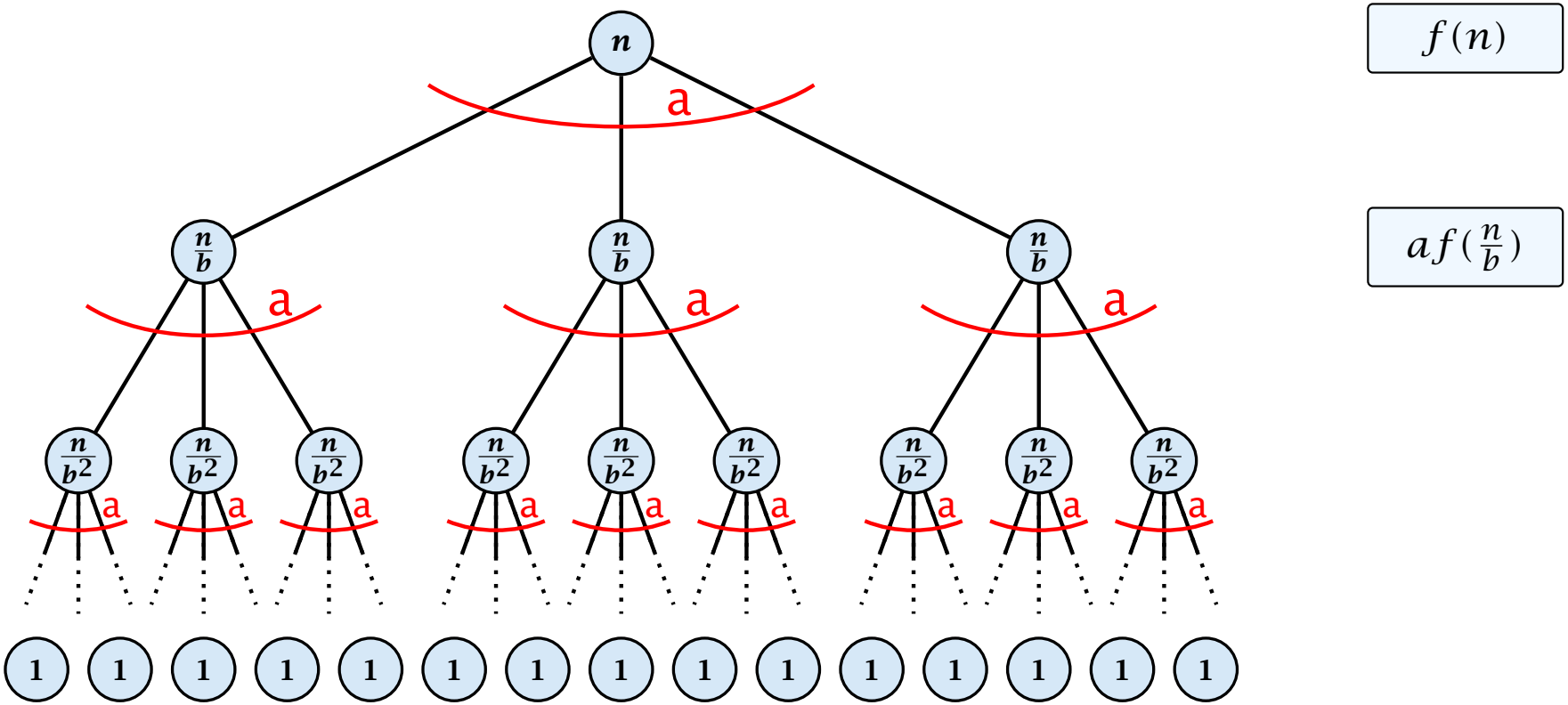
Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



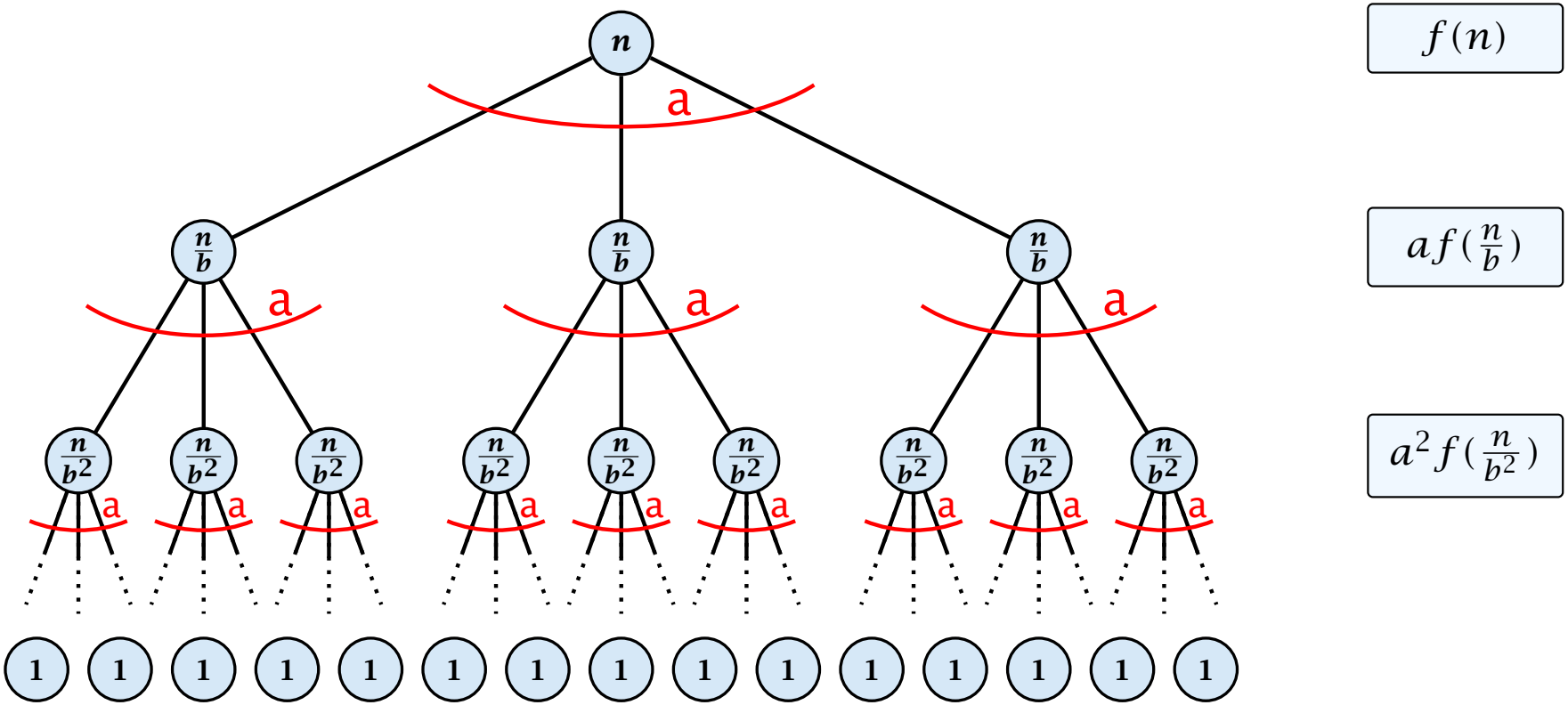
Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



