7 The Master Theorem

This section is about the master equation; it is based on the treatment in Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein which is considered the authority on this subject.

The master method is an approach to solving recurrences of the form

$$T(n) = aT(n/b) + f(n) \tag{1}$$

where $a \geq 1$ and $b \geq 1$ and $f(n)$ is some function which is positive for large enough n. This applies to a problem in which the recursion requires the solution of a sub-problems of size n/b. In the case of binary search we had

$$T(n) = T(n/2) + t_0 \tag{2}$$

The problem requires the solution of just one $n/2$ sized problem and $a = 1$ and $b = 2$. The $f(n)$ measures the extra work required to solve the size n problem when the a size n/b problems have been solved. In the binary search example the size of this part of the problem didn't depend on n so $f(n) = t_0$ for some constant t_0, or put another way, $f(n) \in O(1)$.

The master theorem tells us the big-Oh complexity of $T(n)$ if the large n behavior of $f(n)$ satisfies some conditions which depend on a and b. Furthermore, it is a proper theorem, our aim in this course is to get used to doing rough estimates of complexity without trying for much rigor, but our *ad hoc* approach is embedded in a rigorous one and in the proper study of algorithms and, indeed, in much of cryptography, it is important to prove results about complexity properly. The master theorem is often useful for this.

Before looking at the theorem, we need to note one confusing aspect. The theorem tells us about the large n behaviour of the $T(n)$ that satisfies the recursion relation

$$T(n) = aT(n/b) + f(n) \tag{3}$$

Usually the recursion relation applies to the worst case run time; this means the theorem gives us a big-Theta description for the large n behaviour, but that doesn't mean that it is giving us a big-Theta description for the algorithm. Typically the recursion relation only applies to the worst-case senario, for example, for the binary search algorithm the recursion relation is

$$T(n) = T(n/2) + t_0 \tag{4}$$

if the last element examined is the element we are looking for; if the first element examined is the one we are looking for then the algorithm takes fewer steps and, indeed, the best case, where the first element we look at is the one we want, is $\Omega(1)$. In this way, if we apply the master theorem to

$$T(n) = T(n/2) + t_0 \tag{5}$$

it will tell us $T(n) \in \Theta(\log n)$. However this only applies to $T(n)$s satisfying the recursion relation, the $T(n)$ that is the run time of the algorithm is $O(\log n)$.

Now for the master theorem. It has three cases depending on how $f(n)$ behaves

1. If $f(n) \in O(n^c)$ for $c < \log_b a$ then $T(n) \in \Theta(n^{\log_b a})$
2. If $f(n) \in \Theta(n^c)$ for $c = \log_b a$ then $T(n) \in \Theta(n^c \log n)$

3. If $f(n) \in \Omega(n^c)$ for $c > \log_b a$ then, provided some other conditions on $f(n)$ hold then $T(n) \in \Theta(f(n))$

Obviously we haven't stated the third posibility fully and, indeed, we won't be proving the master theorem. We haven't worried about the fact that n/b may not be an integer, in practise in divide and conquer algorithms n/b gets rounded up or down to the nearest integer. We saw this in the case of binary search and, in fact, it doesn't make any difference to the theorem.

Roughly speaking the large n behaviour of the recursion

$$T(n) = aT(n/b) + f(n) \tag{6}$$

is either dominated by the $aT(n/b)$ term or the $f(n)$ term depending on how fast $f(n)$ grows. This is what gives the first and third cases; in the first case $f(n)$ grows slowly enough to allow the first term to dominate the behaviour, in the third case $f(n)$ grows so fast it takes control. The second case is a kind of in-between case. Since ¡¡¡¡¡¡¡ HEAD the first case is about $f(n)$ growing slower than $n^{\log_b a}$ we specify the behavior of $f(n)$ using big-oh: $f(n) \in O(n^c)$ for ======= the first case is about $f(n)$ being growing slower than $n^{\log_b a}$ we specify the behavior of $f(n)$ using big-Oh: $f(n) \in O(n^c)$ for ¿¿¿¿¿¿¿ 4b994be2a3e4e129db67738906496d7fc4ccd6a3 $c < \log_b a$. Conversely, the third case applies to functions that grow faster than n^c so we specify the behavior of $f(n)$ using big-Omega. Finally, the middle case applies only to functions that grow like n^c, not faster, not slower, so we specify $f(n)$ using big-Theta.

Now lets consider the binary search case we looked at before. In that case $a = 1$ and $b = 2$ so $\log_b a = \log_2 1 = 0$. We also know $f(n) = 1 \in \Theta(1)$ so $c = 0$. This means the second case applies and the master theorem says the $T(n) \in O(\log n)$ which is what we worked out.

We will have the opportunity to see the master equation in action again and it is a standard well-used tool in the study of algorithms. Here we will just look at some more artificial examples where we consider some recursion relations without examining any algorithm they might have come from.

Now say

$$T(n) = 9T(n/3) + n. \tag{7}$$

Here $a = 9$, $b = 3$ and $f(n) = n$, so $f(n) \in O(n)$ and $\log_b a = 2 > 1$. This means the first case applies and

$$T(n) \in \Theta(n^2) \tag{8}$$

If, instead

$$T(n) = 5T(n/4) + n \tag{9}$$

we have $a = 5$, $b = 4$ and $f(n) = n \in O(n)$. Now $\log_b a = \log_4 5 \approx 1.16 > 1$ so we are again in the first case and, approximately

$$T(n) \in \Theta(n^{1.16}) \tag{10}$$

Finally say we have

$$T(n) = 4T(n/2) + n^2 \tag{11}$$

we have $\log_b a = 2$ and $f(n) \in \Theta(n^2)$ so

$$T(n) \in \Theta(n^2 \log n) \tag{12}$$