6.2 Master Theorem

Lemma 1

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

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

Case 1.

If $f(n) = \Theta(n^{\log_b(a)-\epsilon})$ 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 $n$
a $af\left(\frac{n}{b}\right) \leq c f(n)$ for some constant $c < 1$ then $T(n) = \Theta(f(n))$.
We prove the Master Theorem for the case that $n$ is of the form $b^\ell$, 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:

\[ n \]
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:

![Recursion Tree Diagram]

6.2 Master Theorem
The Recursion Tree

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

\[ f(n) \]

\[ af\left(\frac{n}{b}\right) \]

\[ a^2 f\left(\frac{n}{b^2}\right) \]
The Recursion Tree

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

\[
\begin{align*}
    f(n) & \quad \text{root of the tree} \\
    af\left(\frac{n}{b}\right) & \quad \text{second level} \\
    a^2f\left(\frac{n}{b^2}\right) & \quad \text{third level} \\
    \vdots & \\
    a^{\log_b n} & \quad \text{leaves of the tree}
\end{align*}
\]
The Recursion Tree

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

\[
\begin{align*}
\text{n} & \quad a \\
\text{n/b} & \quad a \text{f(n/b)} \\
\text{n/b^2} & \quad a^2 \text{f(n/b^2)} \\
\vdots & \quad \vdots \\
1 & = n^{\log_b a}
\end{align*}
\]
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 \( f(n) \leq cn^{\log_b a - \epsilon} \).

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

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

Hence,

\[
T(n) \leq (c b^{\epsilon - 1} + 1) n^{\log_b a - \epsilon}
\]
Case 1. Now suppose that $f(n) \leq cn^{\log_b a - \epsilon}$.

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

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

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

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

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

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

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

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

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

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

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

$$\sum_{i=0}^{k} q^i = \frac{q^{k+1} - 1}{q - 1}$$
Case 1. Now suppose that \( f(n) \leq cn^{\log_b a - \epsilon} \).

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

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

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

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

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

\[
= cn^{\log_b a - \epsilon} (b^\epsilon \log_b n - 1) / (b^\epsilon - 1)
\]
Case 1. Now suppose that \( f(n) \leq cn^{\log_b a - \epsilon} \).

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

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

\[
\sum_{i=0}^{k} q^i = \frac{q^{k+1} - 1}{q - 1} = c n^{\log_b a - \epsilon} (\frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1})
\]

\[
= c n^{\log_b a - \epsilon} (n^\epsilon - 1) / (b^\epsilon - 1)
\]
Case 1. Now suppose that \( f(n) \leq cn^{\log_b a - \epsilon} \).

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

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

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

\[
= c n^{\log_b a - \epsilon} \left( b^{\epsilon \log_b n - 1} \right) / (b^{\epsilon} - 1)
= c n^{\log_b a - \epsilon} (n^{\epsilon} - 1) / (b^{\epsilon} - 1)
= \frac{c}{b^{\epsilon} - 1} n^{\log_b a} (n^{\epsilon} - 1) / (n^{\epsilon})
\]
Case 1. Now suppose that $f(n) \leq cn^{\log_b a - \epsilon}$.

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

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

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

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

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

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

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

Hence,

$$T(n) \leq \left(\frac{c}{b^{\epsilon} - 1} + 1\right) n^{\log_b a}$$
Case 1. Now suppose that \( f(n) \leq cn^{\log_b a - \epsilon} \).

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

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

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

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 \( f(n) \leq cn^{\log_b a} \).

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

Hence, \( T(n) = O(n^{\log_b a} \log_b n) \).
Case 2. Now suppose that \( f(n) \leq cn^{\log_b a} \).

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right) \\
\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
= cn^{\log_b a} \sum_{i=0}^{\log_b n-1} 1
\]
Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

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

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

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

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

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

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

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

\[
= cn^{\log_b a} \log_b n
\]

Hence,

\[
T(n) = O(n^{\log_b a} \log_b n)
\]
Case 2. Now suppose that $f(n) \leq cn^{\log_b a}$.

