CMPS 101
Algorithms and Abstract Data Types
Spring 2006

Recurrence Relations

When analyzing the run time of recursive algorithms we are often led to consider functions $T(n)$ which are defined by recurrence relations of a certain form. A typical example would be

$$T(n) = \begin{cases} c & n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + dn & n > 1 \end{cases}$$

where *c*, *d* are fixed constants. The specific values of these constants are important for determining the explicit solution to the recurrence. Often however we are only concerned with finding an asymptotic (upper, lower, or tight) bound on the solution. We call such a bound an *asymptotic solution* to the recurrence. In the above example the particular constants *c* and *d* have no effect on the asymptotic solution. We may therefore write our recurrence as

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & n > 1 \end{cases}$$

In what follows we'll show that a tight asymptotic solution to the above recurrence is $T(n) = \Theta(n \log n)$.

We will study the following methods for solving recurrences.

- **Substitution Method.** This method consists of guessing an asymptotic (upper or lower) bound on the solution, and trying to prove it by induction.
- **Recursion Tree Method – Iteration Method.** These are two closely related methods for expressing $T(n)$ as a summation which can then be analyzed. A recursion tree is a graphical depiction of the entire set of recursive invocations of the function *T*. The goal of drawing a recursion tree is to obtain a guess which can then be verified by the more rigorous substitution method. Iteration consists of repeatedly substituting the recurrence into itself to obtain an iterated (i.e. summation) expression.
- **Master Method.** This is a cookbook method for determining asymptotic solutions to recurrences of a specific form.

The Substitution Method

We begin with the following example.

$$T(n) = \begin{cases} 2 & 1 \le n < 3 \\ 3T(\lfloor n/3 \rfloor) + n & n \ge 3 \end{cases}$$

We will see later that $T(n) = \Theta(n \log(n))$, but for the time being, suppose that we are able to guess (somehow) that $T(n) = O(n \log(n))$. In order to prove this we must find positive numbers *c* and n_0 such

that $T(n) \le cn\log(n)$ for all $n \ge n_0$. If we knew appropriate values for these constants, we could prove this inequality by induction. Our goal then is to determine c and n_0 such that the induction proof will work. In what follows it will be useful to take log to mean $\log_3$. We begin by mimicking the induction step (IId) to find c:

> Let $n > n_0$ and assume for all $k < n$ that $T(k) \le ck\log(k)$. In particular, taking $k = \lfloor n/3 \rfloor$ gives us $T(\lfloor n/3 \rfloor) \le c\lfloor n/3 \rfloor \log(\lfloor n/3 \rfloor)$. We must show that $T(n) \le cn\log(n)$. Observe
>
> $$\begin{aligned} T(n) &= 3T(\lfloor n/3 \rfloor) + n && \text{(by the recurrence for } T\text{)} \\ &\le 3c\lfloor n/3 \rfloor \log(\lfloor n/3 \rfloor) + n && \text{(by the induction hypothesis)} \\ &\le 3c(n/3)\log(n/3) + n && \text{(since } \lfloor x \rfloor \le x \text{ for all } x\text{)} \\ &= cn(\log(n) - 1) + n \\ &= cn\log(n) - cn + n \end{aligned}$$
>
> To obtain $T(n) \le cn\log(n)$ we need to have $-cn + n \le 0$, which implies $c \ge 1$.

We see that as long as c is chosen to satisfy the constraint $c \ge 1$, the induction step will go through. It remains to determine the constant n_0. The base step requires $T(n_0) \le cn_0 \log(n_0)$. This can be viewed as a single constraint on the two parameters c and n_0, which indicates that we have some freedom in choosing them. In other words, the constraints do not uniquely determine both constants, so we must make some arbitrary choice. Choosing $n_0 = 3$ is algebraically convenient. This yields $T(3) \le 3c$, whence $c \ge T(3)/3$. Since $T(3) = 9$ (using the recurrence) we have the constraint $c \ge 3$. Clearly if we choose $c = 3$, both constraints ($c \ge 1$ and $c \ge 3$) will be satisfied.

