Chapter 4

Induction, Recursion, and Recurrences

4.1 Mathematical Induction

Smallest Counter-Examples

In Section 3.3, we saw one way of proving statements about infinite universes: we considered a "generic" member of the universe and derived the desired statement about that generic member. When our universe is the universe of integers, or is in a one-to-one correspondence with the integers, there is a second technique we can use.

Recall our our proof of Euclid's Division Theorem (Theorem 2.12), which says that for each pair (m, n) of positive integers, there are nonnegative integers q and r such that $m = nq + r$ and $0 \leq r < n$. For the purpose of a proof by contradiciton, we assumed that the statement was fales. Then we said the following. "Among all pairs (m, n) that make it false, choose the smallest m that makes it false. We cannot have $m < n$ because then the statement would be true with $q = 0$ and $r = m$, and we cannot have $m = n$ because then the statement is true with $q = 1$ and $r = 0$. This means $m - n$ is a positive number smaller than m. We assumed that m was the smallest value that made the theorem false, and so the theorem must be true for the pair $(m - n, n)$. Therefore, there must exist a q' and r' such that

$$m - n = q'n + r', \text{ with } 0 \leq r' < n.$$

Thus $m = (q' + 1)n + r'$. Now, by setting $q = q' + 1$ and $r = r'$, we can satisfy the theorem for the pair (m, n), contradicting the assumption that the statement is false. Thus the only possibility is that the statement is true."

Focus on the sentences "This means $m - n$ is a positive number smaller than m. We assumed that m was the smallest value that made the theorem false, and so the theorem must be true for the pair $(m - n, n)$. Therefore, there must exist a q' and r' such that

$$m - n = q'n + r', \text{ with } 0 \leq r' < n.$$

Thus $m = (q' + 1)n + r'$." To analyze these sentences, let $p(m, n)$ denote the statement "there are nonnegative integers q and r with $0 \leq r < n$ such that $m = nq + r$" The quoted sentences

we focused on provide a proof that $p(m-n,n) \Rightarrow p(m,n)$. This implication is the crux of the proof. Let us give an analysis of the proof that shows the pivotal role of this impliction.

- We assumed a counter-example with a smallest m existed.
- Then using the fact that $p(m',n)$ had to be true for every m' smaller than m, we chose $m' = m-n$, and observed that $p(m',n)$ had to be true.
- Then we used the implication $p(m-n,n) \Rightarrow p(m,n)$ to conclude the truth of $p(m,n)$.
- But we had assumed that $p(m,n)$ was false, so this is the assumption we contradicted in the proof by contradiction.

Exercise 4.1-1 In Chapter 1 we learned Gauss's trick for showing that for all positive integers n,

$$1+2+3+4+...+n = \frac{n(n+1)}{2} \ . \tag{4.1}$$

Use the technique of asserting that if there is a counter-example, there is a smallest counter-example and deriving a contradiction to prove that the sum is $n(n+1)/2$. What implication did you have to prove in the process?

Exercise 4.1-2 For what values of $n \geq 0$ do you think $2^{n+1} \geq n^2+2$? Use the technique of asserting there is a smallest counter-example and deriving a contradiction to prove you are right. What implication did you have to prove in the process?

Exercise 4.1-3 For what values of $n \geq 0$ do you think $2^{n+1} \geq n^2+3$? Is it possible to use the technique of asserting there is a smallest counter-example and deriving a contradiction to prove you are right? If so, do so and describe the implication you had to prove in the process. If not, why not?

Exercise 4.1-4 Would it make sense to say that if there is a counter example there is a *largest* counter-example and try to base a proof on this? Why or why not?

In Exercise 4.1-1, suppose the formula for the sum is false. Then there must be a smallest n such that the formula does not hold for the sum of the first n positive integers. Thus for any positive integer i smaller than n,

$$1+2+\cdots+i = \frac{i(i+1)}{2}. \tag{4.2}$$

Because $1 = 1 \cdot 2/2$, Equation 4.1 holds when $n=1$, and therefore the smallest counterexample is not when $n=1$. So $n>1$, and $n-1$ is one of the positive integers i for which the formula holds. Substituting $n-1$ for i in Equation 4.2 gives us

$$1+2+\cdots+n-1 = \frac{(n-1)n}{2}.$$

Adding n to both sides gives

$$\begin{aligned} 1+2+\cdots+n-1+n &= \frac{(n-1)n}{2}+n \\ &= \frac{n^2-n+2n}{2} \\ &= \frac{n(n+1)}{2} \ . \end{aligned}$$

Thus n is not a counter-example after all, and therefore there is no counter-example to the formula. Thus the formula holds for all positive integers n. Note that the crucial step was proving that $p(n-1) \Rightarrow p(n)$, where $p(n)$ is the formula

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

In Exercise 4.1-2, let $p(n)$ be the statement that $2^{n+1} \geq n^2 + 2$. Some experimenting with small values of n leads us to believe this statement is true for all nonnegative integers. Thus we want to prove $p(n)$ is true for all nonnegative integers n. To do so, we assume that the statement that "$p(n)$ is true for all nonnegative integers n" is false. When a "for all" statement is false, there must be some n for which it is false. Therefore, there is some smallest nonnegative integer n so that $2^{n+1} \not\geq n^2 + 2$. Assume now that n has this value. This means that for all nonnegative integers i with $i < n$, $2^{i+1} \geq i^2 + 2$. Since we know from our experimentation that $n \neq 0$, we know $n - 1$ is a nonnegative integer less than n, so using $n - 1$ in place of i, we get

$$2^{(n-1)+1} \geq (n-1)^2 + 2,$$

or

$$\begin{aligned} 2^n &\geq n^2 - 2n + 1 + 2 \\ &= n^2 - 2n + 3. \end{aligned} \tag{4.3}$$

From this we want to draw a contradiction, presumably a contradiction to $2^{n+1} \not\geq n^2 + 2$.

To get the contradiction, we want to convert the left-hand side of Equation 4.3 to 2^{n+1}. For this purpose, we multiply both sides by 2, giving

$$\begin{aligned} 2^{n+1} &= 2 \cdot 2^n \\ &\geq 2n^2 - 4n + 6\ . \end{aligned}$$

You may have gotten this far and wondered "What next?" Since we want to obtain a contradiction, we want to convert the right hand side into something like $n^2 + 2$. More precisely, we will convert the right-hand side into $n^2 + 2$ plus an additional term. If we can show that the additional term is nonnegative, the proof will be complete. Thus we write

$$\begin{aligned} 2^{n+1} &\geq 2n^2 - 4n + 6 \\ &= (n^2 + 2) + (n^2 - 4n + 4) \\ &= n^2 + 2 + (n-2)^2 \\ &\geq n^2 + 2\ , \end{aligned} \tag{4.4}$$

since $(n-2)^2 \geq 0$. This is a contradiction, so there must not have been a smallest counter-example, and thus there must be no counter-example. Therefore $2^n \geq n^2 + 2$ for all nonnegative integers n.

What implication did we prove above? Let $p(n)$ stand for $2^{n+1} \geq n^2 + 2$. Then in Equations 4.3 and 4.4 we proved that $p(n-1) \Rightarrow p(n)$. Notice that at one point in our proof we had to note that we had considered the case with $n = 0$ already. Although we have given a proof by smallest counterexample, it is natural to ask whether it would make more sense to try to prove the statement directly. Would it make more sense to forget about the contradiction now that we

have $p(n-1) \Rightarrow p(n)$ in hand and just observe that $p(0)$ and $p(n-1) \Rightarrow p(n)$ implies $p(1)$,that $p(1)$ and $p(n-1) \Rightarrow p(n)$ implies $p(2)$, and so on so that we have $p(k)$ for every k? We will address this question shortly.

Now let's consider Exercise 4.1-3. Notice that $2^{n+1} \not> n^2+3$ for $n = 0$ and 1, but $2^{n+1} > n^2+3$ for any larger n we look at at. Let us try to prove that $2^{n+1} > n^2 + 3$ for $n \geq 2$. We now let $p'(n)$ be the statement $2^{n+1} > n^2 + 3$. We can easily prove $p'(2)$: since $8 = 2^3 \geq 2^2 + 3 = 7$. Now suppose that among the integers larger than 2 there is a counter-example m to $p'(n)$. That is, suppose that there is an m such that $m > 2$ and $p'(m)$ is false. Then there is a smallest such m, so that for k between 2 and $m-1$, $p'(k)$ is true. If you look back at your proof that $p(n-1) \Rightarrow p(n)$, you will see that, when $n \geq 2$, essentially the same proof applies to p' as well. That is, with very similar computations we can show that $p'(n-1) \Rightarrow p'(n)$, so long as $n \geq 2$. Thus since $p'(m-1)$ is true, our implication tells us that $p'(m)$ is also true. This is a contradiction to our assumption that $p'(m)$ is false. therefore, $p'(m)$ is true. Again, we could conclude from $p'(2)$ and $p'(2) \Rightarrow p'(3)$ that $p'(3)$ is true, and similarly for $p'(4)$, and so on. The implication we had to prove was $p'(n-1) \Rightarrow p'(n)$.

For Exercise 4.1-4 if we have a counter-example to a statement $p(n)$ about an integer n, this means that there is an m such that $p(m)$ is false. To find a smallest counter example we would need to examine $p(0)$, $p(1)$, ..., perhaps all the way up to $p(m)$ in order to find a smallest counter-example, that is a smallest number k such that $p(k)$ is false. Since this involves only a finite number of cases, it makes sense to assert that there is a smallest counter-example. But, in answer to Exercise 4.1-4, it does not make sense to assert that there is a largest counter example, because there are infinitely many cases n that we would have to check in hopes if finding a largest one, and thus we might never find one. Even if we found one, we wouldn't be able to figure out that we had a largest counter-example just by checking larger and larger values of n, because we would never run out of values of n to check. Sometimes there is a largest counter-example, as in Exercise 4.1-3. To prove this, though, we didn't check all cases. Instead, based on our intuition, we guessed that the largest counter example was $n = 1$. Then we proved that we were right by showing that among numbers greater than or equal to two, there is no smallest counter-example. Sometimes there is no largest counter example n to a statement $p(n)$; for example $n^2 < n$ is false for all all integers n, and therefore there is no largest counter-example.

The Principle of Mathematical Induction

It may seem clear that repeatedly using the implication $p(n-1) \Rightarrow p(n)$ will prove $p(n)$ for all n (or all $n \geq 2$). That observation is the central idea of the Principle of Mathematical Induction, which we are about to introduce. In a theoretical discussion of how one constructs the integers from first principles, the principle of mathematical induction (or the equivalent principle that every set of nonnegative integers has a smallest element, thus letting us use the "smallest counter-example" technique) is one of the first principles we assume. The principle of mathematical induction is usually described in two forms. The one we have talked about so far is called the "weak form." It applies to statements about integers n.

The Weak Principle of Mathematical Induction. If the statement $p(b)$ is true, and the statement $p(n-1) \Rightarrow p(n)$ is true for all $n > b$, then $p(n)$ is true for all integers $n \geq b$.

Suppose, for example, we wish to give a direct inductive proof that $2^{n+1} > n^2 + 3$ for $n \geq 2$. We would proceed as follows. (The material in square brackets is not part of the proof; it is a

running commentary on what is going on in the proof.)

> We shall prove by induction that $2^{n+1} > n^2 + 3$ for $n \geq 2$. First, $2^{2+1} = 2^3 = 8$, while $2^2 + 3 = 7$. [We just proved $p(2)$. We will now proceed to prove $p(n-1) \Rightarrow p(n)$.] Suppose now that $n > 2$ and that $2^n > (n-1)^2 + 3$. [We just made the hypothesis of $p(n-1)$ in order to use Rule 8 of our rules of inference.]
>
> Now multiply both sides of this inequality by 2, giving us
>
> $$\begin{aligned} 2^{n+1} &> 2(n^2 - 2n + 1) + 6 \\ &= n^2 + 3 + n^2 - 4n + 4 + 1 \\ &= n^2 + 3 + (n-2)^2 + 1 \ . \end{aligned}$$
>
> Since $(n-2)^2 + 1$ is positive for $n > 2$, this proves $2^{n+1} > n^2 + 3$. [We just showed that from the hypothesis of $p(n-1)$ we can derive $p(n)$. Now we can apply Rule 8 to assert that $p(n-1) \Rightarrow p(n)$.] Therefore
>
> $$2^n > (n-1)^2 + 3 \ \Rightarrow \ 2^{n+1} > n^2 + 3.$$
>
> Therefore by the principle of mathematical induction, $2^{n+1} > n^2 + 3$ for $n \geq 2$.

In the proof we just gave, the sentence "First, $2^{2+1} = 2^3 = 8$, while $2^2 + 3 = 7$" is called the *base case*. It consisted of proving that $p(b)$ is true, where in this case b is 2 and $p(n)$ is $2^{n+1} > n^2 + 3$. The sentence "Suppose now that $n > 2$ and that $2^n > (n-1)^2 + 3$." is called the *inductive hypothesis*. This is the assumption that $p(n-1)$ is true. In inductive proofs, we always make such a hypothesis[1] in order to prove the implication $p(n-1) \Rightarrow p(n)$. The proof of the implication is called the *inductive step* of the proof. The final sentence of the proof is called the *inductive conclusion*.

Exercise 4.1-5 Use mathematical induction to show that

$$1 + 3 + 5 + \cdots + (2k-1) = k^2$$

for each positive integer k.

Exercise 4.1-6 For what values of n is $2^n > n^2$? Use mathematical induction to show that your answer is correct.

For Exercise 4.1-5, we note that the formula holds when $k = 1$. Assume inductively that the formula holds when $k = n-1$, so that $1 + 3 + \cdots + (2n-3) = (n-1)^2$. Adding $2n-1$ to both sides of this equation gives

$$\begin{aligned} 1 + 3 + \cdots + (2n-3) + (2n-1) &= n^2 - 2n + 1 + 2n - 1 \\ &= n^2. \end{aligned} \tag{4.5}$$

Thus the formula holds when $k = n$, and so by the principle of mathematical induction, the formula holds for all positive integers k.

[1] At times, it might be more convenient to assume that $p(n)$ is true and use this assumption to prove that $p(n+1)$ is true. This proves the implication $p(n) \Rightarrow p(n+1)$, which lets us reason in the same way.

Notice that in our discussion of Exercise 4.1-5 we nowhere mentioned a statement $p(n)$. In fact, $p(n)$ is the statement we get by substituting n for k in the formula, and in Equation 4.5 we were proving $p(n-1) \Rightarrow p(n)$. Next notice that we did not explicitly say we were going to give a proof by induction; instead we told the reader when we were making the inductive hypothesis by saying "Assume inductively that ...." This convention makes the prose flow nicely but still tells the reader that he or she is reading a proof by induction. Notice also how the notation in the statement of the exercise helped us write the proof. If we state what we are trying to prove in terms of a variable other than n, say k, then we can assume that our desired statement holds when this variable (k) is $n-1$ and then prove that the statement holds when $k = n$. Without this notational device, we have to either mention our statement $p(n)$ explicitly, or avoid any discussion of substituting values into the formula we are trying to prove. Our proof above that $2^{n+1} > n^2 + 3$ demonstrates this last approach to writing an inductive proof in plain English. This is usually the "slickest" way of writing an inductive proof, but it is often the hardest to master. We will use this approach first for the next exercise.

For Exercise 4.1-6 we note that $2 = 2^1 > 1^2 = 1$, but then the inequality fails for $n = 2, 3, 4$. However, $32 > 25$. Now we assume inductively that for $n > 5$ we have $2^{n-1} > (n-1)^2$. Multiplying by 2 gives us

$$\begin{aligned} 2^n > 2(n^2 - 2n + 1) &= n^2 + n^2 - 4n + 2 \\ &> n^2 + n^2 - n \cdot n \\ &= n^2 \ , \end{aligned}$$

since $n > 5$ implies that $-4n > -n \cdot n$. (We also used the fact that $n^2+n^2-4n+2 > n^2+n^2-4n$.) Thus by the principle of mathematical induction, $2^n > n^2$ for all $n \geq 5$.

Alternatively, we could write the following. Let $p(n)$ denote the inequality $2^n > n^2$. Then $p(5)$ is true because $32 > 25$. Assume that $n > 5$ and $p(n-1)$ is true. This gives us $2^{n-1} > (n-1)^2$. Multiplying by 2 gives

$$\begin{aligned} 2^n &> 2(n^2 - 2n + 1) \\ &= n^2 + n^2 - 4n + 2 \\ &> n^2 + n^2 - n \cdot n \\ &= n^2 \ , \end{aligned}$$

since $n > 5$ implies that $-4n > -n \cdot n$. Therefore $p(n-1) \Rightarrow p(n)$. Thus by the principle of mathematical induction, $2^n > n^2$ for all $n \geq 5$.

Notice how the "slick" method simply assumes that the reader knows we are doing a proof by induction from our "Assume inductively...," and mentally supplies the appropriate $p(n)$ and observes that we have proved $p(n-1) \Rightarrow p(n)$ at the right moment.

Here is a slight variation of the technique of changing variables. To prove that $2^n > n^2$ when $n \geq 5$, we observe that the inequality holds when $n = 5$ since $32 > 25$. Assume inductively that the inequality holds when $n = k$, so that $2^k > k^2$. Now when $k \geq 5$, multiplying both sides of this inequality by 2 yields

$$\begin{aligned} 2^{k+1} > 2k^2 &= k^2 + k^2 \\ &\geq k^2 + 5k \\ &> k^2 + 2k + 1 \\ &= (k+1)^2 \ , \end{aligned}$$

since $k \geq 5$ implies that $k^2 \geq 5k$ and $5k = 2k+3k > 2k+1$. Thus by the principle of mathematical induction, $2^n > n^2$ for all $n \geq 5$.

This last variation of the proof illustrates two ideas. First, there is no need to save the name n for the variable we use in applying mathematical induction. We used k as our "inductive variable" in this case. Second, as suggested in a footnote earlier, there is no need to restrict ourselves to proving the implication $p(n-1) \Rightarrow p(n)$. In this case, we proved the implication $p(k) \Rightarrow p(k+1)$. Clearly these two implications are equivalent as n ranges over all integers larger than b and as k ranges over all integers larger than or equal to b.

Strong Induction

In our proof of Euclid's division theorem we had a statement of the form $p(m, n)$ and, assuming that it was false, we chose a smallest m such that $p(m, n)$ is false for some n. This meant we could assume that $p(m', n)$ is true for **all** $m' < m$, and we needed this assumption, because we ended up showing that $p(m-n, n) \Rightarrow p(m, n)$ in order to get our contradiction. This situation differs from the examples we used to introduce mathematical induction, for in those we used an implication of the form $p(n-1) \Rightarrow p(n)$. The essence of our method in proving Euclid's division theorem is that we have a statement $q(k)$ we want to prove. We suppose it is false, so that there must be a smallest k for which $q(k)$ is false. This means we may assume $q(k')$ is true for **all** k' in the universe of q with $k' < k$. We then use this assumption to derive a proof of $q(k)$, thus generating our contradiction.