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

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

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

$$= cn^{\log_b a} \log_b n$$

Hence,

$$T(n) = O(n^{\log_b a} \log_b n) \Rightarrow T(n) = O(n^{\log_b a} \log n).$$
Case 2. Now suppose that $f(n) \geq cn^{\log_b a}$.
Case 2. Now suppose that \( f(n) \geq cn^{\log_b a} \).

\[
T(n) - n^{\log_b a}
\]
Case 2. Now suppose that \( f(n) \geq cn^{\log_b a} \).

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)
\]
Case 2. Now suppose that $f(n) \geq cn^{\log_b a}$.

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

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)
\geq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a}
= c n^{\log_b a} \sum_{i=0}^{\log_b n-1} 1
= c n^{\log_b a} \log_b n
\]
Case 2. Now suppose that $f(n) \geq cn^\log_b a$.

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

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

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

\[
= cn^\log_b a \log_b n
\]

Hence,

\[
T(n) = \Omega(n^\log_b a \log_b n)
\]
Case 2. Now suppose that $f(n) \geq cn^{\log_b a}$.

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

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

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

$$= c n^{\log_b a} \log_b n$$

Hence,

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

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

6.2 Master Theorem
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 \( f(n) \leq cn^{\log_b a} (\log_b (n))^k \).

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

\[
\leq c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b \left(\frac{n}{b^i}\right)\right)^k
\]
Case 2. Now suppose that $f(n) \leq cn^{\log_b a} (\log_b(n))^k$.

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

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

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

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

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

\[
n = b^\ell \Rightarrow \ell = \log_b n
\]

\[
= cn^{\log_b a} \sum_{i=0}^{\ell-1} \left( \log_b \left( \frac{b^\ell}{b^i} \right) \right)^k
\]
Case 2. Now suppose that $f(n) \leq cn^{\log_b a} (\log_b (n))^k$. 

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

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

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

\[ = cn^{\log_b a} \sum_{i=0}^{\ell-1} (\ell - i)^k \]
Case 2. Now suppose that \( f(n) \leq c n^{\log_b a} (\log_b(n))^k \).

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

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

\[
n = b^\ell \Rightarrow \ell = \log_b n
\]

\[
= c n^{\log_b a} \sum_{i=0}^{\ell - 1} \left(\log_b \left(\frac{b^\ell}{b^i}\right)\right)^k
\]

\[
= c n^{\log_b a} \sum_{i=0}^{\ell - 1} (\ell - i)^k
\]

\[
= c n^{\log_b a} \sum_{i=1}^{\ell} i^k
\]
Case 2. Now suppose that \( f(n) \leq cn^{\log_b a} (\log_b(n))^k \).

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

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

\[
n = b^\ell \Rightarrow \ell = \log_b n
\]

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

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

\[
= cn^{\log_b a} \sum_{i=1}^{\ell} i^k \approx \frac{1}{k} \ell^{k+1}
\]
Case 2. Now suppose that \( f(n) \leq cn^{\log_b a (\log_b (n))^k} \).

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

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

\[
n = b^\ell \Rightarrow \ell = \log_b n
\]

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

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

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

\[
\approx \frac{c}{k} n^{\log_b a} \ell^{k+1}
\]
Case 2. Now suppose that $f(n) \leq cn^{\log_b a (\log_b(n))^k}$.

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

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

\[
n = b^\ell \Rightarrow \ell = \log_b n
\]

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

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

\[
= cn^{\log_b a} \sum_{i=0}^{\ell} i^k
\]

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

\[
\Rightarrow T(n) = \Theta(n^{\log_b a \log^{k+1} n}).
\]
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 $c < 1$. 

Hence, $T(n) \leq O(f(n))$.

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

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

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
\leq \sum_{i=0}^{\log_b n - 1} c^i f(n) + \mathcal{O}(n^{\log_b a})
\]
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.

