CS 624
Lecture 2: Some Mathematical Tools

1 Growth of some standard functions

There are a lot of common mathematical functions that it is important to be familiar with. The very first ones you need to really have a feeling for are powers, exponential functions, and logarithms. In particular, you really need to understand "in your bones" how they grow for large values of their arguments, and how they compare to each other.

Figure 1 is something that you should study carefully. It will be of great importance in this course.

2 A quick review of exponents and logarithms

Here are some easy-to-prove properties of these functions that you should be familiar with:

1. a^{b^c} means $a^{(b^c)}$. That is, it associates *to the right*, as opposed to the other arithmetic operations. The reason for this is simple: $(a^b)^c$ is just a^{bc}, so there is no reason to have a^{b^c} associate to the left. Compiler writers have to remember this fact when writing parsers for arithmetic expressions.

2. $\log_a x$ is just "the number you have to raise a to in order to get x". In other words, by definition

$$a^{\log_a x} = x$$

3. Sometimes you will see authors say that

$$\ln x \text{ means } \log_e x$$
$$\log x \text{ means } \log_{10} x$$
$$\lg x \text{ means } \log_2 x$$

   There's nothing really wrong with this, and it is actually common in computer science texts. In mathematics, however, the convention is that

$$\log x \text{ means } \log_e x$$

   and $\ln x$ and $\lg x$ are never used. (Well, you sometimes see $\ln x$ used in first-year calculus books; but nowhere else.)

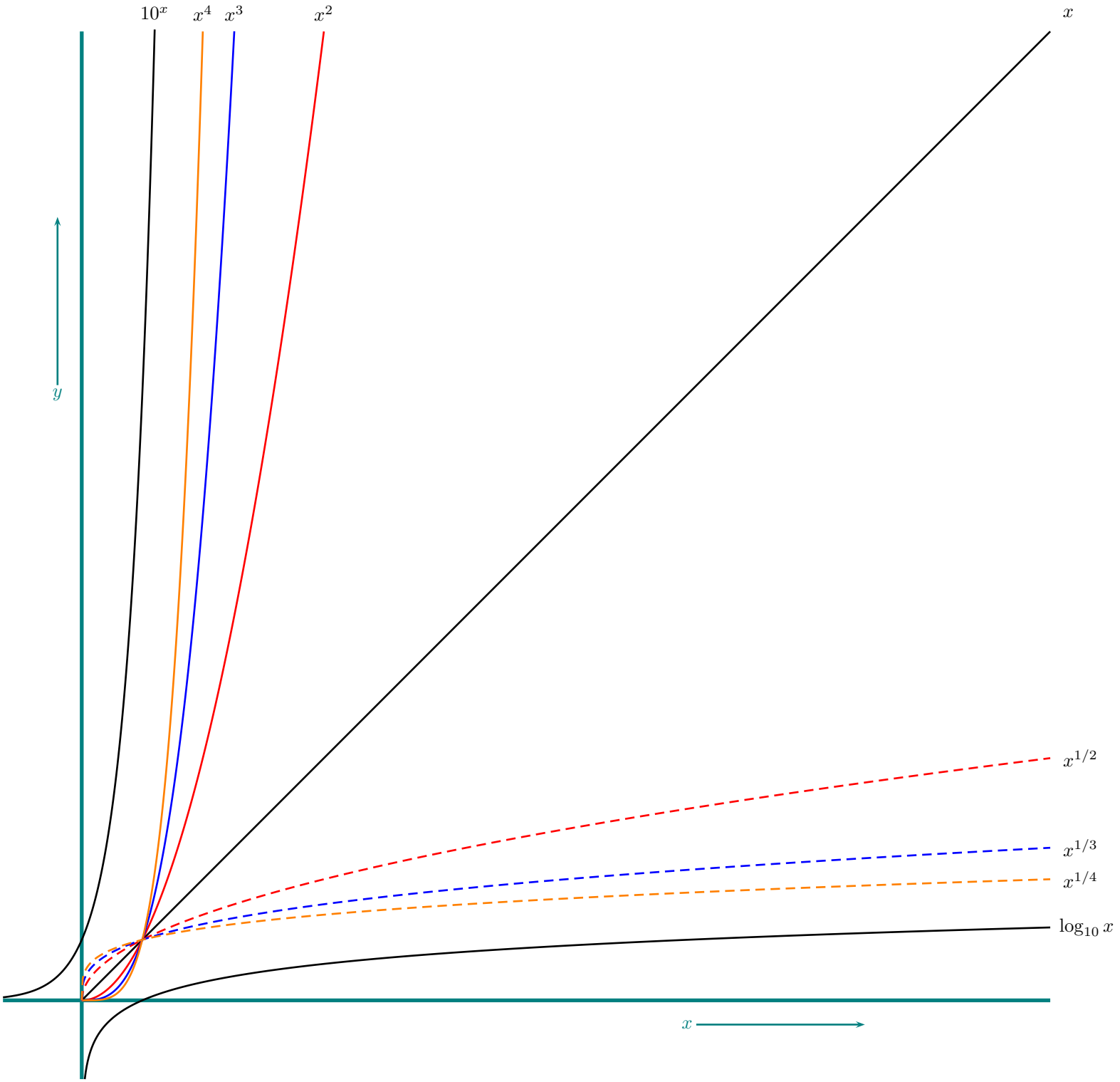

Figure 1: The standard functions—powers, the exponential function, and the logarithm.