Again, we can avoid the step of generating a contradiction in the following way. Suppose first we have a proof of $q(0)$. Suppose also that we have a proof that

$$q(0) \wedge q(1) \wedge q(2) \wedge \ldots \wedge q(k-1) \Rightarrow q(k)$$

for all k larger than 0. Then from $q(0)$ we can prove $q(1)$, from $q(0) \wedge q(1)$ we can prove $q(2)$, from $q(0) \wedge q(1) \wedge q(2)$ we can prove $q(3)$ and so on, giving us a proof of $q(n)$ for any n we desire. This is another form of the mathematical induction principle. We use it when, as in Euclid's division theorem, we can get an implication of the form $q(k') \Rightarrow q(k)$ for *some* $k' < k$ or when we can get an implication of the form $q(0) \wedge q(1) \wedge q(2) \wedge \ldots \wedge q(k-1) \Rightarrow q(k)$. (As is the case in Euclid's division theorem, we often don't really know what the k' is, so in these cases the first kind of situation is really just a special case of the second. Thus, we do not treat the first of the two implications separately.) We have described the method of proof known as the Strong Principle of Mathematical Induction.

The Strong Principle of Mathematical Induction. If the statement $p(b)$ is true, and the statement $p(b) \wedge p(b+1) \wedge \ldots \wedge p(n-1) \Rightarrow p(n)$ is true for all $n > b$, then $p(n)$ is true for all integers $n \geq b$.

Exercise 4.1-7 Prove that every positive integer is either a power of a prime number or the product of powers of prime numbers.

In Exercise 4.1-7 we can observe that 1 is a power of a prime number; for example $1 = 2^0$. Suppose now we know that every number less than n is a power of a prime number or a product of powers of prime numbers. Then if n is not a prime number, it is a product of two smaller

numbers, each of which is, by our supposition, a power of a prime number or a product of powers of prime numbers. Therefore n is a power of a prime number or a product of powers of prime numbers. Thus, by the strong principle of mathematical induction, every positive integer is a power of a prime number or a product of powers of prime numbers.

Note that there was no explicit mention of an implication of the form

$$p(b) \wedge p(b+1) \wedge \ldots \wedge p(n-1) \Rightarrow p(n) \ .$$

This is common with inductive proofs. Note also that we did not explicitly identify the base case or the inductive hypothesis in our proof. This is common too. Readers of inductive proofs are expected to recognize when the base case is being given and when an implication of the form $p(n-1) \Rightarrow p(n)$ or $p(b) \wedge p(b+1) \wedge \cdots \wedge p(n-1) \Rightarrow p(n)$ is being proved.

Mathematical induction is used frequently in discrete math and computer science. Many quantities that we are interested in measuring, such as running time, space, or output of a program, typically are restricted to positive integers, and thus mathematical induction is a natural way to prove facts about these quantities. We will use it frequently throughout this book. We typically will not distinguish between strong and weak induction, we just think of them both as induction. (In Problems 14 and 15 at the end of the section you will be asked to derive each version of the principle from the other.)

Induction in general

To summarize what we have said so far, a typical proof by mathematical induction showing that a statement $p(n)$ is true for all integers $n \geq b$ consists of three steps.

1. First we show that $p(b)$ is true. This is called "establishing a *base case*."

2. Then we show either that for all $n > b$, $p(n-1) \Rightarrow p(n)$, or that for all $n > b$,

   $$p(b) \wedge p(b+1) \wedge \ldots \wedge p(n-1) \Rightarrow p(n).$$

   For this purpose, we make either the *inductive hypothesis* of $p(n-1)$ or the *inductive hypothesis* $p(b) \wedge p(b+1) \wedge \ldots \wedge p(n-1)$. Then we derive $p(n)$ to complete the proof of the implication we desire, either $p(n-1) \Rightarrow p(n)$ or $p(b) \wedge p(b+1) \wedge \ldots \wedge p(n-1) \Rightarrow p(n)$.

   Instead we could

2′. show either that for all $n \geq b$, $p(n) \Rightarrow p(n+1)$ or

   $$p(b) \wedge p(b+1) \wedge \cdots \wedge p(n) \Rightarrow p(n+1).$$

   For this purpose, we make either the *inductive hypothesis* of $p(n)$ or the *inductive hypothesis* $p(b) \wedge p(b+1) \wedge \ldots \wedge p(n)$. Then we derive $p(n=1)$ to complete the proof of the implication we desire, either $p(n) \Rightarrow p(n=1)$ or $p(b) \wedge p(b+1) \wedge \ldots \wedge p(n) \Rightarrow p(n=1)$.

3. Finally, we conclude on the basis of the principle of mathematical induction that $p(n)$ is true for all integers n greater than or equal to b.

The second step is the core of an inductive proof. This is usually where we need the most insight into what we are trying to prove. In light of our discussion of Exercise 4.1-6, it should be clear that step $2'$ is simply a variation on the theme of writing an inductive proof.

It is important to realize that induction arises in some circumstances that do not fit the "pat" typical description we gave above. These circumstances seem to arise often in computer science. However, inductive proofs always involve three things. First we always need a base case or cases. Second, we need to show an implication that demonstrates that $p(n)$ is true given that $p(n')$ is true for some set of $n' < n$, or possibly we may need to show a set of such implications. Finally, we reach a conclusion on the basis of the first two steps.

For example, consider the problem of proving the following statement:

$$\sum_{i=0}^{n} \left\lfloor \frac{i}{2} \right\rfloor = \begin{cases} \frac{n^2}{4} & \text{if } n \text{ is even} \\ \frac{n^2-1}{4} & \text{if } n \text{ is odd} \end{cases} \tag{4.6}$$

In order to prove this, one must show that $p(0)$ is true, $p(1)$ is true, $p(n-2) \Rightarrow p(n)$ if n is odd, and that $p(n-2) \Rightarrow p(n)$, if n is even. Putting all these together, we see that our formulas hold for all $n \geq 0$. We can view this as either two proofs by induction, one for even and one for odd numbers, or one proof in which we have two base cases and two methods of deriving results from previous ones. This second view is more profitable, because it expands our view of what induction means, and makes it easier to find inductive proofs. In particular we could find situations where we have just one implication to prove but several base cases to check to cover all cases, or just one base case, but several different implications to prove to cover all cases.

Logically speaking, we could rework the example above so that it fits the pattern of strong induction. For example, when we prove a second base case, then we have just proved that the first base case implies it, because a true statement implies a true statement. Writing a description of mathematical induction that covers all kinds of base cases and implications one might want to consider in practice would simply give students one more unnecessary thing to memorize, so we shall not do so. However, in the mathematics literature and especially in the computer science literature, inductive proofs are written with multiple base cases and multiple implications with no effort to reduce them to one of the standard forms of mathematical induction. So long as it is possible to "cover" all the cases under consideration with such a proof, it can be rewritten as a standard inductive proof. Since readers of such proofs are expected to know this is possible, and since it adds unnecessary verbiage to a proof to do so, this is almost always left out.

Important Concepts, Formulas, and Theorems

1. *Weak Principle of Mathematical Induction.* The weak principle of mathematical induction states that

   > If the statement $p(b)$ is true, and the statement $p(n-1) \Rightarrow p(n)$ is true for all $n > b$, then $p(n)$ is true for all integers $n \geq b$.

2. *Strong Principle of Mathematical Induction.* The strong principle of mathematical induction states that

   > If the statement $p(b)$ is true, and the statement $p(b) \wedge p(b+1) \wedge \ldots \wedge p(n-1) \Rightarrow p(n)$ is true for all $n > b$, then $p(n)$ is true for all integers $n \geq b$.

3. *Base Case.* Every proof by mathematical induction, strong or weak, begins with a *base case* which establishes the result being proved for at least one value of the variable on which we are inducting. This base case should prove the result for the smallest value of the variable for which we are asserting the result. In a proof with multiple base cases, the base cases should cover all values of the variable which are not covered by the inductive step of the proof.

4. *Inductive Hypothesis.* Every proof by induction includes an inductive hypothesis in which we assume the result $p(n)$ we are trying to prove is true when $n = k - 1$ or when $n < k$ (or in which we assume an equivalent statement).

5. *Inductive Step.* Every proof by induction includes an inductive step in which we prove the implication that $p(k-1) \Rightarrow p(k)$ or the implication that $p(b) \wedge p(b+1) \wedge \cdots \wedge p(k-1) \Rightarrow p(k)$, or some equivalent implication.

6. *Inductive Conclusion.* A proof by mathematical induction should include, at least implicitly, a concluding statement of the form "Thus by the principle of mathematical induction ...," which asserts that by the principle of mathematical induction the result $p(n)$ which we are trying to prove is true for all values of n including and beyond the base case(s).

Problems

1. This exercise explores ways to prove that $\frac{2}{3} + \frac{2}{9} + \cdots + \frac{2}{3^n} = 1 - \left(\frac{1}{3}\right)^n$ for all positive integers n.

   (a) First, try proving the formula by contradiction. Thus you assume that there is some integer n that makes the formula false. Then there must be some smallest n that makes the formula false. Can this smallest n be 1? What do we know about $\frac{2}{3} + \frac{2}{9} + \cdots + \frac{2}{3^i}$ when i is a positive integer smaller than this smallest n? Is $n - 1$ a positive integer for this smallest n? What do we know about $\frac{2}{3} + \frac{2}{9} + \cdots + \frac{2}{3^{n-1}}$ for this smallest n? Write this as an equation and add $\frac{2}{3^n}$ to both sides and simplify the right side. What does this say about our assumption that the formula is false? What can you conclude about the truth of the formula? If $p(k)$ is the statement $\frac{2}{3} + \frac{2}{9} + \cdots + \frac{2}{3^k} = 1 - \left(\frac{1}{3}\right)^k$, what implication did we prove in the process of deriving our contradiction?

   (b) What is the base step in a proof by mathematical induction that $\frac{2}{3} + \frac{2}{9} + \cdots + \frac{2}{3^n} = 1 - \left(\frac{1}{3}\right)^n$ for all positive integers n? What would you assume as an inductive hypothesis? What would you prove in the inductive step of a proof of this formula by induction? Prove it. What does the principle of mathematical induction allow you to conclude? If $p(k)$ is the statement $\frac{2}{3} + \frac{2}{9} + \cdots + \frac{2}{3^k} = 1 - \left(\frac{1}{3}\right)^k$, what implication did we prove in the process of doing our proof by induction?

2. Use contradiction to prove that $1 \cdot 2 + 2 \cdot 3 + \cdots + n(n+1) = \frac{n(n+1)(n+2)}{3}$.

3. Use induction to prove that $1 \cdot 2 + 2 \cdot 3 + \cdots + n(n+1) = \frac{n(n+1)(n+2)}{3}$.

4. Prove that $1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$.

5. Write a careful proof of Euclid's division theorem using strong induction.

6. Prove that $\sum_{i=j}^{n} \binom{i}{j} = \binom{n+1}{j+1}$. As well as the inductive proof that we are expecting, there is a nice "story" proof of this formula. It is well worth trying to figure it out.

7. Prove that every number greater than 7 is a sum of a nonnegative integer multiple of 3 and a nonnegative integer multiple of 5.

8. The usual definition of exponents in an advanced mathematics course (or an intermediate computer science course) is that $a^0 = 1$ and $a^{n+1} = a^n \cdot a$. Explain why this defines a^n for all nonnegative integers n. Prove the rule of exponents $a^{m+n} = a^m a^n$ from this definition.

9. Our arguments in favor of the sum principle were quite intuitive. In fact the sum principle for n sets follows from the sum principle for two sets. Use induction to prove the sum principle for a union of n sets from the sum principle for a union of two sets.

10. We have proved that every positive integer is a power of a prime number or a product of powers of prime numbers. Show that this factorization is unique in the following sense: If you have two factorizations of a positive integer, both factorizations use exactly the same primes, and each prime occurs to the same power in both factorizations. For this purpose, it is helpful to know that if a prime divides a product of integers, then it divides one of the integers in the product. (Another way to say this is that if a prime is a factor of a product of integers, then it is a factor of one of the integers in the product.)

11. Prove that $1^4 + 2^4 + \cdots + n^4 = O(n^5 - n^4)$.

12. Find the error in the following "proof" that all positive integers n are equal. Let $p(n)$ be the statement that all numbers in an n-element set of positive integers are equal. Then $p(1)$ is true. Now assume $p(n-1)$ is true, and let N be the set of the first n integers. Let N' be the set of the first $n-1$ integers, and let N'' be the set of the last $n-1$ integers. Then by $p(n-1)$ all members of N' are equal and all members of N'' are equal. Thus the first $n-1$ elements of N are equal and the last $n-1$ elements of N are equal, and so all elements of N are equal. Thus all positive integers are equal.

13. Prove by induction that the number of subsets of an n-element set is 2^n.

14. Prove that the Strong Principle of Mathematical Induction implies the Weak Principle of Mathematical Induction.

15. Prove that the Weak Principal of Mathematical Induction implies the Strong Principal of Mathematical Induction.

16. Prove (4.6).

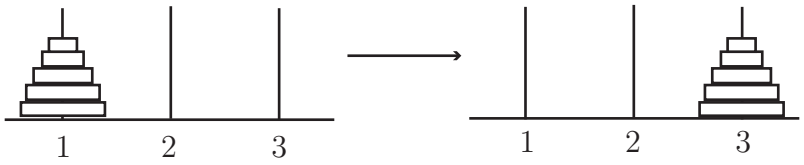
4.2 Recursion, Recurrences and Induction

Recursion

Exercise 4.2-1 Describe the uses you have made of recursion in writing programs. Include as many as you can.

Exercise 4.2-2 Recall that in the Towers of Hanoi problem we have three pegs numbered 1, 2 and 3, and on one peg we have a stack of n disks, each smaller in diameter than the one below it as in Figure 4.1. An allowable move consists of removing a disk

Figure 4.1: The Towers of Hanoi



from one peg and sliding it onto another peg so that it is not above another disk of smaller size. We are to determine how many allowable moves are needed to move the disks from one peg to another. Describe the strategy you have used or would use in a recursive program to solve this problem.

For the Tower of Hanoi problem, to solve the problem with no disks you do nothing. To solve the problem of moving all disks to peg 2, we do the following

1. (Recursively) solve the problem of moving $n-1$ disks from peg 1 to peg 3,
2. move disk n to peg 2,
3. (Recursively) solve the problem of moving $n-1$ disks on peg 3 to peg 2.

Thus if $M(n)$ is the number of moves needed to move n disks from peg i to peg j, we have

$$M(n) = 2M(n-1) + 1.$$

This is an example of a **recurrence equation** or **recurrence**. A recurrence equation for a function defined on the set of integers greater than or equal to some number b is one that tells us how to compute the nth value of a function from the $(n-1)$st value or some or all the values preceding n. To completely specify a function on the basis of a recurrence, we have to give enough information about the function to get started. This information is called the *initial condition (or the initial conditions)* (which we also call the *base case*) for the recurrence. In this case we have said that $M(0) = 0$. Using this, we get from the recurrence that $M(1) = 1$, $M(2) = 3$, $M(3) = 7$, $M(4) = 15$, $M(5) = 31$, and are led to guess that $M(n) = 2^n - 1$.

Formally, we write our recurrence and initial condition together as

$$M(n) = \begin{cases} 0 & \text{if } n = 0 \\ 2M(n-1) + 1 & \text{otherwise} \end{cases} \tag{4.7}$$

Now we give an inductive proof that our guess is correct. The base case is trivial, as we have defined $M(0) = 0$, and $0 = 2^0 - 1$. For the inductive step, we assume that $n > 0$ and $M(n-1) = 2^{n-1} - 1$. From the recurrence, $M(n) = 2M(n-1) + 1$. But, by the inductive hypothesis, $M(n-1) = 2^{n-1} - 1$, so we get that:

$$\begin{aligned} M(n) &= 2M(n-1) + 1 && (4.8)\\ &= 2(2^{n-1} - 1) + 1 && (4.9)\\ &= 2^n - 1. && (4.10) \end{aligned}$$

thus by the principle of mathematical induction, $M(n) = 2^n - 1$ for all nonnegative integers n.

The ease with which we solved this recurrence and proved our solution correct is no accident. Recursion, recurrences and induction are all intimately related. The relationship between recursion and recurrences is reasonably transparent, as recurrences give a natural way of analyzing recursive algorithms. Recursion and recurrences are abstractions that allow you to specify the solution to an instance of a problem of size n as some function of solutions to smaller instances. Induction also falls naturally into this paradigm. Here, you are deriving a statement $p(n)$ from statements $p(n')$ for $n' < n$. Thus we really have three variations on the same theme.

We also observe, more concretely, that the mathematical correctness of solutions to recurrences is naturally proved via induction. In fact, the correctness of recurrences in describing the number of steps needed to solve a recursive problem is also naturally proved by induction. The recurrence or recursive structure of the problem makes it straightforward to set up the induction proof.

First order linear recurrences

Exercise 4.2-3 The empty set ($\emptyset$) is a set with no elements. How many subsets does it have? How many subsets does the one-element set $\{1\}$ have? How many subsets does the two-element $\{1, 2\}$ set have? How many of these contain 2? How many subsets does $\{1, 2, 3\}$ have? How many contain 3? Give a recurrence for the number $S(n)$ of subsets of an n-element set, and prove by induction that your recurrence is correct.

Exercise 4.2-4 When someone is paying off a loan with initial amount A and monthly payment M at an interest rate of p percent, the total amount $T(n)$ of the loan after n months is computed by adding $p/12$ percent to the amount due after $n-1$ months and then subtracting the monthly payment M. Convert this description into a recurrence for the amount owed after n months.

Exercise 4.2-5 Given the recurrence

$$T(n) = rT(n-1) + a,$$

where r and a are constants, find a recurrence that expresses $T(n)$ in terms of $T(n-2)$ instead of $T(n-1)$. Now find a recurrence that expresses $T(n)$ in terms of $T(n-3)$ instead of $T(n-2)$ or $T(n-1)$. Now find a recurrence that expresses $T(n)$ in terms of $T(n-4)$ rather than $T(n-1)$, $T(n-2)$, or $T(n-3)$. Based on your work so far, find a general formula for the solution to the recurrence

$$T(n) = rT(n-1) + a,$$

with $T(0) = b$, and where r and a are constants.

If we construct small examples for Exercise 4.2-3, we see that $\emptyset$ has only 1 subset, $\{1\}$ has 2 subsets, $\{1,2\}$ has 4 subsets, and $\{1,2,3\}$ has 8 subsets. This gives us a good guess as to what the general formula is, but in order to prove it we will need to think recursively. Consider the subsets of $\{1,2,3\}$:

$$\begin{array}{cccc} \emptyset & \{1\} & \{2\} & \{1,2\} \\ \{3\} & \{1,3\} & \{2,3\} & \{1,2,3\} \end{array}$$

The first four subsets do not contain three, and the second four do. Further, the first four subsets are exactly the subsets of $\{1,2\}$, while the second four are the four subsets of $\{1,2\}$ with 3 added into each one. This suggests that the recurrence for the number of subsets of an n-element set (which we may assume is $\{1,2,\ldots,n\}$) is

$$S(n) = \begin{cases} 2S(n-1) & \text{if } n \geq 1 \\ 1 & \text{if } n = 0 \end{cases} . \tag{4.11}$$

To prove this recurrence is correct, we note that the subsets of an n-element set can be partitioned by whether they contain element n or not. The subsets of $\{1,2,\ldots,n\}$ containing element n can be constructed by adjoining the element n to the subsets not containing element n. So the number of subsets containing element n is the same as the number of subsets not containing element n. The number of subsets not containing element n is just the number of subsets of an $n-1$-element set. Therefore each block of our partition has size equal to the number of subsets of an $n-1$-element set. Thus, by the sum principle, the number of subsets of $\{1,2,\ldots,n\}$ is twice the number of subsets of $\{1,2,\ldots,n-1\}$. This proves that $S(n) = 2S(n-1)$ if $n > 0$. We already observed that $\emptyset$ has no subsets, so we have proved the correctness of Recurrence 4.11.

