

## 6.2 Master Theorem

Note that the cases do not cover all possibilities.

### 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  $f(n) = \mathcal{O}(n^{\log_b(a)-\epsilon})$  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  $n$   
 $af\left(\frac{n}{b}\right) \leq cf(n)$  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  $b^{\ell}$ , 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:

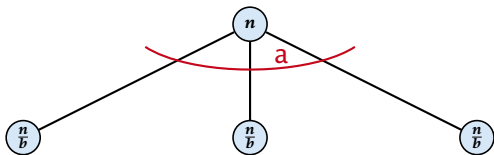
## The Recursion Tree

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



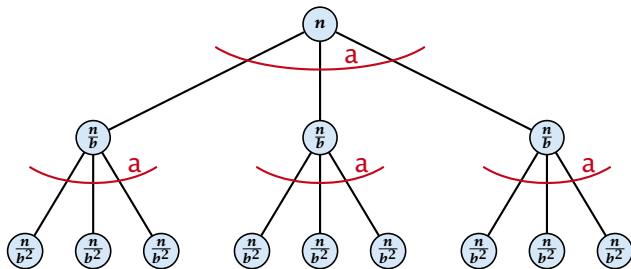
## The Recursion Tree

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



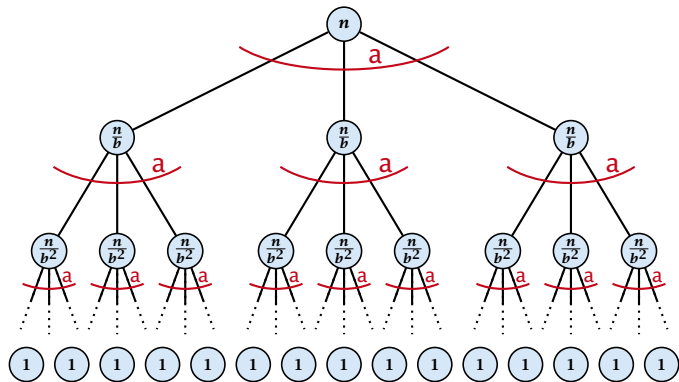
# The Recursion Tree

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



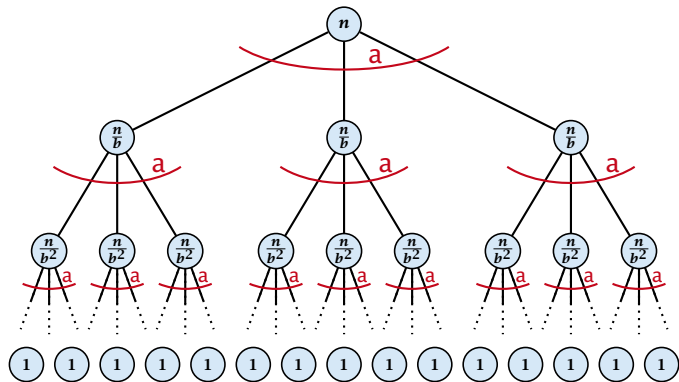
# The Recursion Tree

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



# The Recursion Tree

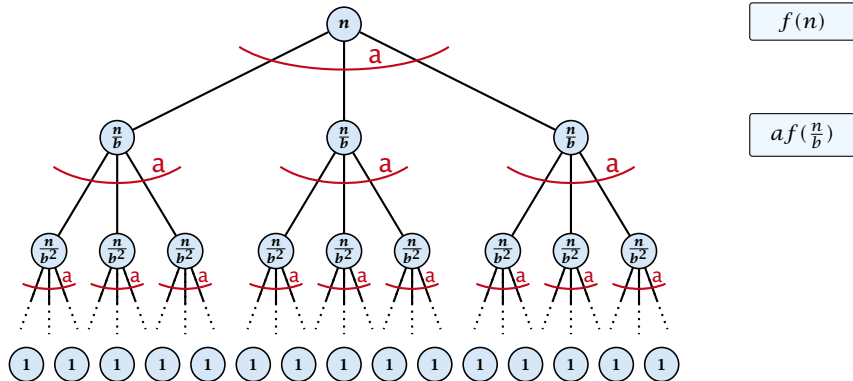
The running time of a recursive algorithm can be visualized by a recursion tree:





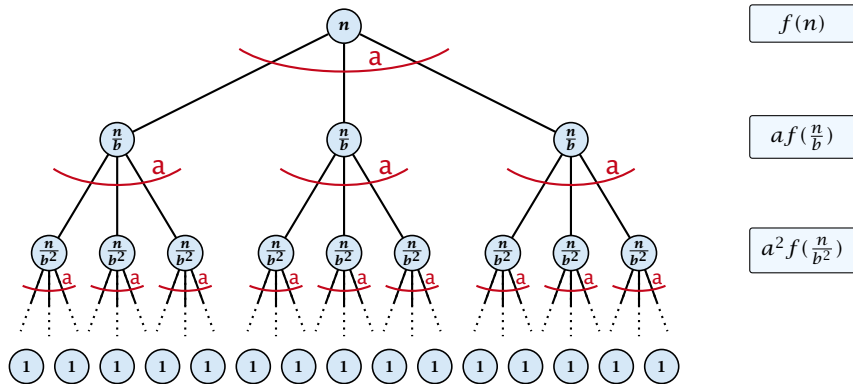
# The Recursion Tree

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



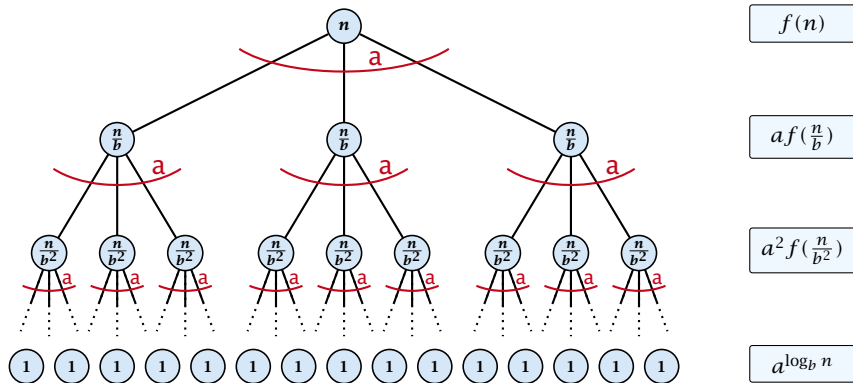
# The Recursion Tree

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



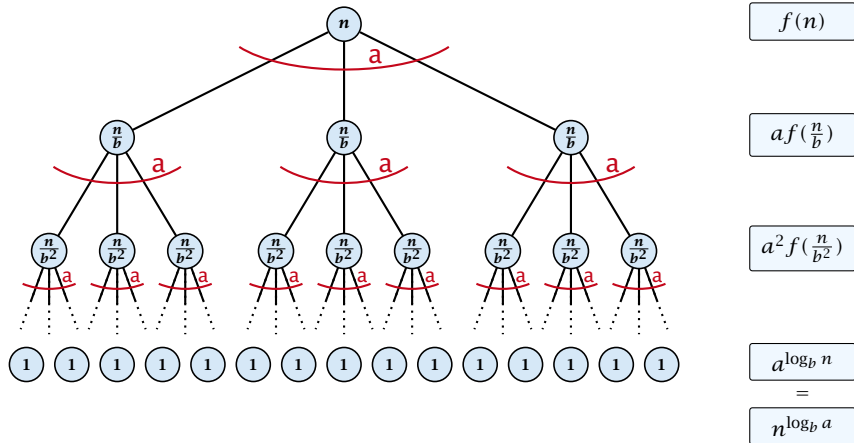
# The Recursion Tree

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:



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\boxed{\sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1}} = cn^{\log_b a - \epsilon} (b^{\epsilon \log_b n} - 1) / (b^{\epsilon} - 1)$$

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

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

$$\begin{aligned}b^{-i(\log_b a - \epsilon)} = b^{\epsilon i} (b^{\log_b a})^{-i} = b^{\epsilon i} a^{-i} &= cn^{\log_b a - \epsilon} \sum_{i=0}^{\log_b n - 1} (b^{\epsilon})^i \\ \sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1} &= cn^{\log_b a - \epsilon} (b^{\epsilon \log_b n} - 1) / (b^{\epsilon} - 1) \\ &= cn^{\log_b a - \epsilon} (n^{\epsilon} - 1) / (b^{\epsilon} - 1)\end{aligned}$$

