6.2 Master Theorem

Note that the cases do not cover all possibilities.

Lemma 5

Let $a \geq 1, b > 1$ and $\epsilon > 0$ denote constants. Consider the recurrence

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

Case 1.

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

Case 2.

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

Case 3.

If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ and for sufficiently large n $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ then $T(n) = \Theta(f(n))$.

6.2 Master Theorem

We prove the Master Theorem for the case that n is of the form b^ℓ, and we assume that the non-recursive case occurs for problem size 1 and incurs cost 1.

The Recursion Tree

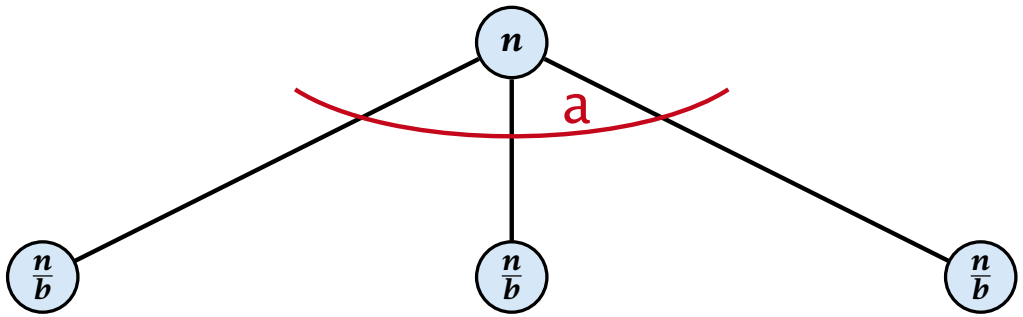
The running time of a recursive algorithm can be visualized by a recursion tree:

The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:

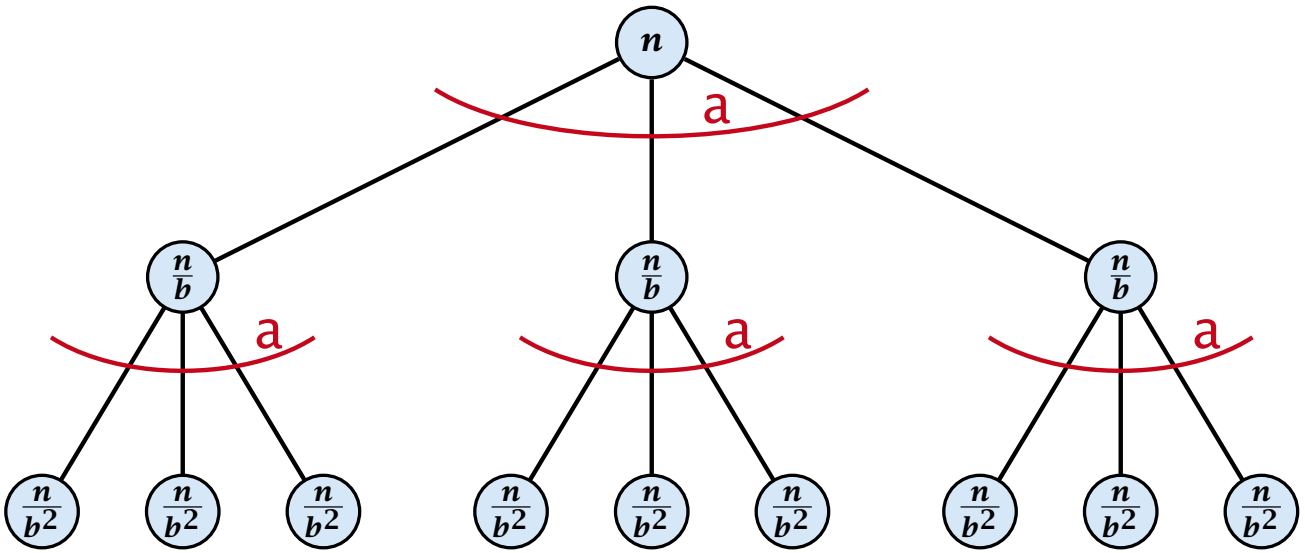
The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



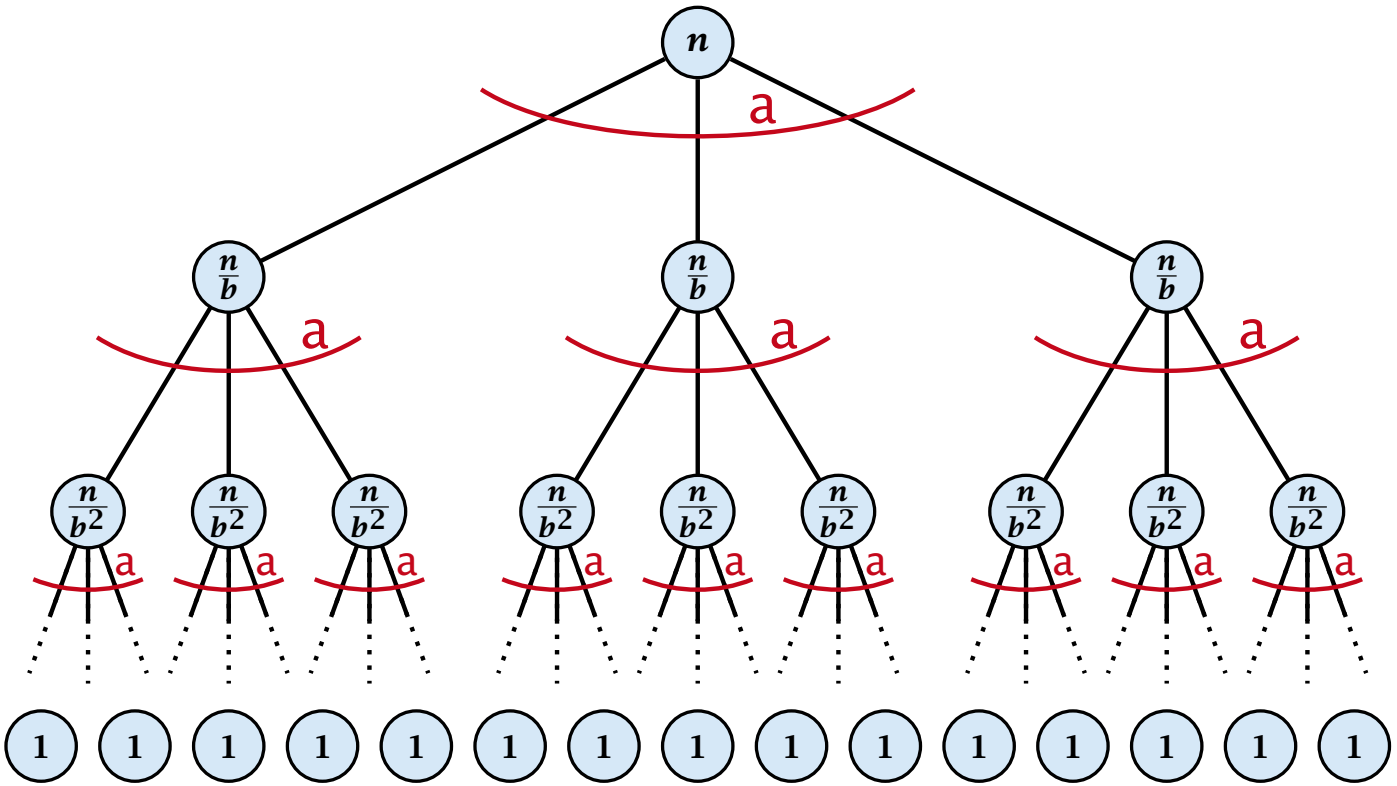
The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