I generally follow this convention—when I say or write $\log x$, I mean $\log_e x$. I won't use $\lg x$ at all.

It is important to note that for the purposes of this class, this really doesn't matter. The reason is this:

4. If a, b, and x are all positive, then

$$\log_b x = \log_a x \cdot \log_b a$$

PROOF. Say $\log_b a = P$ and $\log_a x = Q$. Then we have

$$b^P = a$$
$$a^Q = x$$

and hence

$$b^{PQ} = (b^P)^Q = a^Q = x$$

That is,

$$b^{\log_b a \cdot \log_a x} = x = b^{\log_b x}$$

and so

$$\log_b a \cdot \log_a x = \log_b x$$

□

Thus, $\log_b x$ and $\log_a x$ differ only by a multiplicative constant (namely, $\log_b a$). As we'll see, this is something we by and large don't care about in this course.

Computations like the one above are quite standard, and you should be able to do them pretty easily. For instance, you should be able to prove that

$$a^{b(\log_a x)} = x^b$$

3 Asymptotic notation

3.1 Upper bounds; O-notation

Suppose that f and g are functions defined on the positive reals, or on the positive integers. We will also assume for simplicity in this section that f and g take only only positive values.

We say that $f \leq g$ iff $f(x) \leq g(x)$ for *all* x.

There is a weaker notation that is very important in this field. It's called "big-O" notation. Roughly, $f = O(g)$ means that f is bounded by some constant multiple of g; that is, there is some constant $c > 0$ such that $f \leq cg$. And in fact, this doesn't have to hold for all x—it just has to hold for all *sufficiently large* x.

This may seem picky, but there's a good reason for this. Imagine we have some algorithm—say a sorting algorithm. We want to measure how long this algorithm takes on an input of n elements. Maybe we have something like the picture in Figure 2.

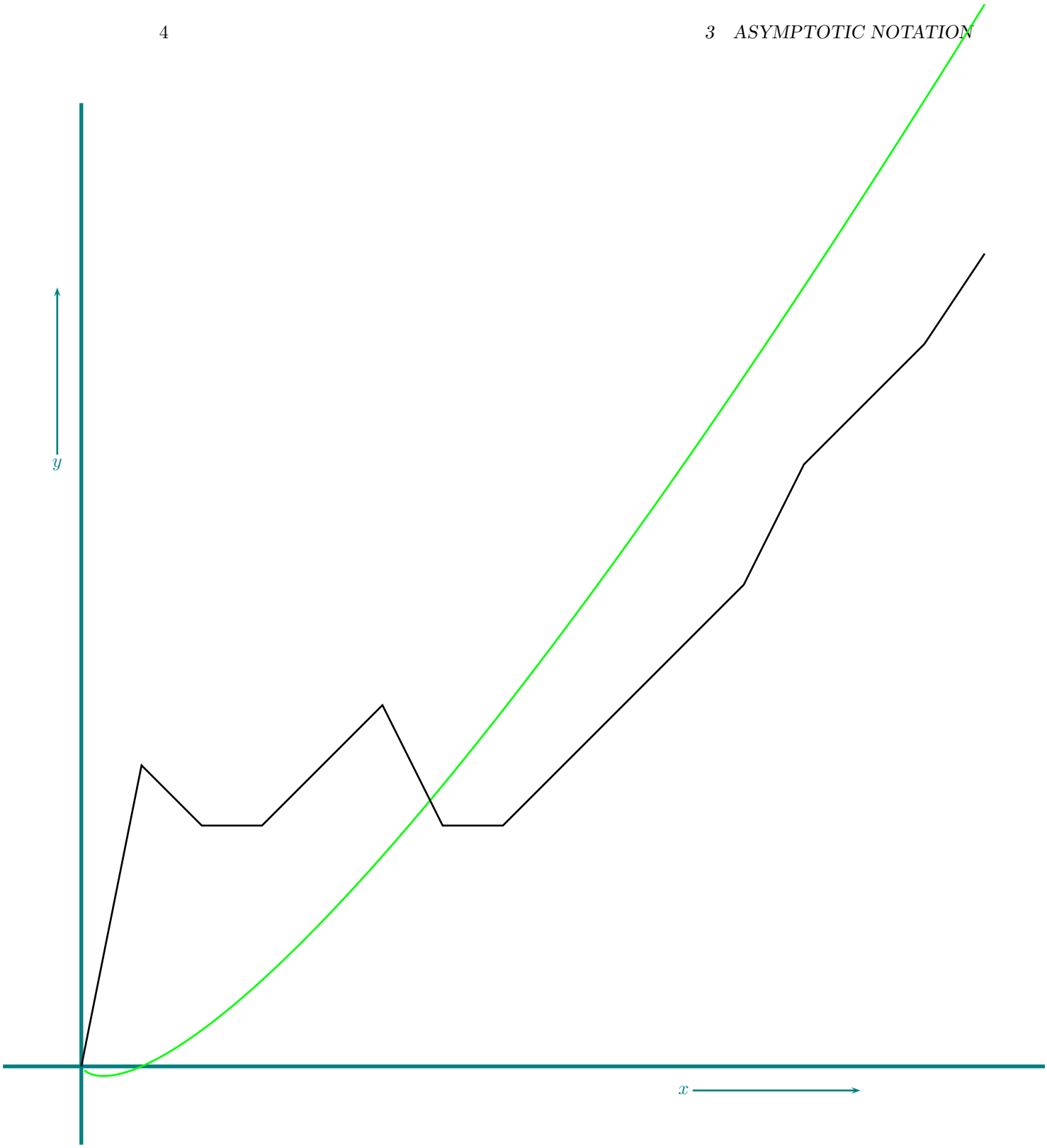

Figure 2: An example of the running time of an algorithm, and an asymptotic upper bound for that running time. The black line represents the actual running time of the algorithm. The green function is an asyptotic upper bound.

