5.3 More general kinds of recurrences

Recurrence Inequalities

The recurrences we have been working with are really idealized versions of what we know about the problems we are working on. For example, in merge-sort on a list of n items, we say we divide the list into two parts of equal size, sort each part, and then merge the two sorted parts. The time it takes to do this is the time it takes to divide the list into two parts plus the time it takes to sort each part, plus the time it takes to merge the two sorted lists. We don't specify how we are dividing the list, or how we are doing the merging. (We assume the sorting is done by applying the same method to the smaller lists, unless they have size 1, in which case we do nothing.) What we do know is that any sensible way of dividing the list into two parts takes no more than some constant multiple of n time units (and might take no more than constant time if we do it by leaving the list in place and manipulating pointers) and that any sensible algorithm for merging two lists will take no more than some (other) constant multiple of n time units. Thus we know that if $T(n)$ is the amount of time it takes to apply merge sort to n data items, then there is a constant c (the sum of the two constant multiples we mentioned) such that

$$T(n) \leq 2T(n/2) + cn. \tag{5.6}$$

Thus real world problems often lead us to *recurrence inequalities* rather than recurrence equations. These are inequalities that state that $T(n)$ is less than or equal to some expression involving values of $T(m)$ for $m < n$. (We could also include inequalities with a greater than or equal to sign, but they do not arise in the applications we are studying.) A *solution* to a recurrence inequality is a function T that satisfies the inequality. For simplicity we will expand what we mean by the word recurrence to include either recurrence inequalities or recurrence equations.

In Recurrence 5.6 we are implicitly assuming that T is defined only on positive integer values and, since we said we divided the list into two equal parts each time, our analysis only makes sense if we assume that n is a power of 2.

Note that there are actually infinitely many solutions to Recurrence 5.6. (For example for any $c' < c$, the unique solution to

$$T(n) = \begin{cases} 2T(n/2) + c'n & \text{if } n \geq 2 \\ k & \text{if } n = 1 \end{cases}$$

satisfies Inequality 5.6 for any constant k.) This idea of infinitely many solutions for a recurrence inequality is analogous to the idea that $x - 3 = 0$ has one solution, but $x - 3 \leq 0$ has infinitely many solutions. Later in this section we shall see how to show that all the solutions to Recurrence 5.6 satisfy $T(n) = O(n \log_2 n)$. In other words, no matter how we sensibly implement merge sort, we have a $O(n \log_2 n)$ time bound on how long the merge sort process takes.

Exercise 5.3-1 Carefully prove by induction that for any function T defined on the non-negative powers of 2, if

$$T(n) \leq 2T(n/2) + cn$$

for some constant c, then $T(n) = O(n \log n)$.

A Wrinkle with induction

We can analyze recurrence inequalities via a recursion tree. The process is virtually identical to our previous use of recursion trees. We must, however, keep in mind that on each level, we are really computing an upper bound on the work done on that level. We can also use a variant of the method we used a few sections ago, guessing an upper bound and verifying by induction. We use this method for the recurrence in Exercise 5.3-1. Here we wish to show that $T(n) = O(n \log n)$. Using the definition of Big-O, we can see that we wish to show that $T(n) \leq kn \log n$ for some positive constant k (so long as n is larger than some value n_0).

We are going to do something you may find rather curious. We will consider the possibility that we have a value of k for which the inequality holds. Then in analyzing the consequences of this possibility, we will discover that there are assumptions that we need to make about k in order for such a k to exist. What we will really be doing is experimenting to see how we will need to choose k to make an inductive proof work.