It is important to note that we have not proved anything yet. Everything we have done up to this point has been scratch work with the goal of finding appropriate values for c and n_0. It remains to present a complete induction proof of the assertion: $T(n) \le 3n\log(n)$ for all $n \ge 3$.

Proof:

I. Since $T(3) = 3T(\lfloor 3/3 \rfloor) + 3 = 3T(1) + 3 = 3 \cdot 2 + 3 = 9$ and $3 \cdot 3\log(3) = 9$ the base case reduces to $9 \le 9$ which is true.

II. Let $n > 3$ and assume for all $k < n$ that $T(k) \le 3k\log(k)$. Then

$$\begin{aligned} T(n) &= 3T(\lfloor n/3 \rfloor) + n && \text{(by the recurrence for } T\text{)} \\ &\le 3 \cdot 3\lfloor n/3 \rfloor \log(\lfloor n/3 \rfloor) + n && \text{(by the induction hypothesis letting } k = \lfloor n/3 \rfloor\text{)} \\ &\le 9(n/3)\log(n/3) + n && \text{(since } \lfloor x \rfloor \le x \text{ for all } x\text{)} \\ &= 3n(\log(n) - 1) + n \\ &= 3n\log(n) - 2n \\ &\le 3n\log(n) \end{aligned}$$

It now follows that $T(n) \le 3n\log(n)$ for all $n \ge 3$. ///

It is a somewhat more difficult exercise to prove by the same technique that $T(n) = \Omega(n\log(n))$, and therefore $T(n) = \Theta(n\log(n))$.

Exercise Determine positive numbers c and n_0 such that $T(n) \geq cn\log(n)$ for all $n \geq n_0$. Hint: Use the following facts: (1) $\lfloor x \rfloor > x - 1$, and (2) $\log_3\lfloor x \rfloor \geq \log_3(x) - 1$ for $x \geq 3/2$. (With some effort, its possible to show that $c = 1/4$ and $n_0 = 4$ work.)

The next example illustrates one complication that can arise in the substitution method. Define $T(n)$ by the recurrence

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\lfloor n/2 \rfloor) + 1 & n \geq 2 \end{cases}$$

We guess that $T(n) = O(n)$. To prove this guess, we attempt to find positive constants c and n_0 such that $T(n) \leq cn$ for all $n \geq n_0$. As before we proceed by mimicking the induction step. Let $n > n_0$ and assume for all k in the range $n_0 \leq k < n$ that $T(k) \leq ck$. In particular, for $k = \lfloor n/2 \rfloor$ we have $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor$, and thus

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + 1 && \text{(by the recurrence for } T(n)\text{)} \\ &\leq 2c\lfloor n/2 \rfloor + 1 && \text{(by the induction hypothesis)} \\ &\leq 2c(n/2) + 1 \\ &= cn + 1 \end{aligned}$$

Our goal is to determine c so that the induction step is feasible. We need to reach the conclusion that $T(n) \leq cn$, and to do this we must determine a number c such that $cn + 1 \leq cn$. But this inequality is obviously false for all positive c. Apparently the induction step cannot be made to work, which might lead us to believe that our guess is incorrect. Actually $T(n) = O(n)$ *is* correct (as can be seen by other methods), so we take a different approach. Our trick is to subtract a lower order term from the right side of the inequality we wish to prove. To be precise, we seek positive constants c, n_0, and b such that $T(n) \leq cn - b$ for all $n \geq n_0$. Observe that this inequality also implies $T(n) = O(n)$ by a result proved in the handout on asymptotic growth rates. It may seem counter-intuitive to attempt to prove an inequality which is stronger than the one that failed, but notice that this strategy allows us to use a stronger induction hypothesis. Again let $n > n_0$ and assume for all k in the range $n_0 \leq k < n$ that $T(k) \leq ck - b$, and in particular $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor - b$. Thus

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + 1 && \text{(by the recurrence for } T(n)\text{)} \\ &\leq 2(c\lfloor n/2 \rfloor - b) + 1 && \text{(by the induction hypothesis)} \\ &\leq 2(c(n/2) - b) + 1 && \text{(since } \lfloor x \rfloor \leq x\text{)} \\ &= cn - 2b + 1 \end{aligned}$$

If we take $b = 1$ then $T(n) \leq cn - b$, as desired. For the base case let $n_0 = 1$. We must show $T(1) \leq c \cdot 1 - 1$, which says $c \geq T(1) + 1 = 2$. Thus we may set $n_0 = 1$, $c = 2$, and $b = 1$.