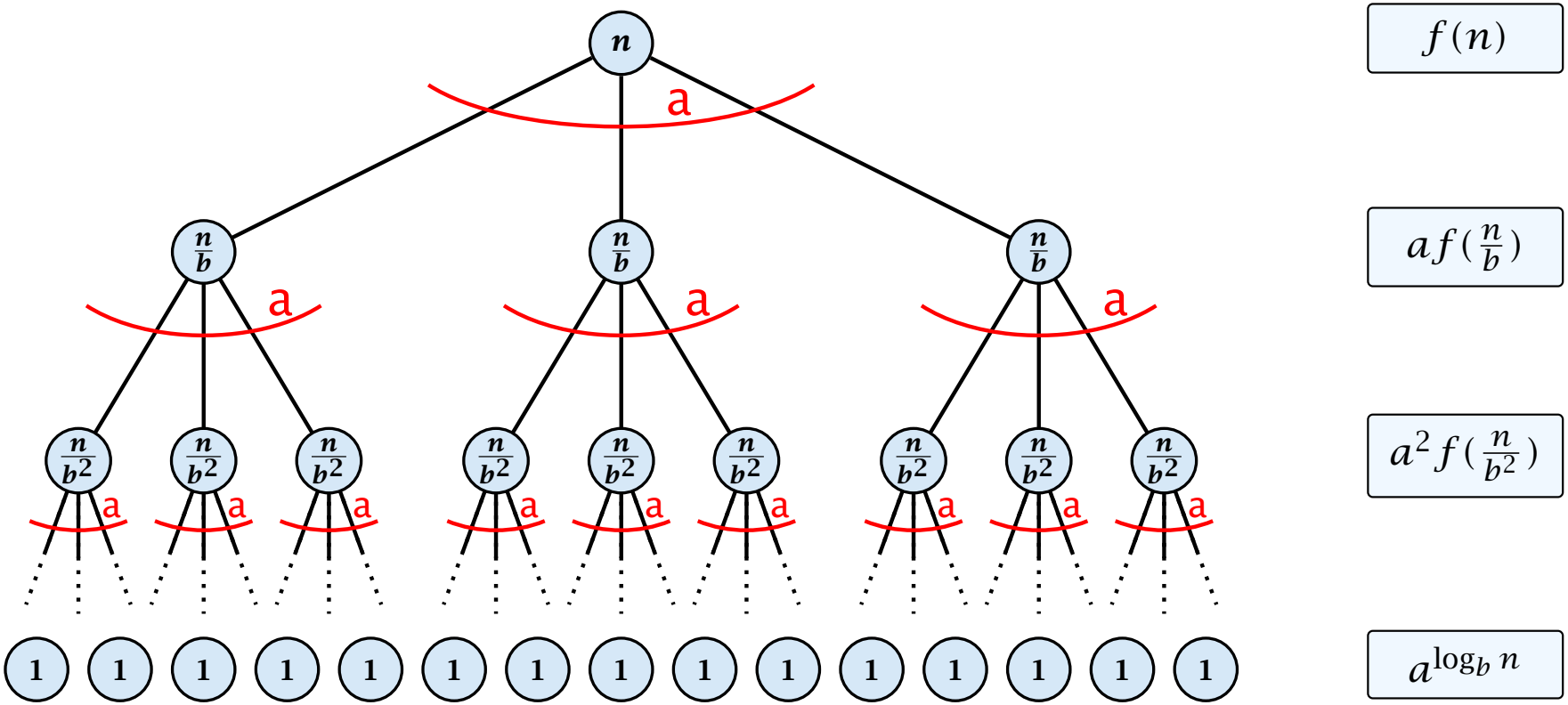
Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



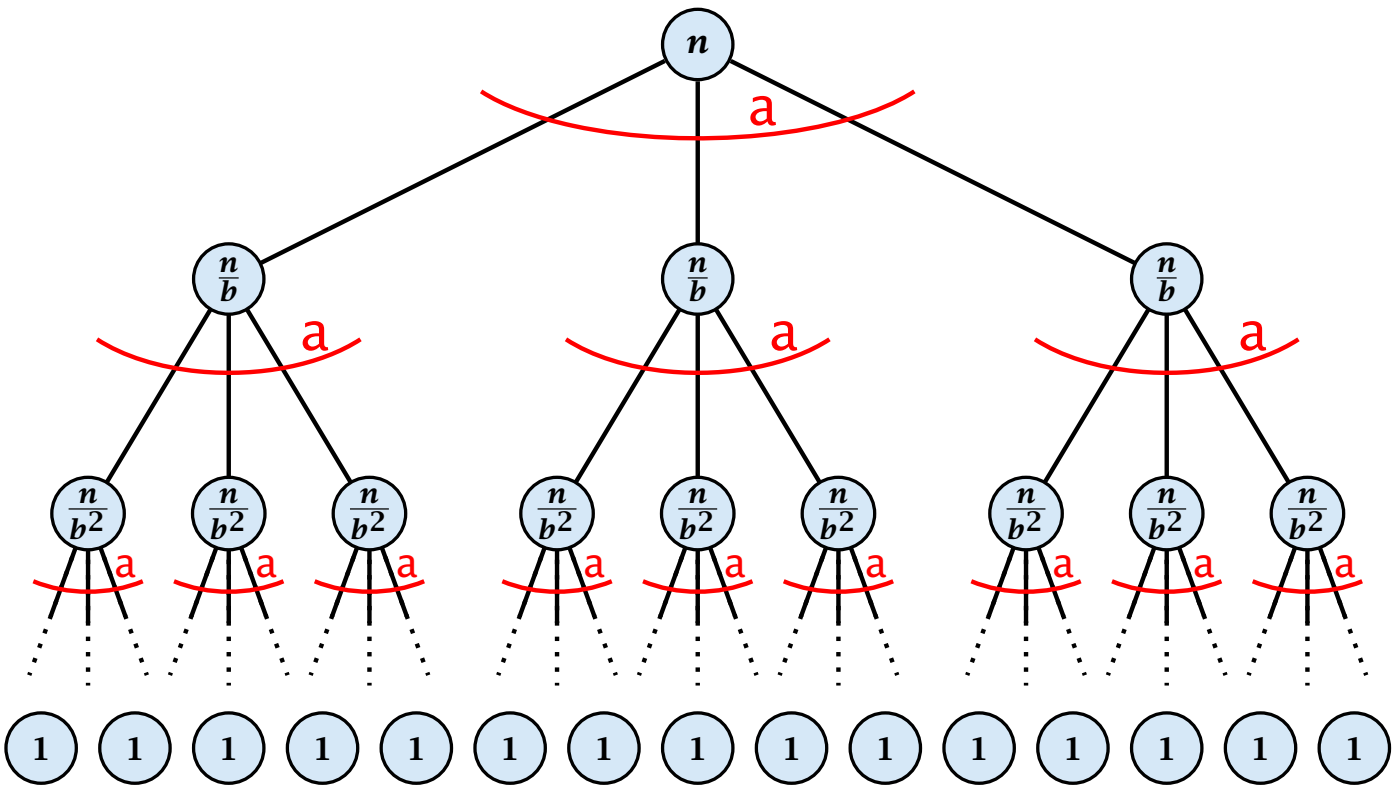
Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



Der Rekursionsbaum

Die Laufzeit eines rekursiven Algorithmus kann durch einen Rekursionsbaum veranschaulicht werden:



$f(n)$

$af(\frac{n}{b})$

$a^2 f(\frac{n}{b^2})$

$a^{\log_b n}$

$=$

$n^{\log_b a}$

Mastertheorem

Das heißt unsere Kosten sind

$$T(n) = n^{\log_b a} + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) .$$

Fall 1. Sei $f(n) \leq cn^{\log_b a - \epsilon}$.

Fall 1. Sei $f(n) \leq cn^{\log_b a - \epsilon}$.

$$T(n) - n^{\log_b a}$$

Fall 1. Sei $f(n) \leq cn^{\log_b a - \epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$$= cn^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n-1} \left(b^\epsilon\right)^i$$

$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$$= cn^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n-1} \left(b^\epsilon\right)^i$$

$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$

$\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$$

$$= cn^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n-1} \left(b^{\epsilon}\right)^i$$

$$\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$$

$$= cn^{\log_b a-\epsilon}(b^{\epsilon \log_b n} - 1)/(b^{\epsilon} - 1)$$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$

$$= cn^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n-1} \left(b^{\epsilon}\right)^i$$

$\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$

$$= cn^{\log_b a-\epsilon}(b^{\epsilon \log_b n} - 1)/(b^{\epsilon} - 1)$$

$$= cn^{\log_b a-\epsilon}(n^{\epsilon} - 1)/(b^{\epsilon} - 1)$$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$

$$= cn^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n-1} (b^\epsilon)^i$$