We are given that $T(n) \leq 2T(n/2) + cn$ for all positive integers n that are powers of 2. We want to prove there is another positive real number $k > 0$ and an $n_0 > 0$ such that for $n > n_0$, $T(n) \leq kn \log n$. We cannot expect to have the inequality $T(n) \leq kn \log n$ hold for $n = 1$, because $\log 1 = 0$. To have $T(2) \leq k \cdot 2 \log 2 = k \cdot 2$, we must choose $k \geq \frac{T(2)}{2}$. This is the first assumption we must make about k. Our inductive hypothesis will be that if n is a power of 2 and m is a power of 2 with $2 < m < n$ then $T(m) \leq km \log m$. Then $n/2 < n$, and since n is a power of 2 greater than 2, we have that $n/2 \geq 2$, so $(n/2) \log n/2 \geq 2$. By the inductive hypothesis, $T(n/2) \leq k(n/2) \log n/2$. But then

$$\begin{aligned}
T(n) \leq 2T(n/2) + cn \quad &\leq \quad 2k\frac{n}{2}\log\frac{n}{2} + cn && (5.7)\\
&= \quad kn \log \frac{n}{2} + cn && (5.8)\\
&= \quad kn \log n - kn \log 2 + cn && (5.9)\\
&= \quad kn \log n - kn + cn. && (5.10)
\end{aligned}$$

This shows that we need to make another assumption about k, namely that $-kn + cn \leq 0$, or $k \geq c$. Then if both our assumptions about k are satisfied, we will have $T(n) < kn \log n$, and we can conclude by the principle of mathematical induction that for all $n > 1$ (so our n_0 is 2), $T(n) \leq kn \log n$, so that $T(n) = O(n \log n)$.

A full inductive proof that $T(n) = O(n \log n)$ is actually embedded in the discussion above, but since it might not appear to you to be a proof, below we will summarize our observations in a more traditional looking proof. However you should be aware that some authors and teachers prefer to write their proofs in a style that shows why we make the choices about k that we do, and so you should learn how to to read discussions like the one above as proofs.

We want to show that if $T(n) \leq T(n/2) + cn$, then $T(n) = O(n \log n)$. We are given a real number $c > 0$ such that for $T(n) \leq 2T(n/2) + cn$ for all $n > 1$. Choose k to be larger than or equal to $\frac{T(2)}{2}$ and c. Then

$$T(2) \leq k \cdot 2 \log 2$$

because $k \geq T(n_0)/2$ and $\log 2 = 1$. Now assume that $n > 2$ and assume that for m with $2 \leq m < n$, we have $T(m) \leq km \log m$. Since n is a power of 2, we have $n \geq 4$, so that $n/2$ is an

m with $2 \leq m < n$. Thus, by the inductive hypothesis,

$$T\left(\frac{n}{2}\right) \leq k\frac{n}{2}\log\frac{n}{2.}$$

Then by the recurrence,

$$\begin{aligned} T(n) &\leq 2k\frac{n}{2}\log\frac{n}{2} + cn \\ &= kn(\log n - 1) + cn \\ &= kn\log n + cn - kn \\ &\leq kn\log n, \end{aligned}$$

since $k \geq c$. Thus by the principle of mathematical induction, $T(n) \leq kn\log n$ for all $n > 2$, and therefore $T(n) = O(n\log n)$.

There are three things to note about this proof. First without the preceding discussion, the choice of k seems arbitrary. Second, without the preceeding discussion, the implicit choice of 2 for the n_0 in the big-O statement also seems arbitrary. Third, the constant k is chosen in terms of the previous constant c. Since c was given to us by the recurrence, it may be used in choosing the constant we use to prove a Big-O statement about solutions to the recurrence.

Further Wrinkles in Induction Proofs

Exercise 5.3-2 Show by induction that any solution $T(n)$ to the recurrence

$$T(n) \leq T(n/3) + cn$$

with n restricted to integer powers of 3 has $T(n) = O(n)$.

Exercise 5.3-3 Show by induction that any solution $T(n)$ to the recurrence

$$T(n) \leq 4T(n/2) + cn$$

with n restricted to integer powers of 2 has $T(n) = O(n^2)$.

In Exercise 5.3-2 we are given a constant c such that $T(n) \leq T(n/3) + cn$ if $n > 1$. Since we want to show that $T(n) = O(n)$, we want to find two more constants n_0 and k such that $T(n) \leq kn$ whenever $n > n_0$.