Exercise Use induction to prove that $T(n) \le 2n-1$ for all $n \ge 1$, whence $T(n) = O(n)$.

Exercise Referring to the previous example, use the substitution method to show that $T(n) = \Omega(n)$, whence $T(n) = \Theta(n)$.

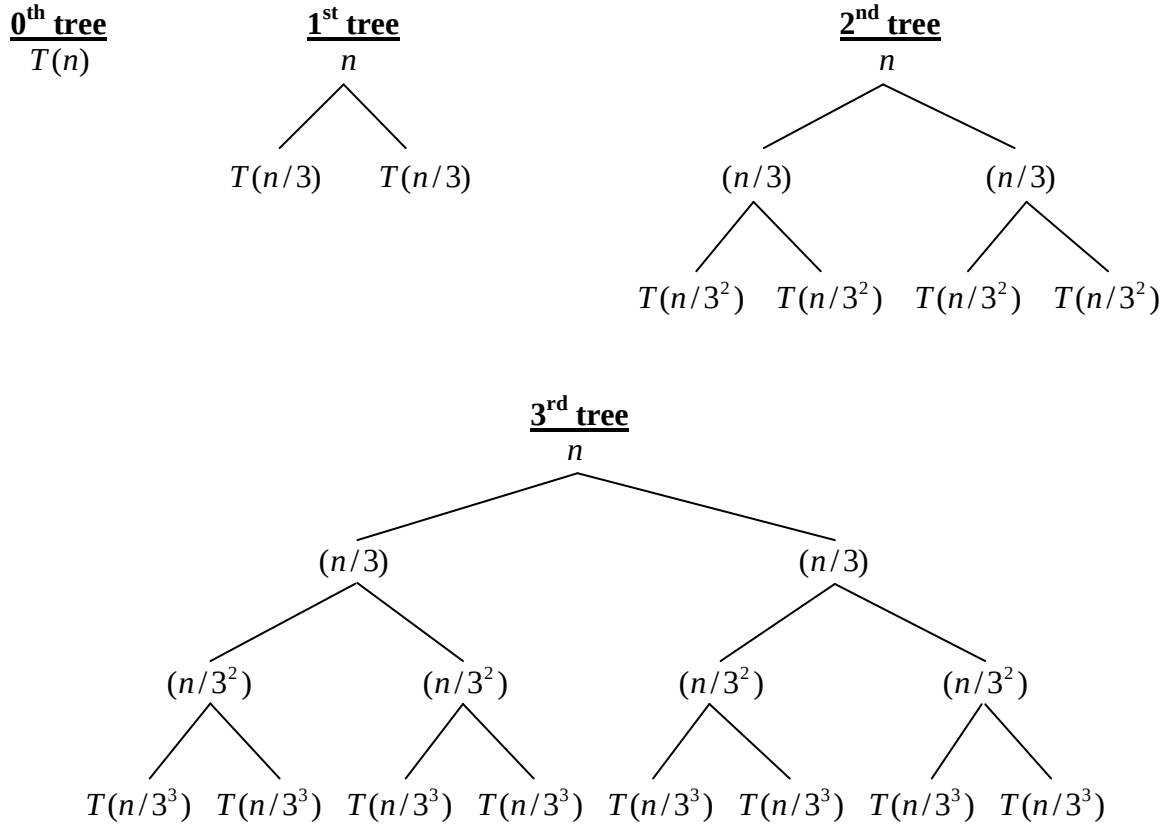
The Recursion Tree – Iteration Method

The *recursion tree method* can be used to generate a good guess for an asymptotic bound on a recurrence. This guess can then be verified by the substitution method. Since our guess will be verified, we can take some liberties in our calculations, such as dropping floors and ceilings or restricting the values of *n*.