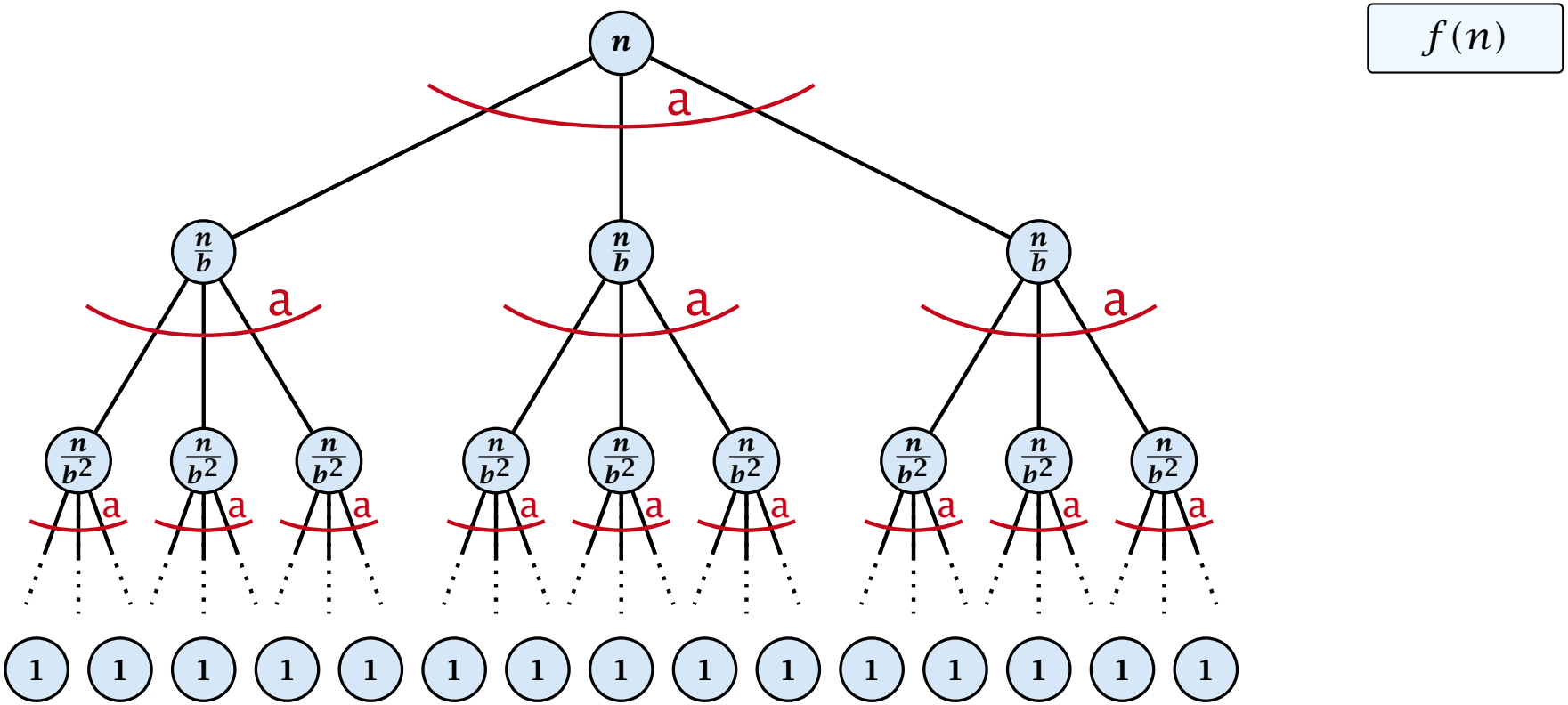
The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



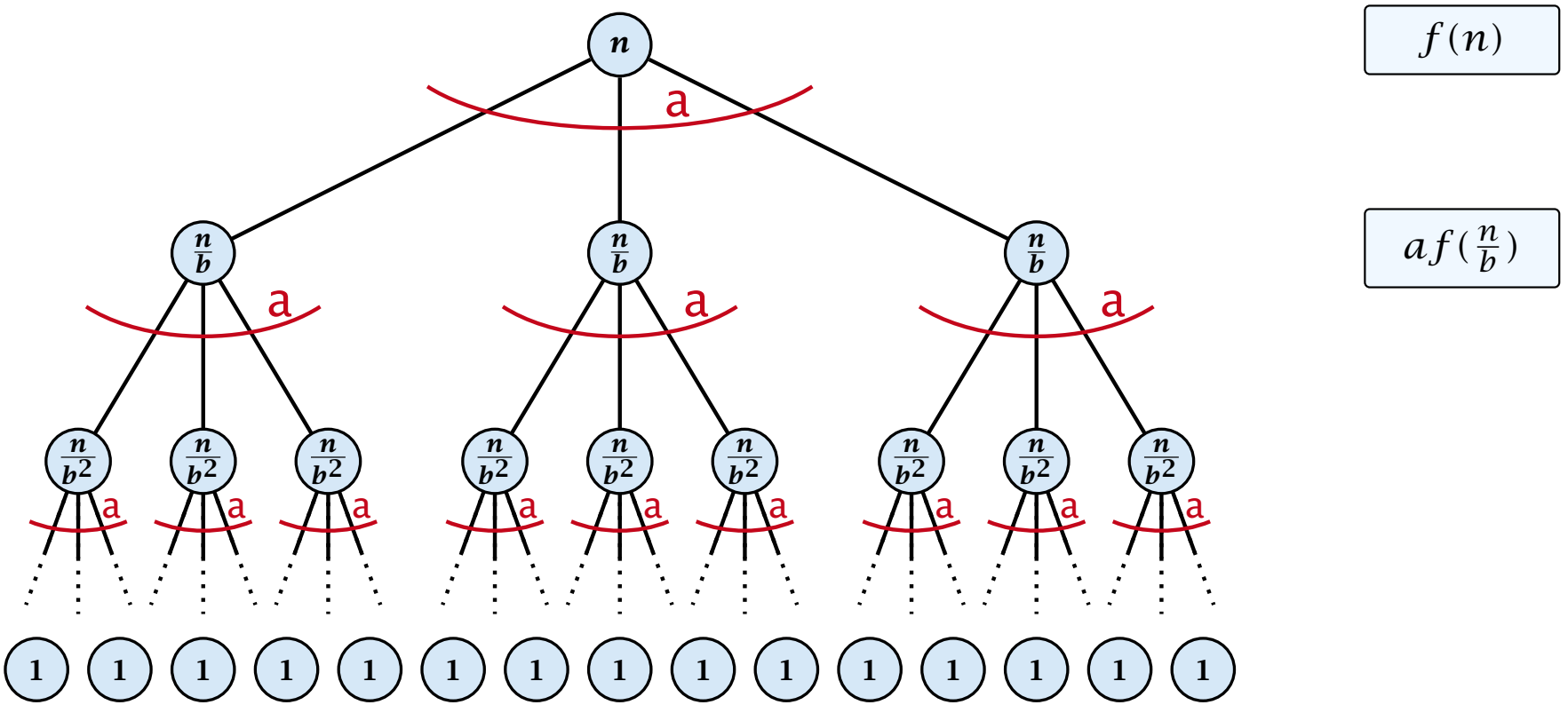
The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