We will choose $n_0 = 1$ here. (This was not an arbitrary choice; it is based on observing that $T(1) \leq kn$ is not an impossible condition to satisfy when $n = 1$.) In order to have $T(n) \leq kn$ for $n = 1$, we must assume $k \geq T(1)$. Now assuming inductively that $T(m) \leq km$ when $1 \leq m < n$ we can write

$$\begin{aligned} T(n) &\leq T(n/3) + cn \\ &\leq k(n/3) + cn \\ &= kn + \left(c - \frac{2k}{3}\right)n \end{aligned}$$

Thus, as long as $c - \frac{2k}{3} \leq 0$, i.e. $k \geq \frac{3}{2}c$, we may conclude by mathematical induction that $T(n) \leq kn$ for all $n \geq 1$. Again, the elements of an inductive proof are in the preceding discussion. Again you should try to learn how to read the argument we just finished as a valid inductive proof. However, we will now present something that looks more like an inductive proof.

We choose k to be the maximum of $T(1)$ and $3c/2$. To prove by induction that $T(x) \leq kx$ we begin by observing that $T(1) \leq k \cdot 1$. Next we assume that $n > 1$ and assume inductively that for m with $1 \leq m < n$ we have $T(m) \leq km$. Now we may write

$$T(n) \leq T(n/3) + cn \leq kn/3 + cn = kn + (c - 2k/3)n \leq kn,$$

because we chose k to be at least as large as $3c/2$, making $c - 2k/3$ negative or zero. Thus by the principle of mathematical induction we have $T(n) \leq kn$ for all $n \geq 1$ and so $T(n) = O(n)$.

Now let's analyze Exercise 5.3-3. We won't dot all the i's and cross all the t's here because there is only one major difference between this exercise and the previous one. We wish to prove there are an n_0 and a k such that $T(n) \leq kn^2$ for $n > n_0$. Assuming that we have chosen k so that the base case holds, we can bound $T(n)$ inductively by assuming that $T(m) \leq km^2$ for $m < n$ and reasoning as follows:

$$\begin{aligned}
T(n) &\leq 4T\left(\frac{n}{2}\right) + cn \\
&\leq 4\left(k\left(\frac{n}{2}\right)^2\right) + cn \\
&= 4\left(\frac{kn^2}{4}\right) + cn \\
&= kn^2 + cn.
\end{aligned}$$

To proceed as before, we would like to choose a value of k so that $cn \leq 0$. But we see that we have a problem because both c and n are always positive! What went wrong? We have a statement that we know is true, and we have a proof method (induction) that worked nicely for similar problems.

The problem is that, while the statement is true, it is too weak to be proved by induction. To have a chance of making the inductive proof work, we will have to make an inductive hypothesis that puts some sort of negative quantity, say a term like $-kn$, into the last line of our display above. Let's see if we can prove something that is actually stronger than we were originally trying to prove, namely that for some positive constants k_1 and k_2, $T(n) \leq k_1n^2 - k_2n$. Now proceeding as before, we get

$$\begin{aligned}
T(n) &\leq 4T(n/2) + cn \\
&\leq 4\left(k_1\left(\frac{n}{2}\right)^2 - k_2\left(\frac{n}{2}\right)\right) + cn \\
&= 4\left(\frac{k_1n^2}{4} - k_2\left(\frac{n}{2}\right)\right) + cn \\
&= k_1n^2 - 2k_2n + cn \\
&= k_1n^2 - k_2n + (c - k_2)n.
\end{aligned}$$

Now we have to make $(c - k_2)n \leq 0$ for the last line to be at most $k_1n^2 - k_2n$, and so we just choose $k_2 \geq c$ (and greater than whatever we need in order to make a base case work). Since $T(n) \leq k_1n^2 - k_2n$ for some constants k_1 and k_2, then $T(n) = O(n^2)$.