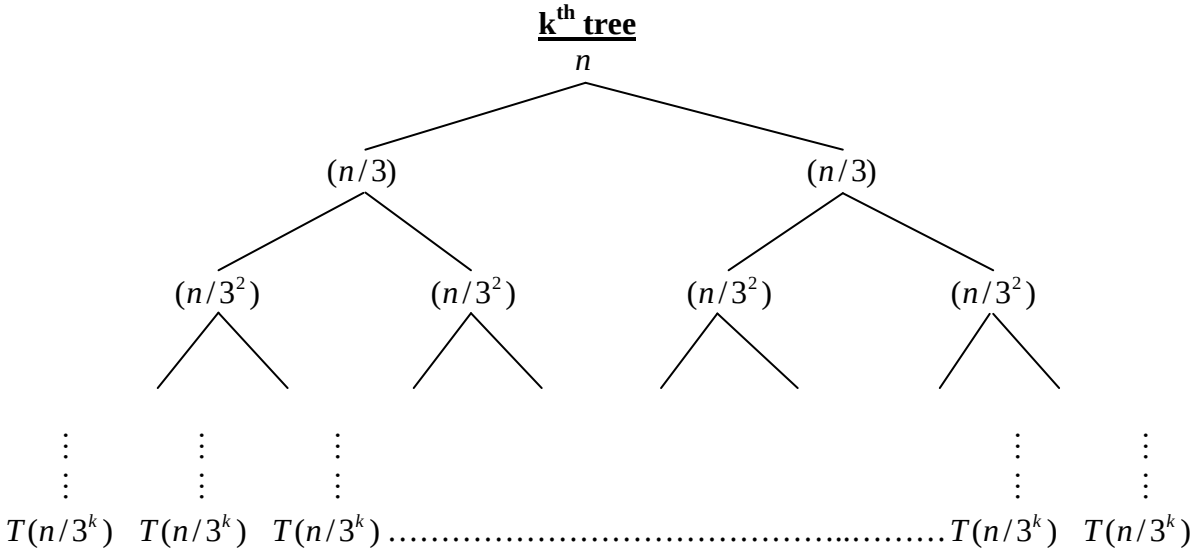
Let us begin with the example

$$T(n) = \begin{cases} \Theta(1) & 1 \le n < 3 \\ 2T(\lfloor n/3 \rfloor) + \Theta(n) & n \ge 3 \end{cases}$$

We simplify the recurrence by dropping the floor and replacing $\Theta(n)$ by n to get $T(n) = 2T(n/3) + n$. Each node in a recursion tree represents one term in the calculation of $T(n)$ obtained by recursively substituting the expression for T into itself. We construct a sequence of such trees of increasing depths.



By summing the nodes in any one of these trees, we obtain an expression for $T(n)$. After k iterations of this process we reach a tree in which all bottom level nodes are $T(n/3^k)$.



Note that there are 2^i nodes at depth *i*, each of which has value $n/3^i$ (for $0 \le i \le k-1$). The sequence of trees terminates when all bottom level nodes are $T(1)$, i.e. when $n/3^k = 1$, which implies $k = \log_3(n)$. The number of nodes at this bottom level is therefore $2^k = 2^{\log_3(n)} = n^{\log_3(2)}$. Summing all nodes in the final recursion tree gives us the following expression for $T(n)$.

$$\begin{aligned} T(n) &= \sum_{i=0}^{k-1} 2^i \cdot (n/3^i) \; + \; n^{\log_3(2)} \cdot T(1) \\ &= n\left(\sum_{i=0}^{k-1} (2/3)^i \right) \; + \; n^{\log_3(2)} \cdot T(1) \\ &= n\left(\frac{1-(2/3)^k}{1-(2/3)} \right) \; + \; n^{\log_3(2)} \cdot T(1) \\ &= 3n\left(1-(2/3)^k\right) \; + \; \Theta(n^{\log_3(2)}) \end{aligned}$$

If we seek an asymptotic upper bound we may drop the negative term to obtain $T(n) \le 3n + \Theta(n^{\log_3(2)})$. Since $\log_3(2) < 1$ the first term dominates, and so we guess: $T(n) = O(n)$.