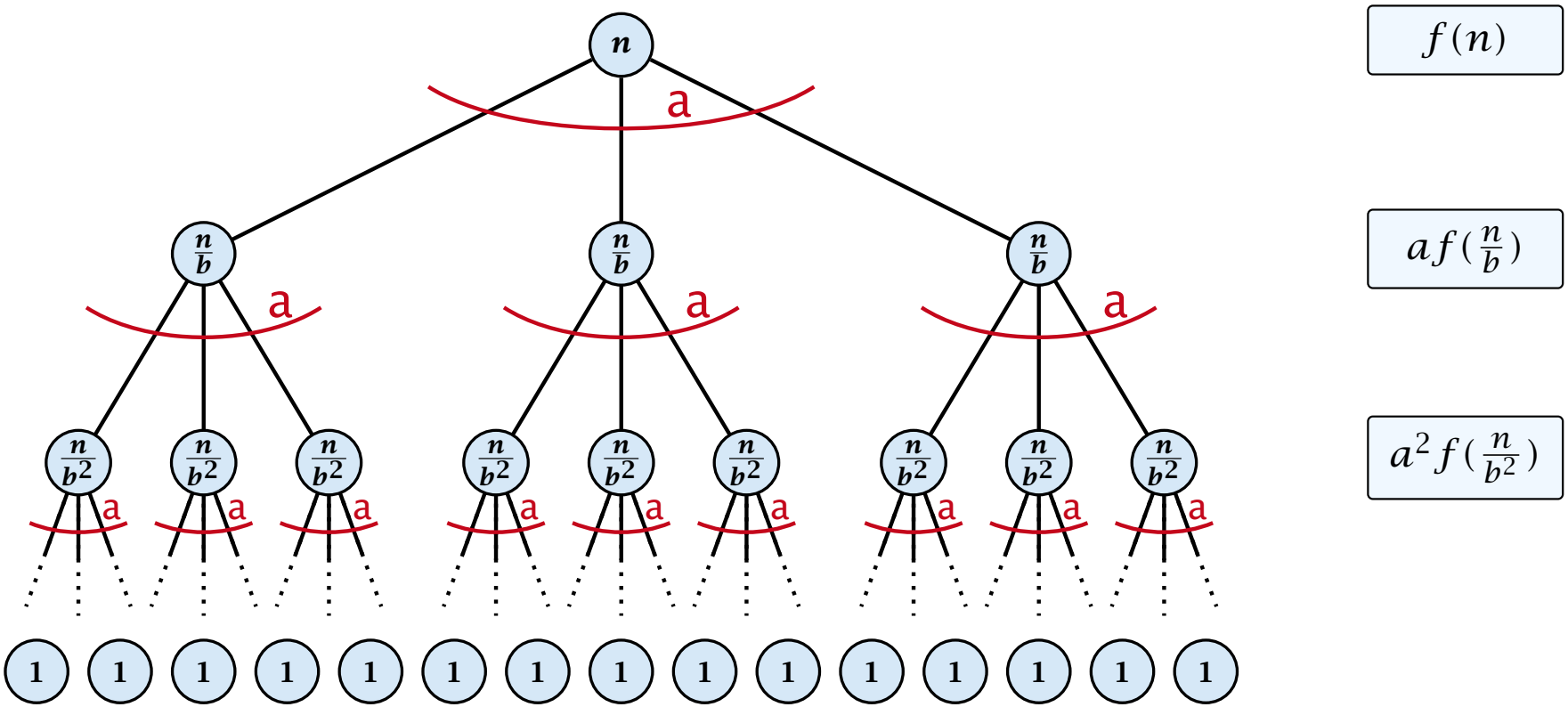
The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



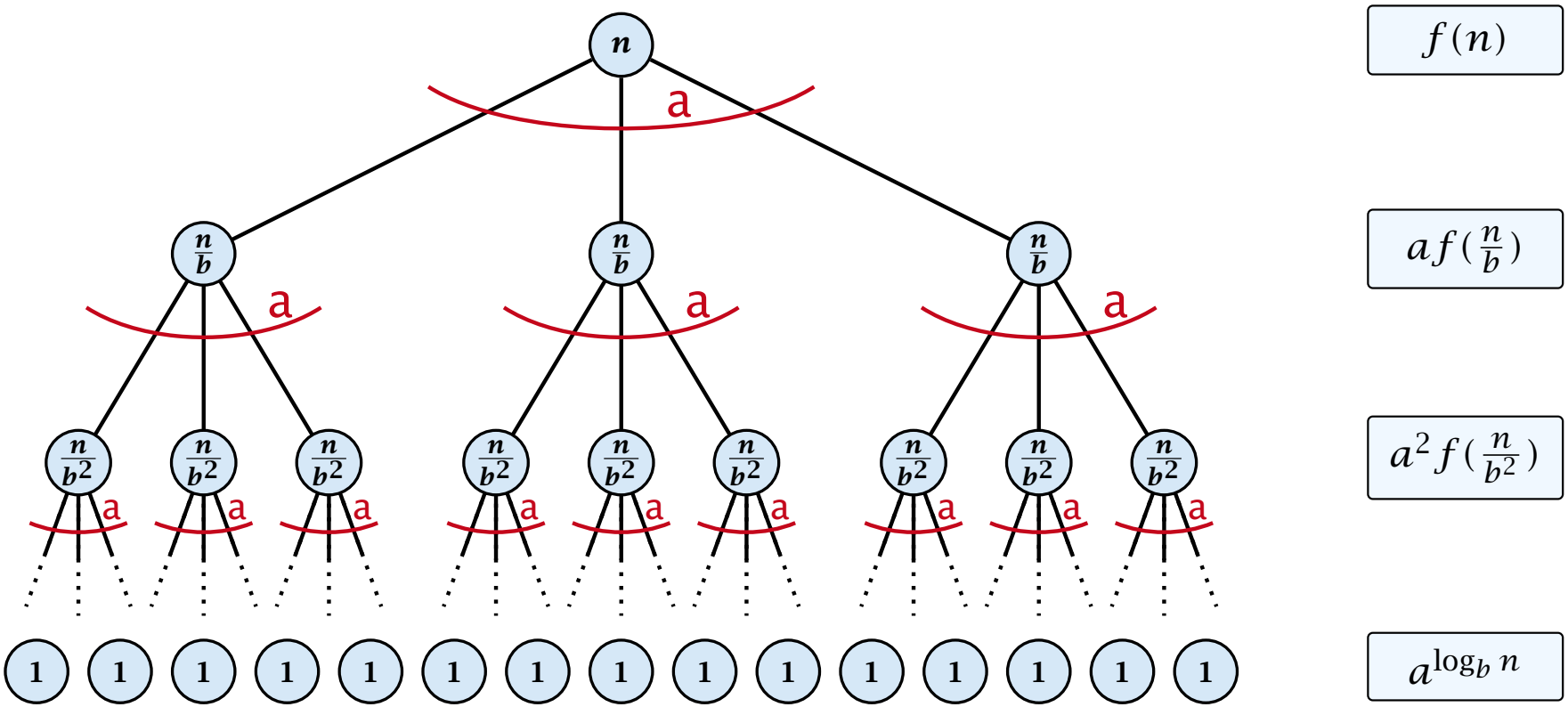
The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



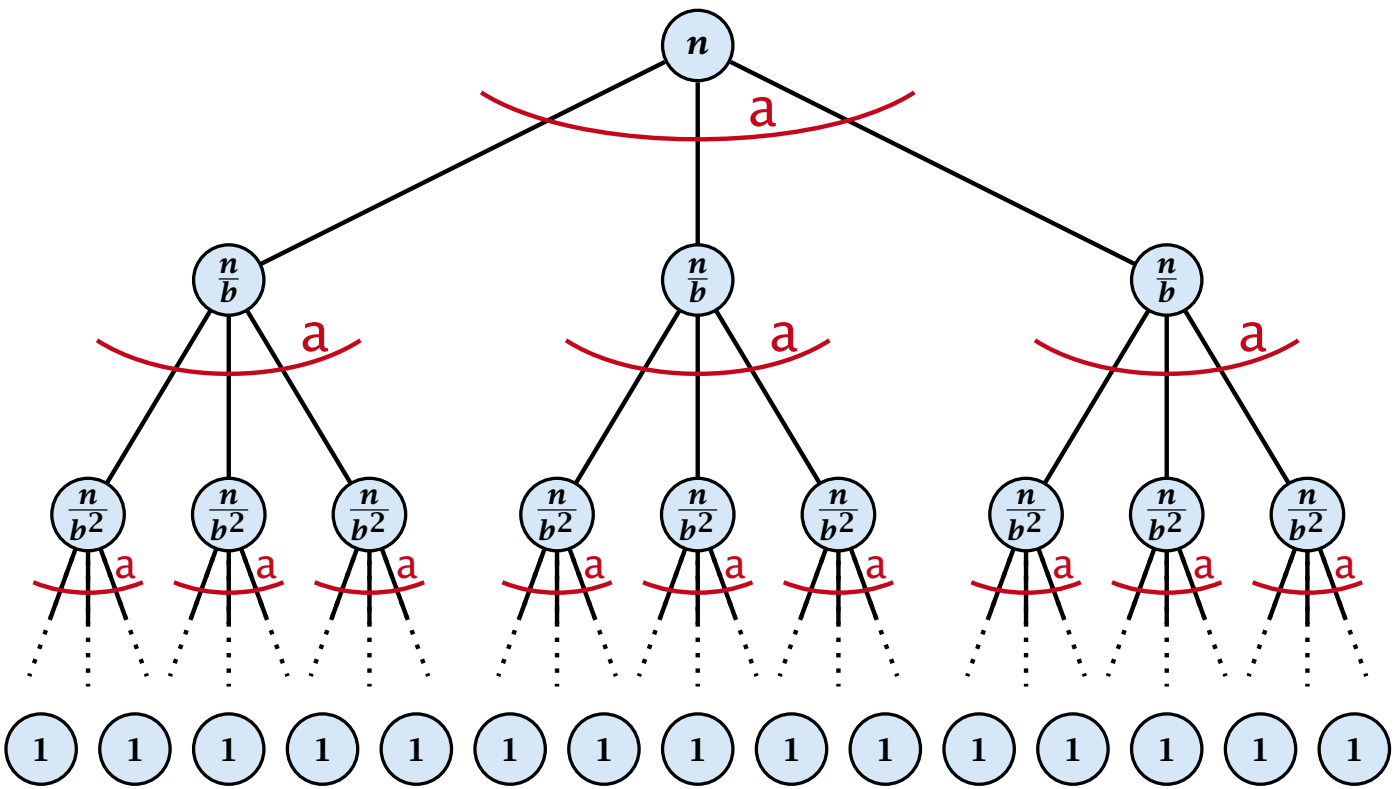
The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



$f(n)$

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

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

$a^{\log_b n}$

$=$

$n^{\log_b a}$

6.2 Master Theorem

This gives

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a - \epsilon}$.