$\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$

$$= cn^{\log_b a-\epsilon}(b^{\epsilon \log_b n} - 1)/(b^\epsilon - 1)$$

$$= cn^{\log_b a-\epsilon}(n^\epsilon - 1)/(b^\epsilon - 1)$$

$$= \frac{c}{b^\epsilon - 1} n^{\log_b a}(n^\epsilon - 1)/(n^\epsilon)$$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$

$$= cn^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n-1} (b^\epsilon)^i$$

$\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$

$$= cn^{\log_b a-\epsilon}(b^{\epsilon \log_b n} - 1)/(b^\epsilon - 1)$$

$$= cn^{\log_b a-\epsilon}(n^\epsilon - 1)/(b^\epsilon - 1)$$

$$= \frac{c}{b^\epsilon - 1} n^{\log_b a}(n^\epsilon - 1)/(n^\epsilon)$$

Also,

$$T(n) \leq \left(\frac{c}{b^\epsilon - 1} + 1\right) n^{\log_b(a)}$$

Fall 1. Sei $f(n) \leq cn^{\log_b a-\epsilon}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a-\epsilon}$$

$b^{-i(\log_b a-\epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i}a^{-i}$

$$= cn^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n-1} (b^\epsilon)^i$$

$\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$

$$= cn^{\log_b a-\epsilon}(b^{\epsilon \log_b n} - 1)/(b^\epsilon - 1)$$

$$= cn^{\log_b a-\epsilon}(n^\epsilon - 1)/(b^\epsilon - 1)$$

$$= \frac{c}{b^\epsilon - 1} n^{\log_b a}(n^\epsilon - 1)/(n^\epsilon)$$

Also,

$$T(n) \leq \left(\frac{c}{b^\epsilon - 1} + 1\right) n^{\log_b(a)}$$

$\Rightarrow T(n) = \mathcal{O}(n^{\log_b a})$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

$$T(n) - n^{\log_b a}$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a}$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

$$
\begin{aligned}
T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
&\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
&= cn^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1
\end{aligned}
$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

$$
\begin{aligned}
T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
&\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
&= cn^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\
&= cn^{\log_b a} \log_b n
\end{aligned}
$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

$$
\begin{aligned}
T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
&\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
&= cn^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\
&= cn^{\log_b a} \log_b n
\end{aligned}
$$

Also,

$$T(n) = \mathcal{O}(n^{\log_b a} \log_b n)$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}$.

$$
\begin{aligned}
T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
&\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
&= cn^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\
&= cn^{\log_b a} \log_b n
\end{aligned}
$$

Also,

$$T(n) = \mathcal{O}(n^{\log_b a} \log_b n) \qquad \boxed{\Rightarrow T(n) = \mathcal{O}(n^{\log_b a} \log n).}$$

Fall 2. Sei $f(n) \geq cn^{\log_b a}$.

Fall 2. Sei $f(n) \geq c n^{\log_b a}$.

$$T(n) - n^{\log_b a}$$

Fall 2. Sei $f(n) \geq c n^{\log_b a}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

Fall 2. Sei $f(n) \geq c n^{\log_b a}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\geq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a}$$

Fall 2. Sei $f(n) \geq c n^{\log_b a}$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\geq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a}$$

$$= c n^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1$$

Fall 2. Sei $f(n) \geq c n^{\log_b a}$.

$$
\begin{aligned}
T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
&\geq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
&= c n^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\
&= c n^{\log_b a} \log_b n
\end{aligned}
$$

Fall 2. Sei $f(n) \geq c n^{\log_b a}$.

$$
\begin{aligned}
T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
&\geq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
&= c n^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\
&= c n^{\log_b a} \log_b n
\end{aligned}
$$

Also,

$$T(n) = \Omega(n^{\log_b a} \log_b n)$$

Fall 2. Sei $f(n) \geq cn^{\log_b a}$.

$$
\begin{aligned}
T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
&\geq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
&= cn^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\
&= cn^{\log_b a} \log_b n
\end{aligned}
$$

Also,

$$T(n) = \Omega(n^{\log_b a} \log_b n) \qquad \boxed{\Rightarrow T(n) = \Omega(n^{\log_b a} \log n).}$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq c n^{\log_b a} (\log_b(n))^k$.

$$T(n) - n^{\log_b a}$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq c n^{\log_b a} (\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b \left(\frac{n}{b^i}\right)\right)^k$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq c n^{\log_b a} (\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b \left(\frac{n}{b^i}\right)\right)^k$$

$$n = b^\ell \Rightarrow \ell = \log_b n$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b\left(\frac{n}{b^i}\right)\right)^k$$

$$n = b^\ell \Rightarrow \ell = \log_b n$$

$$= cn^{\log_b a} \sum_{i=0}^{\ell - 1} \left(\log_b\left(\frac{b^\ell}{b^i}\right)\right)^k$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b\left(\frac{n}{b^i}\right)\right)^k$$

$n = b^\ell \Rightarrow \ell = \log_b n$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} \left(\log_b\left(\frac{b^\ell}{b^i}\right)\right)^k$$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} (\ell - i)^k$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b\left(\frac{n}{b^i}\right)\right)^k$$

$n = b^\ell \Rightarrow \ell = \log_b n$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} \left(\log_b\left(\frac{b^\ell}{b^i}\right)\right)^k$$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} (\ell - i)^k$$

$$= cn^{\log_b a} \sum_{i=1}^{\ell} i^k$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b\left(\frac{n}{b^i}\right)\right)^k$$

$n = b^\ell \Rightarrow \ell = \log_b n$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} \left(\log_b\left(\frac{b^\ell}{b^i}\right)\right)^k$$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} (\ell - i)^k$$

$$= cn^{\log_b a} \boxed{\sum_{i=1}^{\ell} i^k} \approx \tfrac{1}{k}\ell^{k+1}$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b\left(\frac{n}{b^i}\right)\right)^k$$

$n = b^\ell \Rightarrow \ell = \log_b n$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} \left(\log_b\left(\frac{b^\ell}{b^i}\right)\right)^k$$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} (\ell - i)^k$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

$$= cn^{\log_b a} \sum_{i=1}^{\ell} i^k$$

$$\approx \frac{c}{k} n^{\log_b a} \ell^{k+1}$$

Fall 2. Sei $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b\left(\frac{n}{b^i}\right)\right)^k$$

$n = b^\ell \Rightarrow \ell = \log_b n$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} \left(\log_b\left(\frac{b^\ell}{b^i}\right)\right)^k$$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} (\ell - i)^k$$

Beachte, dass $\sum_{i=1}^{\ell} i^k \leq \ell^{k+1}$ trivial ist, und für diese obere Schranke ausreichen würde.
Für eine untere Schranke reicht auch $\sum_{i=1}^{\ell} i^k \geq \sum_{i=\ell/2}^{\ell} i^k \geq (\ell/2)^{k+1}$.

$$= cn^{\log_b a} \sum_{i=1}^{\ell} i^k$$

$$\approx \frac{c}{k} n^{\log_b a} \ell^{k+1}$$

$$\Rightarrow T(n) = \mathcal{O}(n^{\log_b a} \log^{k+1} n).$$

Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

Dafür müssen wir zunächst zwei Zahlen A and B addieren können:

Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

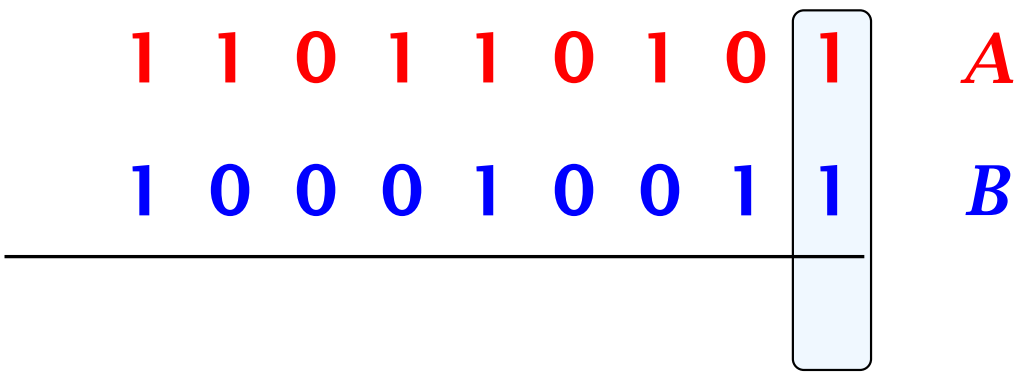
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:

$$\begin{array}{rrrrrrrrrr} & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & A \\ & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & B \\ \hline \end{array}$$

Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

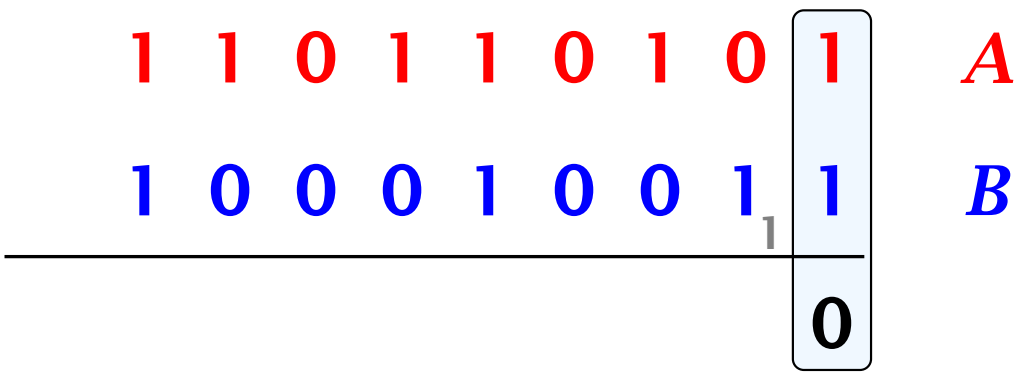
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

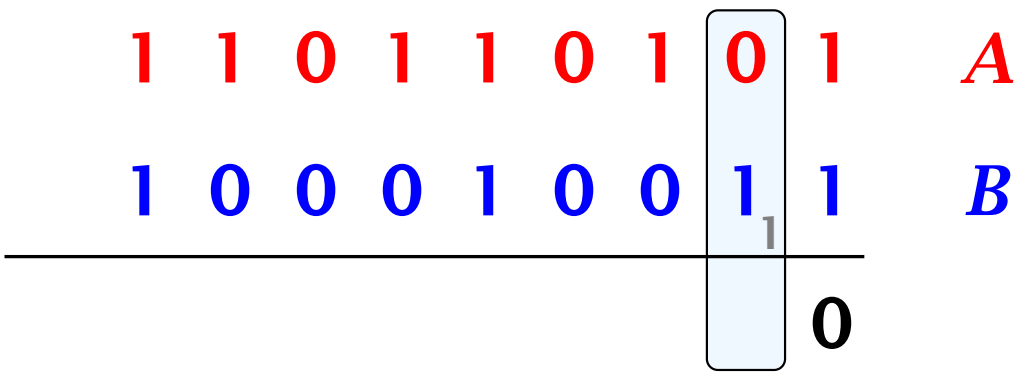
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

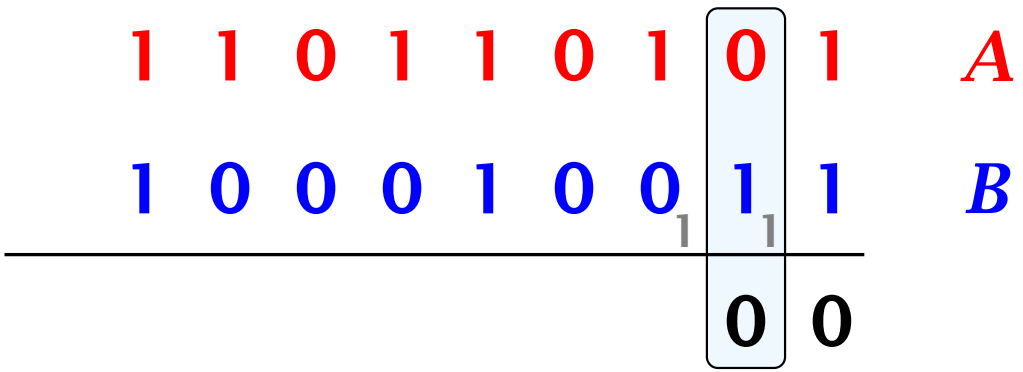
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

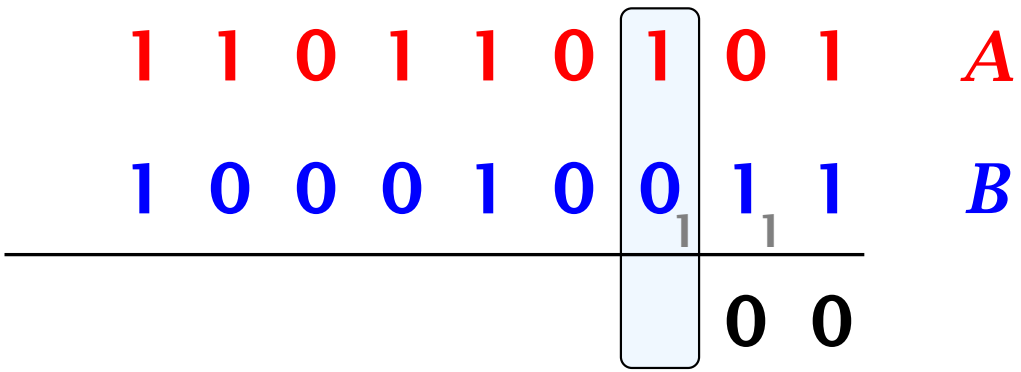
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

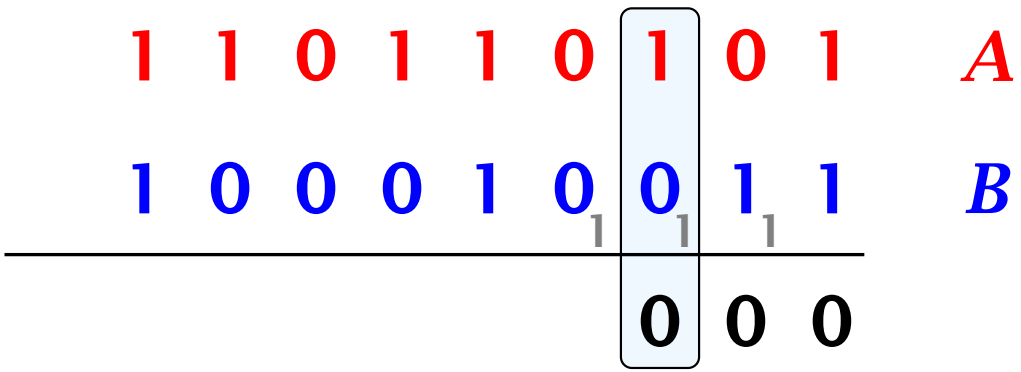
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

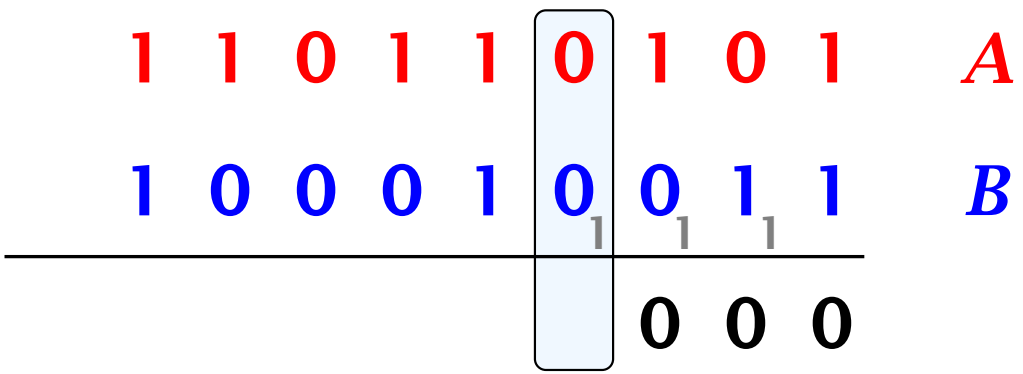
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