Exercise Prove that this guess is correct using the substitution method.

It is sometimes possible to push these computations a little further to obtain an asymptotic estimate directly, with no simplifying assumptions. We call this the *iteration method*. We illustrate on the very same example.

$$T(n) = \begin{cases} 1 & 1 \le n < 3 \\ 2T\left(\lfloor n/3 \rfloor\right) + n & n \ge 3 \end{cases}$$

Here we do not assume that *n* is an exact power of 3, and keep the floor. Substituting this expression into itself *k* times yields:

$$
\begin{aligned}
T(n) &= n + 2T(\lfloor n/3 \rfloor) \\
&= n + 2(\lfloor n/3 \rfloor + 2T(\lfloor \lfloor n/3 \rfloor /3 \rfloor)) \\
&= n + 2\lfloor n/3 \rfloor + 2^2 T(\lfloor n/3^2 \rfloor) \\
&= n + 2\lfloor n/3 \rfloor + 2^2 (\lfloor n/3^2 \rfloor + 2T(\lfloor \lfloor n/3^2 \rfloor /3 \rfloor)) \\
&= n + 2\lfloor n/3 \rfloor + 2^2 \lfloor n/3^2 \rfloor + 2^3 T(\lfloor n/3^3 \rfloor) \\
&\vdots \\
&\vdots \\
&= \sum_{i=0}^{k-1} 2^i \cdot \lfloor n/3^i \rfloor + 2^k T(\lfloor n/3^k \rfloor)
\end{aligned}
$$

The process must terminate when k is chosen so that $1 \le \lfloor n/3^k \rfloor < 3$, which implies

$$
\begin{aligned}
& 1 \le n/3^k < 3 \\
\therefore\ & 3^k \le n < 3^{k+1} \\
\therefore\ & k \le \log_3(n) < k+1 \\
\therefore\ & k = \lfloor \log_3(n) \rfloor
\end{aligned}
$$

With this value of k we have $T(\lfloor n/3^k \rfloor) = 1$, whence $T(n) = \sum_{i=0}^{k-1} 2^i \cdot \lfloor n/3^i \rfloor + 2^k$. This expression in essence converts the computation of $T(n)$ from a recursive algorithm to an iterative algorithm, which is why we call this the *iteration method*. This summation can now be used to obtain asymptotic upper and lower bounds for $T(n)$.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{k-1} 2^i \cdot \lfloor n/3^i \rfloor + 2^k \\
&\le \sum_{i=0}^{k-1} 2^i \cdot (n/3^i) + 2^{\log_3(n)} && \text{(since } \lfloor x \rfloor \le x\text{)} \\
&= n \sum_{i=0}^{k-1} (2/3)^i + n^{\log_3(2)} \\
&= n \left(\frac{1-(2/3)^k}{1-(2/3)} \right) + n^{\log_3(2)} \\
&= 3n(1-(2/3)^k) + n^{\log_3(2)} \\
&\le 3n + n^{\log_3(2)} && \text{(dropping the negative term)} \\
&= O(n) && \text{(since } \log_3(2) < 1\text{)}
\end{aligned}
$$