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

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

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

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

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

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

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)
\leq \sum_{i=0}^{\log_b n - 1} c^i f(n) + O(n^{\log_b a})
\leq \frac{1}{1 - c} f(n) + O(n^{\log_b a})
\]

Hence,

\[
T(n) \leq O\left(f(n)\right)
\]
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.

\[
\begin{align*}
T(n) - n^{\log_b a} & = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\
& \leq \sum_{i=0}^{\log_b n - 1} c^i f(n) + O(n^{\log_b a}) \\
\end{align*}
\]

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

Hence,

\[
T(n) \leq O(f(n)) \quad \Rightarrow \quad T(n) = \Theta(f(n)).
\]
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 $B$:

\[
\begin{array}{c}
1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \\
\hline
1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1
\end{array}
\]

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

6.2 Master Theorem
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 \( B \):

\[
\begin{array}{c}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]
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 $B$:

\[
\begin{array}{cccccccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
\hline
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1
\end{array}
\]

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

\[6.2 \text{ Master Theorem}\]
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 $B$:

\[
\begin{array}{c}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & A \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & B \\
\end{array}
\]

\[
\begin{array}{c}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 \\
\end{array}
\]
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 $B$:

\[
\begin{array}{cccccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}
\]

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

6.2 Master Theorem
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 \( B \):

\[
\begin{array}{c}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & A \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & B \\
\end{array}
\]

\[
\begin{array}{c}
0 & 0 \\
\end{array}
\]
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 $B$:

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\hline
0 & 0 & 0 & 0
\end{array}
\]

This gives that two $n$-bit integers can be added in time $O(n)$. 
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 \( B \):

\[
\begin{array}{c}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

This gives that two \( n \)-bit integers can be added in time \( O(n) \).
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 $B$:

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
\]

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

6.2 Master Theorem
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 $B$:

\[
\begin{array}{cccc}
1&1&0&1 \\
1&0&0&0
\end{array}
\begin{array}{cccc}
1&0&1&0 \\
1&0&0&1
\end{array}
\begin{array}{cccc}
A \\
B
\end{array}
\]

\[
\begin{array}{cccc}
1&0&0&0
\end{array}
\begin{array}{cccc}
0&1&1&1
\end{array}
\begin{array}{cccc}
1&0&0&0
\end{array}
\]
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 $B$:

\[
\begin{array}{c}
1 & 1 & 0 & 1 \\
& & & \\
1 & 0 & 0 & 0
\end{array}
\quad
\begin{array}{c}
1 & 0 & 1 & 0 & 1 \\
& & & \\
1 & 0 & 0 & 1 & 1
\end{array}
\quad
A
\quad
B
\]

\[
\begin{array}{c}
0 & 1 & 0 & 0 & 0
\end{array}
\]

This gives that two $n$-bit integers can be added in time $O(n)$.
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 $B$:

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & A \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & B \\
\hline
0 & 1 & 0 & 0 & 0 & 0
\end{array}
\]
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 $B$:

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
\hline
0 & 0 & 1 & 0 & 0 & 0 &
\end{array}
\]
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 $B$:

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & A \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & B \\
\hline
0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}
\]
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 $B$:

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]
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 $B$: 

\[
\begin{array}{c}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]
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 \( B \):

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
\hline
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]

This gives that two \( n \)-bit integers can be added in time \( O(n) \).
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 \( B \):

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]

This gives that two \( n \)-bit integers can be added in time \( O(n) \).
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 $B$:

\[
\begin{array}{ccccccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]
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 \( B \):

\[
\begin{array}{cccccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]

This gives that two \( n \)-bit integers can be added in time \( O(n) \).
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 $B$:

\[
\begin{array}{c}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & A \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & B \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]

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

6.2 Master Theorem
Example: Multiplying Two Integers

Suppose that we want to multiply an \( n \)-bit integer \( A \) and an \( m \)-bit integer \( B \) \((m \leq n)\).
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
1 & 0 & 0 & 0 & 1 \\
\times & 1 & 0 & 1 & 1
\end{array}
\]

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 & 1 \\
\times & & 1 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 1
\end{array}
\]
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
1 & 0 & 0 & 0 & 1 \\
\times & & & 1 & 0 & 1 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

Time requirement:

- Computing intermediate results: \( O(nm) \).
- Adding \( m \) numbers of length \( \leq 2n \): \( O((m + n)m) = O(nm) \).
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
1 \ 0 \ 0 \ 0 \ 1 \\
\times \\
1 \ 0 \ \boxed{1} \ 1
\end{array}
\]

\[
\begin{array}{c}
1 \ 0 \ 0 \ 0 \ 0 \ 1
\end{array}
\]

Time requirement:

- Computing intermediate results: \( O(\ell^m) \).
- Adding \( m \) numbers of length \( \leq 2n \): \( O((m + n)m) = O(nm) \).