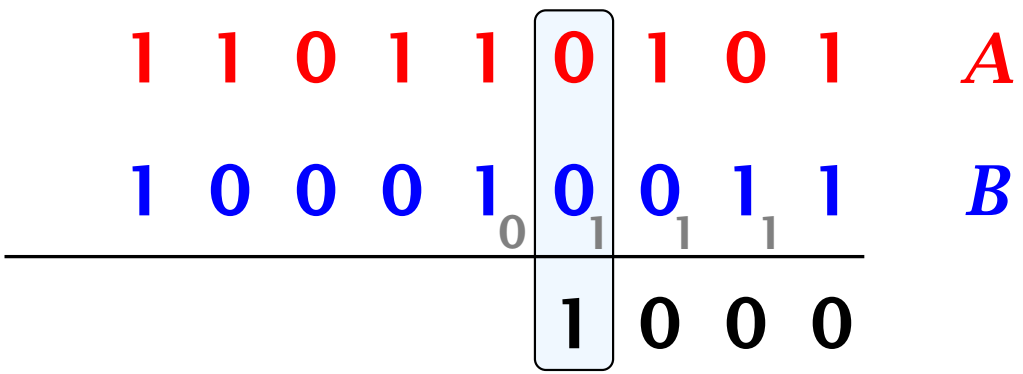
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

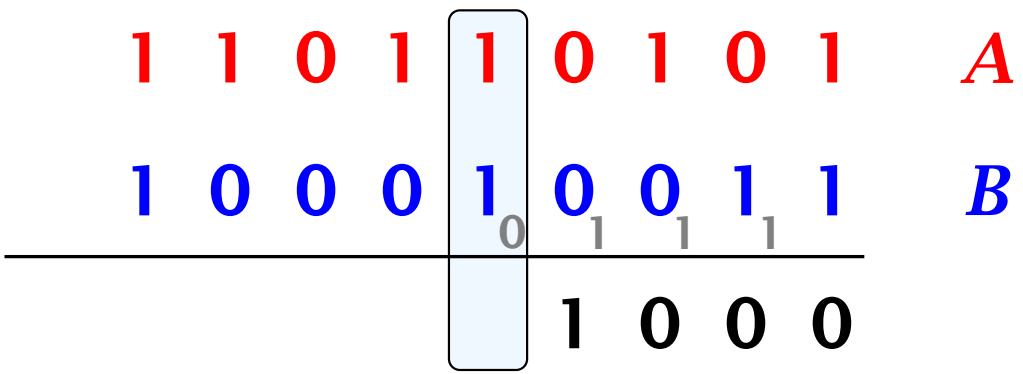
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

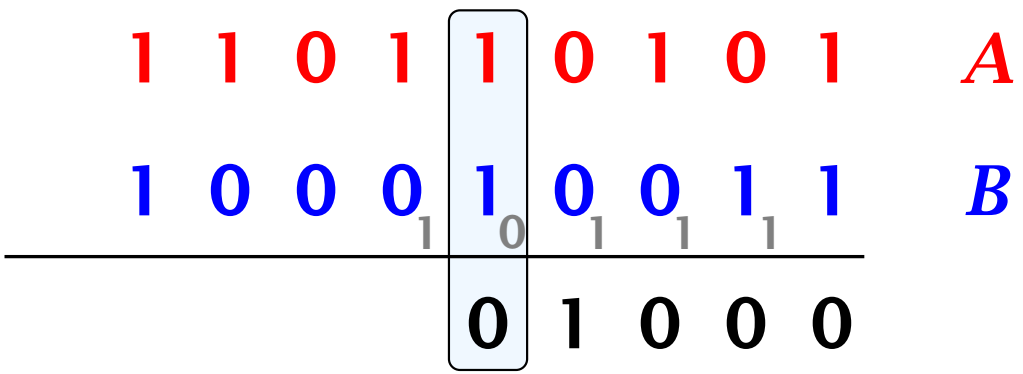
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

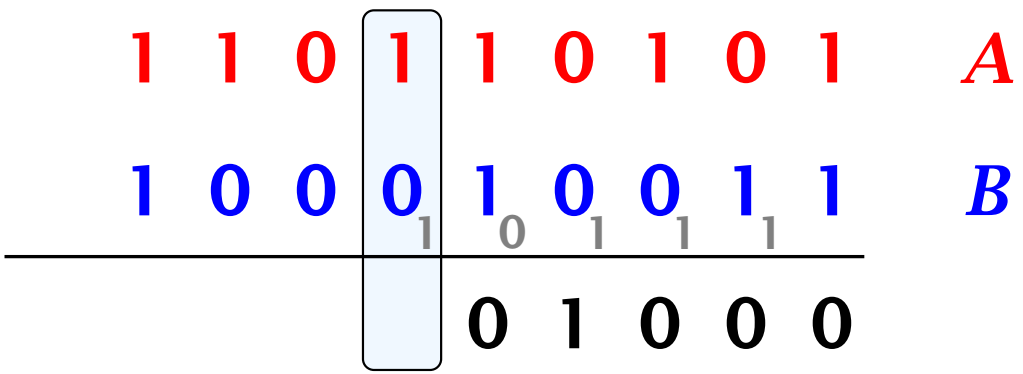
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

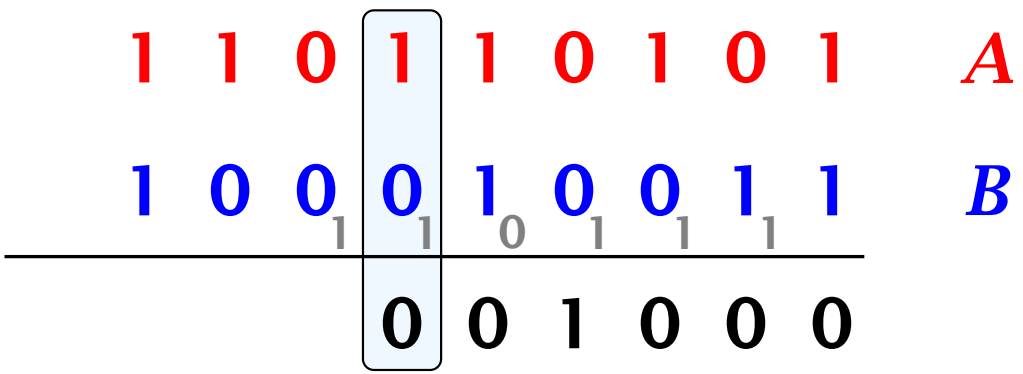
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

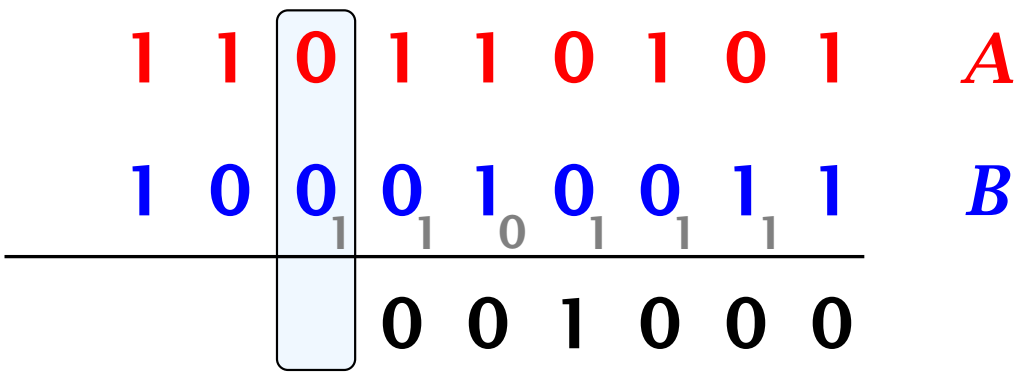
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

