Section IV.6: The Master Method and Applications

Definition IV.6.1: A function f is **asymptotically positive** if and only if there exists a real number n such that $f(x) > 0$ for all $x > n$.

A consequence of this definition is that function f is **asymptotically positive** if and only if the coefficient of its dominant term is positive. The dominant term t is the one with the largest growth rate, i. e. $t >' u$ for all other terms in f, $>'$ defined as in section II.5.

The **master method** provides us a straightforward and "cookbook" method for solving recurrences of the form $T(n) = a\ T(\ n/b\ ) + f(n)$, where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function. This recurrence gives us the running time of an algorithm that divides a problem of size n into a subproblems of size $\frac{n}{b}$. This recurrence is technically correct only when $\frac{n}{b}$ is an integer, so the assumption will be made that $\frac{n}{b}$ is either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$ since such a replacement does not affect the asymptotic behavior of the recurrence. a is a positive integer since one can have only a whole number of subproblems.

Theorem IV.6.1: Master Theorem:
Let $a \geq 1$ and $b > 1$ be constants and $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by $T(n) = a\ T(\frac{n}{b}) + f(n)$ where $\frac{n}{b}$ is treated as above. Then $T(n)$ can be bounded asymptotically as follows:

(1). If $f(n) = O(\ n^{\log_b a - \varepsilon}\ )$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a}\ )$.

(2). If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta\ (n^{\log_b a} \log_2 n\ )$.

(3). If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $a\ f(\frac{n}{b}) \leq c\ f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Note: The relation $a\ f(\frac{n}{b}) \leq c\ f(n)$ is called the **regularity condition** on $f(n)$.

Intuitive explanation: In the Master theorem one compares $n^{\log_b a}$ and $f(n)$. If these functions are in the same Θ class (see section II.5), then we multiply by a logarithmic factor to get the run time of $T(n)$ (case 2). If $f(n)$ is polynomially smaller than $n^{\log_b a}$ (by a factor of n^{ε} **)** then $T(n)$ is in the same Θ class as $n^{\log_b a}$ (case 1). If $f(n)$ is polynomially larger than $n^{\log_b a}$, then $T(n)$ is in the same Θ class as $f(n)$ (case 3). In cases 1 and 3, the functions **must** be polynomially larger or smaller than $n^{\log_b a}$. Example IV.6.4 illustrates

a case where the function is **not** polynomially larger or smaller than $n^{\log_b a}$. Here the Master Theorem does not apply.

Example IV.6.1: Get a Θ estimate, coefficient 1, for $T(n) = 16\ T(\frac{n}{4}) + n$.

Solution: Here $a = 16$ and $b = 4$, so $\log_b a = 2$. $f(n) = n = O(n) = O(n^{2-1})$, so we can take $\varepsilon = 1$. Therefore case (1) of Master Theorem holds, and, therefore, $T(n) = \Theta(n^2)$.

Example IV.6.2: Get a Θ estimate, coefficient 1, for $T(n) = T\left(\frac{3n}{4}\right) + 2$.

Solution: Here $a = 1$, $b = \frac{4}{3}$, and $f(n) = 2$. $\log_b a = 0$ since $a = 1$.
$f(n) = 2 = \Theta(1) = \Theta(n^0)$. Therefore, case (2) of the Master Theorem holds. So
$T(n) = \Theta(1 \log_2 n) = \Theta(\log_2 n)$.

Example IV.6.3: Get a Θ estimate, coefficient 1, for $T(n) = 3T\left(\frac{n}{4}\right) + n \log_2 n$.

Solution: Here $a = 3$, $b = 4$, and $f(n) = n \log_2 n$. $n^{\log_b a} = n^{\log_4 3} \approx n^{0.793}$. It is true that $f(n) = \Omega(n^{.793+.2})$, where $\varepsilon \approx 0.2$, since $n \log_2 n \geq n \geq n^{.993}$. Case 3 of Master Theorem applies if we can find $c < 1$ where $3f(\frac{n}{4}) \leq c\, f(n)$.