For Exercise 4.2-4 we can algebraically describe what the problem said in words by

$$T(n) = (1 + .01p/12) \cdot T(n-1) - M,$$

with $T(0) = A$. Note that we add $.01p/12$ times the principal to the amount due each month, because $p/12$ percent of a number is $.01p/12$ times the number.

Iterating a recurrence

Turning to Exercise 4.2-5, we can substitute the right hand side of the equation $T(n-1) = rT(n-2) + a$ for $T(n-1)$ in our recurrence, and then substitute the similar equations for $T(n-2)$ and $T(n-3)$ to write

$$\begin{aligned} T(n) &= r(rT(n-2)+a)+a \\ &= r^2T(n-2)+ra+a \\ &= r^2(rT(n-3)+a)+ra+a \\ &= r^3T(n-3)+r^2a+ra+a \\ &= r^3(rT(n-4)+a)+r^2a+ra+a \\ &= r^4T(n-4)+r^3a+r^2a+ra+a \end{aligned}$$

From this, we can guess that

$$\begin{aligned} T(n) &= r^n T(0) + a \sum_{i=0}^{n-1} r^i \\ &= r^n b + a \sum_{i=0}^{n-1} r^i. \end{aligned} \tag{4.12}$$

The method we used to guess the solution is called *iterating the recurrence* because we repeatedly use the recurrence with smaller and smaller values in place of n. We could instead have written

$$\begin{aligned} T(0) &= b \\ T(1) &= rT(0) + a \\ &= rb + a \\ T(2) &= rT(1) + a \\ &= r(rb + a) + a \\ &= r^2 b + ra + a \\ T(3) &= rT(2) + a \\ &= r^3 b + r^2 a + ra + a \end{aligned}$$

This leads us to the same guess, so why have we introduced two methods? Having different approaches to solving a problem often yields insights we would not get with just one approach. For example, when we study recursion trees, we will see how to visualize the process of iterating certain kinds of recurrences in order to simplify the algebra involved in solving them.

Geometric series

You may recognize that sum $\sum_{i=0}^{n-1} r^i$ in Equation 4.12. It is called a *finite geometric series with common ratio* r. The sum $\sum_{i=0}^{n-1} ar^i$ is called a *finite geometric series with common ratio* r *and initial value* a. Recall from algebra the factorizations

$$\begin{aligned} (1-x)(1+x) &= 1-x^2 \\ (1-x)(1+x+x^2) &= 1-x^3 \\ (1-x)(1+x+x^2+x^3) &= 1-x^4 \end{aligned}$$

These factorizations are easy to verify, and they suggest that $(1-r)(1+r+r^2+\cdots+r^{n-1}) = 1-r^n$, or

$$\sum_{i=0}^{n-1} r^i = \frac{1-r^n}{1-r}. \tag{4.13}$$

In fact this formula is true, and lets us rewrite the formula we got for $T(n)$ in a very nice form.

Theorem 4.1 *If* $T(n) = rT(n-1) + a$, $T(0) = b$, *and* $r \neq 1$ *then*

$$T(n) = r^n b + a\frac{1-r^n}{1-r} \tag{4.14}$$

for all nonnegative integers n.

Proof: We will prove our formula by induction. Notice that the formula gives $T(0) = r^0 b + a\frac{1-r^0}{1-r}$ which is b, so the formula is true when $n = 0$. Now assume that $n > 0$ and

$$T(n-1) = r^{n-1}b + a\frac{1-r^{n-1}}{1-r}.$$

Then we have

$$\begin{aligned} T(n) &= rT(n-1) + a \\ &= r\left(r^{n-1}b + a\frac{1-r^{n-1}}{1-r}\right) + a \\ &= r^n b + \frac{ar - ar^n}{1-r} + a \\ &= r^n b + \frac{ar - ar^n + a - ar}{1-r} \\ &= r^n b + a\frac{1-r^n}{1-r}\ . \end{aligned}$$

Therefore by the principle of mathematical induction, our formula holds for all integers n greater than 0. ■

We did not prove Equation 4.13. However it is easy to use Theorem 4.1 to prove it.

Corollary 4.2 *The formula for the sum of a geometric series with $r \neq 1$ is*

$$\sum_{i=0}^{n-1} r^i = \frac{1-r^n}{1-r}. \tag{4.15}$$

Proof: Define $T(n) = \sum_{i=0}^{n-1} r^i$. Then $T(n) = rT(n-1) + 1$, and since $T(0)$ is a sum with no terms, $T(0) = 0$. Applying Theorem 4.1 with $b = 0$ and $a = 1$ gives us $T(n) = \frac{1-r^n}{1-r}$. ■

Often, when we see a geometric series, we will only be concerned with expressing the sum in big-O notation. In this case, we can show that the sum of a geometric series is at most the largest term times a constant factor, where the constant factor depends on r, but not on n.

Lemma 4.3 *Let r be a quantity whose value is independent of n and not equal to 1. Let $t(n)$ be the largest term of the geometric series*

$$\sum_{i=0}^{n-1} r^i.$$

Then the value of the geometric series is $O(t(n))$.

Proof: It is straightforward to see that we may limit ourselves to proving the lemma for $r > 0$. We consider two cases, depending on whether $r > 1$ or $r < 1$. If $r > 1$, then

$$\begin{aligned} \sum_{i=0}^{n-1} r^i &= \frac{r^n - 1}{r-1} \\ &\leq \frac{r^n}{r-1} \\ &= r^{n-1}\frac{r}{r-1} \\ &= O(r^{n-1}). \end{aligned}$$

On the other hand, if $r < 1$, then the largest term is $r^0 = 1$, and the sum has value

$$\frac{1 - r^n}{1 - r} < \frac{1}{1 - r}.$$

Thus the sum is $O(1)$, and since $t(n) = 1$, the sum is $O(t(n))$. ■

In fact, when r is nonnegative, an even stronger statement is true. Recall that we said that, for two functions f and g from the real numbers to the real numbers that $f = \Theta(g)$ if $f = O(g)$ and $g = O(f)$.

Theorem 4.4 *Let r be a nonnegative quantity whose value is independent of n and not equal to 1. Let $t(n)$ be the largest term of the geometric series*

$$\sum_{i=0}^{n-1} r^i.$$

Then the value of the geometric series is $\Theta(t(n))$.

Proof: By Lemma 4.3, we need only show that $t(n) = O(\frac{r^n-1}{r-1})$. Since all r^i are nonnegative, the sum $\sum_{i=0}^{n-1} r^i$ is at least as large as any of its summands. But $t(n)$ is one of these summands, so $t(n) = O(\frac{r^n-1}{r-1})$. ■

Note from the proof that $t(n)$ and the constant in the big-O upper bound depend on r. We will use this Theorem in subsequent sections.

First order linear recurrences

A recurrence of the form $T(n) = f(n)T(n-1) + g(n)$ is called a *first order linear recurrence*. When $f(n)$ is a constant, say r, the general solution is almost as easy to write down as in the case we already figured out. Iterating the recurrence gives us

$$\begin{aligned}
T(n) &= rT(n-1) + g(n) \\
&= r\Big(rT(n-2) + g(n-1)\Big) + g(n) \\
&= r^2T(n-2) + rg(n-1) + g(n) \\
&= r^2\Big(rT(n-3) + g(n-2)\Big) + rg(n-1) + g(n) \\
&= r^3T(n-3) + r^2g(n-2) + rg(n-1) + g(n) \\
&= r^3\Big(rT(n-4) + g(n-3)\Big) + r^2g(n-2) + rg(n-1) + g(n) \\
&= r^4T(n-4) + r^3g(n-3) + r^2g(n-2) + rg(n-1) + g(n) \\
&\vdots \\
&= r^nT(0) + \sum_{i=0}^{n-1} r^ig(n-i)
\end{aligned}$$

This suggests our next theorem.

Theorem 4.5 *For any positive constants a and r, and any function g defined on the nonnegative integers, the solution to the first order linear recurrence*

$$T(n) = \begin{cases} rT(n-1) + g(n) & \text{if } n > 0 \\ a & \text{if } n = 0 \end{cases}$$

is

$$T(n) = r^n a + \sum_{i=1}^{n} r^{n-i} g(i). \tag{4.16}$$

Proof: Let's prove this by induction.

Since the sum $\sum_{i=1}^{n} r^{n-i} g(i)$ in Equation 4.16 has no terms when $n = 0$, the formula gives $T(0) = 0$ and so is valid when $n = 0$. We now assume that n is positive and $T(n-1) = r^{n-1}a + \sum_{i=1}^{n-1} r^{(n-1)-i} g(i)$. Using the definition of the recurrence and the inductive hypothesis we get that

$$\begin{aligned} T(n) &= rT(n-1) + g(n) \\ &= r\left(r^{n-1}a + \sum_{i=1}^{n-1} r^{(n-1)-i} g(i)\right) + g(n) \\ &= r^n a + \sum_{i=1}^{n-1} r^{(n-1)+1-i} g(i) + g(n) \\ &= r^n a + \sum_{i=1}^{n-1} r^{n-i} g(i) + g(n) \\ &= r^n a + \sum_{i=1}^{n} r^{n-i} g(i). \end{aligned}$$

Therefore by the principle of mathematical induction, the solution to

$$T(n) = \begin{cases} rT(n-1) + g(n) & \text{if } n > 0 \\ a & \text{if } n = 0 \end{cases}$$

is given by Equation 4.16 for all nonnegative integers n. ■

The formula in Theorem 4.5 is a little less easy to use than that in Theorem 4.1 because it gives us a sum to compute. Fortunately, for a number of commonly occurring functions g the sum $\sum_{i=1}^{n} r^{n-i} g(i)$ is reasonable to compute.

Exercise 4.2-6 Solve the recurrence $T(n) = 4T(n-1) + 2^n$ with $T(0) = 6$.

Exercise 4.2-7 Solve the recurrence $T(n) = 3T(n-1) + n$ with $T(0) = 10$.

For Exercise 4.2-6, using Equation 4.16, we can write

$$\begin{aligned} T(n) &= 6 \cdot 4^n + \sum_{i=1}^{n} 4^{n-i} \cdot 2^i \\ &= 6 \cdot 4^n + 4^n \sum_{i=1}^{n} 4^{-i} \cdot 2^i \end{aligned}$$

$$\begin{aligned} &= 6 \cdot 4^n + 4^n \sum_{i=1}^{n} \left(\frac{1}{2}\right)^i \\ &= 6 \cdot 4^n + 4^n \cdot \frac{1}{2} \cdot \sum_{i=0}^{n-1} \left(\frac{1}{2}\right)^i \\ &= 6 \cdot 4^n + (1 - (\frac{1}{2})^n) \cdot 4^n \\ &= 7 \cdot 4^n - 2^n \end{aligned}$$

For Exercise 4.2-7 we begin in the same way and face a bit of a surprise. Using Equation 4.16, we write

$$\begin{aligned} T(n) &= 10 \cdot 3^n + \sum_{i=1}^{n} 3^{n-i} \cdot i \\ &= 10 \cdot 3^n + 3^n \sum_{i=1}^{n} i 3^{-i} \\ &= 10 \cdot 3^n + 3^n \sum_{i=1}^{n} i \left(\frac{1}{3}\right)^i . \end{aligned} \tag{4.17}$$

Now we are faced with a sum that you may not recognize, a sum that has the form

$$\sum_{i=1}^{n} i x^i = x \sum_{i=1}^{n} i x^{i-1},$$

with $x = 1/3$. However by writing it in in this form, we can use calculus to recognize it as x times a derivative. In particular, using the fact that $0x^0 = 0$, we can write

$$\sum_{i=1}^{n} i x^i = x \sum_{i=0}^{n} i x^{i-1} = x \frac{d}{dx} \sum_{i=0}^{n} x^i = x \frac{d}{dx} \left(\frac{1 - x^{n+1}}{1 - x} \right).$$

But using the formula for the derivative of a quotient from calculus, we may write

$$x \frac{d}{dx} \left(\frac{1 - x^{n+1}}{1 - x} \right) = x \frac{(1-x)(-(n+1)x^n) - (1 - x^{n+1})(-1)}{(1-x)^2} = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(1-x)^2}.$$

Connecting our first and last equations, we get

$$\sum_{i=1}^{n} i x^i = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(1-x)^2}. \tag{4.18}$$

Substituting in $x = 1/3$ and simplifying gives us

$$\sum_{i=1}^{n} i \left(\frac{1}{3}\right)^i = -\frac{3}{2}(n+1)\left(\frac{1}{3}\right)^{n+1} - \frac{3}{4}\left(\frac{1}{3}\right)^{n+1} + \frac{3}{4}.$$

Substituting this into Equation 4.17 gives us

$$\begin{aligned} T(n) &= 10 \cdot 3^n + 3^n \left(-\frac{3}{2}(n+1)\left(\frac{1}{3}\right)^{n+1} - \frac{3}{4}(1/3)^{n+1} + \frac{3}{4} \right) \\ &= 10 \cdot 3^n - \frac{n+1}{2} - \frac{1}{4} + \frac{3^{n+1}}{4} \\ &= \frac{43}{4} 3^n - \frac{n+1}{2} - \frac{1}{4}. \end{aligned}$$

The sum that arises in this exercise occurs so often that we give its formula as a theorem.

Theorem 4.6 *For any real number $x \neq 1$,*

$$\sum_{i=1}^{n} ix^i = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(1-x)^2}. \tag{4.19}$$

Proof: Given before the statement of the theorem.■

Important Concepts, Formulas, and Theorems

1. *Recurrence Equation or Recurrence.* A recurrence equation is one that tells us how to compute the nth term of a sequence from the $(n-1)$st term or some or all the preceding terms.

2. *Initial Condition.* To completely specify a function on the basis of a recurrence, we have to give enough information about the function to get started. This information is called the *initial condition (or the initial conditions)* for the recurrence.

3. *First Order Linear Recurrence.* A recurrence $T(n) = f(n)T(n-1) + g(n)$ is called a *first order linear recurrence.*

4. *Constant Coefficient Recurrence.* A recurrence in which $T(n)$ is expressed in terms of a sum of constant multiples of $T(k)$ for certain values $k < n$ (and perhaps another function of n) is called a *constant coefficient recurrence.*

5. *Solution to a First Order Constant Coefficient Linear Recurrence.* If $T(n) = rT(n-1) + a$, $T(0) = b$, and $r \neq 1$ then
$$T(n) = r^n b + a\frac{1-r^n}{1-r}$$
for all nonnegative integers n.

6. *Finite Geometric Series.* A finite geometric series with common ratio r is a sum of the form $\sum_{i=0}^{n-1} r^i$. The formula for the sum of a geometric series with $r \neq 1$ is
$$\sum_{i=0}^{n-1} r^i = \frac{1-r^n}{1-r}.$$

7. *Big-Theta Bounds on the Sum of a Geometric Series.* Let r be a nonnegative quantity whose value is independent of n and not equal to 1. Let $t(n)$ be the largest term of the geometric series
$$\sum_{i=0}^{n-1} r^i.$$
Then the value of the geometric series is $\Theta(t(n))$.

8. *Solution to a First Order Linear Recurrence.* For any positive constants a and r, and any function g defined on the nonnegative integers, the solution to the first order linear recurrence

$$T(n) = \begin{cases} rT(n-1) + g(n) & \text{if } n > 0 \\ a & \text{if } n = 0 \end{cases}$$

is

$$T(n) = r^n a + \sum_{i=1}^{n} r^{n-i} g(i).$$

9. *Iterating a Recurrence.* We say we are *iterating* a recurrence when we guess its solution by using the equation that expresses $T(n)$ in terms of $T(k)$ for k smaller than n to re-express $T(n)$ in terms of $T(k)$ for k smaller than $n-1$, then for k smaller than $n-2$, and so on until we can guess the formula for the sum.

10. *An Important Sum.* For any real number $x \neq 1$,

$$\sum_{i=1}^{n} ix^i = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(1-x)^2}.$$

Problems

1. Prove Equation 4.15 directly by induction.

2. Prove Equation 4.18 directly by induction.

3. Solve the recurrence $M(n) = 2M(n-1) + 2$, with a base case of $M(1) = 1$. How does it differ from the solution to Recurrence 4.7?

4. Solve the recurrence $M(n) = 3M(n-1) + 1$, with a base case of $M(1) = 1$. How does it differ from the solution to Recurrence 4.7.

5. Solve the recurrence $M(n) = M(n-1) + 2$, with a base case of $M(1) = 1$. How does it differ from the solution to Recurrence 4.7.

6. There are m functions from a one-element set to the set $\{1, 2, \ldots, m\}$. How many functions are there from a two-element set to $\{1, 2, \ldots, m\}$? From a three-element set? Give a recurrence for the number $T(n)$ of functions from an n-element set to $\{1, 2, \ldots, m\}$. Solve the recurrence.

7. Solve the recurrence that you derived in Exercise 4.2-4.

8. At the end of each year, a state fish hatchery puts 2000 fish into a lake. The number of fish in the lake at the beginning of the year doubles due to reproduction by the end of the year. Give a recurrence for the number of fish in the lake after n years and solve the recurrence.

9. Consider the recurrence $T(n) = 3T(n-1) + 1$ with the initial condition that $T(0) = 2$. We know that we could write the solution down from Theorem 4.1. Instead of using the theorem, try to guess the solution from the first four values of $T(n)$ and then try to guess the solution by iterating the recurrence four times.

10. What sort of big-Θ bound can we give on the value of a geometric series $1+r+r^2+\cdots+r^n$ with common ratio $r = 1$?

11. Solve the recurrence $T(n) = 2T(n-1) + n2^n$ with the initial condition that $T(0) = 1$.

12. Solve the recurrence $T(n) = 2T(n-1) + n^3 2^n$ with the initial condition that $T(0) = 2$.

13. Solve the recurrence $T(n) = 2T(n-1) + 3^n$ with $T(0) = 1$.

14. Solve the recurrence $T(n) = rT(n-1) + r^n$ with $T(0) = 1$.

15. Solve the recurrence $T(n) = rT(n-1) + r^{2n}$ with $T(0) = 1$

16. Solve the recurrence $T(n) = rT(n-1) + s^n$ with $T(0) = 1$.

17. Solve the recurrence $T(n) = rT(n-1) + n$ with $T(0) = 1$.

18. The Fibonacci numbers are defined by the recurrence

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{if } n > 0 \\ 1 & \text{if } n = 0 \text{ or } n = 1 \end{cases}$$

    (a) Write down the first ten Fibonacci numbers.

    (b) Show that $(\frac{1+\sqrt{5}}{2})^n$ and $(\frac{1-\sqrt{5}}{2})^n$ are solutions to the equation $F(n) = F(n-1) + F(n-2)$.