Case 1. Now suppose that $f(n) \leq cn^{\log_b a - \epsilon}$.

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a - \epsilon}$.

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

Case 1. Now suppose that $f(n) \leq c n^{\log_b a - \epsilon}$.

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

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

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

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

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

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

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

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

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

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

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

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

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

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

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

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

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

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

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

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

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

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

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

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

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

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

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

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

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

Hence,

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

Case 1. Now suppose that $f(n) \leq cn^{\log_b a-\epsilon}$.

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

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

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

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

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

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

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

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

Hence,

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

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

Case 2. Now suppose that $f(n) \leq c n^{\log_b a}$.

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

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

Case 2. Now suppose that $f(n) \leq c n^{\log_b a}$.

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

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

Hence,

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

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

Hence,

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

Case 2. Now suppose that $f(n) \geq cn^{\log_b a}$.

Case 2. Now suppose that $f(n) \geq c n^{\log_b a}$.

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

Case 2. Now suppose that $f(n) \geq c n^{\log_b a}$.

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

Case 2. Now suppose that $f(n) \geq c n^{\log_b a}$.

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

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

Case 2. Now suppose that $f(n) \geq cn^{\log_b a}$.

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

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

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

Case 2. Now suppose that $f(n) \geq c n^{\log_b a}$.

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

Case 2. Now suppose that $f(n) \geq cn^{\log_b a}$.

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

Hence,

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

Case 2. Now suppose that $f(n) \geq c n^{\log_b a}$.

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

Hence,

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

Case 2. Now suppose that $f(n) \leq c n^{\log_b a} (\log_b(n))^k$.

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

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

Case 2. Now suppose that $f(n) \leq c n^{\log_b a} (\log_b(n))^k$.

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

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

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

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

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

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

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

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

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

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

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

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

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

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

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

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

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

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

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

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

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

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

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

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

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

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

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

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

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

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

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

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

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

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

$$\leq \sum_{i=0}^{\log_b n-1} c^i f(n) + \mathcal{O}(n^{\log_b a})$$

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

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

$$\leq \sum_{i=0}^{\log_b n-1} c^i f(n) + \mathcal{O}(n^{\log_b a})$$

$q < 1 : \sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q} \leq \frac{1}{1-q}$

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

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

$$\leq \sum_{i=0}^{\log_b n-1} c^i f(n) + \mathcal{O}(n^{\log_b a})$$

$$q < 1 : \sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q} \leq \frac{1}{1-q} \qquad \leq \frac{1}{1-c} f(n) + \mathcal{O}(n^{\log_b a})$$

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

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

$$\leq \sum_{i=0}^{\log_b n-1} c^i f(n) + \mathcal{O}(n^{\log_b a})$$

$q < 1 : \sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q} \leq \frac{1}{1-q}$

$$\leq \frac{1}{1-c} f(n) + \mathcal{O}(n^{\log_b a})$$

Hence,

$$T(n) \leq \mathcal{O}(f(n))$$

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Case 3. Now suppose that $f(n) \geq dn^{\log_b a+\epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

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

$$\leq \sum_{i=0}^{\log_b n-1} c^i f(n) + \mathcal{O}(n^{\log_b a})$$

$q < 1 : \sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q} \leq \frac{1}{1-q}$

$$\leq \frac{1}{1-c} f(n) + \mathcal{O}(n^{\log_b a})$$

Hence,

$$T(n) \leq \mathcal{O}(f(n)) \qquad \Rightarrow T(n) = \Theta(f(n)).$$

Where did we use $f(n) \geq \Omega(n^{\log_b a+\epsilon})$?

Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

For this we first need to be able to add two integers A and B:

Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

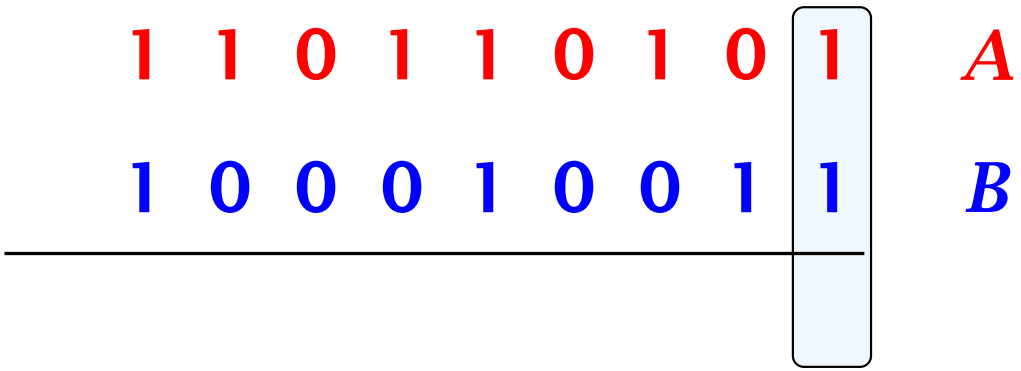
For this we first need to be able to add two integers A and B:

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

Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

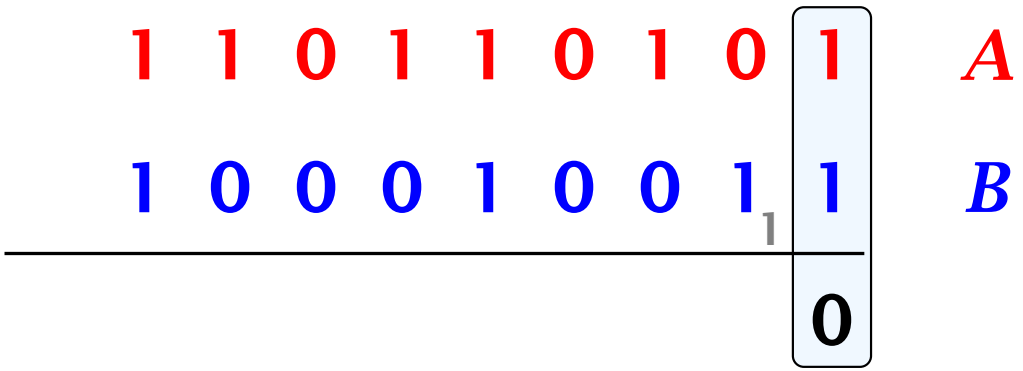
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

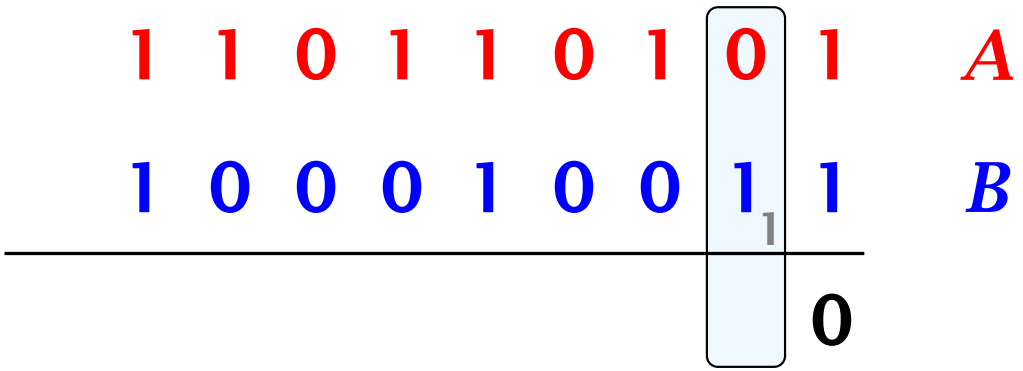
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