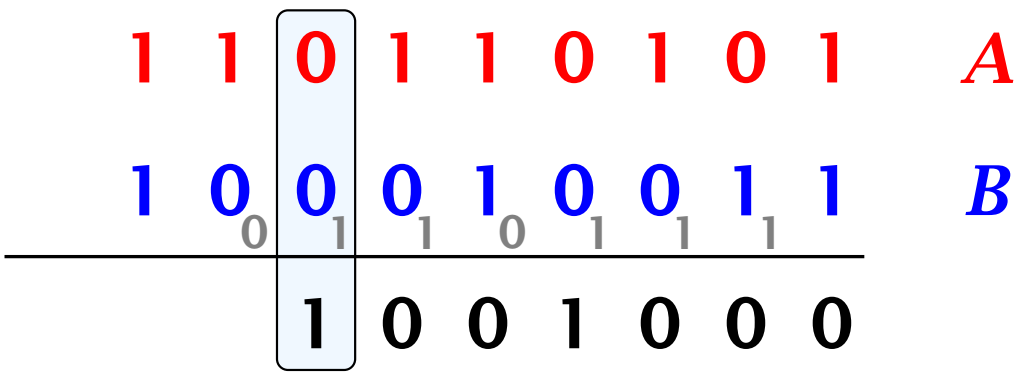
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

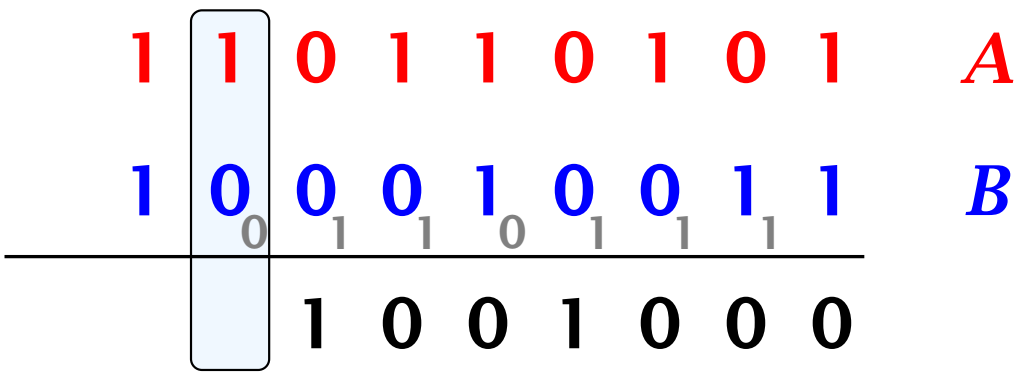
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

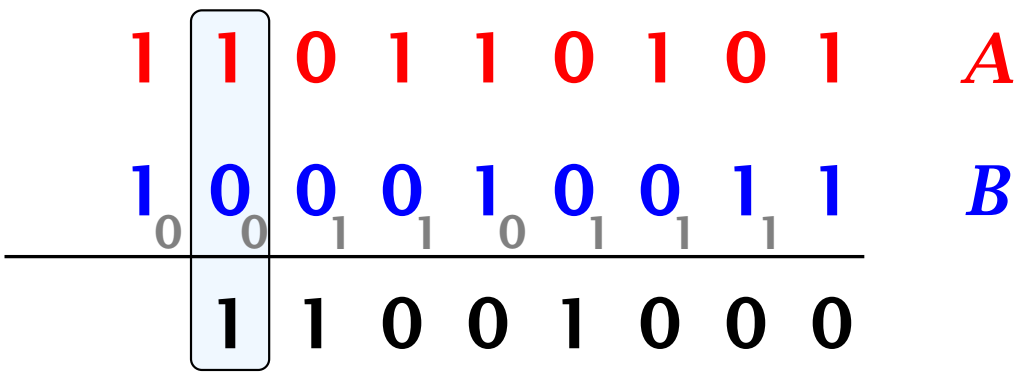
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

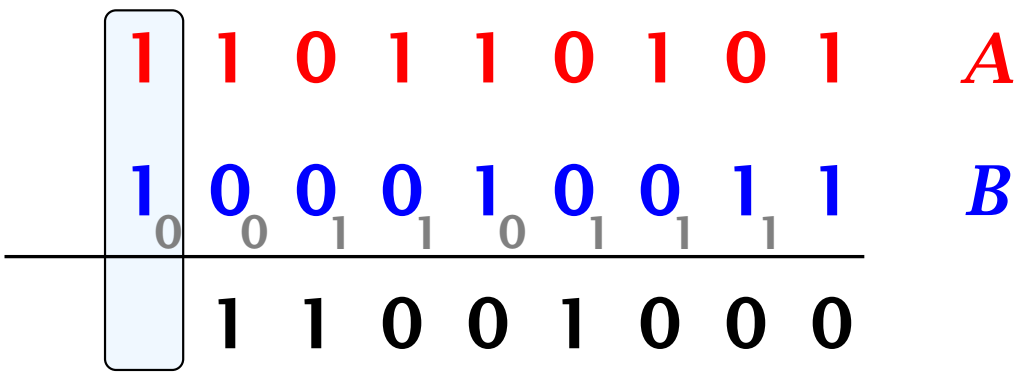
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

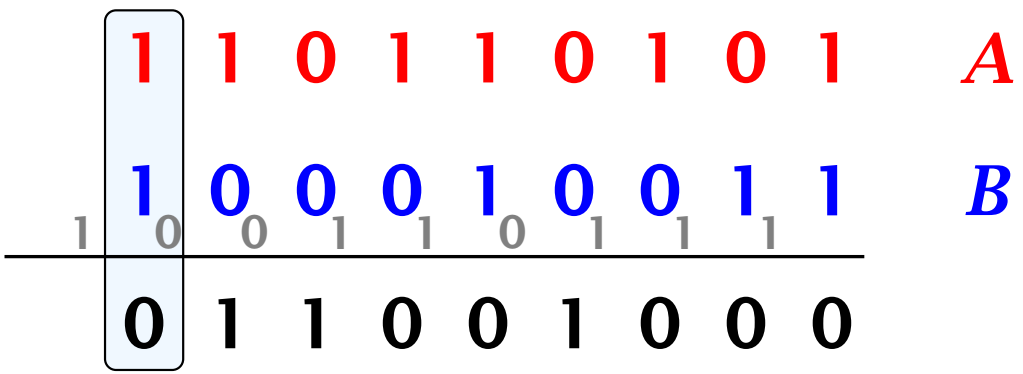
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

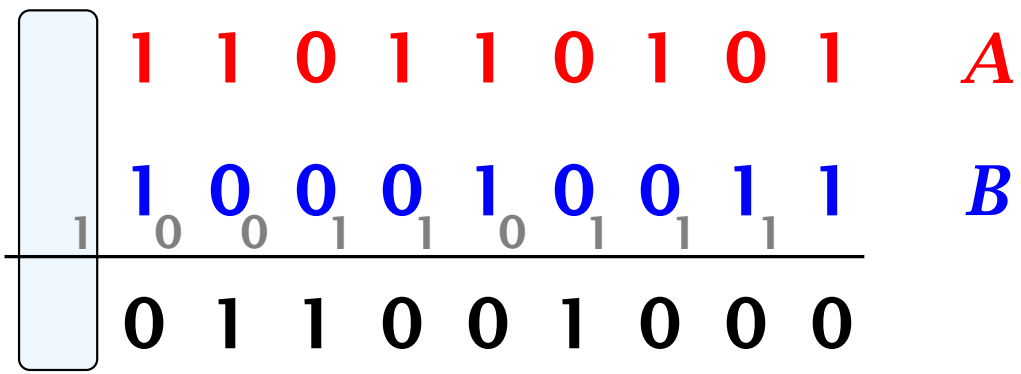
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

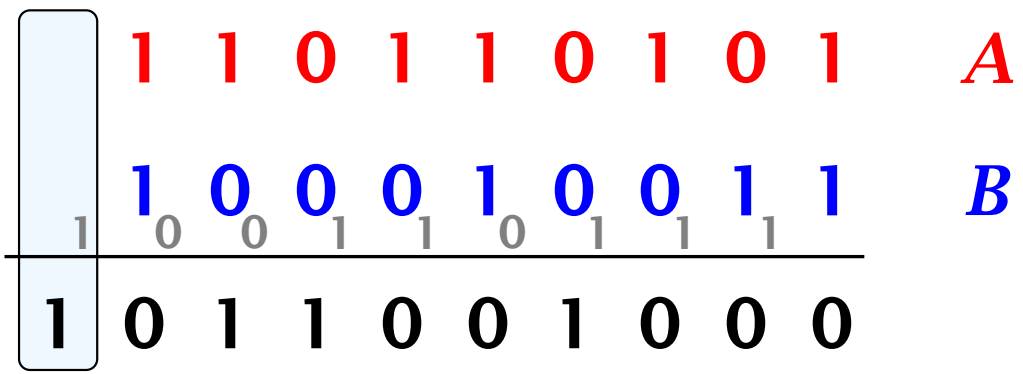
Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

Dafür müssen wir zunächst zwei Zahlen A and B addieren können:



Beispiel: Integermultiplikation

Angenommen wir möchten zwei n-bit Integer multiplizieren, aber unsere Register können nur Operationen auf Integern konstanter Länge ausführen.