In Figure 2, the black line shows the cost of an algorithm for various sizes of inputs. The green line (which is actually the graph of the function $f(x) = x \log x$) seems to be greater than the function represented by the black line, at least from some point on.

Now actually, I made up this example. But the behavior is typical. Often for small inputs, the cost of the algorithm contains a certain amount of "noise". For larger inputs, this noise is insignificant. The point is though, that we almost never care about the cost for small inputs. For a data set of a small size, pretty much any algorithm will work. It's when we get to large size inputs that the cost really matters. So it's reasonable for us to say (assuming that the functions in Figure 2 continue to act like we think they should) that the green function is an upper asymptotic bound for the actual cost of the algorithm.

In fact, we don't really even care that the green function is (eventually) greater than the actual cost of the algorithm—all that really matters is that some constant multiple of it is eventually greater.

So here is the precise definition:

Definition *$f = O(g)$ means that there exist numbers $c > 0$ and $x_0 > 0$ such that*

$$f(x) \leq cg(x) \text{ for all } x \geq x_0 \tag{1}$$

There are three things I have to emphasize here:

1. As we have already pointed out, the clause about "sufficiently large x" is not just a small point. It is very important. For instance, for $x \geq 1$, $x^2 \leq x^3$. But for $0 < x < 1$, $x^2 > x^3$, and in fact there is no constant $c > 0$ such that $x^2 < cx^3$ for all $x > 0$. But there *is* such a constant[1] provided we restrict x to the domain $x \geq 1$. It's important to keep this in mind.

   The "sufficiently large" criterion is what the constant x_0 is used to specify.

2. As we also already pointed out, we don't much care about the value of c, as long as it is constant. If the running time of an algorithm, when run on an input of size n, is eventually bounded by cn^2, for instance, then it doesn't much matter whether c is 1 or 100. If it is 100 today, then pretty soon there will be computers that are 100 times faster in any case.

3. It's also terribly important to understand that this is a real mathematical definition. That is, it is a contract. When I (or anyone anywhere in the world) uses this notation, this definition says exactly what we mean by it.

   So in particular, if I give you two functions f and g, and I ask you to prove that $f = O(g)$, what do you have to do? You have to come up with two numbers $c > 0$ and $x_0 > 0$ such that equation (1) is satisfied.

   For instance, to prove that

   $$2n^2 = O(n^3)$$

   You have to find $c > 0$ and $n_0 > 0$ such that

   $$2n^2 \leq cn^3 \text{ for all } n \geq n_0$$

[1] In fact, the constant $c = 1$ works.

In this case, you should be able to see that $c = 1$ and $n_0 = 2$ works. So you would have to announce that $c = 1$ and $n_0 = 2$. And then you would have to prove that this really does work, probably like this: provided that $n \geq 2$,

$$2n^2 \leq n \cdot n^2 = n^3 = 1 \cdot n^3$$

This is important! I expect you to be able to reason in this fashion, and to write out this sort of reasoning clearly, both on your homework and on tests.

Thus, I would expect you to hand in something like this:

> We need to show that
>
> $$2n^2 = O(n^3)$$
>
> To show this, we need to show that there is a number $c > 0$ and another number $n_0 > 0$ such that
>
> $$2n^2 \leq cn^3 \text{ for all } n \geq n_0$$
>
> In fact, the numbers $c = 1$ and $n_0 = 2$ work. This is true because when $n \geq 2$,
>
> $$2n^2 \leq n \cdot n^2 = n^3 = 1 \cdot n^3$$

Note by the way that we were really abusing notation slightly. What we really should have said is that if $f(n) = 2n^2$ and $g(n) = n^3$, then $f = O(g)$. But instead of giving names to these functions, we just used the expressions that define the functions and wrote $2n^2 = O(n^3)$ directly. This is very common, and should not lead to any confusion.

Another unusual thing about this notation to be aware of is that when we write $f = O(g)$, the equal sign is really not the ordinary equal sign. This is the only kind of place where the equal sign means anything different than strict mathematical equality. Be careful about this!

Also note that although it is true that $n = O(n^2)$, you *cannot* write

$$O(n^2) = n \qquad \text{!!! WRONG !!!}$$

The "O" expression always has to be on the right.