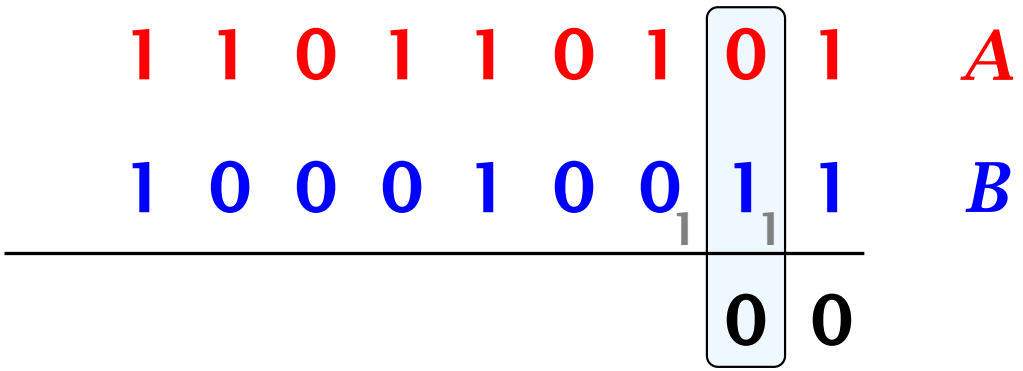
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

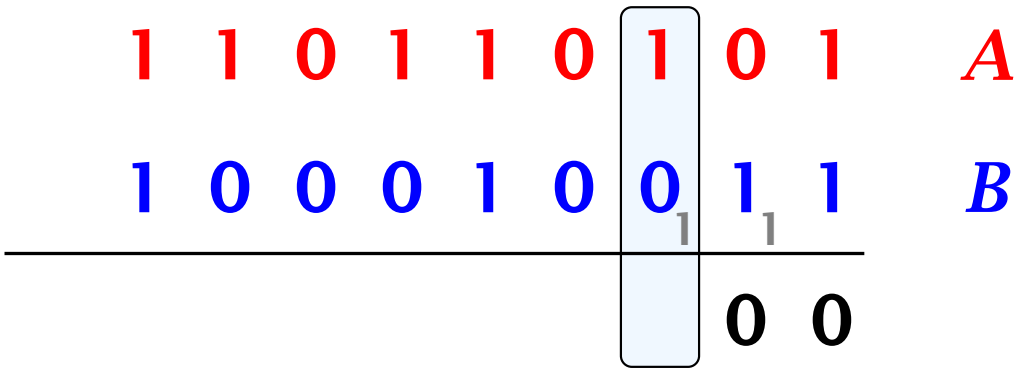
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

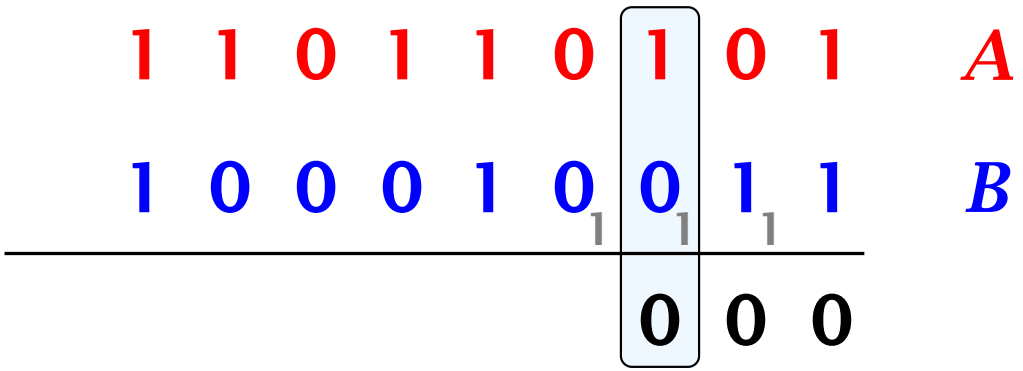
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

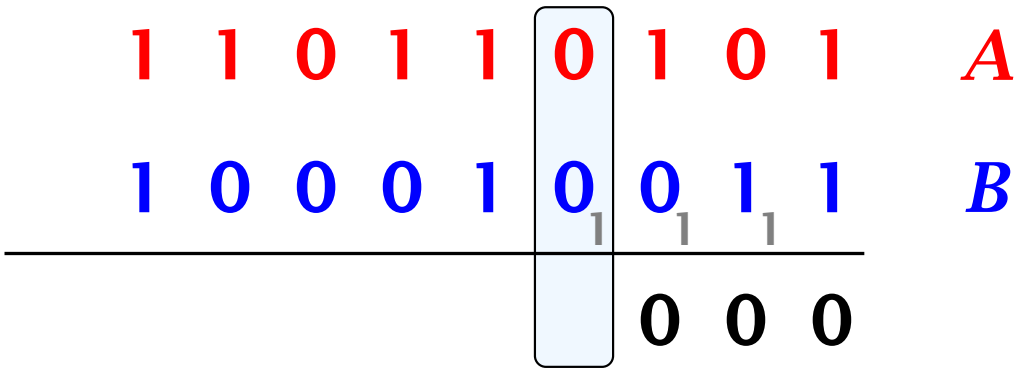
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

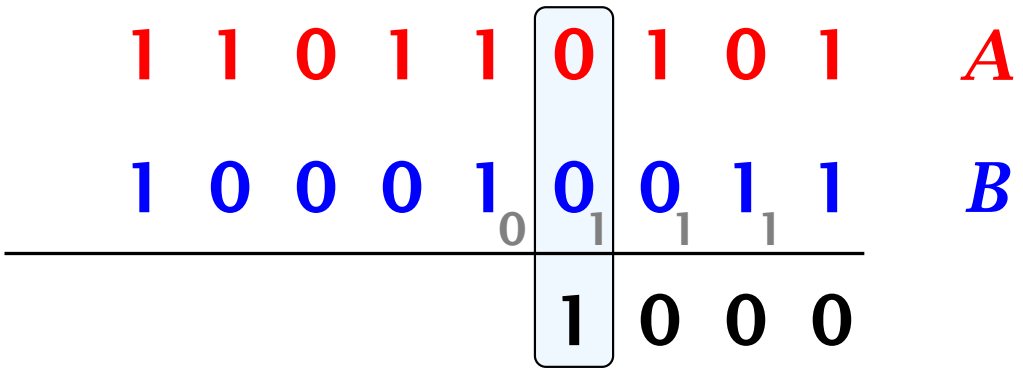
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

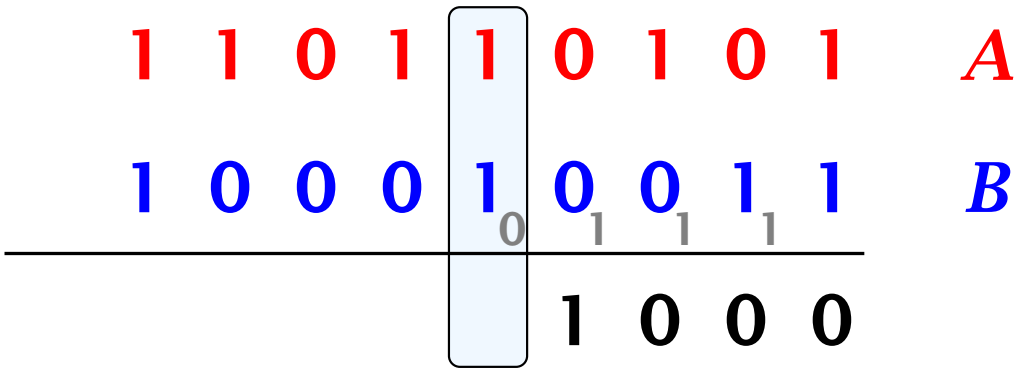
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

