5 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 which the recursion requires the solution of a sub-problems of size n/b. In the case of binary search for example, the problem required the solution of just one $n/2$ sized problem, so $a = 1$ 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 $f(n) = c$ for some constant, or put another way, $f(n) \in O(1)$.

The master theorem tells us how the big-oh complexity of $T(n)$ if the large n behavior of $f(n)$ satisfies some conditions which depend on a and b.

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. Furthermore, the master formula gives us a big-theta complexity for $T(n)$, we usually concentrate in this course on big-oh complexity so we will largely ignore this benefit of the master theorem. Finally, 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 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

$$T(n) = aT(n/b) + f(n) \tag{2}$$

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.

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

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

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

$$T(n) \in \Theta(n^2) \tag{4}$$

If, instead

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

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

$$T(n) \in \Theta(n^{1.16}) \tag{6}$$

Finally say we have

$$T(n) = 4T(n/2) + n^2 \tag{7}$$

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

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