Dafür müssen wir zunächst zwei Zahlen A and B addieren können:

$$\begin{array}{cccccccccc} & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & A \\ & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & B \\ \hline 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & & \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \end{array}$$

Das heißt wir können zwei n-bit Integer in Zeit $\mathcal{O}(n)$ addieren.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{r} 1\ 0\ 0\ 0\ 1 \\ \times\ 1\ 0\ 1\ 1 \\ \hline \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{r} 1\ 0\ 0\ 0\ 1 \times 1\ 0\ 1\ \boxed{1} \\ \hline \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{r} 1\ 0\ 0\ 0\ 1 \times 1\ 0\ 1\ \boxed{1} \\ \hline 1\ 0\ 0\ 0\ 1 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{r} 1\ 0\ 0\ 0\ 1 \times 1\ 0\ \boxed{1}\ 1 \\ \hline 1\ 0\ 0\ 0\ 1 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{rrrrrrrrrr} 1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & \boxed{1} & 1 \\ \hline & & & & & & 1 & 0 & 0 & 0 & 1 \\ & & & & & & & & & 0 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{rrrrrcrrrr} 1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & \boxed{1} & 1 \\ \hline & & & & & & 1 & 0 & 0 & 0 & 1 \\ & & & & & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{rrrrrcrrrr} 1 & 0 & 0 & 0 & 1 & \times & 1 & \boxed{0} & 1 & 1 \\ \hline & & & & & & 1 & 0 & 0 & 0 & 1 \\ & & & & & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{rrrrrrrrrr} 1 & 0 & 0 & 0 & 1 & \times & 1 & \boxed{0} & 1 & 1 \\ \hline & & & & & & 1 & 0 & 0 & 0 & 1 \\ & & & & & 1 & 0 & 0 & 0 & 1 & 0 \\ & & & & & & & & & 0 & 0 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{rrrrrrrrrr} 1 & 0 & 0 & 0 & 1 & \times & 1 & \boxed{0} & 1 & 1 \\ \hline & & & & & & 1 & 0 & 0 & 0 & 1 \\ & & & & & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{rrrrrcrrrr} 1 & 0 & 0 & 0 & 1 & \times & \boxed{1} & 0 & 1 & 1 \\ \hline & & & & & & 1 & 0 & 0 & 0 & 1 \\ & & & & & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$\begin{array}{rrrrrrrrrr} 1 & 0 & 0 & 0 & 1 & \times & \boxed{1} & 0 & 1 & 1 \\ \hline & & & & & & 1 & 0 & 0 & 0 & 1 \\ & & & & & 1 & 0 & 0 & 0 & 1 & 0 \\ & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & & 0 & 0 & 0 \end{array}$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$
\begin{array}{rrrrrcrrrr}
1 & 0 & 0 & 0 & 1 & \times & \boxed{1} & 0 & 1 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 0 & 1 & \\
 & & & 1 & 0 & 0 & 0 & 1 & 0 & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \\
\end{array}
$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$
\begin{array}{rrrrrcrrrr}
1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 0 & 1 & \\
 & & & 1 & 0 & 0 & 0 & 1 & 0 & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \\
\hline
\end{array}
$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$
\begin{array}{rrrrrcrrrr}
1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 0 & 1 & \\
 & & & 1 & 0 & 0 & 0 & 1 & 0 & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & &
\end{array}
$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$
\begin{array}{rrrrrcrrrr}
1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 0 & 1 & \\
 & & & 1 & 0 & 0 & 0 & 1 & 0 & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & &
\end{array}
$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Laufzeit:

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$
\begin{array}{rrrrrcrrrr}
1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 0 & 1 & \\
 & & & 1 & 0 & 0 & 0 & 1 & 0 & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & &
\end{array}
$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Laufzeit:

- Zwischenergebnisse berechnen: $\mathcal{O}(nm)$.

Beispiel: Integermultiplikation

Angenommen wir möchten ein n-bit Integer A und ein m-bit Integer B ($m \leq n$) multiplizieren.

$$
\begin{array}{rrrrrcrrrr}
1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 0 & 1 & \\
 & & & 1 & 0 & 0 & 0 & 1 & 0 & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & &
\end{array}
$$

- Dies nennt man auch die „Schulmethode“ für die Integermultiplikation.
- Die Zahlen der Zwischenergebnisse haben höchstens $m + n \leq 2n$ bits.

Laufzeit:

- Zwischenergebnisse berechnen: $\mathcal{O}(nm)$.
- Addieren von m Zahlen der Länge $\leq 2n$: $\mathcal{O}((m+n)m) = \mathcal{O}(nm)$.

Beispiel: Integermultiplikation

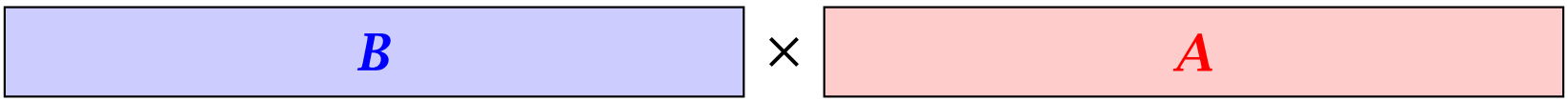
Ein rekursiver Ansatz:

Angenommen die Integer A und B haben Länge $n = 2^k$.

Beispiel: Integermultiplikation

Ein rekursiver Ansatz:

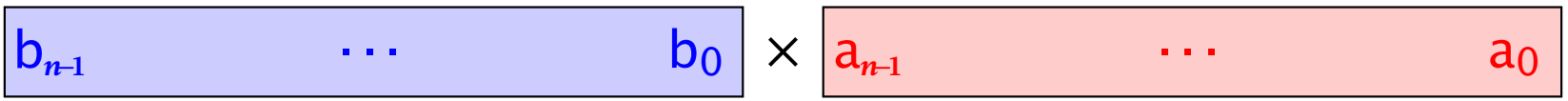
Angenommen die Integer A und B haben Länge $n = 2^k$.



Beispiel: Integermultiplikation

Ein rekursiver Ansatz:

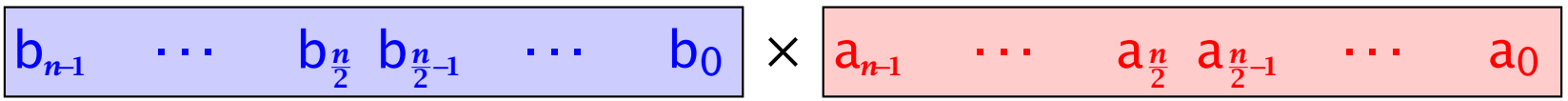
Angenommen die Integer A und B haben Länge $n = 2^k$.



Beispiel: Integermultiplikation

Ein rekursiver Ansatz:

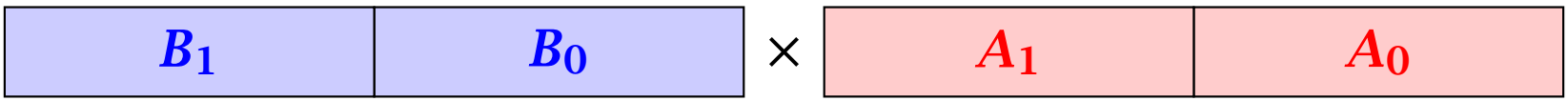
Angenommen die Integer A und B haben Länge $n = 2^k$.