    (c) Why is

$$c_1(\frac{1+\sqrt{5}}{2})^n + c_2(\frac{1-\sqrt{5}}{2})^n$$

    a solution to the equation $F(n) = F(n-1) + F(n-2)$ for any real numbers c_1 and c_2?

    (d) Find constants c_1 and c_2 such that the Fibonacci numbers are given by

$$F(n) = c_1(\frac{1+\sqrt{5}}{2})^n + c_2(\frac{1-\sqrt{5}}{2})^n$$

4.3 Growth Rates of Solutions to Recurrences

Divide and Conquer Algorithms

One of the most basic and powerful algorithmic techniques is *divide and conquer*. Consider, for example, the binary search algorithm, which we will describe in the context of guessing a number between 1 and 100. Suppose someone picks a number between 1 and 100, and allows you to ask questions of the form "Is the number greater than k?" where k is an integer you choose. Your goal is to ask as few questions as possible to figure out the number. Your first question should be "Is the number greater than 50?" Why is this? Well, after asking if the number is bigger than 50, you have learned either that the number is between one and 50, or that the number is between 51 and 100. In either case have reduced your problem to one in which the range is only half as big. Thus you have *divided* the problem up into a problem that is only half as big, and you can now (recursively) *conquer* this remaining problem. (If you ask any other question, the size of one of the possible ranges of values you could end up with would be more than half the size of the original problem.) If you continue in this fashion, always cutting the problem size in half, you will reduce the problem size down to one fairly quickly, and then you will know what the number is. Of course it would be easier to cut the problem size exactly in half each time if we started with a number in the range from one to 128, but the question doesn't sound quite so plausible then. Thus to analyze the problem we will assume someone asks you to figure out a number between 0 and n, where n is a power of 2.

Exercise 4.3-1 Let $T(n)$ be number of questions in binary search on the range of numbers between 1 and n. Assuming that n is a power of 2, give a recurrence for $T(n)$.

For Exercise 4.3-1 we get:

$$T(n) = \begin{cases} T(n/2) + 1 & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases} \tag{4.20}$$

That is, the number of guesses to carry out binary search on n items is equal to 1 step (the guess) plus the time to solve binary search on the remaining $n/2$ items.

What we are really interested in is how much time it takes to use binary search in a computer program that looks for an item in an ordered list. While the number of questions gives us a feel for the amount of time, processing each question may take several steps in our computer program. The exact amount of time these steps take might depend on some factors we have little control over, such as where portions of the list are stored. Also, we may have to deal with lists whose length is not a power of two. Thus a more realistic description of the time needed would be

$$T(n) \leq \begin{cases} T(\lceil n/2 \rceil) + C_1 & \text{if } n \geq 2 \\ C_2 & \text{if } n = 1, \end{cases} \tag{4.21}$$

where C_1 and C_2 are constants.

Note that $\lceil x \rceil$ stands for the smallest integer larger than or equal to x, while $\lfloor x \rfloor$ stands for the largest integer less than or equal to x. It turns out that the solution to (4.20) and (4.21) are roughly the same, in a sense that will hopefully become clear later. (This is almost always

the case.) For now, let us not worry about floors and ceilings and the distinction between things that take 1 unit of time and things that take no more than some constant amount of time.

Let's turn to another example of a divide and conquer algorithm, *mergesort*. In this algorithm, you wish to sort a list of n items. Let us assume that the data is stored in an array A in positions 1 through n. Mergesort can be described as follows:

```
MergeSort(A,low,high)
     if (low == high)
          return
     else
          mid = (low + high)/2
          MergeSort(A,low,mid)
          MergeSort(A,mid+1,high)
          Merge the sorted lists from the previous two steps
```

More details on mergesort can be found in almost any algorithms textbook. Suffice to say that the base case (low = high) takes one step, while the other case executes 1 step, makes two recursive calls on problems of size $n/2$, and then executes the Merge instruction, which can be done in n steps.

Thus we obtain the following recurrence for the running time of mergesort:

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases} \tag{4.22}$$

Recurrences such as this one can be understood via the idea of a recursion tree, which we introduce below. This concept allows us to analyze recurrences that arise in divide-and-conquer algorithms, and those that arise in other recursive situations, such as the Towers of Hanoi, as well. A recursion tree for a recurrence is a visual and conceptual representation of the process of iterating the recurrence.

Recursion Trees

We will introduce the idea of a recursion tree via several examples. It is helpful to have an "algorithmic" interpretation of a recurrence. For example, (ignoring for a moment the base case) we can interpret the recurrence

$$T(n) = 2T(n/2) + n \tag{4.23}$$

as "in order to solve a problem of size n we must solve 2 problems of size $n/2$ and do n units of additional work." Similarly we can interpret

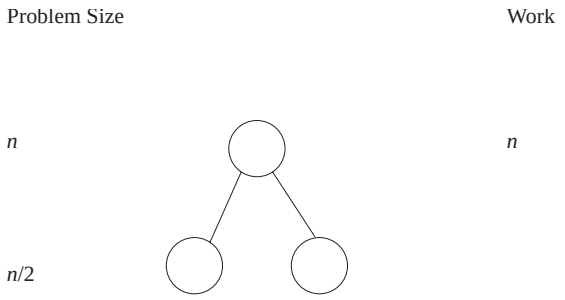
$$T(n) = T(n/4) + n^2$$

as "in order to solve a problem of size n we must solve one problem of size $n/4$ and do n^2 units of additional work."

We can also interpret the recurrence

$$T(n) = 3T(n-1) + n$$

Figure 4.2: The initial stage of drawing a recursion tree diagram.



as "in order to solve a problem of size n, we must solve 3 subproblems of size $n-1$ and do n additional units of work.

In Figure 4.2 we draw the beginning of the recursion tree diagram for (4.23). For now, assume n is a power of 2. A recursion tree diagram has three parts, a left, a middle, and a right. On the left, we keep track of the problem size, in the middle we draw the tree, and on right we keep track of the work done. We draw the diagram in levels, each level of the diagram representing a level of recursion. Equivalently, each level of the diagram represents a level of iteration of the recurrence. So to begin the recursion tree for (4.23), we show, in level 0 on the left, that we have problem of size n. Then by drawing a root vertex with two edges leaving it, we show in the middle that we are splitting our problem into 2 problems. We note on the right that we do n units of work in addition to whatever is done on the two new problems we created. In the next level, we draw two vertices in the middle representing the two problems into which we split our main problem and show on the left that each of these problems has size $n/2$.

You can see how the recurrence is reflected in levels 0 and 1 of the recursion tree. The top vertex of the tree represents $T(n)$, and on the next level we have two problems of size $n/2$, representing the recursive term $2T(n/2)$ of our recurrence. Then after we solve these two problems we return to level 0 of the tree and do n additional units of work for the nonrecursive term of the recurrence.

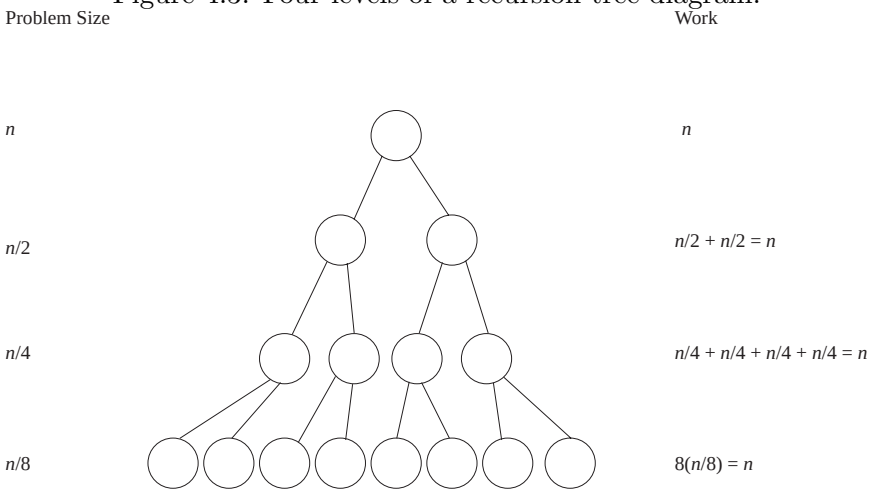
Now we continue to draw the tree in the same manner. Filling in the rest of level one and adding a few more levels, we get Figure 4.3.

Let us summarize what the diagram tells us so far. At level zero (the top level), n units of work are done. We see that at each succeeding level, we halve the problem size and double the number of subproblems. We also see that at level 1, each of the two subproblems requires $n/2$ units of additional work, and so a total of n units of additional work are done. Similarly level 2 has 4 subproblems of size $n/4$ and so $4(n/4) = n$ units of additional work are done. Notice that to compute the total work done on a level we multiply the number of subproblems by the amount of additional work per subproblem.

To see how iteration of the recurrence is reflected in the diagram, we iterate the recurrence once, getting

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ T(n) &= 2(2T(n/4) + n/2) + n \\ T(n) &= 4T(n/4) + n + n = 4T(n/4) + 2n \end{aligned}$$

Figure 4.3: Four levels of a recursion tree diagram.



If we examine levels 0, 1, and 2 of the diagram, we see that at level 2 we have four vertices which represent four problems, each of size $n/4$ This corresponds to the recursive term that we obtained after iterating the recurrence. However after we solve these problems we return to level 1 where we twice do $n/2$ additional units of work and to level 0 where we do another n additional units of work. In this way each time we add a level to the tree we are showing the result of one more iteration of the recurrence.

We now have enough information to be able to describe the recursion tree diagram in general. To do this, we need to determine, for each level, three things:

- the number of subproblems,
- the size of each subproblem,
- the total work done at that level.

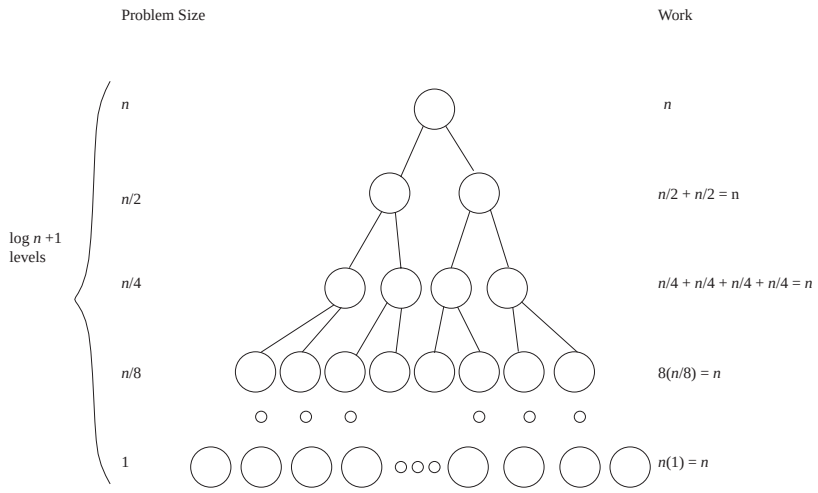
We also need to figure out how many levels there are in the recursion tree.

We see that for this problem, at level i, we have 2^i subproblems of size $n/2^i$. Further, since a problem of size 2^i requires 2^i units of additional work, there are $(2^i)[n/(2^i)] = n$ units of work done per level. To figure out how many levels there are in the tree, we just notice that at each level the problem size is cut in half, and the tree stops when the problem size is 1. Therefore there are $\log_2 n + 1$ levels of the tree, since we start with the top level and cut the problem size in half $\log_2 n$ times.[2] We can thus visualize the whole tree in Figure 4.4.

The computation of the work done at the bottom level is different from the other levels. In the other levels, the work is described by the recursive equation of the recurrence; in this case the amount of work is the n in $T(n) = 2T(n/2) + n$. At the bottom level, the work comes from the base case. Thus we must compute the number of problems of size 1 (assuming that one is the base case), and then multiply this value by $T(1) = 1$. In our recursion tree in Figure 4.4, the number of nodes at the bottom level is $2^{\log_2 n} = n$. Since $T(1) = 1$, we do n units of work at

[2]To simplify notation, for the remainder of the book, if we omit the base of a logarithm, it should be assumed to be base 2.

Figure 4.4: A finished recursion tree diagram.



the bottom level of the tree. Had we chosen to say that $T(1)$ was some constant other than 1, this would not have been the case. We emphasize that the correct value always comes from the base case; it is just a coincidence that it sometimes also comes from the recursive equation of the recurrence.

The bottom level of the tree represents the final stage of iterating the recurrence. We have seen that at this level we have n problems each requiring work $T(1) = 1$, giving us total work n at that level. After we solve the problems represented by the bottom level, we have to do all the additional work from all the earlier levels. For this reason, we sum the work done at all the levels of the tree to get the total work done. *Iteration of the recurrence shows us that the solution to the recurrence is the sum of all the work done at all the levels of the recursion tree.*

The important thing is that we now know how much work is done at each level. Once we know this, we can sum the total amount of work done over all the levels, giving us the solution to our recurrence. In this case, there are $\log_2 n + 1$ levels, and at each level the amount of work we do is n units. Thus we conclude that the total amount of work done to solve the problem described by recurrence (4.23) is $n(\log_2 n + 1)$. The total work done throughout the tree is the solution to our recurrence, because the tree simply models the process of iterating the recurrence. Thus the solution to recurrence (4.22) is $T(n) = n(\log n + 1)$.

Since one unit of time will vary from computer to computer, and since some kinds of work might take longer than other kinds, we are usually interested in the big-θ behavior of $T(n)$. For example, we can consider a recurrence that it identical to (4.22), except that $T(1) = a$, for some constant a. In this case, $T(n) = an + n \log n$, because an units of work are done at level 1 and n additional units of work are done at each of the remaining $\log n$ levels. It is still true that $T(n) = \Theta(n \log n)$, because the different base case did not change the solution to the recurrence by more than a constant factor[3]. Although recursion trees can give us the exact solutions (such as $T(n) = an + n \log n$ above) to recurrences, our interest in the big-Θ behavior of solutions will usually lead us to use a recursion tree to determine the big-Θ or even, in complicated cases, just the big-O behavior of the actual solution to the recurrence. In Problem 10 we explore whether

[3]More precisely, $n \log n < an + n \log n < (a + 1)n \log n$ for any $a > 0$.

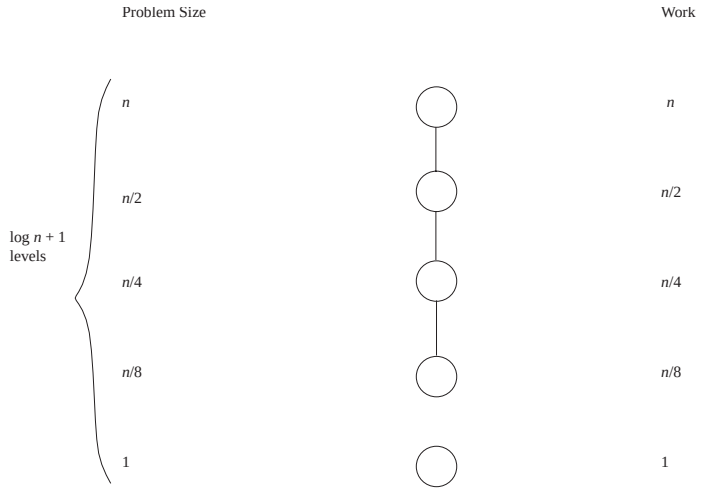
the value of $T(1)$ actually influences the big-Θ behavior of the solution to a recurrence.

Let's look at one more recurrence.

$$T(n) = \begin{cases} T(n/2) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases} \tag{4.24}$$

Again, assume n is a power of two. We can interpret this as follows: to solve a problem of size n, we must solve one problem of size $n/2$ and do n units of additional work. We draw the tree for this problem in Figure 4.5 and see that the problem sizes are the same as in the previous tree. The remainder, however, is different. The number of subproblems does not double, rather

Figure 4.5: A recursion tree diagram for Recurrence 4.24.



it remains at one on each level. Consequently the amount of work halves at each level. Note that there are still $\log n + 1$ levels, as the number of levels is determined by how the problem size is changing, not by how many subproblems there are. So on level i, we have 1 problem of size $n/2^i$, for total work of $n/2^i$ units.

We now wish to compute how much work is done in solving a problem that gives this recurrence. Note that the additional work done is different on each level, so we have that the total amount of work is

$$n + n/2 + n/4 + \cdots + 2 + 1 = n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \left(\frac{1}{2}\right)^{\log_2 n}\right),$$

which is n times a geometric series. By Theorem 4.4, the value of a geometric series in which the largest term is one is $\Theta(1)$. This implies that the work done is described by $T(n) = \Theta(n)$.

We emphasize that there is exactly one solution to recurrence (4.24); it is the one we get by using the recurrence to compute $T(2)$ from $T(1)$, then to compute $T(4)$ from $T(2)$, and so on. What we have done here is show that $T(n) = \Theta(n)$. In fact, for the kinds of recurrences we have been examining, once we know $T(1)$ we can compute $T(n)$ for any relevant n by repeatedly using the recurrence, so there is no question that solutions do exist and can, in principle, be computed for any value of n. In most applications, we are not interested in the exact form of the solution, but a big-O upper bound, or Big-Θ bound on the solution.

Exercise 4.3-2 Find a big-Θ bound for the solution to the recurrence

$$T(n) = \begin{cases} 3T(n/3) + n & \text{if } n \geq 3 \\ 1 & \text{if } n < 3 \end{cases}$$

using a recursion tree. Assume that n is a power of 3.

Exercise 4.3-3 Solve the recurrence

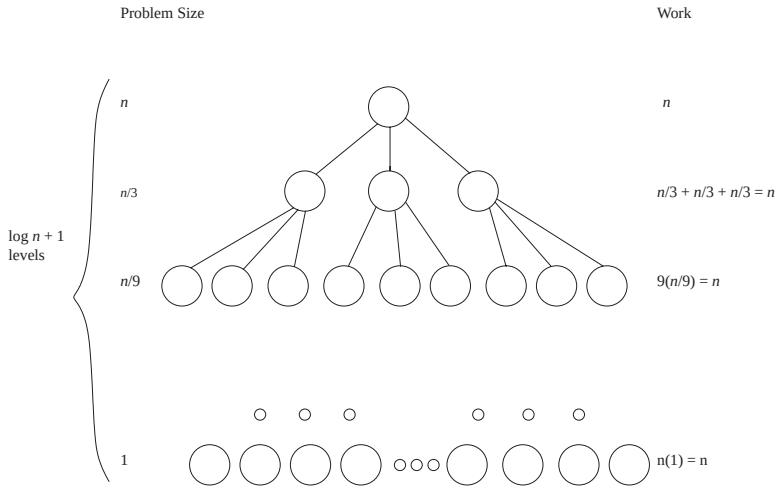
$$T(n) = \begin{cases} 4T(n/2) + n & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$

using a recursion tree. Assume that n is a power of 2. Convert your solution to a big-Θ statement about the behavior of the solution.

Exercise 4.3-4 Can you give a general big-Θ bound for solutions to recurrences of the form $T(n) = aT(n/2) + n$ when n is a power of 2? You may have different answers for different values of a.