At first glance, this approach seems paradoxical: why is it easier to prove a stronger statement than it is to prove a weaker one? This phenomenon happens often in mathematics: a stronger statement is often easier to prove than a weaker one. It happens particularly often when using induction. Think carefully about an inductive proof where you have assumed that a bound holds for values smaller than n and you are trying to prove a statement for n. You use the bound you have assumed for smaller values to help prove the bound for n. Thus if the bound you used for smaller values is actually weak, then that is hindering you in proving the bound for n. In other words when you want to prove something about $p(n)$ you are using $p(1) \wedge \ldots \wedge p(n-1)$. Thus if these are stronger, they will be of greater help in proving $p(n)$. In the case above, the problem was that these values, $p(1), \ldots, p(n-1)$ were too weak, and thus we were not able to prove $p(n)$. By using a stronger $p(1), \ldots, p(n-1)$, however, we were able to prove a stronger $p(n)$, one that implied the original $p(n)$ we wanted. When we give an induction proof in this way, we say that we are using a *stronger inductive hypothesis.*

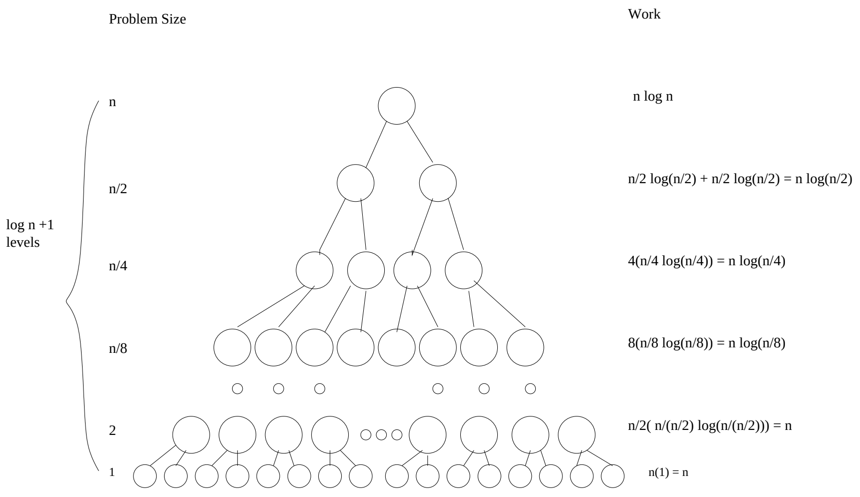
Dealing with functions other than n^c

Our statement of the Master Theorem involved a recursive term plus an added term that was $\Theta(n^c)$. Sometimes algorithmic problems lead us to consider other kinds of functions. The most common such is example is when that added function involves logarithms. For example, consider the recurrence:

$$T(n) = \begin{cases} 2T(n/2) + n\log n & \text{if } n > 1 \\ 1 & \text{if } n = 1, \end{cases}$$

where n is a power of 2. Just as before, we can draw a recursion tree; the whole methodology works, but our sums may be a little more complicated. The tree for this recurrence is shown in Figure 5.7.

Figure 5.7: The recursion tree for $T(n) = 2T(n/2) + n\log n$ if $n > 1$ and $T(1) = 1$.


Problem Size
Work
n
n log n
n/2
n/2 log(n/2) + n/2 log(n/2) = n log(n/2)
log n +1 levels
n/4
4(n/4 log(n/4)) = n log(n/4)
n/8
8(n/8 log(n/8)) = n log(n/8)
2
n/2(n/(n/2) log(n/(n/2))) = n
1
n(1) = n

This is similar to the tree for $T(n) = 2T(n/2)+n$, except that the work on level i is $n\log\left(\frac{n}{2^i}\right)$ for $i \geq 2$, and for level 1 it is n, the number of subproblems, times 1. Thus if we sum the work per level we get

$$\begin{aligned}
\sum_{i=0}^{\log_2 n-1} n\log\left(\frac{n}{2^i}\right)+n &= n\sum_{i=0}^{\log_2 n-1}\log\left(\frac{n}{2^i}\right)+n \\
&= n\sum_{i=0}^{\log_2 n-1}(\log n-\log 2^i)+O(n) \\
&= n\left(\sum_{i=0}^{\log_2 n-1}\log n-\sum_{i=0}^{\log_2 n-1} i\right)+n \\
&= n\left((\log_2 n)(\log_2 n)-\frac{(\log_2 n)(\log_2 n-1)}{2}\right)+n \\
&= O(n\log^2 n)\ .
\end{aligned}$$

A bit of mental arithmetic in the second last line of our equations shows that the $\log^2 n$ will not cancel out, so our solution is in fact $\Theta(n\log^2 n)$.