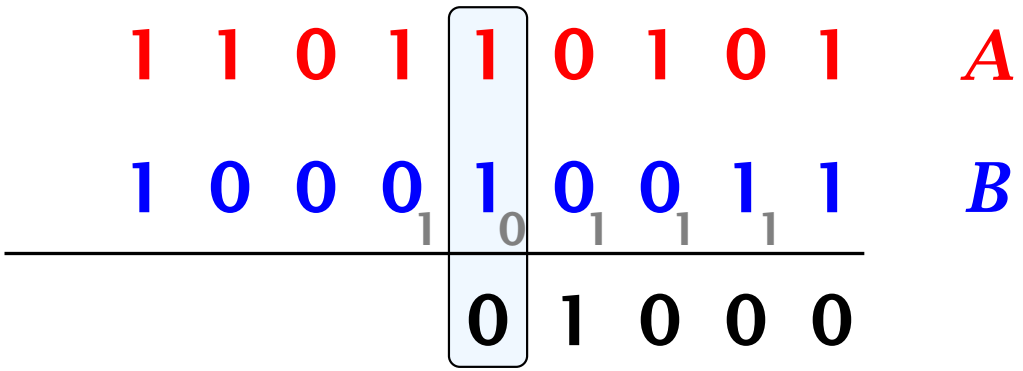
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

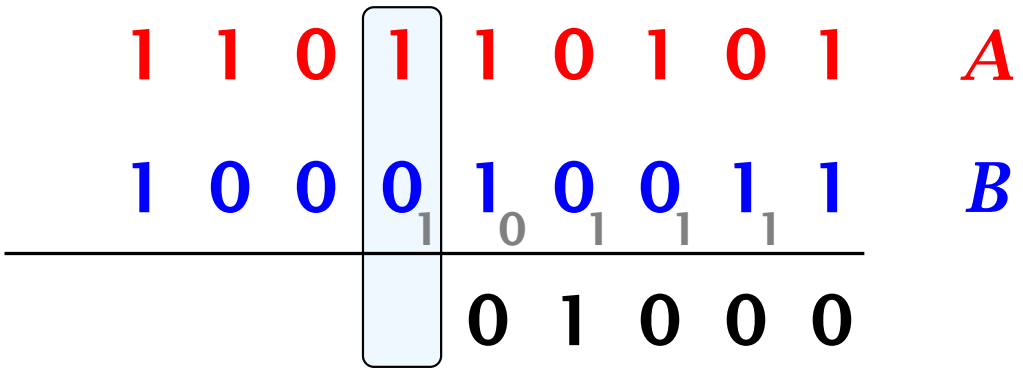
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

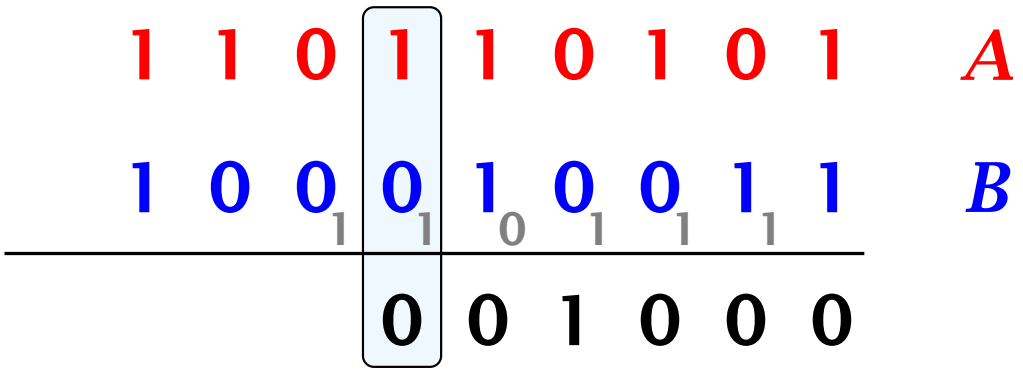
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

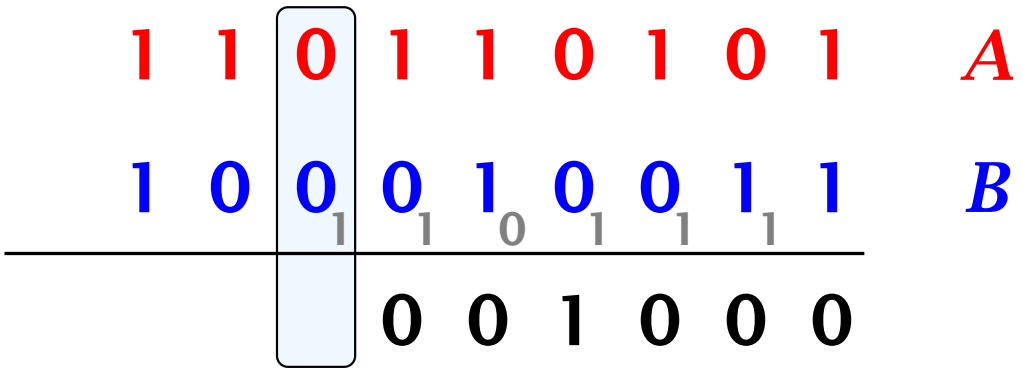
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

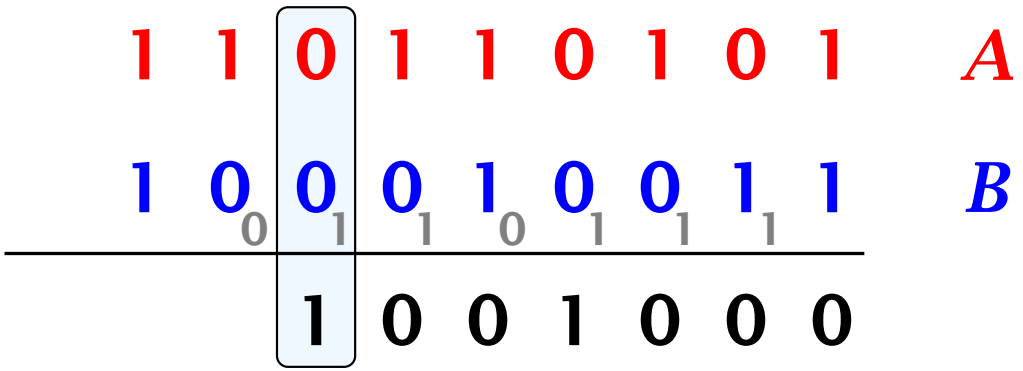
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

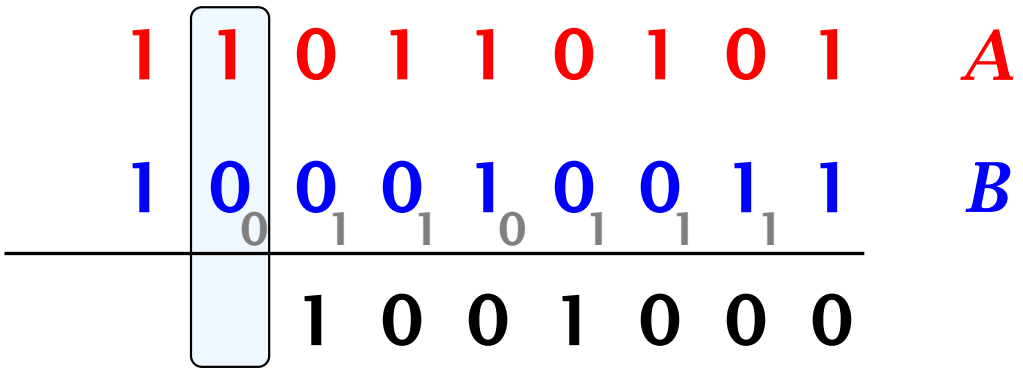
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

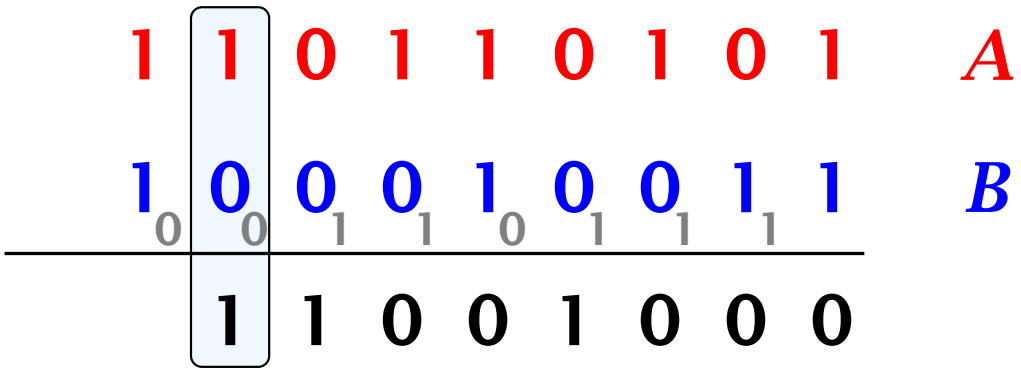
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

