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

Case 2.

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

Case 3.

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

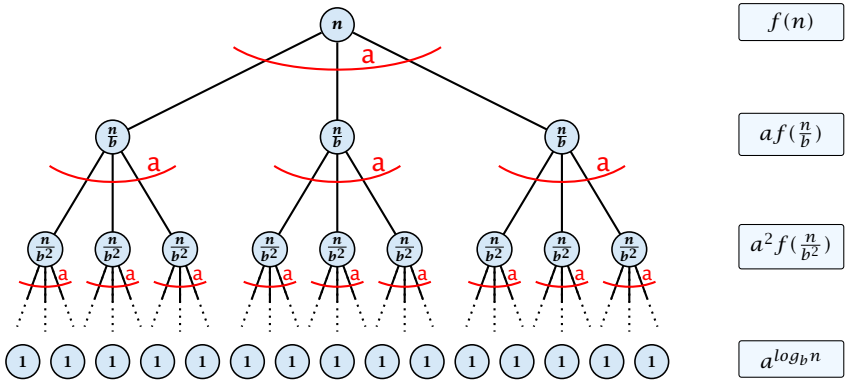
Note that the cases do not cover all possibilities.

6.2 Master Theorem

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

The Recursion Tree

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



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

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

Hence,

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

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

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

Hence,

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

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,

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

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

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

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

Case 3. Now suppose that $f(n) \geq d n^{\log_b a+\epsilon}$, 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.

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

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

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

Hence,

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

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:

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

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

Example: Multiplying Two Integers

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

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

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

Time requirement:

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

Example: Multiplying Two Integers

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

B_1	B_0	×	A_1	A_0

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,

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

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

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

We get the following recurrence:

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

Example: Multiplying Two Integers

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

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

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 then the "school method".

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

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

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

Example: Multiplying Two Integers

We get the following recurrence:

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

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

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

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

A huge improvement over the "school method".