$a\, f(\frac{n}{b}) = 3(\frac{n}{4} \log_2 \frac{n}{4}) \leq \frac{3}{4} n \log_2 n = c\, f(n)$ for $c = \frac{3}{4}$. Therefore, the regularity condition holds for $f(n)$. Case 3 holds, so $T(n) = \Theta(n \log_2 n)$.

Example IV.6.4: Show that the Master Theorem cannot be applied to the function
$T(n) = 2T(\frac{n}{2}) + n \log_2 n$.

Solution: Here $a = 2$, $b = 2$, and $f(n) = n \log_2 n$. $n^{\log_b a} = n$. $f(n) = n \log_2 n$ is asymptotically larger than n (faster growth rate) but is not polynomially larger than n since $\log_2 n <' n^{\varepsilon}$ for any $\varepsilon > 0$. (Theorem II.5.6).

Corman's book *Introduction to Algorithms* gives a detailed proof of the Master Theorem. Here we give lemmas and draw a recursion tree to illustrate the cases of the Master theorem. See Corman's text for a more in-depth approach.

Lemma IV.6.1: Let $a \geq 1$ and $b > 1$ be constants and $f(n)$ be a nonnegative function defined on exact powers of b. Define $T(n)$ on exact powers of b by the recurrence

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1 \\ aT(n/b) + f(n) \text{ if } n = b^i \text{ where } i \text{ is a positive integer.} \end{cases}$$

Then $T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$.

Proof: Iterating the recurrence gives

$$T(n) = f(n) + aT(n/b) = f(n) + af(n/b) + a^2 T(n/b^2)$$
$$= f(n) + af(n/b) + a^2 f(n/b^2) + \ldots + a^{\log_b n - 1} f(n/b^{\log_b n - 1}) + a^{\log_b n} T(1).$$

We stop the above process when $(n/b^k) \leq 1$, i.e., when $k \geq \log_b n$.

Since

(*) $a^{\log_b n} = n^{\log_b a}$,

the last term of the above expression becomes

$a^{\log_b n} T(1) = \Theta(n^{\log_b a})$,

using the condition $T(1) = \Theta(1)$. The remaining terms can be expressed as the sum $\sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$. Here we are summing across the levels of the recursion tree below.

At the level j, there are a^j nodes.