Here's another example of this kind of usage: Suppose I have a function f which I don't know an exact formula for (or maybe I do but it's pretty complicated). Maybe I can still write this:

$$f(n) = n^3 + O(n^2)$$

What this means is that there is a function h, defined by

$$f(n) = n^3 + h(n)$$

such that

$$h(n) = O(n^2)$$

Here are some more examples. I won't prove any of these statements (remember that to do so, I would have to give you two numbers—$c > 0$ and $n_0 > 0$ for each statement—and prove that they really worked in the definition we gave above), but you should be able to prove them.

$$\begin{aligned} n^2 &= O(n^2 - 3) \\ n^2 &= O(n^2 + 3) \\ 100n^2 &= O(n^2) \\ n^2 &= O(n^2 + 7n + 2) \\ n^2 + 7n + 2 &= O(n^2) \end{aligned}$$

Further,

$$\text{If } 0 < p < q, \text{ then } x^p = O(x^q)$$

$$\text{For all } a > 0 \text{ and } b > 0, \log_a x = O(\log_b x)$$

Here's another important and simple property. I'm going to prove it because I want you to see just how careful you have to be.

3.1 Lemma *If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.*

PROOF. Since $f = O(h)$, we know that there are constants $c > 0$ and $x_0 > 0$ such that for all $x > x_0$, $f(x) \leq ch(x)$.

We also know that $g = O(h)$. So we should be able to write down something similar. However, it is important to realize that the same values of c and x_0 might not work for g and h. So we have to say that there are constants $d > 0$ and $x_1 > 0$ such that for all $x > x_1$, $g(x) \leq dh(x)$

We need to put c and d and x_0 and x_1 together somehow to come up with two numbers that work for $f + g$ and h. It's not hard to see that we can use $c + d$ and $\max(x_0, x_1)$. That is, I assert that for $x \geq \max(x_0, x_1)$, $f(x) + g(x) \leq (c + d)h(x)$. This is because if $x \geq \max(x_0, x_1)$, then certainly $x \geq x_0$. Hence

$$f(x) \leq ch(x)$$

And if $x \geq \max(x_0, x_1)$ then certainly $x \geq x_1$. Hence

$$g(x) \leq dh(x)$$

and adding these two results together, we see that if $x \geq \max(x_0, x_1)$, we have

$$f(x) + g(x) \leq ch(x) + dh(x) = (c + d)h(x)$$

and we are done. □

3.2 Lower bounds; Ω-notation

We can think of O-notation as providing a kind of upper bound. There is an analogous notation for lower bounds:

Definition *$f = \Omega(g)$ means that there exist $c > 0$ and $x_0 > 0$ such that for all $x \geq x_0$,*

$$f(x) \geq cg(x)$$

You should be able to show pretty easily that

$$f = O(g) \iff g = \Omega(f)$$
$$\sqrt{n} = \Omega(\log n)$$

3.3 Tight bounds; Θ-notation

Definition *$f = \Theta(g)$ means that both $f = O(g)$ and $f = \Omega(g)$*

You should be able to show that $f = \Theta(g)$ iff there are constants $a > 0$, $b > 0$, and $x_0 > 0$ such that for all $x > x_0$,

$$ag(x) \leq f(x) \leq bg(x)$$

And it should be pretty obvious to you that

$$\frac{1}{2}n^2 - 2n = \Theta(n^2)$$

4 Solving Recurrences

Recurrences typically arise in analyzing divide-and-conquer problems. In the last lecture when we talked about MERGE-SORT, we found a recurrence for the running time $T(n)$:

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

and we were able (by using a "recurrence tree" to see that $T(n) = \Theta(n \log n)$.

Recurrences are a bit like differential equations They give you information about a function, but don't give a formula for the function. For instance,

$$f'(x) = f(x)$$

is a differential equation. It has the family of solutions

$$f(x) = ae^x$$

(for any constant a). This solution is perhaps not so obvious just by looking at the differential equation, but it is one of the things you learn in first-year calculus.

Sometimes we can't get an exact solution to a differential equation, but we can use the differential equation to derive properties of the solution, and this may be all we really need. In fact, even if we

have managed to solve the differential equation exactly, it's usually the properties of the solution we are actually interested in. We're happy when we can solve the equation above to get $f(x) = ae^x$ because we know a lot about the exponential function.

Recurrences are similar. We'd like to get an exact solution where possible. But in any case, we'd like to find out "properties" of the solution. For the purposes of this course, "properties" means "order of growth".

There are three techniques shown in the book for dealing with recurrences:

4.1 The "substitution method"—proof by induction

To use the substitution method, here's what you do:

1. Guess a formula or bound for the solution.
2. Prove the formula or bound by induction. In doing so, you generally have to solve for any necessary constants.

Here's an example:

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

($T(1)$, whatever it is, is just a constant.)

> First, note an obvious question: shouldn't we really write something like this:
>
> $$T(n) = 4T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$$
>
> The answer is yes, we really should, unless we know for sure that n is a power of 2, which of course it really doesn't have to be. But we're not going to worry about this here. It turns out that this is an irritating but minor point. If you can understand these computations without the floors and ceilings, you've got the main point.

So here are the two steps we take:

1. Guess $T(n) = O(n^3)$.
2. Prove this by induction:

   First, we have $T(1) \le c(1^3)$ provided that c is big enough. So just start out by assuming that c is big enough to make that true. (We may make it bigger later.) Further, we will start by guessing that we can take $n_0 = 1$—that is, we want to prove that $T(k) \le ck^3$ for all $k \ge 1$.