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 & 1 \\
\times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 \\
0 &
\end{array}
\]

Time requirement:

\begin{itemize}
  \item Computing intermediate results: $O(nm)$.
  \item Adding $m$ numbers of length $\leq 2n$: $O((m+n)m) = O(nm)$.
\end{itemize}
Example: Multiplying Two Integers

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

\[
\begin{array}{ccccccc}
1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0
\end{array}
\]

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

6.2 Master Theorem

Ernst Mayr, Harald Räcke
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
  1 & 0 & 0 & 0 & 1 \\
\times & \begin{array}{c} 1 \ \ 0 \ \ 1 \ \ 1 \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
  1 & 0 & 0 & 0 & 1 \\
  \hline
  1 & 0 & 0 & 0 & 1 \\
  1 & 0 & 0 & 0 & 1 \ 0
\end{array}
\]

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{cc}
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \end{array} \times \begin{array}{cc}
1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 & 1 \\
\times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

Time requirement:

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 & 1 \\
\times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Time requirement:
- Computing intermediate results: \( O(nm) \).
- Adding \( m \) numbers of length \( \leq 2n \): \( O((m+n)m) = O(nm) \).

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 & 1 \\
\times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
1 & 0 & 0 & 0 & 1 \\
\times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
\times & 1 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]

Time requirement:

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
1 \ 0 \ 0 \ 0 \ 1 \\
\times \\
1 \ 0 \ 1 \ 1 \\
\hline
1 \ 0 \ 0 \ 0 \ 1 \\
1 \ 0 \ 0 \ 0 \ 1 \ 0 \\
0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \\
\hline
1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \\
\end{array}
\]

Time requirement:
- Computing intermediate results: \( O(nm) \).
- Adding \( m \) numbers of length \( \leq 2n \): \( O((m+n)m) = O(nm) \).

6.2 Master Theorem

Ernst Mayr, Harald Räcke
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
1 \quad 0 \quad 0 \quad 0 \quad 1 \\
\times \quad 1 \quad 0 \quad 1 \quad 1 \\
\hline
1 \quad 0 \quad 0 \quad 0 \quad 1 \\
1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \\
0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \\
\hline
1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 
\end{array}
\]

Time requirement:
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
\times & 1 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1
\end{array}
\]

Time requirement:
- Computing intermediate results: \( \Theta(nm) \).
Example: Multiplying Two Integers

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 & 1 \\
\times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

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 $k$. 

$B \times A$
Example: Multiplying Two Integers

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

\[
\begin{array}{c}
\begin{array}{c}
b_{n-1} \\
\cdots \\
b_0
\end{array}
\end{array}
\times
\begin{array}{c}
\begin{array}{c}
a_{n-1} \\
\cdots \\
a_0
\end{array}
\end{array}\]
Example: Multiplying Two Integers

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

$$
\begin{array}{c}
\text{b}_{n-1} \cdots \text{b}_n \text{b}_{n-1} \cdots \text{b}_0 \\
\text{a}_{n-1} \cdots \text{a}_n \text{a}_{n-1} \cdots \text{a}_0
\end{array}
$$
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 $A = A_1 \cdot 2^{n_2} + A_0$ and $B = B_1 \cdot 2^{n_2} + B_0$.
Hence, $A \cdot B = A_1 B_1 \cdot 2^{n_2} + (A_1 B_0 + A_0 B_1) \cdot 2^{n_2} + A_0 B_0$. 

$\square$
A recursive approach:
Suppose that integers $A$ and $B$ are of length $n = 2^k$, for some $k$.