The recurrence in Exercise 4.3-2 is similar to the mergesort recurrence. One difference is that at each step we divide into 3 problems of size $n/3$. Thus we get the picture in Figure 4.6. Another difference is that the number of levels, instead of being $\log_2 n + 1$ is now $\log_3 n + 1$, so

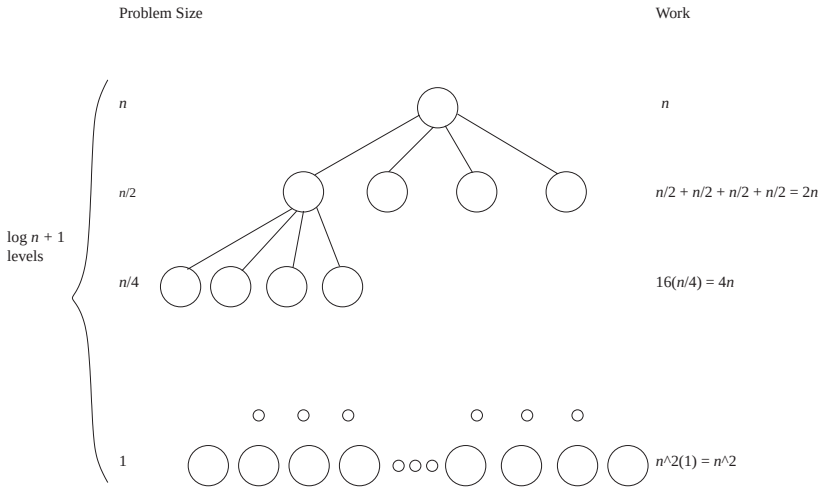
Figure 4.6: The recursion tree diagram for the recurrence in Exercise 4.3-2.



the total work is still $\Theta(n \log n)$ units. (Note that $\log_b n = \Theta(\log_2 n)$ for any $b > 1$.)

Now let's look at the recursion tree for Exercise 4.3-3. Here we have 4 children of size $n/2$, and we get Figure 4.7. Let's look carefully at this tree. Just as in the mergesort tree there are $\log_2 n + 1$ levels. However, in this tree, each node has 4 children. Thus level 0 has 1 node, level 1 has 4 nodes, level 2 has 16 nodes, and in general level i has 4^i nodes. On level i each node corresponds to a problem of size $n/2^i$ and hence requires $n/2^i$ units of additional work. Thus the total work on level i is $4^i(n/2^i) = 2^i n$ units. This formula applies on level $\log_2 n$ (the bottom

Figure 4.7: The Recursion tree for Exercise 4.3-3.



level) as well since there are $n^2 = 2^{\log_2 n} n$ nodes, each requiring $T(1) = 1$ work. Summing over the levels, we get

$$\sum_{i=0}^{\log_2 n} 2^i n = n \sum_{i=0}^{\log_2 n} 2^i.$$

There are many ways to simplify that expression, for example from our formula for the sum of a geometric series we get

$$\begin{aligned} T(n) &= n \sum_{i=0}^{\log_2 n} 2^i \\ &= n \frac{1 - 2^{(\log_2 n)+1}}{1 - 2} \\ &= n \frac{1 - 2n}{-1} \\ &= 2n^2 - n \\ &= \Theta(n^2). \end{aligned}$$

More simply, by Theorem 4.4 we have that $T(n) = n\Theta(2^{\log n}) = \Theta(n^2)$.

Three Different Behaviors

Now let's compare the recursion tree diagrams for the recurrences $T(n) = 2T(n/2) + n$, $T(n) = T(n/2) + n$ and $T(n) = 4T(n/2) + n$. Note that all three trees have depth $1 + \log_2 n$, as this is determined by the size of the subproblems relative to the parent problem, and in each case, the size of each subproblem is $1/2$ the size of of the parent problem. The trees differ, however, in the amount of work done per level. In the first case, the amount of work on each level is the same. In the second case, the amount of work done on a level decreases as you go down the tree, with the most work being at the top level. In fact, it decreases geometrically, so by Theorem 4.4 the

total work done is bounded above and below by a constant times the work done at the root node. In the third case, the number of nodes per level is growing at a faster rate than the problem size is decreasing, and the level with the largest amount of work is the bottom one. Again we have a geometric series, and so by Theorem 4.4 the total work is bounded above and below by a constant times the amount of work done at the last level.

If you understand these three cases and the differences among them, you now understand the great majority of the recursion trees that arise in algorithms.

So to answer Exercise 4.3-4, which asks for a general Big-Θ bound for the solutions to recurrences of the form $T(n) = aT(n/2) + n$, we can conclude the following:

Lemma 4.7 *Suppose that we have a recurrence of the form*

$$T(n) = aT(n/2) + n,$$

where a is a positive integer and $T(1)$ is nonnegative. Thus we have the following big-Theta bounds on the solution.

1. *If $a < 2$ then $T(n) = \Theta(n)$.*
2. *If $a = 2$ then $T(n) = \Theta(n \log n)$*
3. *If $a > 2$ then $T(n) = \Theta(n^{\log_2 a})$*

Proof: Cases 1 and 2 follow immediately from our observations above. We can verify case 3 as follows. At each level i we have a^i nodes, each corresponding to a problem of size $n/2^i$. Thus at level i the total amount of work is $a^i(n/2^i) = n(a/2)^i$ units. Summing over the $\log_2 n$ levels, we get

$$a^{\log_2 n} T(1) + n \sum_{i=0}^{(\log_2 n)-1} (a/2)^i.$$

The sum given by the summation sign is a geometric series, so, since $a/2 \neq 1$, the sum will be big-Θ of the largest term (see Theorem 4.4). Since $a > 2$, the largest term in this case is clearly the last one, namely $n(a/2)^{(\log_2 n)-1}$, and applying rules of exponents and logarithms, we get that n times the largest term is

$$\begin{aligned} n\left(\frac{a}{2}\right)^{(\log_2 n)-1} &= \frac{2}{a} \cdot \frac{n \cdot a^{\log_2 n}}{2^{\log_2 n}} = \frac{2}{a} \cdot \frac{n \cdot a^{\log_2 n}}{n} = \frac{2}{a} \cdot a^{\log_2 n} \\ &= \frac{2}{a} a^{\log_2 (a^{log_2 a}} = \frac{2}{a} \cdot 2^{\log_2 a \log_2 n} = \frac{2}{a} \cdot n^{\log_2 a}. \end{aligned} \tag{4.25}$$

Thus $T(1)a^{\log_2 n} = T(1)n^{\log_2 a}$. Since $\frac{2}{a}$ and $T(1)$ are both nonnegative, the total work done is $\Theta(n^{\log_2 a})$. ■

In fact Lemma 4.7 holds for all positive real numbers a; we can iterate the recurrence to see this. Since a recursion tree diagram is a way to visualize iterating the recurrence when a is an integer, iteration is the natural thing to try when a is not an integer.

Notice that in the last two equalities of computation we made in Equation 4.25, we showed that $a^{\log n} = n^{\log a}$. This is a useful and, perhaps, surprising fact, so we state it (in slightly more generality) as a corollary to the proof.

Corollary 4.8 *For any base b, we have $a^{\log_b n} = n^{\log_b a}$.*

Important Concepts, Formulas, and Theorems

1. *Divide and Conquer Algorithm.* A *divide and conquer algorithm* is one that solves a problem by dividing it into problems that are smaller but otherwise of the same type as the original one, recursively solves these problems, and then assembles the solution of these so-called subproblems into a solution of the original one. Not all problems can be solved by such a strategy, but a great many problems of interest in computer science can.

2. *Mergesort.* In *mergesort* we sort a list of items that have some underlying order by dividing the list in half, sorting the first half (by recursively using mergesort), sorting the second half (by recursively using mergesort), and then merging the two sorted list. For a list of length one mergesort returns the same list.

3. *Recursion Tree.* A *recursion tree diagram* for a recurrence of the form $T(n) = aT(n/b)+g(n)$ has three parts, a left, a middle, and a right. On the left, we keep track of the problem size, in the middle we draw the tree, and on right we keep track of the work done. We draw the diagram in levels, each level of the diagram representing a level of recursion. The tree has a vertex representing the initial problem and one representing each subproblem we have to solve. Each non-leaf vertex has a children. The vertices are divided into levels corresponding to (sub-)problems of the same size; to the left of a level of vertices we write the size of the problems the vertices correspond to; to the right of the vertices on a given level we write the total amount of work done at that level by an algorithm whose work is described by the recurrence, not including the work done by any recursive calls from that level.

4. *The Base Level of a Recursion Tree.* The amount of work done on the lowest level in a recursion tree is the number of nodes times the value given by the initial condition; it is not determined by attempting to make a computation of "additional work" done at the lowest level.

5. *Bases for Logarithms.* We use $\log n$ as an alternate notation for $\log_2 n$. A fundamental fact about logarithms is that $\log_b n = \Theta(\log_2 n)$ for any real number $b > 1$.

6. *An Important Fact About Logarithms.* For any $b > 0$, $a^{\log_b n} = n^{\log_b a}$.

7. *Three behaviors of solutions.* The solution to a recurrence of the form $T(n) = aT(n/2) + n$ behaves in one of the following ways:

   (a) if $a < 2$ then $T(n) = \Theta(n)$.
   (b) if $a = 2$ then $T(n) = \Theta(n \log n)$
   (c) if $a > 2$ then $T(n) = \Theta(n^{\log_2 a})$.

Problems

1. Draw recursion trees and find big-Θ bounds on the solutions to the following recurrences. For all of these, assume that $T(1) = 1$ and n is a power of the appropriate integer.

   (a) $T(n) = 8T(n/2) + n$
   (b) $T(n) = 8T(n/2) + n^3$

(c) $T(n) = 3T(n/2) + n$

(d) $T(n) = T(n/4) + 1$

(e) $T(n) = 3T(n/3) + n^2$

2. Draw recursion trees and find exact solutions to the following recurrences. For all of these, assume that $T(1) = 1$ and n is a power of the appropriate integer.

(a) $T(n) = 8T(n/2) + n$

(b) $T(n) = 8T(n/2) + n^3$

(c) $T(n) = 3T(n/2) + n$

(d) $T(n) = T(n/4) + 1$

(e) $T(n) = 3T(n/3) + n^2$

3. Find the exact solution to Recurrence 4.24.

4. Show that $\log_b n = \Theta(\log_2 n)$, for any constant $b > 1$.

5. Prove Corollary 4.8 by showing that $a^{\log_b n} = n^{\log_b a}$ for any $b > 0$.

6. Recursion trees will still work, even if the problems do not break up geometrically, or even if the work per level is not n^c units. Draw recursion trees and and find the best big-O bounds you can for solutions to the following recurrences. For all of these, assume that $T(1) = 1$.

(a) $T(n) = T(n-1) + n$

(b) $T(n) = 2T(n-1) + n$

(c) $T(n) = T(\lfloor\sqrt{n}\rfloor) + 1$ (You may assume n has the form $n = 2^{2^i}$.)

(d) $T(n) = 2T(n/2) + n \log n$ (You may assume n is a power of 2.)

7. In each case in the previous problem, is the big-O bound you found a big-Θ bound?

8. If $S(n) = aS(n-1) + g(n)$ and $g(n) < c^n$ with $0 \leq c < a$, how fast does $S(n)$ grow (in big-Θ terms)?

9. If $S(n) = aS(n-1) + g(n)$ and $g(n) = c^n$ with $0 < a \leq c$, how fast does $S(n)$ grow in big-Θ terms?

10. Given a recurrence of the form $T(n) = aT(n/b) + g(n)$ with $T(1) = c > 0$ and $g(n) > 0$ for all n and a recurrence of the form $S(n) = aS(n/b) + g(n)$ with $S(1) = 0$ (and the same a, b, and $g(n)$), is there any difference in the big-Θ behavior of the solutions to the two recurrences? What does this say about the influence of the initial condition on the big-Θ behavior of such recurrences?

4.4 The Master Theorem

Master Theorem

In the last section, we saw three different kinds of behavior for recurrences of the form

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

These behaviors depended upon whether $a < 2$, $a = 2$, or $a > 2$. Remember that a was the number of subproblems into which our problem was divided. Dividing by 2 cut our problem size in half each time, and the n term said that after we completed our recursive work, we had n additional units of work to do for a problem of size n. There is no reason that the amount of additional work required by each subproblem needs to be the size of the subproblem. In many applications it will be something else, and so in Theorem 4.9 we consider a more general case. Similarly, the sizes of the subproblems don't have to be 1/2 the size of the parent problem. We then get the following theorem, our first version of a theorem called the *Master Theorem*. (Later on we will develop some stronger forms of this theorem.)

Theorem 4.9 *Let a be an integer greater than or equal to 1 and b be a real number greater than 1. Let c be a positive real number and d a nonnegative real number. Given a recurrence of the form*

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

in which n is restricted to be a power of b,

1. *if $\log_b a < c$, $T(n) = \Theta(n^c)$,*
2. *if $\log_b a = c$, $T(n) = \Theta(n^c \log n)$,*
3. *if $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$.*

Proof: In this proof, we will set $d = 1$, so that the work done at the bottom level of the tree is the same as if we divided the problem one more time and used the recurrence to compute the additional work. As in Footnote 3 in the previous section, it is straightforward to show that we get the same big-Θ bound if d is positive. It is only a little more work to show that we get the same big-Θ bound if d is zero.

Let's think about the recursion tree for this recurrence. There will be $1 + \log_b n$ levels. At each level, the number of subproblems will be multiplied by a, and so the number of subproblems at level i will be a^i. Each subproblem at level i is a problem of size (n/b^i). A subproblem of size n/b^i requires $(n/b^i)^c$ additional work and since there are a^i problems on level i, the total number of units of work on level i is

$$a^i(n/b^i)^c = n^c \left(\frac{a^i}{b^{ci}}\right) = n^c \left(\frac{a}{b^c}\right)^i. \tag{4.26}$$

Recall from Lemma 4.7 that the different cases for $c = 1$ were when the work per level was decreasing, constant, or increasing. The same analysis applies here. From our formula for work

on level i, we see that the work per level is decreasing, constant, or increasing exactly when $(\frac{a}{b^c})^i$ is decreasing, constant, or increasing, respectively. These three cases depend on whether $(\frac{a}{b^c})$ is less than one, equal to one, or greater than one, respectively. Now observe that

$$\begin{aligned} & (\tfrac{a}{b^c}) = 1 \\ \Leftrightarrow \quad & a = b^c \\ \Leftrightarrow \quad & \log_b a = c \log_b b \\ \Leftrightarrow \quad & \log_b a = c. \end{aligned}$$

This shows us where the three cases in the statement of the theorem come from. Now we need to show the bound on $T(n)$ in the different cases. In the following paragraphs, we will use the facts (whose proof is a straightforward application of the definition of logarithms and rules of exponents) that for any x, y and z, each greater than 1, $x^{\log_y z} = z^{\log_y x}$ (see Corollary 4.8, Problem 5 at the end of the previous section, and Problem 3 at the end of this section) and that $\log_x y = \Theta(\log_2 y)$ (see Problem 4 at the end of the previous section).

In general, the total work done is computed by summing the expression for the work per level given in Equation 4.26 over all the levels, giving

$$\sum_{i=0}^{\log_b n} n^c \left(\frac{a}{b^c}\right)^i = n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i$$

In case 1, (part 1 in the statement of the theorem) this is n^c times a geometric series with a ratio of less than 1. Theorem 4.4 tells us that

$$n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i = \Theta(n^c).$$

Exercise 4.4-1 Prove Case 2 (part 2 of the statement) of the Master Theorem.

Exercise 4.4-2 Prove Case 3 (part 3 of the statement) of the Master Theorem.

In Case 2 we have that $\frac{a}{b^c} = 1$ and so

$$\begin{aligned} n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i &= n^c \sum_{i=0}^{\log_b n} 1^i \\ &= n^c(1 + \log_b n) \\ &= \Theta(n^c \log n). \end{aligned}$$

In Case 3, we have that $\frac{a}{b^c} > 1$. So in the series

$$\sum_{i=0}^{\log_b n} n^c \left(\frac{a}{b^c}\right)^i = n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i,$$

the largest term is the last one, so by Theorem 4.4,the sum is $\Theta\left(n^c \left(\frac{a}{b^c}\right)^{\log_b n}\right)$. But

$$\begin{aligned} n^c \left(\frac{a}{b^c}\right)^{\log_b n} &= n^c \cdot \frac{a^{\log_b n}}{(b^c)^{\log_b n}} \\ &= n^c \cdot \frac{n^{\log_b a}}{n^{\log_b b^c}} \\ &= n^c \cdot \frac{n^{\log_b a}}{n^c} \\ &= n^{\log_b a}. \end{aligned}$$

Thus the solution is $\Theta(n^{\log_b a})$. ■

We note that we may assume that a is a real number with $a > 1$ and give a somewhat similar proof (replacing the recursion tree with an iteration of the recurrence), but we do not give the details here.

Solving More General Kinds of Recurrences

Exercise 4.4-3 What can you say about the big-θ behavior of the solution to

$$T(n) = \begin{cases} 2T(n/3) + 4n^{3/2} & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

where n can be any nonnegative power of three?

Exercise 4.4-4 If $f(n) = n\sqrt{n+1}$, what can you say about the Big-Θ behavior of solutions to

$$S(n) = \begin{cases} 2S(n/3) + f(n) & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

where n can be any nonnegative power of three?

For Exercise 4.4-3, the work done at each level of the tree except for the bottom level will be four times the work done by the recurrence

$$T'(n) = \begin{cases} 2T'(n/3) + n^{3/2} & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

Thus the work done by T will be no more than four times the work done by T', but will be larger than the work done by T'. Therefore $T(n) = \Theta(T'(n))$. Thus by the master theorem, since $\log_3 2 < 1 < 3/2$, we have that $T(n) = \Theta(n^{3/2})$.

For Exercise 4.4-4, Since $n\sqrt{n+1} > n\sqrt{n} = n^{3/2}$ we have that $S(n)$ is at least as big as the solution to the recurrence

$$T'(n) = \begin{cases} 2T'(n/3) + n^{3/2} & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

where n can be any nonnegative power of three. But the solution to the recurrence for S will be no more than the solution to the recurrence in Exercise 4.4-3 for T, because $n\sqrt{n+1} \leq 4n^{3/2}$ for $n \geq 0$. Since $T(n) = \Theta(T'(n))$, then $S(n) = \Theta(T'(n))$ as well.

Extending the Master Theorem

As Exercise 4.4-3 and Exercise 4.4-4 suggest, there is a whole range of interesting recurrences that do not fit the master theorem but are closely related to recurrences that do. These recurrences have the same kind of behavior predicted by our original version of the Master Theorem, but the original version of the Master Theorem does not apply to them, just as it does not apply to the recurrences of Exercise 4.4-3 and Exercise 4.4-4.

We now state a second version of the Master Theorem that covers these cases. A still stronger version of the theorem may be found in *Introduction to Algorithms* by Cormen, et. al., but the version here captures much of the interesting behavior of recurrences that arise from the analysis of algorithms.