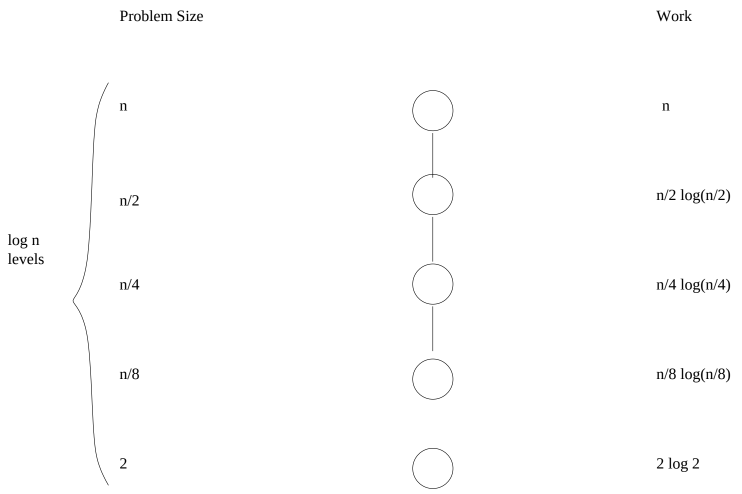
Exercise 5.3-4 Find the best big-O bound you can on the solution to the recurrence

$$T(n)=\begin{cases} T(n/2)+n\log n & \text{if } n>1 \\ 1 & \text{if } n=1, \end{cases}$$

assuming n is a power of 2. Is this bound a big-Θ bound?

The tree for this recurrence is in Figure 5.8

Figure 5.8: The recursion tree for the recurrence $T(n)=T(n/2)+n\log n$ if $n>1$ and $T(1)=1$.



Notice that the work done at the bottom nodes of the tree is determined by the statement $T(1)=1$ in our recurrence, not by the recursive equation. Summing the work, we get

$$\begin{aligned}
\sum_{i=0}^{\log_2 n-1} \frac{n}{2^i}\log\left(\frac{n}{2^i}\right)+1 &= n\left(\sum_{i=0}^{\log_2 n-1}\frac{1}{2^i}(\log n-\log 2^i)\right)+1 \\
&= n\left(\sum_{i=0}^{\log_2 n-1}\left(\frac{1}{2}\right)^i(\log(n)-i)\right)+1 \\
&\le n\left(\log n\sum_{i=0}^{\log_2 n-1}\left(\frac{1}{2}\right)^i\right)+1 \\
&\le n(\log n)(2)+1 \\
&= O(n\log n).
\end{aligned}$$

In fact we have $T(n) = \Theta(n \log n)$, because the largest term in the sum in our second line of equations is $\log(n)$, and none of the terms in the sum are negative. This means that n times the sum is at least $n \log n$.

Removing Ceilings and Using Powers of b.

We showed that in our versions of the master theorem, we could ignore ceilings and assume our variables were powers of b. It might appear that these two theorems do not apply to the more general functions we have studied in this section any more than the master theorem does. However, they actually only depend on properties of the powers n^c and not the three different kinds of cases, so it turns out we can extend them.

Notice that $(xb)^c = b^c x^c$, and this proportionality holds for all values of x with constant of proportionality b^c. Putting this just a bit less precisely, we can write $(xb)^c = O(x^c)$. This suggests that we might be able to obtain Big-Θ bounds on $T(n)$ when T satisfies a recurrence of the form

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

with $f(nb) = \Theta(f(n))$, and we might be able to obtain Big-O bounds on T when T satisfies a recurrence of the form

$$T(n) \le aT(n/b) + f(n)$$

with $f(nb) = O(f(n))$. But are these conditions satisfied by any functions of practical interest? Yes. For example if $f(x) = \log(x)$, then

$$f(bx) = \log(b) + \log(x) = \Theta(\log(x)).$$

Exercise 5.3-5 Show that if $f(x) = x^2 \log x$, then $f(bx) = \Theta(f(x))$.