   Now for the inductive step, assume that $T(k) \le ck^3$ for $1 \le k < n$. We need to show that this remains true for $k = n$.

   What do we have at our disposal? Two things:

The inductive hypothesis: $T(k) \le ck^3$ for all $1 \le k < n$. This is really what we are trying to prove is true for *all* n.

Actually, it's important to realize that this inductive hypothesis is not a single statement. It is actually a sequence of statements, one statement for each value of n. So for instance,

- Statement 5 is this: "$T(k) \le ck^3$ for all $1 \le k < 5$."
- Statement 6 is this: "$T(k) \le ck^3$ for all $1 \le k < 6$."

and so on. You should be able to see that Statement 1 is empty.

Statement 2, however, is true for some c. (Just pick some c that is big enough[2].) That's important. We have to start somewhere. And the important thing is that the value of c does not change—it's the same for all the statements.

The recurrence formula: $T(n) = 4T\left(\frac{n}{2}\right) + n$. This is *not* something we are trying to prove. We know from the beginning that it is true for all n.

So let us put these two together: We have

$$\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + n && \text{by the recurrence formula, which is always true} \\
&\le 4c\left(\frac{n}{2}\right)^3 + n && \text{by the inductive hypothesis, since } n/2 < n \\
&= \frac{c}{2}n^3 + n \\
&= cn^3 - \left(\frac{c}{2}n^3 - n\right) && (= \text{desired - residual}) \\
&\le cn^3
\end{aligned}$$

the last inequality being true whenever

$$\frac{c}{2}n^3 - n \ge 0$$

and this is certainly true if for instance $c \ge 2$ and $n \ge 1$. (Can you prove this?)

This shows that if Statement n in our sequence is true, then Statement $n+1$ is true as well. It then follows that *all* the statements must be true. Since we know that Statement 2 is true, it follows that Statement 3 must be true. It then follows that Statement 4 is true, and the computation we have just performed enables us to keep on going. And so we see that for all values $n \ge 2$, Statement n is true.

And that completes the proof.

I have deliberately written the inductive hypothesis as a sequence of statements. This is a very good way to think about it. You should get used to it. Now mostly people don't express it this way—we usually just refer to "the inductive hypothesis" as if it were one (parametrized) statement. But I suggest strongly that you practice thinking of it as a sequence of statements until this becomes completely natural.

Now this bound that we just proved is not "best possible", or as computer scientists often say, this bound is not "tight". In fact $T(n) = O(n^2)$, and this is a much better bound. Let's see how to prove this, again using the substitution method:

[2] $c = T(1)$ would work, right?

We already have our guess. We again go through the inductive step:

Again, we take $n_0 = 1$. We can assume that c is big enough so that $T(1) \le c(1^2)$. The inductive hypothesis is that $T(k) \le ck^2$ for $1 \le k < n$.

That is, the inductive hypothesis is a sequence of statements:

- Statement 5 is this: "$T(k) \le ck^2$ for $1 \le k < 5$."
- Statement 6 is this: "$T(k) \le ck^2$ for $1 \le k < 6$."

and so on. As before, Statement 1 is vacuous, but there is a constant c such that Statement 2 is true. And we need to show that *all* the statements in this sequence are true with the same value of c.

So suppose we know that Statement n is true for some particular value of n. We need to show that it follows that Statement $n+1$ is also true. We start out as before—we set $k = n$ and compute:

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n && \text{by the recurrence formula, which is always true} \\ &\le 4c\left(\frac{n}{2}\right)^2 + n && \text{by the inductive hypothesis, since } n/2 < n \\ &= cn^2 + n \\ &= O(n^2) && \text{!!! WRONG !!!} \end{aligned}$$

The reason the last step is wrong is that we need exactly the bound cn^2 in the last line, and we don't have it. In fact,

$$cn^2 + n \le cn^2 \text{ for } no \text{ choice of } c > 0$$

So this proof doesn't quite work. We can fix it, though, because it really is pretty close.

Here's how to do this: strengthen the inductive hypothesis by subtracting a lower-order term: Let's take this as our inductive hypothesis:

$$T(k) \le c_1k^2 - c_2k \text{ for } 1 \le k < n$$

I won't write this out as a sequence of statements, but you should be able to do that at this point.

And we'll take care of the base case $n = 2$ later. That is, we will show below that there are numbers c_1 and c_2 making the inductive hypothesis true for $n = 2$.

Here's the inductive step:

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n && \text{by the recurrence formula, which is always true} \\ &\le 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\frac{n}{2}\right) + n && \text{by the inductive hypothesis, since } n/2 < n \\ &= c_1n^2 - 2c_2n + n \\ &= c_1n^2 - c_2n - (c_2 - 1)n \\ &\le c_1n^2 - c_2n \end{aligned}$$

where the last inequality holds so long as we assume that $c_2 \geq 1$.

So let's take $c_2 = 1$. Then to show that the inductive hypothesis is true when $n = 2$, we just need to show that there is some c_1 such that $T(1) \leq c_1 1^2 - c_2 1$. This will be true provided we let c_1 be some fixed number that is "large enough"[3]. That completes the proof. So we've proved that $T(n) = O(n^2)$.

The really important thing to remember about all this is that this technique is a proof by induction. You need to state the inductive hypothesis explicitly and show how the inductive step works.