Theorem 4.10 *Let a and b be positive real numbers with $a \geq 1$ and $b > 1$. Let $T(n)$ be defined for powers n of b by*

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

Then

1. *if $f(n) = \Theta(n^c)$ where $\log_b a < c$, then $T(n) = \Theta(n^c) = \Theta(f(n))$.*
2. *if $f(n) = \Theta(n^c)$, where $\log_b a = c$, then $T(n) = \Theta(n^{\log_b a} \log_b n)$*
3. *if $f(n) = \Theta(n^c)$, where $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.*

Proof: We construct a recursion tree or iterate the recurrence. Since we have assumed that $f(n) = \Theta(n^c)$, there are constants c_1 and c_2, independent of the level, so that the work at each level is between $c_1 n^c \left(\frac{a}{b^c}\right)^i$ and $c_2 n^c \left(\frac{a}{b^c}\right)^i$ so from this point on the proof is largely a translation of the original proof. ■

Exercise 4.4-5 What does the Master Theorem tell us about the solutions to the recurrence

$$T(n) = \begin{cases} 3T(n/2) + n\sqrt{n+1} & \text{if } n > 1 \\ 1 & \text{if } n = 1? \end{cases}$$

As we saw in our solution to Exercise 4.4-4 $x\sqrt{x+1} = \Theta(x^{3/2})$. Since $2^{3/2} = \sqrt{2^3} = \sqrt{8} < 3$, we have that $\log_2 3 > 3/2$. Then by conclusion 3 of version 2 of the Master Theorem, $T(n) = \Theta(n^{\log_2 3})$.

The remainder of this section is devoted to carefully analyzing divide and conquer recurrences in which n is not a power of b and $T(n/b)$ is replaced by $T(\lceil n/b \rceil)$. While the details are somewhat technical, the end result is that the big-Θ behavior of such recurrences is the same as the corresponding recurrences for functions defined on powers of b. In particular, the following theorem is a consequence of what we prove.

Theorem 4.11 *Let a and b be positive real numbers with $a \geq 1$ and $b \geq 2$. Let $T(n)$ satisfy the recurrence*

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

Then

1. *if $f(n) = \Theta(n^c)$ where $\log_b a < c$, then $T(n) = \Theta(n^c) = \Theta(f(n))$.*
2. *if $f(n) = \Theta(n^c)$, where $\log_b a = c$, then $T(n) = \Theta(n^{\log_b a} \log_b n)$*
3. *if $f(n) = \Theta(n^c)$, where $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.*

(The condition that $b \geq 2$ can be changed to $B > 1$ with an appropriate change in the base case of the recurrence, but the base case will then depend on b.) The reader should be able to skip over the remainder of this section without loss of continuity.

More realistic recurrences (Optional)

So far, we have considered divide and conquer recurrences for functions $T(n)$ defined on integers n which are powers of b. In order to consider a more realistic recurrence in the master theorem, namely

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

or

$$T(n) = \begin{cases} aT(\lfloor n/b \rfloor) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

or even

$$T(n) = \begin{cases} a'T(\lceil n/b \rceil) + (a - a')T(\lfloor n/b \rfloor) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

it turns out to be easiest to first extend the domain for our recurrences to a much bigger set than the nonnegative integers, either the real or rational numbers, and then to work backwards.

For example, we can write a recurrence of the form

$$t(x) = \begin{cases} f(x)t(x/b) + g(x) & \text{if } x \geq b \\ k(x) & \text{if } 1 \leq x < b \end{cases}$$

for two (known) functions f and g defined on the real [or rational] numbers greater than 1 and one (known) function k defined on the real [or rational] numbers x with $1 \leq x < b$. Then so long as $b > 1$ it is possible to prove that there is a unique function t defined on the real [or rational] numbers greater than or equal to 1 that satisfies the recurrence. We use the lower case t in this situation as a signal that we are considering a recurrence whose domain is the real or rational numbers greater than or equal to 1.

Exercise 4.4-6 How would we compute $t(x)$ in the recurrence

$$t(x) = \begin{cases} 3t(x/2) + x^2 & \text{if } x \geq 2 \\ 5x & \text{if } 1 \leq x < 2 \end{cases}$$

if x were 7? How would we show that there is one and only one function t that satisfies the recurrence?

Exercise 4.4-7 Is it the case that there is one and only one solution to the recurrence

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

when f and g are (known) functions defined on the positive integers, and k and b are (known) constants with b an integer larger than or equal to 2?

To compute $t(7)$ in Exercise 4.4-6 we need to know $t(7/2)$. To compute $t(7/2)$, we need to know $t(7/4)$. Since $1 < 7/4 < 2$, we know that t$(7/4) = 35/4$. Then we may write

$$t(7/2) = 3 \cdot \frac{35}{4} + \frac{49}{4} = \frac{154}{4} = \frac{77}{2}.$$

Next we may write

$$\begin{aligned} t(7) &= 3t(7/2) + 7^2 \\ &= 3 \cdot \frac{77}{2} + 49 \\ &= \frac{329}{2}. \end{aligned}$$

Clearly we can compute $t(x)$ in this way for any x, though we are unlikely to enjoy the arithmetic. On the other hand suppose all we need to do is to show that there is a unique value of $t(x)$ determined by the recurrence, for all real numbers $x \geq 1$. If $1 \leq x < 2$, then $t(x) = 5x$, which uniquely determines $t(x)$. Given a number $x \geq 2$, there is a smallest integer i such that $x/2^i < 2$, and for this i, we have $1 \leq x/2^i$. We can now prove by induction on i that $t(x)$ is uniquely determined by the recurrence relation.

In Exercise 4.4-7 there is one and only one solution. Why? Clearly $T(1)$ is determined by the recurrence. Now assume inductively that $n > 1$ and that $T(m)$ is uniquely determined for positive integers $m < n$. We know that $n \geq 2$, so that $n/2 \leq n - 1$. Since $b \geq 2$, we know that $n/2 \geq n/b$, so that $n/b \leq n - 1$. Therefore $\lceil n/b \rceil < n$, so that we know by the inductive hypothesis that $T(\lceil n/b \rceil)$ is uniquely determined by the recurrence. Then by the recurrence,

$$T(n) = f(n)T\left(\left\lceil \frac{n}{b} \right\rceil\right) + g(n),$$

which uniquely determines $T(n)$. Thus by the principle of mathematical induction, $T(n)$ is determined for all positive integers n.

For every kind of recurrence we have dealt with, there is similarly one and only one solution. Because we know solutions exist, we don't find formulas for solutions to demonstrate that solutions exist, but rather to help us understand properties of the solutions. In this section and the last section, for example, we were interested in how fast the solutions grew as n grew large. This is why we were finding Big-O and Big-Θ bounds for our solutions.

Recurrences for general n (Optional)

We will now show how recurrences for arbitrary real numbers relate to recurrences involving floors and ceilings. We begin by showing that the conclusions of the Master Theorem apply to recurrences for arbitrary real numbers when we replace the real numbers by "nearby" powers of b.

Theorem 4.12 *Let a and b be positive real numbers with $b > 1$ and c and d be real numbers. Let $t(x)$ be the solution to the recurrence*

$$t(x) = \begin{cases} at(x/b) + x^c & \text{if } x \geq b \\ d & \text{if } 1 \leq x < b. \end{cases}$$

Let $T(n)$ be the solution to the recurrence

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n \geq 0 \\ d & \text{if } n = 1, \end{cases}$$

defined for n a nonnegative integer power of b. Let $m(x)$ be the largest integer power of b less than or equal to x. Then $t(x) = \Theta(T(m(x)))$

Proof: If we iterate (or, in the case that a is an integer, draw recursion trees for) the two recurrences, we can see that the results of the iterations are nearly identical. This means the solutions to the recurrences have the same big-Θ behavior. See the Appendix to this Section for details. ■

Removing Floors and Ceilings (Optional)

We have also pointed out that a more realistic Master Theorem would apply to recurrences of the form $T(n) = aT(\lfloor n/b \rfloor) + n^c$, or $T(n) = aT(\lceil n/b \rceil) + n^c$, or even $T(n) = a'T(\lceil n/b \rceil) + (a - a')T(\lfloor n/b \rfloor) + n^c$. For example, if we are applying mergesort to an array of size 101, we really break it into pieces, of size 50 and 51. Thus the recurrence we want is not really $T(n) = 2T(n/2) + n$, but rather $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$.

We can show, however, that one can essentially "ignore" the floors and ceilings in typical divide-and-conquer recurrences. If we remove the floors and ceilings from a recurrence relation, we convert it from a recurrence relation defined on the integers to one defined on the rational numbers. However we have already seen that such recurrences are not difficult to handle.

The theorem below says that in recurrences covered by the master theorem, if we remove ceilings, our recurrences still have the same big-Θ bounds on their solutions. A similar proof shows that we may remove floors and still get the same big-Θ bounds. Without too much more work we can see that we can remove floors and ceilings simultaneously without changing the big-Θ bounds on our solutions. Since we may remove either floors or ceilings, that means that we may deal with recurrences of the form $T(n) = a'T(\lceil n/b \rceil) + (a - a')T(\lfloor n/b \rfloor) + n^c$. The condition that $b > 2$ can be replaced by $b > 1$, but the base case for the recurrence will depend on b.

Theorem 4.13 *Let a and b be positive real numbers with $b \geq 2$ and let c and d be real numbers. Let $T(n)$ be the function defined on the integers by the recurrence*

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

and let $t(x)$ be the function on the real numbers defined by the recurrence

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

Then $T(n) = \Theta(t(n))$. The same statement applies with ceilings replaced by floors.

Proof: As in the previous theorem, we can consider iterating the two recurrences. It is straightforward (though dealing with the notation is difficult) to show that for a given value of n, the iteration for computing $T(n)$ has at most two more levels than the iteration for computing $t(n)$. The work per level also has the same Big-Θ bounds at each level, and the work for the two additional levels of the iteration for $T(n)$ has the same Big-Θ bounds as the work at the bottom level of the recursion tree for $t(n)$. We give the details in the appendix at the end of this section. ■

Theorem 4.12 and Theorem 4.13 tell us that the Big-Θ behavior of solutions to our more realistic recurrences

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

is determined by their Big-Θ behavior on powers of the base b.

Floors and ceilings in the stronger version of the Master Theorem (Optional)

In our first version of the master theorem, we showed that we could ignore ceilings and assume our variables were powers of b. In fact we can ignore them in circumstances where the function telling us the "work" done at each level of our recursion tree is $\Theta(x^c)$ for some positive real number c. This lets us apply the second version of the master theorem to recurrences of the form $T(n) = aT(\lceil n/b \rceil) + f(n)$.

Theorem 4.14 *Theorems 4.12 and 4.13 apply to recurrences in which the x^c or n^c term is replaced by $f(x)$ or $f(n)$ for a function f with $f(x) = \Theta(x^c)$.*

Proof: We iterate the recurrences or construct recursion trees in the same way as in the proofs of the original theorems, and find that the condition $f(x) = \Theta(x^c)$ gives us enough information to again bound the solution above and below with multiples of the solution of the recurrence with x^c. The details are similar to those in the original proofs. ■

Appendix: Proofs of Theorems (Optional)

For convenience, we repeat the statements of the earlier theorems whose proofs we merely outlined.

Theorem 4.12 *Let a and b be positive real numbers with $b > 1$ and c and d be real numbers. Let $t(x)$ be the solution to the recurrence*

$$t(x) = \begin{cases} at(x/b) + x^c & \text{if } x \geq b \\ d & \text{if } 1 \leq x < b. \end{cases}$$

Let $T(n)$ be the solution to the recurrence

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n \geq 0 \\ d & \text{if } n = 1, \end{cases}$$

defined for n is a nonnegative integer power of b. Let $m(x)$ be the largest integer power of b less than or equal to x. Then $t(x) = \Theta(T(m(x)))$

Proof: By iterating each recursion 4 times (or using a four level recursion tree in the case that a is an integer), we see that

$$t(x) = a^4 t\Big(\frac{x}{b^4}\Big) + \Big(\frac{a}{b^c}\Big)^3 x^c + \Big(\frac{a}{b^c}\Big)^2 x^c + \frac{a}{b^c} x^c$$

and

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

Thus, continuing until we have a solution, in both cases we get a solution that starts with a raised to an exponent that we will denote as either $e(x)$ or $e(n)$ when we want to distinguish between them and e when it is unnecessary to distinguish. The solution for t will be a^e times $t(x/b^e)$ plus x^c times a geometric series $\sum_{i=0}^{e-1}\Big(\frac{a}{b^c}\Big)^i$. The solution for T will be a^e times d plus n^c times a geometric series $\sum_{i=0}^{e-1}\Big(\frac{a}{b^c}\Big)^i$. In both cases $t(x/b^e)$ (or $T(n/b^e)$) will be d. In both cases the geometric series will be $\Theta(1)$, $\Theta(e)$ or $\Theta\Big(\frac{a}{b^c}\Big)^e$, depending on whether $\frac{a}{b^c}$ is less than 1, equal to 1, or greater than one. Clearly $e(n) = \log_b n$. Since we must divide x by b an integer number greater than $\log_b x - 1$ times in order to get a value in the range from 1 to b, $e(x) = \lfloor \log_b x \rfloor$. Thus, if m is the largest integer power of b less than or equal to x, then $0 \le e(x) - e(m) < 1$. Let us use r to stand for the real number $\frac{a}{b^c}$. Then we have $r^0 \le r^{e(x)-e(m)} < r$, or $r^{e(m)} \le r^{e(x)} \le r \cdot r^{e(m)}$. Thus we have $r^{e(x)} = \Theta(r^{e(m)})$ Finally, $m^c \le x^c \le b^c m^c$, and so $x^c = \Theta(m^c)$. Therefore, every term of $t(x)$ is Θ of the corresponding term of $T(m)$. Further, there are only a fixed number of different constants involved in our Big-Θ bounds. Therefore since $t(x)$ is composed of sums and products of these terms, $t(x) = \Theta(T(m))$. ■

Theorem 4.13 *Let a and b be positive real numbers with $b \ge 2$ and let c and d be real numbers. Let $T(n)$ be the function defined on the integers by the recurrence*

$$T(n) = \begin{cases} aT(\lceil n/b \rceil) + n^c & \text{if } n \ge b \\ d & n = 1, \end{cases}$$

and let $t(x)$ be the function on the real numbers defined by the recurrence

$$t(x) = \begin{cases} at(x/b) + x^c & \text{if } x \ge b \\ d & \text{if } 1 \le x < b. \end{cases}$$

Then $T(n) = \Theta(t(n))$.

Proof: As in the previous proof, we can iterate both recurrences. Let us compare what the results will be of iterating the recurrence for $t(n)$ and the recurrence for $T(n)$ the same number of times. Note that

$$\begin{aligned} \lceil n/b \rceil &< n/b + 1 \\ \lceil \lceil n/b \rceil / b \rceil < \lceil n/b^2 + 1/b \rceil &< n/b^2 + 1/b + 1 \\ \lceil \lceil \lceil n/b \rceil / b \rceil / b \rceil < \lceil n/b^3 + 1/b^2 + 1/b \rceil &< n/b^3 + 1/b^2 + 1/b + 1 \end{aligned}$$

This suggests that if we define $n_0 = n$, and $n_i = \lceil n_{i-1}/b \rceil$, then, using the fact that $b \ge 2$, it is straightforward to prove by induction, or with the formula for the sum of a geometric series,

that $n_i < n/b^i + 2$. The number n_i is the argument of T in the ith iteration of the recurrence for T. We have just seen that it differs from the argument of t in the ith iteration of t by at most 2. In particular, we might have to iterate the recurrence for T twice more than we iterate the recurrence for t to reach the base case. When we iterate the recurrence for t, we get the same solution we got in the previous theorem, with n substituted for x. When we iterate the recurrence for T, we get for some integer j that

$$T(n) = a^j d + \sum_{i=0}^{j-1} a^i n_i^c,$$

with $\frac{n}{b^i} \le n_i \le \frac{n}{b^i} + 2$. But, so long as $n/b^i \ge 2$, we have $n/b^i + 2 \le n/b^{i-1}$. Since the number of iterations of T is at most two more than the number of iterations of t, and since the number of iterations of t is $\lfloor \log_b n \rfloor$, we have that j is at most $\lfloor \log_b n \rfloor + 2$. Therefore all but perhaps the last three values of n_i are less than or equal to n/b^{i-1}, and these last three values are at most b^2, b, and 1. Putting all these bounds together and using $n_0 = n$ gives us

$$\begin{aligned} \sum_{i=0}^{j-1} a^i \Big(\frac{n}{b^i}\Big)^c &\le \sum_{i=0}^{j-1} a^i n_i^c \\ &\le n^c + \sum_{i=1}^{j-4} a^i \Big(\frac{n}{b^{i-1}}\Big)^c + a^{j-2}(b^2)^c + a^{j-1}b^c + a^j 1^c, \end{aligned}$$

or

$$\begin{aligned} \sum_{i=0}^{j-1} a^i \Big(\frac{n}{b^i}\Big)^c &\le \sum_{i=0}^{j-1} a^i n_i^c \\ &\le n^c + b\sum_{i=1}^{j-4} a^i \Big(\frac{n}{b^i}\Big)^c + a^{j-2}\left(\frac{b^j}{b^{j-2}}\right)^c + a^{j-1}\left(\frac{b^j}{b^{j-1}}\right)^c + a^j\left(\frac{b^j}{b^j}\right)^c. \end{aligned}$$

As we shall see momentarily these last three "extra" terms and the b in front of the summation sign do not change the Big-Θ behavior of the right-hand side.

As in the proof of the master theorem, the Big-Θ behavior of the left hand side depends on whether a/b^c is less than 1, in which case it is $\Theta(n^c)$, equal to 1, in which case it is $\Theta(n^c \log_b n)$, or greater than one in which case it is $\Theta(n^{\log_b a})$. But this is exactly the Big-Θ behavior of the right-hand side, because $n < b^j < nb^2$, so $b^j = \Theta(n)$, which means that $\left(\frac{b^j}{b^i}\right)^c = \Theta\left(\left(\frac{n}{b^i}\right)^c\right)$, and the b in front of the summation sign does not change its Big-Θ behavior. Adding $a^j d$ to the middle term of the inequality to get $T(n)$ does not change this behavior. But this modified middle term is exactly $T(n)$. Since the left and right hand sides have the same big-Θ behavior as $t(n)$, we have T(n) = $\Theta(t(n))$. ■

Important Concepts, Formulas, and Theorems

1. *Master Theorem, simplified version.* The simplified version of the *Master Theorem* states: Let a be an integer greater than or equal to 1 and b be a real number greater than 1. Let c be a positive real number and d a nonnegative real number. Given a recurrence of the form

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

then for n a power of b,

(a) if $\log_b a < c$, $T(n) = \Theta(n^c)$,

(b) if $\log_b a = c$, $T(n) = \Theta(n^c \log n)$,

(c) if $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$.

2. *Properties of Logarithms.* For any x, y and z, each greater than 1, $x^{\log_y z} = z^{\log_y x}$. Also, $\log_x y = \Theta(\log_2 y)$.

3. *Master Theorem, More General Version.* Let a and b be positive real numbers with $a \geq 1$ and $b \geq 2$. Let $T(n)$ be defined for powers n of b by

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

Then

(a) if $f(n) = \Theta(n^c)$ where $\log_b a < c$, then $T(n) = \Theta(n^c) = \Theta(f(n))$.

(b) if $f(n) = \Theta(n^c)$, where $\log_b a = c$, then $T(n) = \Theta(n^{\log_b a} \log_b n)$

(c) if $f(n) = \Theta(n^c)$, where $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.