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

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

$$\begin{aligned}b^{-i(\log_b a - \epsilon)} = b^{\epsilon i} (b^{\log_b a})^{-i} = b^{\epsilon i} a^{-i} &= cn^{\log_b a - \epsilon} \sum_{i=0}^{\log_b n - 1} (b^\epsilon)^i \\ \sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1} &= cn^{\log_b a - \epsilon} (b^{\epsilon \log_b n} - 1) / (b^\epsilon - 1) \\ &= cn^{\log_b a - \epsilon} (n^\epsilon - 1) / (b^\epsilon - 1) \\ &= \frac{c}{b^\epsilon - 1} n^{\log_b a} (n^\epsilon - 1) / (n^\epsilon)\end{aligned}$$

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

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

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

$$\begin{aligned} \boxed{\sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1}} &= cn^{\log_b a - \epsilon} (b^{\epsilon \log_b n} - 1) / (b^{\epsilon} - 1) \\ &= cn^{\log_b a - \epsilon} (n^{\epsilon} - 1) / (b^{\epsilon} - 1) \\ &= \frac{c}{b^{\epsilon} - 1} n^{\log_b a} (n^{\epsilon} - 1) / (n^{\epsilon}) \end{aligned}$$

Hence,

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



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

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

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

$$\begin{aligned} \boxed{\sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1}} &= cn^{\log_b a - \epsilon} (b^{\epsilon \log_b n} - 1) / (b^{\epsilon} - 1) \\ &= cn^{\log_b a - \epsilon} (n^{\epsilon} - 1) / (b^{\epsilon} - 1) \\ &= \frac{c}{b^{\epsilon} - 1} n^{\log_b a} (n^{\epsilon} - 1) / (n^{\epsilon}) \end{aligned}$$

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

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

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

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

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

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

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,

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

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

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

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

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

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

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

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

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

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

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

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

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

Hence,

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

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.

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

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

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.

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

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

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.

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

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

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.

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

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

Hence,

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

$$\Rightarrow T(n) = \Theta(f(n)).$$

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  $B$ :

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









## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>								0	0

The diagram illustrates the addition of two 9-bit integers, A and B. The bits of A are 1, 1, 0, 1, 1, 0, 1, 0, 1. The bits of B are 1, 0, 0, 0, 1, 0, 0, 1, 1. A horizontal line is drawn under the 7th bit of B. A vertical box highlights the 8th and 9th bits of both numbers, which are 0 and 1 for A, and 1 and 1 for B. Below the line, the result of the addition for these two bits is shown as 0 and 0.

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
						0	0		

The diagram illustrates the addition of two integers, A and B, using a register of constant size. The integers are represented as binary strings: A = 110110101 and B = 100010011. The addition is performed bit-by-bit, with a carry bit (1) being passed to the next higher bit position. The result of the addition is shown as 00 in the final column, indicating that the carry bit is 0.

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>							0	0	0

The diagram illustrates the addition of two 9-bit integers, A and B. The bits of A are 1, 1, 0, 1, 1, 0, 1, 0, 1. The bits of B are 1, 0, 0, 0, 1, 0, 0, 1, 1. A horizontal line is drawn under the bits of B. The result of the addition is shown below the line, with the bits 0, 0, 0. A vertical box highlights the bits 1, 0, 1 of A and the bits 0, 1, 1 of B, which are the bits that are added to produce the result. Small '1' characters are placed below the 6th, 7th, and 8th bits of B, indicating carry bits.

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
						0	0	0	

The diagram illustrates the addition of two integers, A and B, using a register of constant size. The integers are represented as binary strings: A = 110110101 and B = 100010011. The addition is performed bit-by-bit, with carry bits (1) shown below the digits. The result of the addition is 000, indicating that the sum of A and B is zero. A vertical box highlights the 5th bit (0) of A and the 5th bit (0) of B, which are the bits being added in the current step.

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
					0	1	1	1	
					1	0	0	0	

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
					0	1	1	1	
						1	0	0	0



## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
				0	1	0	0	0	

Carry bits: 1, 0, 1, 1, 1

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
				1	0	1	1	1	
				0	1	0	0	0	

The diagram illustrates the addition of two 9-bit integers, A and B. A vertical box highlights the carry propagation from the 4th bit to the 5th bit. The carry bits are shown below the horizontal line.