Then it holds that

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

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

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 $\text{mult}(A, B)$

1: if $|A| = |B| = 1$ then
2: \hspace{1em} 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 $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

We get the following recurrence:

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n).$$
Algorithm 3 \text{mult}(A, B)
\begin{align*}
1: & \text{ if } |A| = |B| = 1 \text{ then} \\
2: & \quad \text{return } a_0 \cdot b_0 \\
3: & \text{ split } A \text{ into } A_0 \text{ and } A_1 \\
4: & \text{ split } B \text{ into } B_0 \text{ 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: & \text{return } Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0
\end{align*}

\(O(1)\)
Example: Multiplying Two Integers

Algorithm 3 $\text{mult}(A, B)$

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 $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

\(\mathcal{O}(1)\)

We get the following recurrence:

\(T(n) = 4T\left(\frac{n}{2}\right) + \mathcal{O}(n)\)
Example: Multiplying Two Integers

Algorithm 3 \mult(A, B)  

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

$O(1)$

$O(1)$

$O(n)$

We get the following recurrence:

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

6.2 Master Theorem
Example: Multiplying Two Integers

Algorithm 3 $\text{mult}(A, B)$

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 $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

$O(1)$

$O(1)$

$O(n)$

$O(n)$
Example: Multiplying Two Integers

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

We get the following recurrence:

\[ T(n) = 4 \cdot T\left(\frac{n}{2}\right) + \mathcal{O}(n) \]
Example: Multiplying Two Integers

Algorithm 3 \( \text{mult}(A, B) \)

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

\( \mathcal{O}(1) \)
\( \mathcal{O}(1) \)
\( \mathcal{O}(n) \)
\( \mathcal{O}(n) \)
\( T(\frac{n}{2}) \)
\( 2T(\frac{n}{2}) + \mathcal{O}(n) \)

6.2 Master Theorem
Algorithm 3 \texttt{mult}(A, B)

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

We get the following recurrence:

\[ T(n) = 4 \mathcal{T}(n^{2}) + \mathcal{O}(n) \]
Example: Multiplying Two Integers

Algorithm 3 \text{mult}(A, B)

1: \textbf{if } |A| = |B| = 1 \textbf{ then} \hfill \mathcal{O}(1)
2: \hspace{1em} \textbf{return } a_0 \cdot b_0 \hfill \mathcal{O}(1)
3: \text{split } A \text{ into } A_0 \text{ and } A_1 \hfill \mathcal{O}(n)
4: \text{split } B \text{ into } B_0 \text{ and } B_1 \hfill \mathcal{O}(n)
5: Z_2 \leftarrow \text{mult}(A_1, B_1) \hfill T\left(\frac{n}{2}\right)
6: Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1) \hfill 2T\left(\frac{n}{2}\right) + \mathcal{O}(n)
7: Z_0 \leftarrow \text{mult}(A_0, B_0) \hfill T\left(\frac{n}{2}\right)
8: \textbf{return } Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0 \hfill \mathcal{O}(n)
Example: Multiplying Two Integers

Algorithm 3 $\text{mult}(A, B)$

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 $Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0$

We get the following recurrence:

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

6.2 Master Theorem
Example: Multiplying Two Integers

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

- **Case 1:** \( f(n) = \Theta(n^{\log_b a - \epsilon}) \Rightarrow T(n) = \Theta(n^{\log_b a}) \)
- **Case 2:** \( f(n) = \Theta(n^{\log_b a \log^k n}) \Rightarrow T(n) = \Theta(n^{\log_b a \log^{k+1} n}) \)
- **Case 3:** \( f(n) = \Omega(n^{\log_b a + \epsilon}) \Rightarrow T(n) = \Theta(f(n)) \)
Example: Multiplying Two Integers

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

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

In our case \( a = 4, b = 2 \), and \( f(n) = \Theta(n) \). Hence, we are in Case 1, since \( n = \Theta(n^{2-\epsilon}) = \Theta(n^{\log_b a - \epsilon}) \).
Example: Multiplying Two Integers

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

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

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

We get a running time of $\Theta(n^2)$ for our algorithm.
Example: Multiplying Two Integers

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

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

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

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

\( \Rightarrow \) Not better then the “school method”.