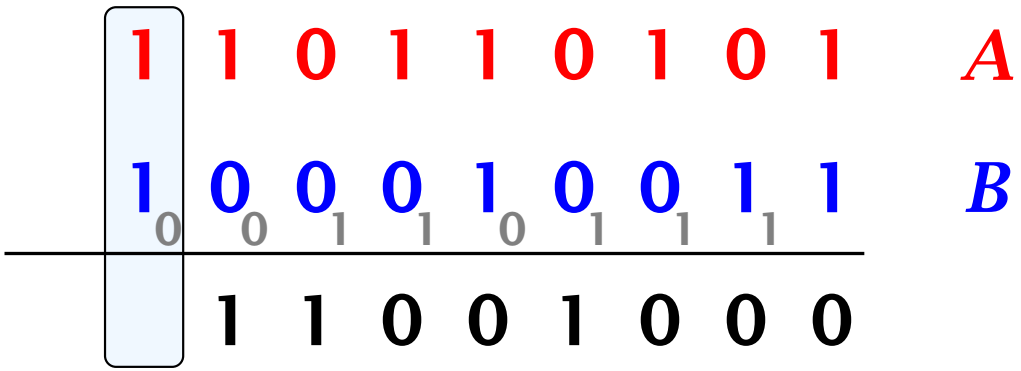
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

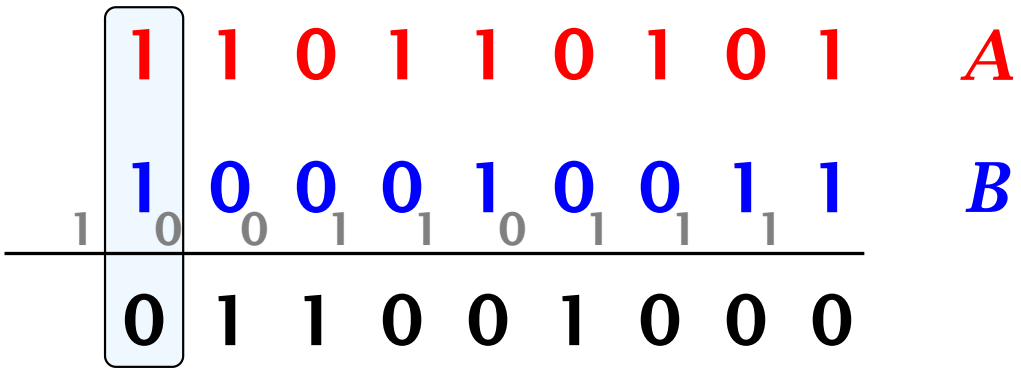
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

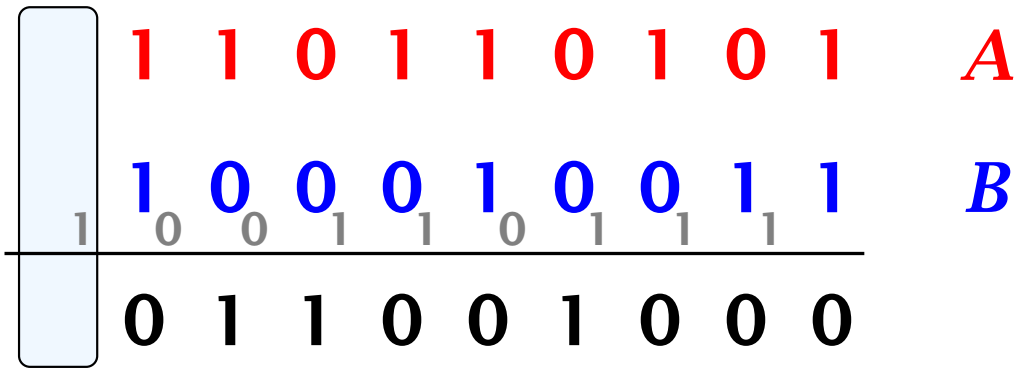
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

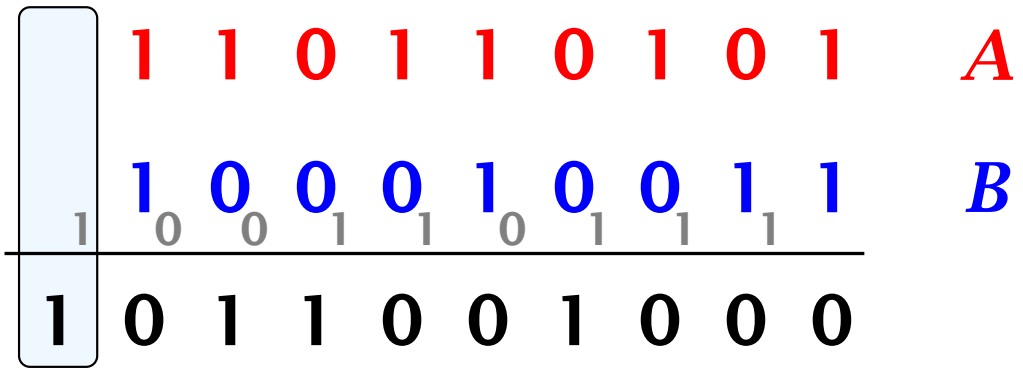
For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

For this we first need to be able to add two integers A and B:



Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

For this we first need to be able to add two integers A and B:

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

This gives that two n-bit integers can be added in time $\mathcal{O}(n)$.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Time requirement:

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Time requirement:

- Computing intermediate results: $\mathcal{O}(nm)$.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

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

- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Time requirement:

- Computing intermediate results: $\mathcal{O}(nm)$.
- Adding m numbers of length $\leq 2n$: $\mathcal{O}((m+n)m) = \mathcal{O}(nm)$.

Example: Multiplying Two Integers

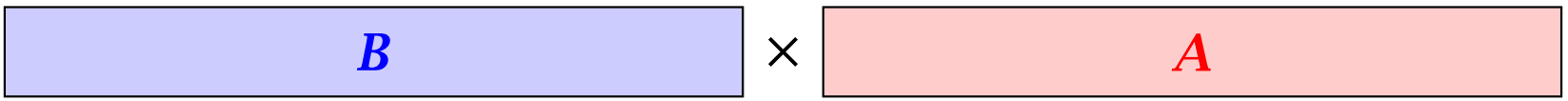
A recursive approach:

Suppose that integers A and B are of length $n = 2^k$, for some k.

Example: Multiplying Two Integers

A recursive approach:

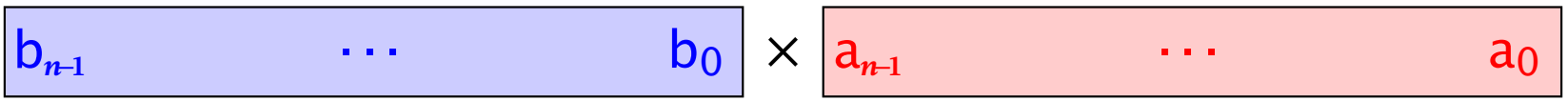
Suppose that integers A and B are of length $n = 2^k$, for some k.



Example: Multiplying Two Integers

A recursive approach:

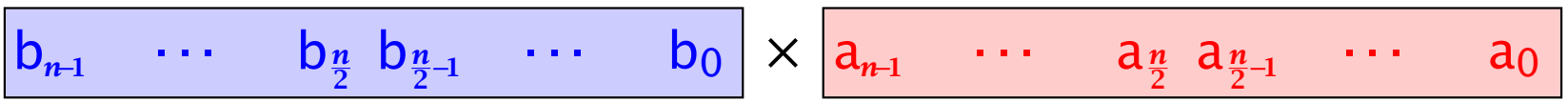
Suppose that integers A and B are of length $n = 2^k$, for some k.