## 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  $B$ :

1	1	0	1	1	0	1	0	1		$A$
1	0	0	0	1	0	0	1	1		$B$
			0	0	1	0	0	0		

Carry bits:  $\underset{1}{\phantom{0}} \underset{1}{\phantom{0}} \underset{0}{\phantom{0}} \underset{1}{\phantom{0}} \underset{1}{\phantom{0}} \underset{1}{\phantom{0}}$

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
			0	0	1	0	0	0	

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
		1	0	0	1	0	0	0	

The diagram illustrates the addition of two 9-bit integers, A and B. The bits of A are 1, 1, 0, 1, 1, 0, 1, 0, 1. The bits of B are 1, 0, 0, 0, 1, 0, 0, 1, 1. A vertical box highlights the third bit (0) of A and the third bit (0) of B. Below the horizontal line, the result of the addition is shown: 1, 0, 0, 1, 0, 0, 0. Small subscripts are placed below the bits of B: 0, 1, 1, 0, 1, 1, 1.

## 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
		1	0	0	1	0	0	0	

*Note: A light blue vertical box highlights the first two bits of A and B. Below the box, the carry bits 0, 1, 1, 0, 1, 1, 1 are written.*

## 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  $B$ :

	1	1	0	1	1	0	1	0	1	$A$
	1	0	0	0	1	0	0	1	1	$B$
	<hr/>									
	1	1	0	0	1	0	0	0		

The diagram shows the addition of two 10-bit integers, A and B. The bits of A are 1, 1, 0, 1, 1, 0, 1, 0, 1, 1. The bits of B are 1, 0, 0, 0, 1, 0, 0, 1, 1, 1. The result of the addition is 1, 1, 0, 0, 1, 0, 0, 0, 0. A vertical box highlights the first two bits of the result, 1 and 1, which correspond to the carry bits from the addition of the first two bits of 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  $B$ :

1	1	0	1	1	0	1	0	1	$A$
1	0	0	0	1	0	0	1	1	$B$
<hr/>									
	1	1	0	0	1	0	0	0	

*Note: In the original image, a light blue box highlights the leading '1' of integer A, and small subscripts (0, 0, 1, 1, 0, 1, 1, 1) are placed below the bits of integer 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  $B$ :

	1	1	0	1	1	0	1	0	1	$A$
	1	0	0	0	1	0	0	1	1	$B$
1	0	0	1	1	0	1	1	1		
	0	1	1	0	0	1	0	0	0	

## 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  $B$ :

1	1	1	0	1	1	0	1	0	1	$A$
1	0	0	1	1	0	1	0	1	1	$B$
	0	1	1	0	0	1	0	0	0	

## 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  $B$ :

The diagram illustrates the addition of two 9-bit integers, A and B, to produce a 10-bit result. The numbers are aligned to the right, and a horizontal line is drawn under the second row. A light blue vertical box highlights the carry bits from the previous step, which are 1, 0, 0, 1, 1, 0, 1, 1, 1, and 1, positioned below the corresponding bits of the second row.

	1	1	0	1	1	0	1	0	1	$A$
	1	0	0	0	1	0	0	1	1	$B$
	<hr/>									
1	0	1	1	0	0	1	0	0	0	

## 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  $B$ :

$$\begin{array}{r} 110110101 \quad A \\ 100010011 \quad B \\ \hline 1011001000 \end{array}$$

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 known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \\ \times 1011 \\ \hline \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \\ \times 1011 \\ \hline \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 00000 \\ 00000 \\ 10001 \\ \hline \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 1000100 \\ 10001000 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 1000100 \\ 10001000 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 0000000 \\ 00000000 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

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

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 0000000 \\ 00000000 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

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

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 0000000 \\ 00000000 \\ 100010000 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 0000000 \\ 10001000 \\ \hline 10001000 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 0000000 \\ 10001000 \\ \hline 10111011 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

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

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 0000000 \\ 10001000 \\ \hline 10111011 \end{array}$$

- This is also known 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  $B$  ( $m \leq n$ ).

$$\begin{array}{r} 10001 \times 1011 \\ \hline 10001 \\ 100010 \\ 0000000 \\ 10001000 \\ \hline 10111011 \end{array}$$

- This is also known 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  $k$ .

$$\boxed{b_{n-1} \quad \dots \quad b_0} \times \boxed{a_{n-1} \quad \dots \quad a_0}$$