Exercise 5.3-6 If $f(x) = 3^x$ and $b = 2$, is $f(bx) = \Theta(f(x))$? Is $f(b(x)) = O(f(x))$?

Notice that if $f(x) = x^2 \log x$, then

$$f(bx) = (bx)^2 \log bx = b^2x^2(\log b + \log x) = \Theta(x^2 \log x).$$

However, if $f(x) = 3^x$, then
$$f(2x) = 3^{2x} = (3^x)^2 = 3^x \cdot 3^x,$$
and there is no way that this can be less than or equal to a constant multiple of 3^x, so it is neither $\Theta(3^x)$ nor $O(3^x)$. Our exercises suggest the kinds of functions that satisfy the condition $f(bx) = O(f(x))$ might include at least some of the kinds of functions of x which arise in the study of algorithms. They certainly include the power functions and thus polynomial functions and root functions, or functions bounded by such functions.

There was one other property of power functions n^c that we used implicitly in our discussions of removing floors and ceilings and assuming our variables were powers of b. Namely, if $x > y$ (and $c \geq 0$) then $x^c \geq y^c$. A function f from the real numbers to the real numbers is called *(weakly) increasing* if whenever $x > y$, then $f(x) \geq f(y)$. Functions like $f(x) = \log x$ and $f(x) = x \log x$ are increasing functions. On the other hand, the function defined by

$$f(x) = \begin{cases} x & \text{if } x \text{ is a power of } b \\ x^2 & \text{otherwise} \end{cases}$$

is not increasing even though it does satisfy the condition $f(bx) = \Theta(f(x))$.

Theorem 5.8 *Theorems 5.2 and 5.3 apply to recurrences in which the x^c term is replaced by an increasing function for which $f(bx) = \Theta(f(x))$.*

Proof: We iterate the recurrences in the same way as in the proofs of the original theorems, and find that the condition $f(bx) = \Theta(f(x))$ applied to an increasing function gives us enough information to again bound the solution to one kind of recurrence above and below with a multiple of the solution of the other kind. The details are similar to those in the original proofs so we omit them. ■

In fact there are versions of Theorems 5.2 and 5.3 for recurrence inequalities also. The proofs involve a similar analysis of iterated recurrences or recursion trees, and so we omit them.

Theorem 5.9 *Let a and b be positive real numbers with $b > 2$ and let $f : R^+ \to R^+$ be an increasing function such that $f(bx) = O(f(x))$. Then every solution $t(x)$ to the recurrence*

$$t(x) \leq \begin{cases} at(x/b) + f(x) & \text{if } x \geq b \\ c & \text{if } 1 \leq x < b, \end{cases}$$

where a, b, and c are constants, satisfies $t(x) = O(h(x))$ if and only if every solution $T(x)$ to the recurrence

$$T(n) \leq \begin{cases} aT(n/b) + f(n) & \text{if } n > 10 \\ d & \text{if } n = 1, \end{cases}$$

where n is restricted to powers of b, satisfies $T(n) = O(h(n))$.

Theorem 5.10 *Let a and b be positive real numbers with $b \geq 2$ and let $f : R^+ \to R^+$ be an increasing function such that $f(bx) = O(f(x))$. Then every solution $T(n)$ to the recurrence*

$$T(n) \leq \begin{cases} at(\lceil n/b \rceil) + f(n) & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

satisfies $T(n) = O(h(n))$ if and only if every solution $t(x)$ to the recurrence

$$t(x) \leq \begin{cases} aT(x/b) + f(x) & \textit{if } x \geq b \\ d & \textit{if } 1 \leq x < b, \end{cases}$$

satisfies $t(x) = O(h(x))$.

Important Concepts, Formulas, and Theorems

1. *Recurrence Inequality.* Recurrence inequalities are inequalities that state that $T(n)$ is less than or equal to some expression involving values of $T(m)$ for $m < n$. A *solution* to a recurrence inequality is a function T that satisfies the inequality.