4.2 The recursion tree method

We saw an example of this last time. Here's a more complicated example. Suppose we have the recursion

$$T(n) = T\Big(\frac{n}{4}\Big) + T\Big(\frac{n}{2}\Big) + n^2$$

Figure 3 shows the recursion tree.

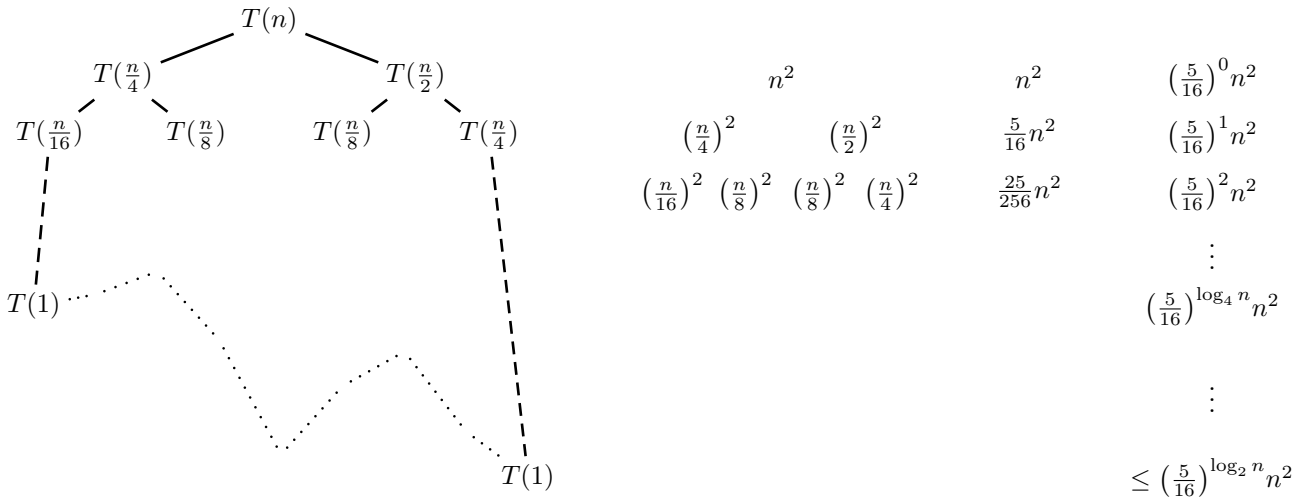

Figure 3: Recursion tree for $T(n) = T\Big(\frac{n}{4}\Big) + T\Big(\frac{n}{2}\Big) + n^2$

The tree is completely filled in down to level $\log_4 n$, containing the left-hand $T(1)$. And it is then partially filled in down to level $\log_2 n$, containing the right-hand $T(1)$. So we can see that the total cost $T(n)$ is bounded below by what you get if you cut off the tree at level $\log_4 n$, and it is bounded

[3]Whenever you see a statement like this, you should stop and figure out what would work. In this case, what value of c_1 would work?

above by what you get if you keep on adding up the right hand column down to level $\log_2 n$. That is,

$$\begin{aligned} T(n) &\geq n^2 \sum_{k=0}^{\log_4 n} \left(\frac{5}{16}\right)^k \\ &= n^2 \frac{\left(\frac{5}{16}\right)^{(\log_4 n)+1} - 1}{\frac{5}{16} - 1} \end{aligned}$$

and

$$\begin{aligned} T(n) &\leq n^2 \sum_{k=0}^{\log_2 n} \left(\frac{5}{16}\right)^k \\ &= n^2 \frac{\left(\frac{5}{16}\right)^{(\log_2 n)+1} - 1}{\frac{5}{16} - 1} \end{aligned}$$

In fact, we really didn't need to get the exact expressions in each of these cases. This is because the first sum is obviously bounded below by 1, and the second is the beginning of a convergent geometric series, and so it is bounded above by some constant (in fact, the constant is

$$\frac{1}{1 - \frac{5}{16}}$$

but we really don't care about the exact number). Therefore we have

$$n^2 \leq T(n) \leq cn^2$$

for all $n > 0$ (where c is the constant we were just talking about). And therefore we have

$$T(n) = \Theta(n^2)$$

4.1 Exercise *Suppose $n = 16$. Draw Figure 3 completely, including every node in the tree. (Since n is finite, there are only a finite number of nodes in the tree. You need to show every one of them.)*

4.3 The "master method"

This name is given[4] to a method that only applies to a certain class of recurrences. For instance, it does not apply to the recurrence of the last section. It applies to recurrences of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$, $b > 1$, and f is ultimately positive. (That is, there is some $x_0 > 0$ such that for all $x > x_0$, $f(x) > 0$. The text uses the phrase "asymptotically positive" for this.)

[4]The name is due to the authors of our text, although they didn't discover the method itself.

Rather than just stating the master theorem, it helps to first get an idea of what it is getting at. We can do this by first considering the recurrence

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

This kind of recurrence would typically occur in a divide-and-conquer problem, where each problem of size n was divided into a subproblems, each of size n/b. It's understood here that a and b are fixed constants.