Example: Multiplying Two Integers

A recursive approach:

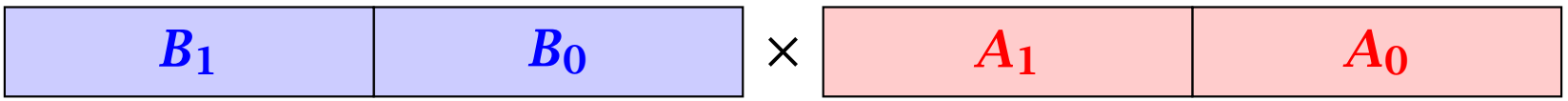
Suppose that integers A and B are of length $n = 2^k$, for some k.



Example: Multiplying Two Integers

A recursive approach:

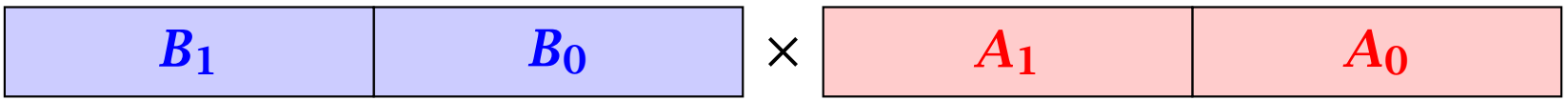
Suppose that integers A and B are of length $n = 2^k$, for some k.



Example: Multiplying Two Integers

A recursive approach:

Suppose that integers A and B are of length $n = 2^k$, for some k.



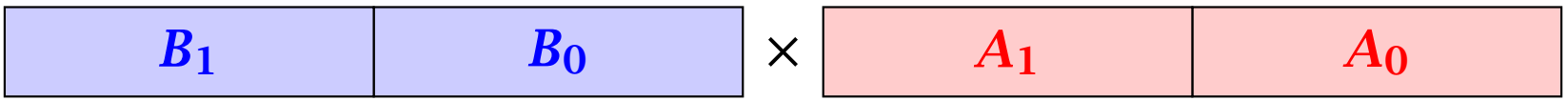
Then it holds that

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

Example: Multiplying Two Integers

A recursive approach:

Suppose that integers A and B are of length $n = 2^k$, for some k.



Then it holds that

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

Hence,

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

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1: **if** $|A| = |B| = 1$ **then**
2: **return** $a_0 \cdot b_0$
3: split A into A_0 and A_1
4: split B into B_0 and B_1
5: $Z_2 \leftarrow \text{mult}(A_1, B_1)$
6: $Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$
7: $Z_0 \leftarrow \text{mult}(A_0, B_0)$
8: **return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1: **if** $|A| = |B| = 1$ **then** $\mathcal{O}(1)$
2: **return** $a_0 \cdot b_0$
3: split A into A_0 and A_1
4: split B into B_0 and B_1
5: $Z_2 \leftarrow \text{mult}(A_1, B_1)$
6: $Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$
7: $Z_0 \leftarrow \text{mult}(A_0, B_0)$
8: **return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

```
1: if |A| = |B| = 1 then                                   O(1)
2:         return a_0 · b_0                                O(1)
3: split A into A_0 and A_1
4: split B into B_0 and B_1
5: Z_2 ← mult(A_1, B_1)
6: Z_1 ← mult(A_1, B_0) + mult(A_0, B_1)
7: Z_0 ← mult(A_0, B_0)
8: return Z_2 · 2^n + Z_1 · 2^(n/2) + Z_0
```

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

```
1: if |A| = |B| = 1 then                                   O(1)
2:         return a_0 · b_0                                O(1)
3: split A into A_0 and A_1                                O(n)
4: split B into B_0 and B_1
5: Z_2 ← mult(A_1, B_1)
6: Z_1 ← mult(A_1, B_0) + mult(A_0, B_1)
7: Z_0 ← mult(A_0, B_0)
8: return Z_2 · 2^n + Z_1 · 2^{n/2} + Z_0
```

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1:	**if** $\|A\| = \|B\| = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	
6:	$Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	
7:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1:	**if** $\|A\| = \|B\| = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	
7:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	$2T(\frac{n}{2}) + \mathcal{O}(n)$
7:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1:	**if** $\|A\| = \|B\| = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	$2T(\frac{n}{2}) + \mathcal{O}(n)$
7:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1:	**if** $\|A\| = \|B\| = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	$2T(\frac{n}{2}) + \mathcal{O}(n)$
7:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	$2T(\frac{n}{2}) + \mathcal{O}(n)$
7:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

We get the following recurrence:

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

Example: Multiplying Two Integers

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

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

Example: Multiplying Two Integers

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

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

In our case $a = 4$, $b = 2$, and $f(n) = \Theta(n)$. Hence, we are in Case 1, since $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$.

Example: Multiplying Two Integers

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

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

In our case $a = 4$, $b = 2$, and $f(n) = \Theta(n)$. Hence, we are in Case 1, since $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$.

We get a running time of $\mathcal{O}(n^2)$ for our algorithm.

Example: Multiplying Two Integers

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

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

In our case $a = 4$, $b = 2$, and $f(n) = \Theta(n)$. Hence, we are in Case 1, since $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$.

We get a running time of $\mathcal{O}(n^2)$ for our algorithm.

$\Longrightarrow$ Not better then the "school method".

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2}+1)+\mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

$$Z_1 = A_1B_0 + A_0B_1$$

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2}+1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

Hence,

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

Hence,

Algorithm 4 mult(A, B)

1: **if** $|A| = |B| = 1$ **then**
2: **return** $a_0 \cdot b_0$
3: split A into A_0 and A_1
4: split B into B_0 and B_1
5: $Z_2 \leftarrow \text{mult}(A_1, B_1)$
6: $Z_0 \leftarrow \text{mult}(A_0, B_0)$
7: $Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$
8: **return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

Hence,

Algorithm 4 mult(A, B)

1: **if** $|A| = |B| = 1$ **then** $\mathcal{O}(1)$
2: **return** $a_0 \cdot b_0$
3: split A into A_0 and A_1
4: split B into B_0 and B_1
5: $Z_2 \leftarrow \text{mult}(A_1, B_1)$
6: $Z_0 \leftarrow \text{mult}(A_0, B_0)$
7: $Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$
8: **return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

$$Z_1 = A_1B_0 + A_0B_1 \qquad \overbrace{}^{=Z_2} \qquad \overbrace{}^{=Z_0}$$
$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,

Algorithm 4 mult(A, B)

```
1: if |A| = |B| = 1 then                                   O(1)
2:         return a_0 · b_0                                O(1)
3: split A into A_0 and A_1
4: split B into B_0 and B_1
5: Z_2 ← mult(A_1, B_1)
6: Z_0 ← mult(A_0, B_0)
7: Z_1 ← mult(A_0 + A_1, B_0 + B_1) − Z_2 − Z_0
8: return Z_2 · 2^n + Z_1 · 2^(n/2) + Z_0
```

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

$$Z_1 = A_1B_0 + A_0B_1 \qquad \overbrace{}^{=Z_2} \qquad \overbrace{}^{=Z_0}$$
$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,

Algorithm 4 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	
6:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	
7:	$Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$	
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

Hence,

Algorithm 4 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	
6:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	
7:	$Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$	
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

Hence,

Algorithm 4 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	
7:	$Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$	
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

Hence,

Algorithm 4 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
7:	$Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$	
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

Hence,

Algorithm 4 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
7:	$Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$	$T(\frac{n}{2}) + \mathcal{O}(n)$
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

$$Z_1 = A_1B_0 + A_0B_1 \qquad \overbrace{}^{= Z_2} \quad \overbrace{}^{= Z_0}$$
$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,

Algorithm 4 mult(A, B)

1:	**if** $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
2:	**return** $a_0 \cdot b_0$	$\mathcal{O}(1)$
3:	split A into A_0 and A_1	$\mathcal{O}(n)$
4:	split B into B_0 and B_1	$\mathcal{O}(n)$
5:	$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
6:	$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
7:	$Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$	$T(\frac{n}{2}) + \mathcal{O}(n)$
8:	**return** $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We get the following recurrence:

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

Example: Multiplying Two Integers

We get the following recurrence:

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

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

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

Example: Multiplying Two Integers

We get the following recurrence:

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

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

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

Again we are in Case 1. We get a running time of $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$.

Example: Multiplying Two Integers

We get the following recurrence:

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

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

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

Again we are in Case 1. We get a running time of $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$.

A huge improvement over the "school method".