Beispiel: Integermultiplikation

Ein rekursiver Ansatz:

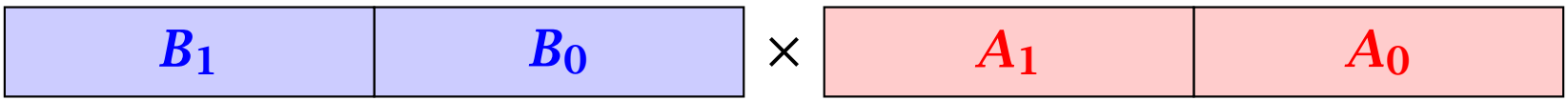
Angenommen die Integer A und B haben Länge $n = 2^k$.



Beispiel: Integermultiplikation

Ein rekursiver Ansatz:

Angenommen die Integer A und B haben Länge $n = 2^k$.



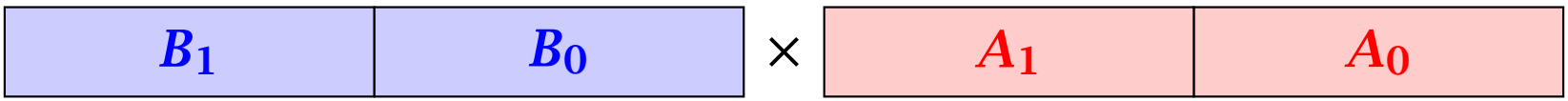
Dann gilt

$$A = A_1 \cdot 2^{\frac{n}{2}} + A_0 \text{ und } B = B_1 \cdot 2^{\frac{n}{2}} + B_0$$

Beispiel: Integermultiplikation

Ein rekursiver Ansatz:

Angenommen die Integer A und B haben Länge $n = 2^k$.



Dann gilt

$$A = A_1 \cdot 2^{\frac{n}{2}} + A_0 \text{ und } B = B_1 \cdot 2^{\frac{n}{2}} + B_0$$

Also,

$$A \cdot B = A_1 B_1 \cdot 2^n + (A_1 B_0 + A_0 B_1) \cdot 2^{\frac{n}{2}} + A_0 B_0$$

Beispiel: Integermultiplikation

```
1  Input:  Zahlen A, B, repraesentiert durch Bitarrays
2          der Laenge n
3  Output: Bitarray, das A*B enthaelt
4
5  mult(A, B, n)
6      if (n == 1)
7          return new int(A[0]*B[0]);
8      split(A,A0,A1);
9      split(B,B0,B1);
10     Z2 = mult(A1,B1,n/2);
11     Z1 = mult(A0,B1,n/2) + mult(A1,B0,n/2);
12     Z0 = mult(A0,B0,n/2);
13     return Z2*2^n + Z1*2^(n/2) + Z0;
```

Die Addition + addiert Bitarrays. Das funktioniert in C++ nicht direkt, aber man kann stattdessen z.B. eine Funktion add(A,B) benutzen. Wenn man den Algorithmus nach C++ übertragen möchte ist das Speichermanagement etwas unübersichtlich, da man jedes Zwischenergebnis vor Funktionsende wieder von Hand löschen muss.

Wir erhalten folgende Rekurrenzgleichung:

$$T(n) = 4T\left(\frac{n}{2}\right) + \mathcal{O}(n) \ .$$

Beispiel: Integermultiplikation

Mastertheorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Fall 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Fall 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

Beispiel: Integermultiplikation

Mastertheorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Fall 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Fall 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

In unserem Fall $a = 4$, $b = 2$, und $f(n) = \Theta(n)$. Also, Fall 1, da $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$.

Beispiel: Integermultiplikation

Mastertheorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Fall 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Fall 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

In unserem Fall $a = 4$, $b = 2$, und $f(n) = \Theta(n)$. Also, Fall 1, da $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$.

Wir erhalten Laufzeit $\mathcal{O}(n^2)$.

Beispiel: Integermultiplikation

Mastertheorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Fall 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Fall 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

In unserem Fall $a = 4$, $b = 2$, und $f(n) = \Theta(n)$. Also, Fall 1, da $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$.

Wir erhalten Laufzeit $\mathcal{O}(n^2)$.

$\Longrightarrow$ Nicht besser als „Schulmethode“.

Beispiel: Integermultiplikation

We can use the following identity to compute Z_1:

Beispiel: Integermultiplikation

We can use the following identity to compute Z_1:

$$Z_1 = A_1 B_0 + A_0 B_1$$

Beispiel: Integermultiplikation

We can use the following identity to compute Z_1:

$$\begin{aligned} Z_1 &= A_1B_0 + A_0B_1 \\ &= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0 \end{aligned}$$

Beispiel: Integermultiplikation

We can use the following identity to compute Z_1:

$$\begin{aligned} Z_1 &= A_1B_0 + A_0B_1 \\ &= (A_0 + A_1) \cdot (B_0 + B_1) - \underbrace{A_1B_1}_{= Z_2} - \underbrace{A_0B_0}_{= Z_0} \end{aligned}$$

Beispiel: Integermultiplikation

We can use the following identity to compute Z_1:

$$\begin{aligned} Z_1 &= A_1B_0 + A_0B_1 \qquad\qquad \underbrace{= Z_2} \quad \underbrace{= Z_0} \\ &= (A_0 + A_1) \cdot (B_0 + B_1) - \underbrace{A_1B_1} - \underbrace{A_0B_0} \end{aligned}$$

```
1  Input:  Zahlen A, B, repraesentiert durch Bitarrays
2          der Laenge n
3  Output: Bitarray, das A*B enthaelt
4
5  mult(A, B, n)
6      if (n == 1)
7          return new int(A[0]*B[0]);
8      split(A,A0,A1);
9      split(B,B0,B1);
10     Z2 = mult(A1,B1,n/2);
11     Z1 = mult(A0+A1,B0+B1,n/2)-Z0-Z2;
12     Z0 = mult(A0,B0,n/2);
13     return Z2*2^n + Z1*2^(n/2) + Z0;
```

Beispiel: Integermultiplikation

Zeile 11 ist leider nicht korrekt, da A0 + A1 bzw. B0 + B1 eventuell $n/2+1$ bits haben können. Wenn man eine $n/2+1$-bit Zahl X in das höchstwertige Bit $X_{\frac{n}{2}}$ und die restlichen Bits $\bar{X}$ zerlegt kann man folgendes nutzen:

```
...
 X = A0 + A1;
 Y = B0 + B1;
Z1 = X_{n/2}*Y_{n/2}*2^n + (X_{n/2}*Ȳ + Y_{n/2}*X̄)*2^{n/2} + mult(X̄,Ȳ) - Z0 - Z2;
...
```

Laufzeit hierfür ist $T(n/2)+\mathcal{O}(n)$.

Beispiel: Integermultiplikation

Wir erhalten folgende Rekurrenz:

$$T(n) = 3T\left(\frac{n}{2}\right) + \mathcal{O}(n) \ .$$

Master Theorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

Wir sind im Fall 1. Laufzeit $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$.

Eine deutliche Verbesserung der „Schulmethode“.

Beispiel: Integermultiplikation

Wir erhalten folgende Rekurrenz:

$$T(n) = 3T\left(\frac{n}{2}\right) + \mathcal{O}(n) \ .$$

Master Theorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

Wir sind im Fall 1. Laufzeit $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$.

Eine deutliche Verbesserung der „Schulmethode“.

Beispiel: Integermultiplikation

Wir erhalten folgende Rekurrenz:

$$T(n) = 3T\left(\frac{n}{2}\right) + \mathcal{O}(n) \ .$$

Master Theorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

Wir sind im Fall 1. Laufzeit $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$.

Eine deutliche Verbesserung der „Schulmethode“.

Beispiel: Integermultiplikation

Wir erhalten folgende Rekurrenz:

$$T(n) = 3T\left(\frac{n}{2}\right) + \mathcal{O}(n) \ .$$

Master Theorem: Rekurrenz: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

Wir sind im Fall 1. Laufzeit $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$.

Eine deutliche Verbesserung der „Schulmethode“.