(There is also a difficulty that occurs: if n is not divisible by b, then we probably need to consider something a little more complicated, like $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. It turns out that dealing with these cases is possible and doesn't change the conclusion of the theorem. So we're going to ignore this matter here. The text does go into this.)

Suppose we have reason to believe (as in fact we do) that the solution of this problem is of the form

$$T(n) = n^p$$

for some power p. Let's see if this is true, and if we can figure out what p could be.

Substituting $T(n) = n^p$ into the recurrence, we get

$$n^p = a(n/b)^p = a\frac{n^p}{b^p}$$

so $b^p = a$, and (taking the $\log_b$ of both sides) we wind up with

$$p = \log_b a$$

(And if this isn't enough you can convince yourself by substituting $T(n) = n^{\log_b a}$ in the recurrence that it is indeed a solution.)

The master theorem is based on this fact. It actually considers a slightly more complicated recursion:

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

where f is some function that occurs in the following way: in our divide-and-conquer problem, we divide each problem of size n into a problems of size n/b. The total cost of the problem of size n is the sum of the costs of the subproblems (this is where the term $aT(n/b)$ comes from) *plus* the cost of merging those subproblem solutions together (this is what the term $f(n)$ represents).

There are three cases that the master theorem deals with (remember that p is $\log_b a$):

- The function $f(n)$ is small compared with n^p.
- The function $f(n)$ is comparable to n^p.
- The function $f(n)$ is large compared with n^p.

Of course this is pretty vague. There are lots of ways to say that one function is "small" compared with another. For the purpose of this theorem (and *not* necessarily of any other theorem!), when

we say that "the function $f(n)$ is small compared with n^p", what we mean is that there is some number $\epsilon > 0$ (which could be very small, but at any rate is really > 0) such that

$$f(n) = O(n^{p-\epsilon})$$

Note that this means that $f(n) = O(n^p/n^\epsilon)$. That is, $f(n)$ grows more slowly than n^p by a (possibly small but positive) power of n.

Similarly, when we say that "f(n) is large compared with n^p", what we mean (again, only for the purpose of this theorem) is that there is some $\epsilon > 0$ such that $f(n) = \Omega(n^{p+\epsilon}) = \Omega(n^p n^\epsilon)$. That is, $f(n)$ grows faster than n^p by a (possibly small but positive) power of n.

In fact, in this last case, we need one additional condition on f—we need to ensure that f doesn't "wiggle too fast". What that means here is that there is some constant c with $0 < c < 1$ such that $af(n/b) \le cf(n)$ for all sufficiently large n.

So here is the actual statement of the master theorem:

4.2 Theorem *If $a \ge 1$ and $b \ge 1$ are constants, $f(n)$ is a function, and $T(n)$ is another function satisfying the recurrence*

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$, then $T(n)$ can be estimated asymptotically as follows:

1. *If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$*

   In other words, when $f(n)$ is small compared with n^p, then f essentially has no effect on the growth of T, and $T(n) = \Theta(n^p)$, just as it would if $f \equiv 0$.

2. *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.*

   This case is significant in that it applies to algorithms which are $O(n \lg n)$.

3. *If $f(n) = \Omega(n^{\log_b a + \epsilon})$* and *if $af(n/b) \le cf(n)$ for some constant c with $0 < c < 1$ and all sufficiently large n, then $T(n) = \Theta\big(f(n)\big)$.*

   In this case, the function f is what really contributes to the growth of T, and the recursion is immaterial.

Remark *In fact, Case 2 really can be split into two parts. If you look at the actual proof of Case 2 in the text, you can see that what is really being proved is this:*

2a. If $f(n) = O(n^{\log_b a})$, then $T(n) = O(n^{\log_b a} \lg n)$.

2b. If $f(n) = \Omega(n^{\log_b a})$, then $T(n) = \Omega(n^{\log_b a} \lg n)$.

Putting these two results together yields the statement of Case 2, but Case 2 does not immediately imply either one of them.

Equivalently, these two sub-cases could be written like this:

2a. If $T(n) \leq aT(n/b) + f(n)$ *where* $f(n) = O(n^{\log_b a})$, *then* $T(n) = O(n^{\log_b a} \lg n)$.

2b. If $T(n) \geq aT(n/b) + f(n)$ *where* $f(n) = \Omega(n^{\log_b a})$, *then* $T(n) = \Omega(n^{\log_b a} \lg n)$.

Here are some examples:

Example 1

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

Here we have

$$a = 4$$
$$b = 2$$
$$f(n) = n$$

so $n^{\log_b a} = n^2$ and since $f(n) = n$, we are in Case 1. (That is $f(n) = O(n^{2-\epsilon})$ for $0 < \epsilon < 1$.)

Therefore the conclusion of the master theorem is that $T(n) = \Theta(n^2)$. ◇

Example 2

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

Here we have

$$a = 4$$
$$b = 2$$
$$f(n) = n^2$$

so again $n^{\log_b a} = n^2$ and since $f(n) = n^2$, we are clearly in Case 2. Therefore the conclusion of the master theorem is that $T(n) = \Theta(n^2 \log n)$. ◇

Example 3

$$T(n) = 4T(n/2) + n^3$$

Now we have

$$a = 4$$
$$b = 2$$
$$f(n) = n^3$$

so again $n^{\log_b a} = n^2$. Since $f(n) = n^3$, we have $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $0 < \epsilon < 1$. Thus we will be in Case 3 provided we can show that the additional condition needed for Case 3 holds. That is, we need to show that there is some constant c with $0 < c < 1$ and some n_0 such that for all $n > n_0$,

$$af(n/b) \leq cf(n)$$

That is,

$$4f(n/2) \leq cf(n)$$

That is,

$$4(n/2)^3 \leq cn^3$$

or equivalently,

$$\frac{n^3}{2} \leq cn^3$$

This certainly holds for any $c > 1/2$. So we could take $c = 3/4$, for example.

