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 \( n \)
\( a f\left(\frac{n}{b}\right) \leq c f(n) \) for some constant \( c < 1 \) then \( T(n) = \Theta(f(n)) \).

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

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

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

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

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

Hence,

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

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

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

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

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

Hence,

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

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

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

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

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

$$= c n^{\log_b a} \sum_{i=0}^{\ell - 1} i^k \approx \frac{c}{k} \ell^{k+1}$$

$$= \frac{c}{k} n^{\log_b a} \ell^{k+1} \Rightarrow T(n) = O\left(n^{\log_b a \log^{k+1} n}\right).$$
Case 3. Now suppose that $f(n) \geq dn^{\log_b a + \epsilon}$, 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 $n/b^i \geq n_0$ is still sufficiently large.

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

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

Hence,

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

\[\text{⇒ } T(n) = \Theta(f(n)).\]

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.

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

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

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

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

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

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

Hence,

\[A \cdot B = A_1 B_1 \cdot 2^n + (A_1 B_0 + A_0 B_1) \cdot 2^n + A_0 B_0\]
Example: Multiplying Two Integers

We get the following recurrence:

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

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,

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

Example: Multiplying Two Integers

We get the following recurrence:

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

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

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

⇒ Not better then the “school method”.

Example: Multiplying Two Integers

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

A huge improvement over the “school method”.

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

Example: Multiplying Two Integers

A huge improvement over the “school method”.

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