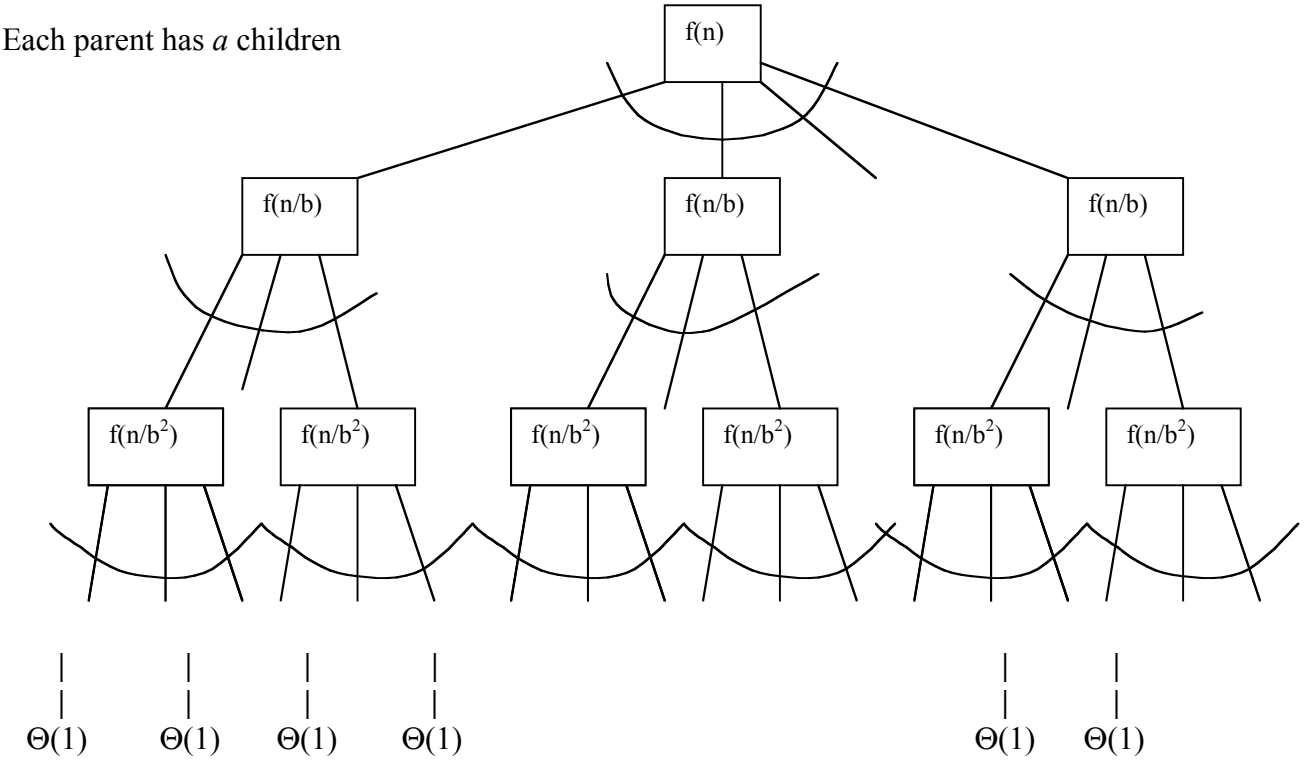
Therefore $T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$, which completes the proof.

Derivation of (*):

$\log_b n = \log_b a \cdot \log_a n$	by the change of base formula
$\log_a (a^{\log_b n}) = \log_b a \cdot \log_a n$	by $\log_a a^x = x$
$\log_a (a^{\log_b n}) = \log_a n^{\log_b a}$	by $x \log_a n = \log_a n^x$
$a^{\log_b n} = n^{\log_b a}$	since a log function is 1 to 1.

Below is drawn a recursion tree for $T(n) = a\ T(n/b) + f(n)$

Each parent has a children



The height of the tree is $\log_b n$ which is the number of times we divide n by b before getting 1. Using (*), it follows that there are $n^{\log_b a}$ leaves. Their sum is $\Theta(n^{\log_b a})$. From the tree, one can see that the the sum of the values in the nodes is $\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$. This is true, since as we go down to the next level of the tree, the number of nodes is multiplied by a, and a new factor of b appears in the denominator of the argument of f.

The recursion tree illustrates the three cases of the Master Theorem.

In case 1, the total cost of the tree is dominated by the cost of the leaves, which is $\Theta(n^{\log_b a})$. The cost of the internal nodes, contributed by $\sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$ is polynomially smaller than that of the leaves, so by Theorem II.5.3, the growth rate of T is the maximum of the two costs, namely $\Theta(n^{\log_b a})$.

In case 2, the total cost is evenly distributed among the levels of the tree. The analysis of this cost is a generalization of that of Merge Sort, found in section IV.5. The cost at each level of the tree is $\Theta(n^{\log_b a})$, and there are $\log_b n$ levels of the tree. So the total cost is

$\Theta(n^{\log_b a} \cdot \log_2 n)$. Here we use Theorem II.5.4 (multiplicative property of Θ) and the fact that the growth rate of a log function is independent of the base.

In case 3, the total cost is dominated by the root. The constraint $a\,f(n/b) \le c\,f(n)$ gives us a convergent infinite series in the derivation of cost of $T(n)$ and guarantees the dominance of $f(n)$.