2. *Recursion Trees for Recurrence Inequalities.* We can analyze recurrence inequalities via a recursion tree. The process is virtually identical to our previous use of recursion trees. We must, however, keep in mind that on each level, we are really computing an upper bound on the work done on that level.

3. *Discovering Necessary Assumptions for an Inductive Proof.* If we are trying to prove a statement that there is a value k such that an inequality of the form $f(n) \leq kg(n)$ or some other statement that involves the parameter k is true, we may start an inductive proof without knowing a value for k and determine conditions on k by assumptions that we need to make in order for the inductive proof to work. When written properly such an exploration is actually a valid proof.

4. *Making a Stronger Inductive Hypothesis.* If we are trying to prove by induction a statement of the form $p(n) \Rightarrow q(n)$ and we have a statement $s(n)$ such that $s(n) \Rightarrow q(n)$, it is sometimes useful to try to prove the statement $p(n) \Rightarrow s(n)$. This process is known as proving a *stronger* statement or making an *stronger* inductive hypothesis. It sometimes works because it gives us an inductive hypothesis which suffices to prove the stronger statement even though our original statement $q(n)$ did not give an inductive hypothesis sufficient to prove the original statement. However we must be careful in our choice of $s(n)$, because we have to be able to succeed in proving $p(n) \Rightarrow s(n)$.

5. *When the Master Theorem does not Apply.* To deal with recurrences of the form

$$T(n) = \begin{cases} aT(\lceil n/b \rceil) + f(n) & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

where $f(n)$ is not $\Theta(n^c)$, recursion trees are an appropriate tool even though the Master Theorem does not apply. The same holds for recurrence inequalities.

6. *Increasing function.* A function $f : R \to R$ is said to be *(weakly) increasing* if whenever $x > y$, $f(x) \geq f(y)$

7. *Removing Floors and Ceilings when the Master Theorem does not Apply.* To deal with big-Θ bounds with recurrences of the form

$$T(n) = \begin{cases} aT(\lceil n/b \rceil) + f(n) & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

where $f(n)$ is not $\Theta(n^c)$, we may remove floors and ceilings and replace n by powers of b if f is increasing and satisfies the condition $f(nb) = Theta(f(n))$. To deal with big-O bounds for a similar recurrence inequality we may remove floors and ceilings if f is increasing and satisfies the condition that $f(nb) = O(f(n))$.

Problems

1. (a) Find the best big-O upper bound you can to any solution to the recurrence

$$T(n) = \begin{cases} 4T(n/2) + n\log n & \text{if } n > 1 \\ 1 & \text{if } n = 1. \end{cases}$$

   (b) Assuming that you were able to guess the result you got in part (a), prove, by induction, that your answer is correct.

2. Is the big-O upper bound in the previous exercise actually a big-Θ bound?

3. Show by induction that

$$T(n) = \begin{cases} 8T(n/2) + n\log n & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

   has $T(n) = O(n^3)$ for any solution $T(n)$.

4. Is the big-O upper bound in the previous problem actually a big-Θ upper bound?

5. Show by induction that any solution to a recurrence of the form

$$T(n) \leq 2T(n/3) + c\log_3 n$$

   is $O(n\log_3 n)$. What happens if you replace 2 by 3 (explain why)? Would it make a difference if we used a different base for the logarithm (only an intuitive explanation is needed here).

6. What happens if you replace the 2 in Problem 5 by 4? (Hint: one way to attack this is with recursion trees.)

7. Is the big-O upper bound in Problem 5 actually a big Θ upper bound?

8. Give an example (different from any in the text) of a function for which $f(bx) = O(f(x))$. Give an example (different from any in the text) of a function for which $f(bx)$ is not $O(f(x))$.

9. Give the best big O upper bound you can for the recurrence $T(n) = 2T(n/3 - 3) + n$, and then prove by induction that your upper bound is correct.

10. Find the best big-O upper bound you can to any solution to the recurrence defined on nonnegative integer powers of two by

$$T(n) \leq 2T(\lceil n/2 \rceil + 1) + cn.$$

    Prove, by induction, that your answer is correct.