Thus $T(n) = O(n)$ has been proved. Note that our proof is rigorous since at no point have we made any simplifying assumptions (such as n being an exact power of 3.) Therefore there is no need to verify $T(n) = O(n)$ by another method, such as substitution. In a similar fashion, one can show $T(n) = \Omega(n)$.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{k-1} 2^i \cdot \lfloor n/3^i \rfloor + 2^k \\
&\geq \sum_{i=0}^{k-1} 2^i \left((n/3^i) - 1 \right) + 2^{\log_3(n)-1} && \text{(since } \lfloor x \rfloor > x-1\text{)} \\
&= n \sum_{i=0}^{k-1} (2/3)^i - \sum_{i=0}^{k-1} 2^i + \frac{1}{2} n^{\log_3(2)} \\
&\geq n - \frac{2^k - 1}{2-1} + \frac{1}{2} n^{\log_3(2)} \\
&\geq n - 2^{\log_3(n)} + 1 + \frac{1}{2} n^{\log_3(2)} && \text{(since } \lfloor x \rfloor \leq x\text{)} \\
&= n + 1 - \frac{1}{2} n^{\log_3(2)} \\
&= \Omega(n) && \text{(since } \log_3(2) < 1\text{)}
\end{aligned}
$$

We've shown $T(n) = \Omega(n)$, and since $T(n) = O(n)$, we now have $T(n) = \Theta(n)$.

The iteration method can even be used to find *exact solutions* to some recurrences, as opposed to the *asymptotic solutions* we have been satisfied with up to now. For example define the function $T(n)$ by $T(0) = T(1) = 5$, and $T(n) = T(n-2) + n$ for $n \geq 2$. Iterating *k* times we find

$$
\begin{aligned}
T(n) &= n + T(n-2) \\
&= n + (n-2) + T(n-4) \\
&= n + (n-2) + (n-4) + T(n-6) \\
&\vdots \\
&\vdots \\
&= \sum_{i=0}^{k-1} (n - 2i) + T(n-2k) \\
&= \sum_{i=0}^{k-1} n + 2 \sum_{i=0}^{k-1} i + T(n-2k) \\
&= kn + 2\left(\frac{k(k-1)}{2} \right) + T(n-2k) \\
&= kn + k(k-1) + T(n-2k)
\end{aligned}
$$

This process must terminate when *k* is chosen so that either $n - 2k = 0$ or $n - 2k = 1$, which is where the recurrence 'bottoms out' so to speak. This implies

$$
\begin{aligned}
& 0 \leq n - 2k < 2 \\
\therefore\ & 2k \leq n < 2k + 2 \\
\therefore\ & k \leq \frac{n}{2} < k + 1 \\
\therefore\ & k = \lfloor n/2 \rfloor
\end{aligned}
$$

Thus if $k = \lfloor n/2 \rfloor$ we have $T(n-2k) = 5$, and therefore the exact solution to the above recurrence is given by

$$T(n) = \lfloor n/2 \rfloor \cdot n \; - \; \lfloor n/2 \rfloor \cdot \left(\lfloor n/2 \rfloor - 1 \right) \; + \; 5$$

The asymptotic solution is now readily seen to be $T(n) = \Theta(n^2)$.

Exercise Define $T(n)$ by $T(1) = 0$ and $T(n) = T\left(\lfloor n/2 \rfloor \right) + 1$ for $n \geq 2$. (Recall this was example 5 in the induction handout.) Use the iteration method to show $T(n) = \lfloor \lg(n) \rfloor$ for all *n*, whence $T(n) = \Theta(\log(n))$.

The Master Method

This is a method for finding (asymptotic) solutions to recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and the function $f(n)$ is asymptotically positive. Here $T(n/b)$ denotes either $T(\lfloor n/b \rfloor)$ or $T(\lceil n/b \rceil)$, and it is understood that $T(n) = \Theta(1)$ for some finite set of initial terms. Such a recurrence describes the run time of a 'divide and conquer' algorithm which divides a problem of size *n* into *a* subproblems, each of size n/b. In this context $f(n)$ represents the cost of doing the dividing and re-combining.

Master Theorem

Let $a \geq 1$, $b > 1$, $f(n)$ be asymptotically positive, and let $T(n)$ be defined by $T(n) = aT(n/b) + f(n)$. Then we have three cases:

(1) If $f(n) = O\left(n^{\log_b(a) - \varepsilon} \right)$ for some $\varepsilon > 0$, then $T(n) = \Theta\left(n^{\log_b(a)} \right)$.
(2) If $f(n) = \Theta\left(n^{\log_b(a)} \right)$, then $T(n) = \Theta\left(n^{\log_b(a)} \cdot \log(n) \right)$.
(3) If $f(n) = \Omega\left(n^{\log_b(a) + \varepsilon} \right)$ for some $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $0 < c < 1$ and for all sufficiently large *n*, then $T(n) = \Theta\left(f(n) \right)$.