A similar result with a base case that depends on b holds when $1 < b < 2$.

4. *Important Recurrences have Unique Solutions. (Optional.)* The recurrence

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

has a unique solution when f and g are (known) functions defined on the positive integers, and k and b are (known) constants with b an integer larger than 2.

5. *Recurrences Defined on the Positive Real Numbers and Recurrences Defined on the Positive Integers. (Optional.)* Let a and b be positive real numbers with $b > 1$ and c and d be real numbers. Let $t(x)$ be the solution to the recurrence

$$t(x) = \begin{cases} at(x/b) + x^c & \text{if } x \geq b \\ d & \text{if } 1 \leq x < b. \end{cases}$$

Let $T(n)$ be the solution to the recurrence

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n \geq 0 \\ d & \text{if } n = 1, \end{cases}$$

where n is a nonnegative integer power of b. Let $m(x)$ be the largest integer power of b less than or equal to x. Then $t(x) = \Theta(T(m(x)))$

6. *Removing Floors and Ceilings from Recurrences. (Optional.)* Let a and b be positive real numbers with $b \geq 2$ and let c and d be real numbers. Let $T(n)$ be the function defined on the integers by the recurrence

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

and let $t(x)$ be the function on the real numbers defined by the recurrence

$$t(x) = \begin{cases} at(x/b) + x^c & \text{if } x \geq b \\ d & \text{if } 1 \leq x < b \end{cases}.$$

Then $T(n) = \Theta(t(n))$. The same statement applies with ceilings replaced by floors.

7. *Extending 5 and 6 (Optional.)* In the theorems summarized in 5 and 6 the n^c or x^c term may be replaced by a function f with $f(x) = \Theta(x^c)$.

8. *Solutions to Realistic Recurrences.* The theorems summarized in 5, 6, and 7 tell us that the Big-Θ behavior of solutions to our more realistic recurrences

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

where $f(n) = \Theta(n^c)$, is determined by their Big-Θ behavior on powers of the base b and with $f(n) = n^c$.

Problems

1. Use the master theorem to give Big-Θ bounds on the solutions to the following recurrences. For all of these, assume that $T(1) = 1$ and n is a power of the appropriate integer.

   (a) $T(n) = 8T(n/2) + n$
   (b) $T(n) = 8T(n/2) + n^3$
   (c) $T(n) = 3T(n/2) + n$
   (d) $T(n) = T(n/4) + 1$
   (e) $T(n) = 3T(n/3) + n^2$

2. Extend the proof of the Master Theorem, Theorem 4.9 to the case $T(1) = d$.

3. Show that for any x, y and z, each greater than 1, $x^{\log_y z} = z^{\log_y x}$.

4. (Optional) Show that for each real number $x \geq 0$ there is one and only one value of $t(x)$ given by the recurrence

$$t(x) = \begin{cases} 7xt(x-1) + 1 & \text{if } x \geq 1 \\ 1 & \text{if } 0 \leq x < 1. \end{cases}$$

5. (Optional) Show that for each real number $x \geq 1$ there is one and only one value of $t(x)$ given by the recurrence

$$t(x) = \begin{cases} 3xT(x/2) + x^2 & \text{if } x \geq 2 \\ 1 & \text{if } 1 \leq x < 2 \end{cases}.$$

6. (Optional) How many solutions are there to the recurrence

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

if $b < 2$? If $b = 10/9$, by what would we have to replace the condition that $T(n) = k$ if $n = 1$ in order to get a unique solution?

7. Give a big-Θ bound on the solution to the recurrence

$$T(n) = \begin{cases} 3T(\lceil n/2 \rceil) + \sqrt{n+3} & \text{if } n > 1 \\ d & \text{if } n = 1. \end{cases}$$

8. Give a big-Θ bound on the solution to the recurrence

$$T(n) = \begin{cases} 3T(\lceil n/2 \rceil) + \sqrt{n^3+3} & \text{if } n > 1 \\ d & \text{if } n = 1. \end{cases}$$

9. Give a big-Θ bound on the solution to the recurrence

$$T(n) = \begin{cases} 3T(\lceil n/2 \rceil) + \sqrt{n^4+3} & \text{if } n > 1 \\ d & \text{if } n = 1. \end{cases}$$

10. Give a big-Θ bound on the solution to the recurrence

$$T(n) = \begin{cases} 2T(\lceil n/2 \rceil) + \sqrt{n^2+3} & \text{if } n > 1 \\ d & \text{if } n = 1. \end{cases}$$

11. (Optional) Explain why theorem 4.11 is a consequence of Theorem 4.12 and Theorem 4.13

4.5 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

$$T(n) \leq 2T(n/2) + cn. \tag{4.27}$$

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 4.27 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 4.27. (For example for any $c' < c$, the unique solution to

$$T(n) = \begin{cases} 2T(n/2) + c'n & \text{if } n \geq 2 \\ k & \text{if } n = 1 \end{cases} \tag{4.28}$$

satisfies Inequality 4.27 for any constant k.) The idea that Recurrence 4.27 has infinitely many solutions, while Recurrence 4.28 has exactly one solution is analogous to the idea that $x - 3 \leq 0$ has infinitely many solutions while $x - 3 = 0$ has one solution. Later in this section we shall see how to show that all the solutions to Recurrence 4.27 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 4.5-1 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 4.5-1. Here we wish to show that $T(n) = O(n \log n)$. From 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 $2 \leq m < n$ then $T(m) \leq km \log m$. Now $n/2 < n$, 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

$$\begin{aligned} T(n) \leq 2T(n/2) + cn &\leq 2k\frac{n}{2} \log \frac{n}{2} + cn && (4.29) \\ &= kn \log \frac{n}{2} + cn && (4.30) \\ &= kn \log n - kn \log 2 + cn && (4.31) \\ &= kn \log n - kn + cn. && (4.32) \end{aligned}$$

Recall that we are trying to show that $T(n) \leq kn \log n$. But that is not quite what Line 4.32 tells us. 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 everyone 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 $T(n) \leq 2T(n/2) + cn$ for all $n > 1$. Choose k to be larger than or equal to $\frac{T(2)}{2}$ and larger than or equal to 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 preceding 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. If you compare the formal proof we just gave with the informal discussion that preceded it, you will find each step of the formal proof actually corresponds to something we said in the informal discussion. Since the informal discussion explained why we were making the choices we did, it is natural that some people prefer the informal explanation to the formal proof.

Further Wrinkles in Induction Proofs

Exercise 4.5-2 Suppose that c is a real number greater than zero. 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 4.5-3 Suppose that c is a real number greater than zero. 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 4.5-2 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 $3c/2$ and we choose $n_0 = 1$. 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 4.5-3. 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 that there are an n_0 and a k such that $T(n) \leq kn^2$ for $n > n_0$. Assuming that we have chosen n_0 and 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 usual way to describe the problem we are facing 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, $T(n) \leq k_1 n^2 - k_2 n$. Now proceeding as before, we get

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

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

Now we have to make $(c - k_2)n \leq 0$ for the last line to be at most $k_1 n^2 - k_2 n$, and so we just choose $k_2 \geq c$ (and greater than whatever we need in order to make a base case work). Since $T(n) \leq k_1 n^2 - k_2 n$ 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 induction: a stronger statement is often easier to prove than a weaker one. 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 the statements, $p(1), \ldots, p(n-1)$ 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:

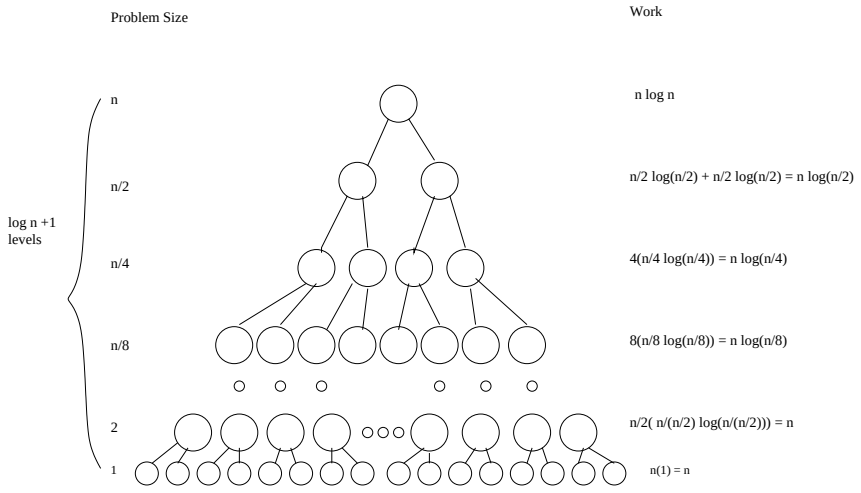
$$T(n) = \begin{cases} 2T(n/2) + n\log n & \text{if } n > 1 \\ 1 & \text{if } n = 1, \end{cases}$$

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 4.8.

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 the bottom level, it is n, the number of subproblems, times 1. Thus if we sum the work per level we get

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

Figure 4.8: The recursion tree for $T(n) = 2T(n/2) + n \log n$ if $n > 1$ and $T(1) = 1$.



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

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 4.5-4 Find the best big-O bound you can on the solution to the recurrence

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

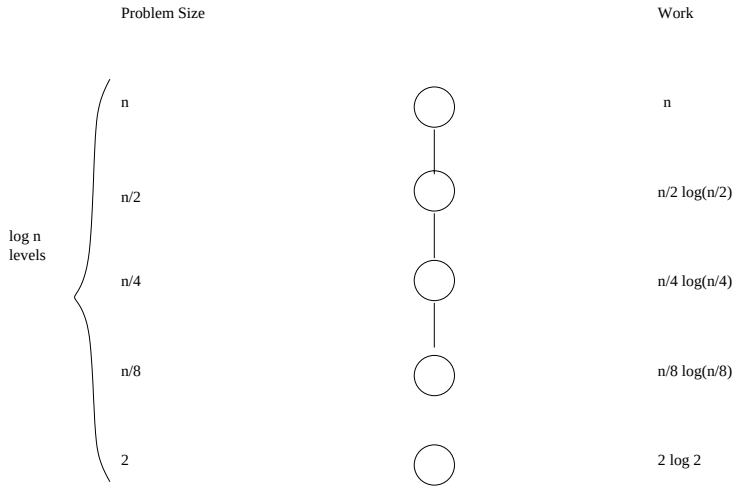
assuming n is a power of 2. Is this bound a big-Θ bound?

The tree for this recurrence is in Figure 4.9

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

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

Figure 4.9: The recursion tree for the recurrence $T(n) = T(n/2) + n \log n$ if $n > 1$ and $T(1) = 1$.



$$\begin{aligned} &\leq 1 + n(\log n)(2) \\ &= O(n \log n). \end{aligned}$$

Note that 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 $n \log n$. Therefore, we have $T(n) = \Theta(n \log n)$.

Removing Ceilings and Using Powers of b. (Optional)

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 the two theorems we used 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

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

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 4.5-5 Show that if $f(x) = x^2 \log x$, then $f(bx) = \Theta(f(x))$.

Exercise 4.5-6 If $f(x) = 3^x$ and $b = 2$, is $f(bx) = \Theta(f(x))$? Is $f(b(x)) = O(f(x))$?

For Exercise 4.5-5 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 4.15 *Theorems 4.12 and 4.13 apply to recurrences in which the x^c term is replaced by an increasing function f 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 4.12 and 4.13 for recurrence inequalities also. The proofs involve a similar analysis of iterated recurrences or recursion trees, and so we omit them.

Theorem 4.16 *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 $T(n)$ to the recurrence

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

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

Theorem 4.17 *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

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

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 explanation 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 a *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 and iterating the recurrence are appropriate tools even though the Master Theorem does not apply. The same holds for recurrence inequalities.

6. *Increasing function. (Optional.)* 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. (Optional.)* 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 problem 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 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 bound?

8. (Optional) 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 solution to 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 integers by

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

Prove by induction that your answer is correct.

4.6 Recurrences and Selection

The idea of selection

One common problem that arises in algorithms is that of *selection*. In this problem you are given n distinct data items from some set which has an underlying order. That is, given any two items a and b, you can determine whether $a < b$. (Integers satisfy this property, but colors do not.) Given these n items, and some value i, $1 \leq i \leq n$, you wish to find the ith smallest item in the set. For example in the set

$$\{3, 1, 8, 6, 4, 11, 7\}, \tag{4.33}$$

the first smallest ($i = 1$) is 1, the third smallest ($i = 3$) is 4 and the seventh smallest ($i = n = 7$) is 11. An important special case is that of finding the *median*, which is the case of $i = \lceil n/2 \rceil$. Another important special case is finding percentiles; for example the 90th percentile is the case $i = \lceil .9n \rceil$. As this suggests, i is frequently given as some fraction of n.

Exercise 4.6-1 How do you find the minimum ($i = 1$) or maximum ($i = n$) in a set? What is the running time? How do you find the second smallest element? Does this approach extend to finding the ith smallest? What is the running time?

Exercise 4.6-2 Give the fastest algorithm you can to find the median ($i = \lceil n/2 \rceil$).

In Exercise 4.6-1, the simple $O(n)$ algorithm of going through the list and keeping track of the minimum value seen so far will suffice to find the minimum. Similarly, if we want to find the second smallest, we can go through the list once, find the smallest, remove it and then find the smallest in the new list. This also takes $O(n + n - 1) = O(n)$ time. If we extend this to finding the ith smallest, the algorithm will take $O(in)$ time. Thus for finding the median, this method takes $O(n^2)$ time.

A better idea for finding the median is to first sort the items, and then take the item in position $n/2$. Since we can sort in $O(n \log n)$ time, this algorithm will take $O(n \log n)$ time. Thus if $i = O(\log n)$ we might want to run the algorithm of the previous paragraph, and otherwise run this algorithm.[4]

All these approaches, when applied to the median, take at least some multiple of $(n \log n)$ units of time.[5] The best sorting algorithms take $O(n \log n)$ time also, and one can prove every comparison-based sorting algorithm takes $\Omega(n \log n)$ time. This raises the natural question of whether it is possible to do selection any faster than sorting. In other words, is the problem of finding the median element, or of finding the ith smallest element of a set, significantly easier than the problem of ordering (sorting) the whole set?

A recursive selection algorithm

Suppose for a minute that we magically knew how to find the median in $O(n)$ time. That is, we have a routine MagicMedian, that given as input a set A, returns the median. We could then use this in a divide and conquer algorithm for Select as follows:

[4] We also note that the running time can be improved to $O(n + i \log n)$ by first creating a *heap*, which takes $O(n)$ time, and then performing a Delete-Min operation i times.

[5] An alternate notation for $f(x) = O(g(x))$ is $g(x) = \Omega(f(x))$. Notice the change in roles of f and g. In this notation, we say that all of these algorithms take $\Omega(n \log n)$ time.

```
Select(A, i, n)
(selects the ith smallest element in set A, where n = |A|)
(1)  if (n = 1)
(2)       return the one item in A
(3)  else
(4)       p = MagicMedian(A)
(5)       Let H be the set of elements greater than p
(6)       Let L be the set of elements less than or equal to p
(7)       if (i ≤ |L|)
(8)            Return Select(L, i, |L|)
(9)       else
(10)           Return Select(H, i − |L|, |H|).
```

By H we do not mean the elements that come after p in the list, but the elements of the list which are larger than p in the underlying ordering of our set. This algorithm is based on the following simple observation. If we could divide the set A up into a "lower half" (L) and an "upper" half (H), then we know in which of these two sets the ith smallest element in A will be. Namely, if $i \leq \lceil n/2 \rceil$, it will be in L, and otherwise it will be in H. Thus, we can recursively look in one or the other set. We can easily partition the data into two sets by making two passes, in the first we copy the numbers smaller than p into L, and in the second we copy the numbers larger than p into H.[6]

The only additional detail is that if we look in H, then instead of looking for the ith smallest, we look for the $i - \lceil n/2 \rceil$th smallest, as H is formed by removing the $\lceil n/2 \rceil$ smallest elements from A.

For example, if the input is the set given in 4.33, and p is 6, the set L would be $\{3, 1, 6, 4\}$, and H would be $\{8, 11, 7\}$. If i were 2, we would recurse on the set L, with $i = 2$. On the other hand, if i were 6, we would recurse on the set H, with $i = 6 - 4 = 2$. Observe that the second smallest element in H is 8, as is the sixth smallest element in the original set.

We can express the running time of Select by the following recurrence:

$$T(n) \leq T(n/2) + cn \ . \tag{4.34}$$

From the master theorem, we know any function which satisfies this recurrence has $T(n) = O(n)$.

So we can conclude that if we already know how to find the median in linear time, we can design a divide and conquer algorithm that will solve the selection problem in linear time. However, this is nothing to write home about (yet)!

Selection without knowing the median in advance

Sometimes a knowledge of solving recurrences can help us design algorithms. What kinds of recurrences do we know about that have solutions $T(n)$ with $T(n) = O(n)$? In particular, consider recurrences of the form $T(n) \leq T(n/b) + cn$, and ask when they have solutions with $T(n) = O(n)$. Using the master theorem, we see that as long as $\log_b 1 < 1$ (and since $\log_b 1 = 0$

[6] We can do this more efficiently, and "in place", using the partition algorithm of quicksort.

for any b, then any b allowed by the master theorem works; that is, any $b > 1$ will work), all solutions to this recurrence will have $T(n) = O(n)$. (Note that b does not have to be an integer.) If we let $b' = 1/b$, we can say equivalently that as long as we can solve a problem of size n by solving (recursively) a problem of size $b'n$, for some $b' < 1$, and also doing $O(n)$ additional work, our algorithm will run in $O(n)$ time. Interpreting this in the selection problem, it says that as long as we can, in $O(n)$ time, choose p to ensure that both L and H have size at most $b'n$, we will have a linear time algorithm. (You might ask "What about actually dividing our set into L and H, doesn't that take some time too?" The answer is yes it does, but we already know we can do the division into H and L in time $O(n)$, so if we can find p in time $O(n)$ also, then we can do both these things in time $O(n)$.)

In particular, suppose that, in $O(n)$ time, we can choose p to ensure that both L and H have size at most $(3/4)n$. Then the running time is described by the recurrence $T(n) = T(3n/4)+O(n)$ and we will be able to solve the selection problem in linear time.

To see why $(3/4)n$ is relevant, suppose instead of the "black box" MagicMedian, we have a much weaker magic black box, one which only guarantees that it will return some number in the middle half of our set in time $O(n)$. That is, it will return a number that is guaranteed to be somewhere between the $n/4$th smallest number and the $3n/4$th smallest number. If we use the number given by this magic box to divide our set into H and L, then neither will have size more than $3n/4$. We will call this black box a MagicMiddle box, and can use it in the following algorithm:

```
Select1(A,i,n)
(selects the ith smallest element in set A, where n = |A| )
(1)  if (n = 1)
(2)       return the one item in A
(3)  else
(4)       p = MagicMiddle(A)
(5)       Let H be the set of elements greater than p
(6)       Let L be the set of elements less than or equal to p
(7)       if (i ≤ |L|)
(8)            Return Select1(L, i, |L|)
(9)       else
(10)           Return Select1(H, i − |L|, |H|).
```

The algorithm Select1 is similar to Select. The only difference is that p is now only guaranteed to be in the middle half. Now, when we recurse, we decide whether to recruse on L or H based on whether i is less than or equal to $|L|$. The element p is called a *partition element*, because it is used to partition our set A into the two sets L and H.