## Example: Multiplying Two Integers

**A recursive approach:**

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

$$\boxed{b_{n-1} \quad \cdots \quad b_{\frac{n}{2}} \quad b_{\frac{n}{2}-1} \quad \cdots \quad b_0} \times \boxed{a_{n-1} \quad \cdots \quad a_{\frac{n}{2}} \quad a_{\frac{n}{2}-1} \quad \cdots \quad a_0}$$

## Example: Multiplying Two Integers

A recursive approach:

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

$$\begin{array}{|c|c|} \hline B_1 & B_0 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline A_1 & A_0 \\ \hline \end{array}$$

## 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

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

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

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

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

$\mathcal{O}(1)$

## Example: Multiplying Two Integers

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

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  $A_1$

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

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  $B_1$

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

- 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:  $Z_2 \leftarrow \text{mult}(A_1, B_1)$
- 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 $\text{mult}(A, B)$

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

$\mathcal{O}(1)$

$\mathcal{O}(1)$

$\mathcal{O}(n)$

$\mathcal{O}(n)$

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

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

$\mathcal{O}(1)$

$\mathcal{O}(1)$

$\mathcal{O}(n)$

$\mathcal{O}(n)$

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

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

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

$\mathcal{O}(1)$

$\mathcal{O}(1)$

$\mathcal{O}(n)$

$\mathcal{O}(n)$

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

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

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

## Example: Multiplying Two Integers

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

1: <b>if</b> $ A  =  B  = 1$ <b>then</b>	$\mathcal{O}(1)$
2: <b>return</b> $a_0 \cdot b_0$	$\mathcal{O}(1)$
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: <b>return</b> $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

## Example: Multiplying Two Integers

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

1: <b>if</b> $ A  =  B  = 1$ <b>then</b>	$\mathcal{O}(1)$
2: <b>return</b> $a_0 \cdot b_0$	$\mathcal{O}(1)$
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: $Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T\left(\frac{n}{2}\right)$
6: $Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	$2T\left(\frac{n}{2}\right) + \mathcal{O}(n)$
7: $Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T\left(\frac{n}{2}\right)$
8: <b>return</b> $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

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:  $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$        $T(n) = \Theta(n^{\log_b a})$
- ▶ 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:  $f(n) = \Omega(n^{\log_b a + \epsilon})$        $T(n) = \Theta(f(n))$



## Example: Multiplying Two Integers

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

- ▶ Case 1:  $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$        $T(n) = \Theta(n^{\log_b a})$
- ▶ 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  $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$ .

## Example: Multiplying Two Integers

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

- ▶ Case 1:  $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$        $T(n) = \Theta(n^{\log_b a})$
- ▶ 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  $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$ .

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:  $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$        $T(n) = \Theta(n^{\log_b a})$
- ▶ 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  $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$ .

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

⇒ Not better than 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  $Z_1$ :

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

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  $Z_1$ :

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

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  $Z_1$ :

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

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  $Z_1$ :

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

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

$\mathcal{O}(1)$

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  $Z_1$ :

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

Hence,

### Algorithm 4 mult( $A, B$ )

1: **if**  $|A| = |B| = 1$  **then**

$\mathcal{O}(1)$

2:     **return**  $a_0 \cdot b_0$

$\mathcal{O}(1)$

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  $Z_1$ :

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

Hence,

### Algorithm 4 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  $B_1$

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  $Z_1$ :

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

Hence,

### Algorithm 4 mult( $A, B$ )

1: **if**  $|A| = |B| = 1$  **then**

$\mathcal{O}(1)$

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  $Z_1$ :

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

Hence,

### Algorithm 4 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**  $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$

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  $Z_1$ :

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

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

$\mathcal{O}(1)$

$\mathcal{O}(1)$

$\mathcal{O}(n)$

$\mathcal{O}(n)$

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

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

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  $Z_1$ :

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

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

$\mathcal{O}(1)$

$\mathcal{O}(1)$

$\mathcal{O}(n)$

$\mathcal{O}(n)$

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

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

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

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 can use the following identity to compute  $Z_1$ :

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

Hence,

### Algorithm 4 mult( $A, B$ )

1: <b>if</b> $ A  =  B  = 1$ <b>then</b>	$\mathcal{O}(1)$
2: <b>return</b> $a_0 \cdot b_0$	$\mathcal{O}(1)$
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: <b>return</b> $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

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

- ▶ 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:  $f(n) = \Omega(n^{\log_b a + \epsilon})$        $T(n) = \Theta(f(n))$

## 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  $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$ .

## 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”.