Remarks In each case we compare $f(n)$ to the function $n^{\log_b(a)}$, and the solution is determined by which function is asymptotically larger. In case (1) $n^{\log_b(a)}$ is polynomially larger than $f(n)$ and the solution is in the class $\Theta\left(n^{\log_b(a)} \right)$. In case (3) $f(n)$ is polynomially larger (and an additional *regularity condition* is met) so the solution is $\Theta\left(f(n) \right)$. In case (2) the functions are asymptotically equivalent and the solution is in the class $\Theta\left(n^{\log_b(a)} \cdot \log(n) \right)$, which is the same as $\Theta\left(f(n) \cdot \log(n) \right)$. To say that $n^{\log_b(a)}$ is *polynomially larger* than $f(n)$ as in (1), means that $f(n)$ is bounded above by a function which is smaller than $n^{\log_b(a)}$ by a polynomial factor, namely n^{ε} for some $\varepsilon > 0$. Note that the conclusion reached by the master theorem does not change if we replace $f(n)$ by a function asymptotically equivalent to it. For this reason the recurrence may simply be given as $T(n) = aT(n/b) + \Theta\left(f(n) \right)$. Notice also that there is no mention of initial terms. It is part of the content of the master theorem that the initial values of the recurrence do not effect it's asymptotic solution.

Examples

Let $T(n) = 8T(n/2) + n^3$. Then $a = 8$, $b = 2$, $\log_b(a) = 3$, and hence $f(n) = n^3 = \Theta\left(n^{\log_b(a)}\right)$, so we are in case (2). Therefore $T(n) = \Theta(n^3 \log(n))$.

Now let $T(n) = 5T(n/4) + n$. Here $a = 5$, $b = 4$, $\log_b(a) = 1.609... > 1$. Let $\varepsilon = \log_4(5) - 1$ so that $\varepsilon > 0$, and $f(n) = n = O\left(n^{\log_4(5)-\varepsilon}\right)$. Therefore we are in case (1) and $T(n) = \Theta\left(n^{\log_4(5)}\right)$.

Next consider $T(n) = 5T(n/4) + n^2$. Again $a = 5$, $b = 4$, so $\log_b(a) = 1.609... < 2$. Let $\varepsilon = 2 - \log_4(5)$, so that $\varepsilon > 0$, and $f(n) = n^2 = \Omega\left(n^{\log_4(5)+\varepsilon}\right)$, and therefore case (3) seems to apply. In this case we must check the regularity condition: $5f(n/4) \le cf(n)$ for some $0 < c < 1$ and all sufficiently large *n*. This inequality say $5(n/4)^2 \le cn^2$, i.e. $(5/16)n^2 \le cn^2$. Thus we may choose any *c* satisfying $5/16 \le c < 1$. Since the regularity condition is satisfied we have $T(n) = \Theta(n^2)$

Note that even though this is called the 'Master Theorem' the three cases do not cover all possibilities. There is a gap between cases (1) and (2) when $n^{\log_b(a)}$ is larger than $f(n)$, but not polynomially larger. For example consider the recurrence $T(n) = 2T(n/2) + n/\lg(n)$. Observe that $\log_2(2) = 1$, and check that $n/\lg(n) = \omega(n^{1-\varepsilon})$, whence $n/\lg(n) \ne O\left(n^{1-\varepsilon}\right)$ for any $\varepsilon > 0$, and therefore we are not in case (1). But also $n/\lg(n) = o(n\log(n))$, so that $n/\lg(n) \ne \Theta(n\log(n))$, and neither are we in case (2). Similar comments hold for cases (2) and (3). In addition, it is possible that the regularity condition in case (3) fails to hold.