---

6.2 Master Theorem

Ernst Mayr, Harald Räcke
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1 B_0 + A_0 B_1 = (A_0 + A_1) (B_0 + B_1) - A_1 B_1 - A_0 B_0$$

Hence,

Algorithm 4 $\text{mult}(A, B)$

1. if $|A| = |B| = 1$
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 $Z_2 \cdot 2^n + Z_1 \cdot 2^{n-1} + Z_0$
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1B_0 + A_0B_1$$
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1B_0 + A_0B_1$$

$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

\[
Z_1 = A_1B_0 + A_0B_1 = Z_2 = Z_0
\]

\[
= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0
\]
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1B_0 + A_0B_1 \quad \Rightarrow \quad Z_2 = Z_0$$

$$\Rightarrow \quad (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

\[
Z_1 = A_1B_0 + A_0B_1 = Z_2 = Z_0
\]

\[
= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0
\]

Hence,

```
Algorithm 4 mult(A, B)
1: if |A| = |B| = 1 then
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: $Z_2 \leftarrow$ mult($A_1, B_1$)
6: $Z_0 \leftarrow$ mult($A_0, B_0$)
7: $Z_1 \leftarrow$ mult($A_0 + A_1, B_0 + B_1$) − $Z_2$ − $Z_0$
8: return $Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0$
```
Example: Multiplying Two Integers

We can use the following identity to compute \( Z_1 \):

\[
Z_1 = A_1 B_0 + A_0 B_1 = Z_2 = Z_0
\
= (A_0 + A_1) \cdot (B_0 + B_1) - A_1 B_1 - A_0 B_0
\]

Hence,

<table>
<thead>
<tr>
<th>Algorithm 4 ( \text{mult}(A, B) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: if (</td>
</tr>
<tr>
<td>2: return ( a_0 \cdot b_0 )</td>
</tr>
<tr>
<td>3: split ( A ) into ( A_0 ) and ( A_1 )</td>
</tr>
<tr>
<td>4: split ( B ) into ( B_0 ) and ( B_1 )</td>
</tr>
<tr>
<td>5: ( Z_2 \leftarrow \text{mult}(A_1, B_1) )</td>
</tr>
<tr>
<td>6: ( Z_0 \leftarrow \text{mult}(A_0, B_0) )</td>
</tr>
<tr>
<td>7: ( Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0 )</td>
</tr>
<tr>
<td>8: return ( Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0 )</td>
</tr>
</tbody>
</table>

\[ O(1) \]
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1B_0 + A_0B_1 = Z_2 = Z_0$$

$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,

**Algorithm 4** \texttt{mult}(A, B)

1: if $|A| = |B| = 1$ then \hspace{1cm} $O(1)$
2: \textbf{return} $a_0 \cdot b_0$ \hspace{1cm} $O(1)$
3: split $A$ into $A_0$ and $A_1$
4: split $B$ into $B_0$ and $B_1$
5: $Z_2 \leftarrow \texttt{mult}(A_1, B_1)$
6: $Z_0 \leftarrow \texttt{mult}(A_0, B_0)$
7: $Z_1 \leftarrow \texttt{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$
8: \textbf{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 $T(n^2 + 1) + O(n)$.
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$ Z_1 = A_1 B_0 + A_0 B_1 = Z_2 = Z_0 $$