This is progress, as we now don't need to assume that we can find the median in order to have a linear time algorithm, we only need to assume that we can find one number in the middle half of the set. This problem seems simpler than the original problem, and in fact it is. Thus our knowledge of which recurrences have solutions which are $O(n)$ led us toward a more plausible algorithm.

It takes a clever algorithm to find an item in the middle half of our set. We now describe such an algorithm in which we first choose a subset of the numbers and then *recursively* find the median of that subset.

An algorithm to find an element in the middle half

More precisely, consider the following algorithm in which we assume that $|A|$ is a multiple of 5. (The condition that $n < 60$ in line 2 is a technical condition that will be justified later.)

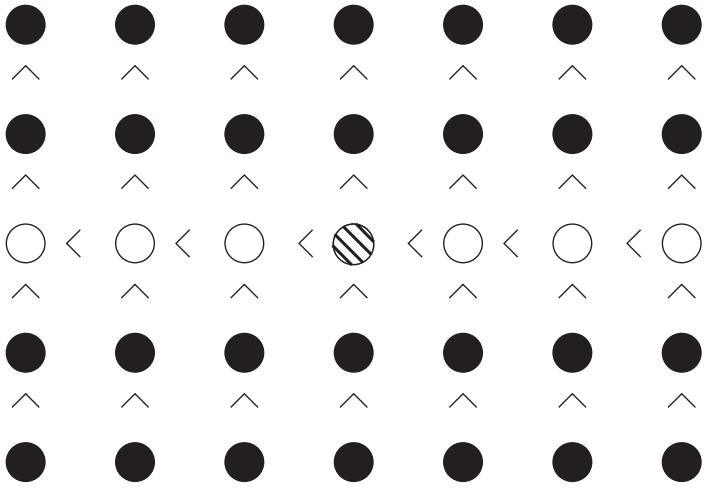
```
MagicMiddle(A)
(1)  Let n = |A|
(2)  if (n < 60)
(3)       use sorting to return the median of A
(4)  else
(5)       Break A into k = n/5 groups of size 5, G_1, ..., G_k
(6)       for i = 1 to k
(7)            find m_i, the median of G_i (by sorting)
(8)       Let M = {m_1, ..., m_k}
(9)       return Select1 (M, ⌈k/2⌉, k).
```

In this algorithm, we break A into $n/5$ sets of size 5, and then find the median of each set. We then (using Select1 recursively) find the median of medians and return this as our p.

Lemma 4.18 *The value returned by MagicMiddle(A) is in the middle half of A.*

Proof: Consider arranging the elements as follows. List each set of 5 vertically in sorted order, with the smallest element on top. Then line up all $n/5$ of these lists, ordered by their medians, smallest on the left. We get the picture in Figure 4.10. In this picture, the medians

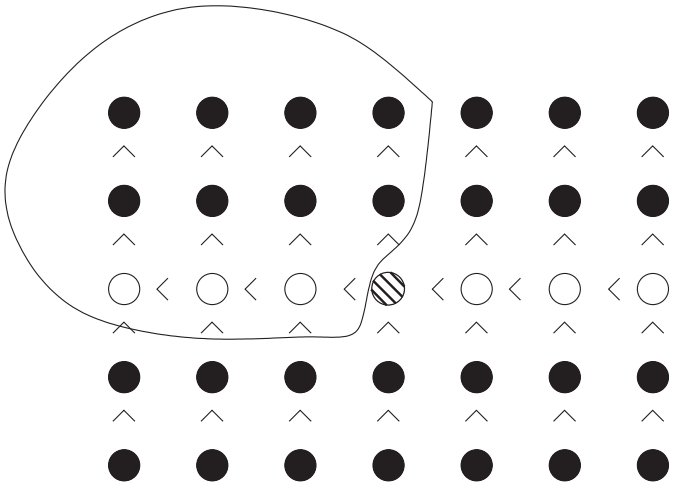
Figure 4.10: Dividing a set into $n/5$ parts of size 5, finding the median of each part and the median of the medians.



are in white, the median of medians is cross-hatched, and we have put in all the inequalities that we know from the ordering information that we have. Now, consider how many items are less than or equal to the median of medians. Every smaller median is clearly less than the median

of medians and, in its 5 element set, the elements smaller than the median are also smaller than the median of medians. Now in Figure 4.11 we circle a set of elements that is guaranteed to be smaller than the median of medians. In one fewer (or in the case of an odd number of columns

Figure 4.11: The circled elements are less than the median of the medians.



as in Figure 4.11, one half fewer) than half the columns, we have circled 3 elements and in one column we have circled 2 elements. Therefore, we have circled at least[7]

$$\left(\frac{1}{2}\left(\frac{n}{5}\right)-1\right)3+2=\frac{3n}{10}-1$$

elements.

So far we have assumed n is an exact multiple of 5, but we will be using this idea in circumstances when it is not. If it is not an exact multiple of 5, we will have $\lceil n/5 \rceil$ columns (in particular more than $n/5$ columns), but in one of them we might have only one element. It is possible that this column is one of the ones we counted on for 3 elements, so our estimate could be two elements too large.[8] Thus we have circled at least

$$\frac{3n}{10}-1-2=\frac{3n}{10}-3$$

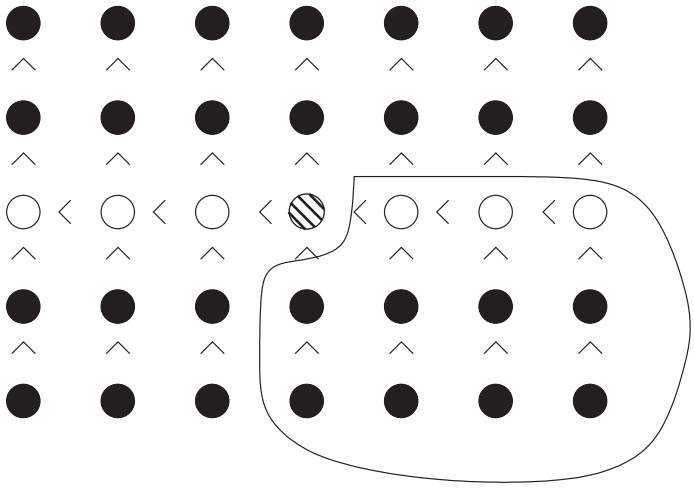
elements. It is a straightforward argument with inequalities that as long as $n \geq 60$, this quantity is at least $n/4$. So if at least $n/4$ items are guaranteed to be less than the median, then at most $3n/4$ items can be greater than the median, and hence $|H| \leq 3n/4$.

A set of elements that is guaranteed to be larger than the median of medians is circled in the Figure 4.12. We can make the same argument about the number of larger elements circled when the number of columns is odd; when the number of columns is even, a similar argument shows that we circle even more elements. By the same argument as we used with $|H|$, this shows that the size of L is at most $3n/4$. ■

[7] We say "at least" because our argument applies exactly when n is even, but underestimates the number of circled elements when n is odd.

[8] A bit less than 2 because we have more than $n/5$ columns.

Figure 4.12: The circled elements are greater than the median of the medians.



Note that we don't actually identify all the nodes that are guaranteed to be, say, less than the median of medians, we are just guaranteed that the proper number exists.

Since we only have the guarantee that MagicMiddle gives us an element in the middle half of the set if the set has at least sixty elements, we modify Select1 to start out by checking to see if $n < 60$, and sorting the set to find the element in position i if $n < 60$. Since 60 is a constant, sorting and finding the desired element takes at most a constant amount of time.

An analysis of the revised selection algorithm

Exercise 4.6-3 Let $T(n)$ be the running time of the modified Select1 on n items. How can you express the running time of Magic Middle in terms of $T(n)$?

Exercise 4.6-4 What is a recurrence for the running time of Select1? Hint: how could Exercise 4.6-3 help you?

Exercise 4.6-5 Can you prove by induction that each solution to the recurrence for Select1 is $O(n)$?

For Exercise 4.6-3, we have the following steps.

- The first step of MagicMiddle is to divide the items into sets of five; this takes $O(n)$ time.
- We then have to find the median of each five-element set. (We can find this median by any straightforward method we choose and still only take at most a constant amount of time; we don't use recursion here.) There are $n/5$ sets and we spend no more than some constant time per set, so the total time is $O(n)$.
- Next we recursively call Select1 to find the median of medians; this takes $T(n/5)$ time.
- Finally, we partition A into those elements less than or equal to the "magic middle" and those that are not, which takes $O(n)$ time.

Thus the total running time is $T(n/5)+O(n)$, which implies that for some n_0 there is a constant $c_0 > 0$ such that, for all $n > n_0$, the running time is no more than $c_0 n$. Even if $n_0 > 60$, there are only finitely many cases between 60 and n_0 so there is a constant c such that for $n \geq 60$, the running time of Magic Middle is no more than $T(n/5) + cn$.

We now get a recurrence for the running time of Select1. Note that for $n \geq 60$ Select1 has to call Magic Middle and then recurse on either L or H, each of which has size at most $3n/4$. For $n < 60$, note that it takes time no more than some constant amount d of time to find the median by sorting. Therefore we get the following recurrence for the running time of Select1:

$$T(n) \leq \begin{cases} T(3n/4) + T(n/5) + c'n & \text{if } n \geq 60 \\ d & \text{if } n < 60. \end{cases} \tag{4.35}$$

This answers Exercise 4.6-4.

As Exercise 4.6-5 requests, we can now verify by induction that $T(n) = O(n)$. What we want to prove is that there is a constant k such that $T(n) \leq kn$. What the recurrence tells us is that there are constants c and d such that $T(n) \leq T(3n/4) + T(n/5) + cn$ if $n \geq 60$, and otherwise $T(n) \leq d$. For the base case we have $T(n) \leq d \leq dn$ for $n < 60$, so we choose k to be at least d and then $T(n) \leq kn$ for $n < 60$. We now assume that $n \geq 60$ and $T(m) \leq km$ for values $m < n$, and get

$$\begin{aligned} T(n) &\leq T(3n/4) + T(n/5) + cn \\ &\leq 3kn/4 + kn/5 + cn \\ &= 19/20kn + cn \\ &= kn + (c - k/20)n\ . \end{aligned}$$

As long as $k \geq 20c$, this is at most kn; so we simply choose k this big and by the principle of mathematical induction, we have $T(n) < kn$ for all positive integers n.

Uneven Divisions

The kind of recurrence we found for the running time of Select1 is actually an instance of a more general class which we will now explore.

Exercise 4.6-6 We already know that when $g(n) = O(n)$, then every solution of $T(n) = T(n/2) + g(n)$ satisfies $T(n) = O(n)$. Use the master theorem to find a Big-O bound to the solution of $T(n) = T(cn) + g(n)$ for any constant $c < 1$, assuming that $g(n) = O(n)$.

Exercise 4.6-7 Use the master theorem to find Big-O bounds to all solutions of $T(n) = 2T(cn) + g(n)$ for any constant $c < 1/2$, assuming that $g(n) = O(n)$.

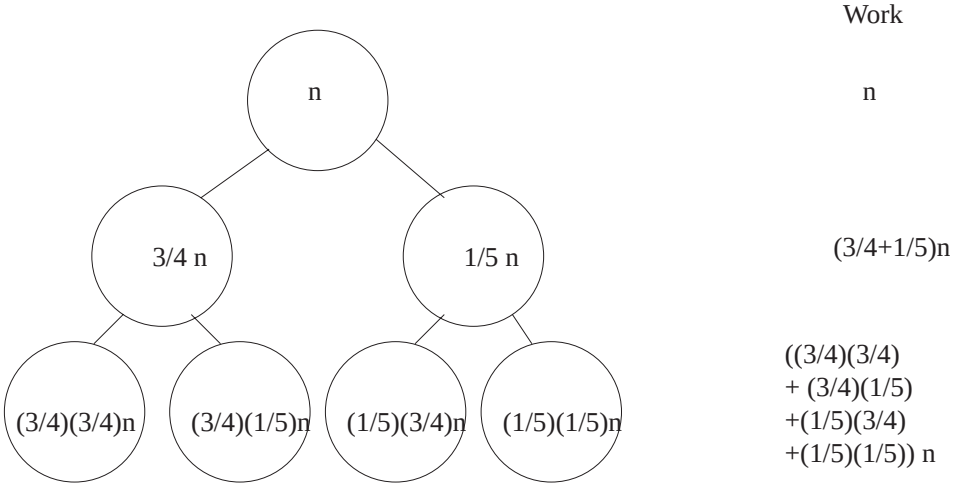
Exercise 4.6-8 Suppose $g(n) = O(n)$ and you have a recurrence of the form $T(n) = T(an) + T(bn) + g(n)$ for some constants a and b. What conditions on a and b guarantee that all solutions to this recurrence have $T(n) = O(n)$?

Using the master theorem for Exercise 4.6-6, we get $T(n) = O(n)$, since $\log_{1/c} 1 < 1$. We also get $T(n) = O(n)$ for Exercise 4.6-7, since $\log_{1/c} 2 < 1$ for $c < 1/2$. You might now guess that as

long as $a + b < 1$, any solution to the recurrence $T(n) \leq T(an) + T(bn) + cn$ has $T(n) = O(n)$. We will now see why this is the case.

First, let's return to the recurrence we had, $T(n) = T(3/4n)+T(n/5)+g(n)$, were $g(n) = O(n)$ and let's try to draw a recursion tree. This recurrence doesn't quite fit our model for recursion trees, as the two subproblems have unequal size (thus we can't even write down the problem size on the left), but we will try to draw a recursion tree anyway and see what happens. As we draw

Figure 4.13: Attempting a recursion tree for $T(n) = T(3/4n) + T(n/5) + g(n)$.



levels one and two, we see that at level one, we have $(3/4 + 1/5)n$ work. At level two we have $((3/4)^2 + 2(3/4)(1/5) + (1/5)^2)n$ work. Were we to work out the third level we would see that we have $((3/4)^3 + 3(3/4)^2(1/5) + 3(3/4)(1/5)^2 + (1/5)^3)n$. Thus we can see a pattern emerging. At level one we have $(3/4 + 1/5)n$ work. At level 2 we have, by the binomial theorem, $(3/4 + 1/5)^2 n$ work. At level 3 we have, by the binomial theorem, $(3/4 + 1/5)^3 n$ work. And, similarly, at level i of the tree, we have $\left(\frac{3}{4} + \frac{1}{5}\right)^i n = \left(\frac{19}{20}\right)^i n$ work. Thus summing over all the levels, the total amount of work is

$$\sum_{i=0}^{O(\log n)} \left(\frac{19}{20}\right)^i n \leq \left(\frac{1}{1 - 19/20}\right) n = 20n.$$

We have actually ignored one detail here. In contrast to a recursion tree in which all subproblems at a level have equal size, the "bottom" of the tree is more complicated. Different branches of the tree will reach problems of size 1 and terminate at different levels. For example, the branch that follows all 3/4's will bottom out after $\log_{4/3} n$ levels, while the one that follows all 1/5's will bottom out after $\log_5 n$ levels. However, the analysis above *overestimates the work*. That is, it assumes that nothing bottoms out until everything bottoms out, i.e. at $\log_{20/19} n$ levels. In fact, the upper bound we gave on the sum "assumes" that the recurrence never bottoms out.

We see here something general happening. It seems as if to understand a recurrence of the form $T(n) = T(an) + T(bn) + g(n)$, with $g(n) = O(n)$, we can study the simpler recurrence $T(n) = T((a + b)n) + g(n)$ instead. This simplifies things (in particular, it lets us use the Master Theorem) and allows us to analyze a larger class of recurrences. Turning to the median algorithm, it tells us that the important thing that happened there was that the sizes of the two recursive calls, namely $3/4n$ and $n/5$, summed to less than 1. As long as that is the case for an

algorithm with two recursive calls and an $O(n)$ additional work term, whose recurrence has the form $T(n) = T(an) + T(bn) + g(n)$, with $g(n) = O(n)$, the algorithm will work in $O(n)$ time.

Important Concepts, Formulas, and Theorems

1. *Median.* The *median* of a set (with an underlying order) of n elements is the element that would be in position $\lceil n/2 \rceil$ if the set were sorted into a list in order.

2. *Percentile.* The pth percentile of a set (with an underlying order) is the element that would be in position $\lceil \frac{p}{100} n \rceil$ if the set were sorted into a list in order.

3. *Selection.* Given an n-element set with some underlying order, the problem of *selection* of the ith smallest element is that of finding the element that would be in the ith position if the set were sorted into a list in order. Note that often i is expressed as a fraction of n.

4. *Partition Element.* A *partition element* in an algorithm is an element of a set (with an underlying order) which is used to divide the set into two parts, those that come before or are equal to the element (in the underlying order), and the remaining elements. Notice that the set as given to the algorithm is not necessarily (in fact not usually) given in the underlying order.

5. *Linear Time Algorithms.* If the running time of an algorithm satisfies a recurrence of the form $T(n) \leq T(an) + cn$ with $0 \leq a < 1$, or a recurrence of the form $T(n) \leq T(an) + T(bn) + cn$ with a and b nonnegative and $a + b < 1$, then $T(n) = O(n)$.

6. *Finding a Good Partition Element.* If a set (with an underlying order) has sixty or more elements, then the procedure of breaking the set into pieces of size 5 (plus one leftover piece if necessary), finding the median of each piece and then finding the median of the medians gives an element guaranteed to be in the middle half of the set.

7. *Selection algorithm.* The Selection algorithm that runs in linear time sorts a set of size less than sixty to find the element in the ith position; otherwise

   - it recursively uses the median of medians of five to find a partition element,
   - it uses that partition element to divide the set into two pieces and
   - then it looks for the appropriate element in the appropriate piece recursively.

Problems

1. In the MagicMiddle algorithm, suppose we broke our data up into $n/3$ sets of size 3. What would the running time of Select1 be?

2. In the MagicMiddle algorithm, suppose we broke our data up into $n/7$ sets of size 7. What would the running time of Select1 be?

3. Let
$$T(n) = \begin{cases} T(n/3) + T(n/2) + n & \text{if } n \geq 6 \\ 1 & \text{otherwise,} \end{cases}$$

and let

$$S(n) = \begin{cases} S(5n/6) + n & \text{if } n \geq 6 \\ 1 & \text{otherwise.} \end{cases}$$

Draw recursion trees for T and S. What are the big-O bounds we get on solutions to the recurrences? Use the recursion trees to argue that, for all n, $T(n) \leq S(n)$.

4. Find a (big-O) upper bound (the best you know how to get) on solutions to the recurrence $T(n) = T(n/3) + T(n/6) + T(n/4) + n$.

5. Find a (big-O) upper bound (the best you know how to get) on solutions the recurrence $T(n) = T(n/4) + T(n/2) + n^2$.

6. Note that we have chosen the median of an n-element set to be the element in position $\lceil n/2 \rceil$. We have also chosen to put the median of the medians into the set L of algorithm Select1. Show that this lets us prove that $T(n) \leq T(3n/4) + T(n/5) + cn$ for $n \geq 40$ rather than $n \geq 60$. (You will need to analyze the case where $\lceil n/5 \rceil$ is even and the case where it is odd separately.) Is 40 the least value possible?