The next lemma, given without proof, with the previous lemma, leads directly to the Master Theorem, for it defines $g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$ and asserts that $T(n)$ has the same behavior as $g(n)$ in cases 2 and 3 of the Master Theorem.

Lemma IV.6.2: Let $a \ge 1$ and $b > 1$ be constants and $f(n)$ be a nonnegative function defined on exact powers of b. A function $g(n)$ defined over exact powers of b by $g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$ can be bounded asymptotically for exact powers of b as follows.

(1) If $f(n) = O(n^{\log_b a-\epsilon})$ for some $\epsilon > 0$, then $g(n) = O(n^{\log_b a})$.

(2) If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \log_2 n)$.

(3) If $a \cdot f(\frac{n}{b}) \le c\,f(n)$ for some constant $c < 1$ and all $n \ge b$, then $g(n) = \Theta(f(n))$.

Claim: Lemmas IV.6.1 and IV.6.2 directly lead to the Master Theorem.

Proof of Claim:

Case 1: Since $g(n) = O(n^{\log_b a})$ (by Lemma IV.6.2), we have
$T(n) = \Theta(n^{\log_b a}) + g(n)$ (by Lemma IV.6.1)
$= \Theta(n^{\log_b a})$ as asserted in case 1 of the Master Theorem.

Case 2: Since $g(n) = \Theta(n^{\log_b a} \log_2 n)$ (by Lemma IV.6.2)
and $n^{\log_b a} = O(n^{\log_b a} \log_2 n)$,
we have $T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log_2 n)$ (by Lemma IV.6.1)
$= \Theta(n^{\log_b a} \log_2 n)$ as asserted in case 2 of the Master Theorem.

Case 3: Since $f(n)$ is polynomially larger than $n^{\log_b a}$ (as stated in the Master Theorem) and $g(n) = \Theta(f(n))$ (by Lemma IV.6.2), we have
$T(n) = \Theta(n^{\log_b a}) + \Theta(f(n))$ (by Lemma IV.6.1)
$= \Theta(f(n))$ as asserted in case 3 of the Master Theorem, since $f(n)$ is polynomially larger that $g(n)$.

Note: Corman's book *Introduction to Algorithms, p.72* illustrates a function between case 2 and case 3 of the Master Theorem. Here $f(n) = \Theta(n^{\log_b a} \log_2^k n)$ and $T(n) = \Theta(n^{\log_b a} \log_2^{k+1} n)$.

Exercises:

(1) Use the Master theorem to get the Θ estimate with leading coefficient 1 for the following recurrences. Clearly indicate which of the three cases hold, and, for case 3, show that the regularity condition holds.

(a) $T(n) = 4T(\frac{n}{2}) + n.$ **(b)** $T(n) = 4T(\frac{n}{2}) + n^2$ **(c)** $T(n) = 4T(\frac{n}{2}) + n^3$

(2) (a) Use the Master theorem to show run time of binary search is $\Theta(\log_2 n)$. Here $T(n) = T(\frac{n}{2}) + 1$.

(b) Use the Master theorem to show that the run time of Merge Sort is $\Theta(n \cdot \log_2 n)$ Here $T(n) = 2\ T(\frac{n}{2}) + n.$

(3) Use the Master theorem to get the Θ estimate with leading coefficient 1 for $T(n) = 3T(\frac{n}{4}) + n.$ Your answer should be the same as Example IV.5.5 in section IV.5.

(4) Use the Master theorem to get the Θ estimate with leading coefficient 1 for $T(n) = 3T(\frac{n}{2}) + n.$ Your answer should be the same as Example IV.5.6 in section IV.5.

(5) The running time of algorithm A is described by $T(n) = 7T(\frac{n}{2}) + n^2$. Another algorithm A′ has running time $T'(n) = a\ T'(\frac{n}{4}) + n^2$. What is the largest value of a such that A′ is asymptotically faster than A? Justify your answer, showing relevant calculations.