$$ = (A_0 + A_1) \cdot (B_0 + B_1) - A_1 B_1 - A_0 B_0 $$

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: $Z_2 \leftarrow$ mult($A_1, B_1$)
6: $Z_0 \leftarrow$ mult($A_0, B_0$)
7: $Z_1 \leftarrow$ mult($A_0 + A_1, B_0 + B_1$) − $Z_2$ − $Z_0$
8: return $Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0$
```

$\mathcal{O}(1)$

$\mathcal{O}(1)$

$\mathcal{O}(n)$
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1B_0 + A_0B_1 = Z_2 = Z_0$$

$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,

**Algorithm 4** \texttt{mult}(A, B)

1: if $|A| = |B| = 1$ then $O(1)$
2: return $a_0 \cdot b_0$ $O(1)$
3: split $A$ into $A_0$ and $A_1$ $O(n)$
4: split $B$ into $B_0$ and $B_1$ $O(n)$
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 $T(n^2 + 1) + O(n)$. 6.2 Master Theorem
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

\[
Z_1 = A_1B_0 + A_0B_1 = \underbrace{Z_2}_\text{=} - A_1B_1 - A_0B_0
\]

\[
= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0
\]

Hence,

\[
\text{Algorithm 4 } \text{mult}(A, B)
\]

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: $Z_0 \leftarrow \text{mult}(A_0, B_0)$
7: $Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$
8: return $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

6.2 Master Theorem
**Example: Multiplying Two Integers**

We can use the following identity to compute $Z_1$:

\[
Z_1 = A_1B_0 + A_0B_1 = Z_2 = Z_0
\]

\[
= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0
\]

Hence,

<table>
<thead>
<tr>
<th>Algorithm 4 $\text{mult}(A, B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: if $</td>
</tr>
<tr>
<td>2: return $a_0 \cdot b_0$ $\mathcal{O}(1)$</td>
</tr>
<tr>
<td>3: split $A$ into $A_0$ and $A_1$ $\mathcal{O}(n)$</td>
</tr>
<tr>
<td>4: split $B$ into $B_0$ and $B_1$ $\mathcal{O}(n)$</td>
</tr>
<tr>
<td>5: $Z_2 \leftarrow \text{mult}(A_1, B_1)$ $T\left(\frac{n}{2}\right)$</td>
</tr>
<tr>
<td>6: $Z_0 \leftarrow \text{mult}(A_0, B_0)$ $T\left(\frac{n}{2}\right)$</td>
</tr>
<tr>
<td>7: $Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$ $\mathcal{O}(n)$</td>
</tr>
<tr>
<td>8: return $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$ $\mathcal{O}(n)$</td>
</tr>
</tbody>
</table>

**6.2 Master Theorem**

Ernst Mayr, Harald Räcke
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1B_0 + A_0B_1 = Z_2 = Z_0$$

$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,

```
Algorithm 4 mult(A, B)
1: if |A| = |B| = 1 then
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: $Z_2 \leftarrow$ mult($A_1, B_1$)
6: $Z_0 \leftarrow$ mult($A_0, B_0$)
7: $Z_1 \leftarrow$ mult($A_0 + A_1, B_0 + B_1$) - $Z_2 - Z_0$
8: return $Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0$
```

Algorithm complexity:

$\mathcal{O}(1)$

$\mathcal{O}(n)$

$\mathcal{T}(n^2) + \mathcal{O}(n)$
Example: Multiplying Two Integers

We can use the following identity to compute $Z_1$:

$$Z_1 = A_1B_0 + A_0B_1 = Z_2 = Z_0$$

$$= (A_0 + A_1) \cdot (B_0 + B_1) - A_1B_1 - A_0B_0$$

Hence,

**Algorithm 4** $\text{mult}(A, B)$

1: if $|A| = |B| = 1$ then
2: \quad 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 $Z_2 \cdot 2^n + Z_1 \cdot 2^{n \frac{n}{2}} + Z_0$

$O(1)$

$O(1)$

$O(n)$

$O(n)$

$T(\frac{n}{2})$

$T(\frac{n}{2})$

$T(\frac{n}{2}) + O(n)$

$O(n)$

6.2 Master Theorem
Ernst Mayr, Harald Räcke
Example: Multiplying Two Integers

We get the following recurrence:

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

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

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

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”.

6.2 Master Theorem
Example: Multiplying Two Integers

We get the following recurrence:

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

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

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

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”.

6.2 Master Theorem
Example: Multiplying Two Integers

We get the following recurrence:

\[ T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n). \]

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

- Case 1: \( f(n) = \Theta(n^{\log_b a - \epsilon}) \quad T(n) = \Theta(n^{\log_b a}) \)
- 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”.

6.2 Master Theorem
Example: Multiplying Two Integers

We get the following recurrence:

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

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

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

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”.

6.2 Master Theorem