Therefore we really are in Case 3, and the conclusion of the master theorem is that $T(n) = \Theta(n^3)$.◇

Example 4

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

Here

$$a = 4$$
$$b = 2$$
$$f(n) = n^2/\log n$$

As before $n^{\log_b a} = n^2$. Now $f(n) = n^2/\log n$.

In this case the master theorem does not apply. (Can you see why?) ◇

Example 5

$$T(n) = 2T(n/2) + cn$$

This is the recursion from the MERGE-SORT application in the first lecture. Here

$$a = 2$$
$$b = 2$$
$$f(n) = cn$$

Now we have $n^{\log_b a} = n$. So certainly $f(n) = \Theta(n^{\log_b a})$, and so we are in Case 2. The conclusion of the master theorem is then that $T(n) = \Theta(n \log n)$. ◇

5 Generating functions

When you studied calculus, you undoubtedly learned about power series, and how many common and important functions can be represented as power series. For instance,

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots$$

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

and there are some simple power series that we all learned in high school, like the *geometric series*:

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + x^4 + \cdots$$

Of course this last series only makes sense (that is, it only converges) for numbers x for which $|x| < 1$.

We usually first learn about power series as a way of starting with a function (like $f(x) = e^x$) and coming up with a set of coefficients (like $a_n = \frac{1}{n!}$) which yield a power series for the function $f(x)$:

$$f(x) = \sum_{n=0}^{\infty} a_n x^n$$

And this is indeed very important.

But there is another way to look at this. Sometimes we are faced with a sequence $\{a_n : n = 1, 2, 3, \ldots\}$ that we need some information about. We can (formally, at least) form the power series

$$f(x) = \sum_{n=0}^{\infty} a_n x^n$$

and if we are fortunate, we may be able to prove that due to some special properties of the coefficients a_n, f itself has some specal properties that enable us to characterize it in some way that then enables us to go back and infer some stronger results about the coefficients a_n.

When we do this, we are using the technique of *generating functions*. We say that the function f is the generating function of the sequence $\{a_n\}$.

Example 6 For a very simple example, we know that the binomial theorem says that

$$(1 + x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

This just tells us that $(1 + x)^n$ is the generating function for the finite sequence $\left\{\binom{n}{k} : 0 \leq k \leq n\right\}$.

Now suppose we set $x = 1$ in that identity. We get

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

This is not a hard result to prove by induction. But certainly this proof using a generating function is much easier and more elegant. (And notice, by the way, that the power series in this case has only a finite number of terms. That's OK—it's still a power series.) ◇

Example 7 Here's another simple example: we know that

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$$

Now we can differentiate both sides of this identity. (And the power series can be differentiated term-by-term. This is actually a theorem about power series.) When we do this, we get

$$\frac{1}{(1-x)^2} = \sum_{n=0}^{\infty} nx^{n-1} = \sum_{n=1}^{\infty} nx^{n-1}$$

and if we substitute $x = 1/2$, we get

$$4 = \sum_{n=1}^{\infty} \frac{n}{2^{n-1}}$$

or equivalently (multiplying both sides by 1/2 to make it look a bit simpler),

$$\sum_{n=1}^{\infty} \frac{n}{2^n} = 2$$

Again, this could proved in other ways, but this is certainly a very clean way to do this. ◇

Finally, here is a really serious application that you should find amazing:

5.1 A formula for the Fibonacci numbers via generating functions

We let $\{f_0, f_1, f_2, \dots\}$ denote the Fibonacci numbers:

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_2 &= 1 \\ f_3 &= 2 \\ f_4 &= 3 \\ f_5 &= 5 \\ f_6 &= 8 \end{aligned}$$

and

$$f_n = f_{n-1} + f_{n-2} \qquad \text{for } n \geq 2 \tag{2}$$

(Why did we start off with 0, rather than 1? There's no deep reason for this. But it does make the algebra turn out a little easier.)

(2) is a recursion relation for the Fibonacci numbers. It would be nice if we had an actual formula giving f_n as a function only of n.

It turns out that there is such a formula, but it's not at all an obvious one. However, we can use the basic recursion (2), together with a generating function, to derive it. Here's how:

Let us define the generating function F of the Fibonacci numbers:

$$F(x) = f_0 + f_1 x + f_2 x^2 + \cdots = \sum_{n=0}^{\infty} f_n x^n$$

Now right now we don't know anything about the function F. But we do know something about the coefficents $\{f_n\}$—we know they satisfy the recursion (2). We can use that fact to derive an important property of the function F. We do it like this: first we write the power series for $F(x)$, $xF(x)$, and $x^2F(x)$.

$$\begin{array}{rcccccccccccc} F(x) & = f_0 & + & f_1 x & + & f_2 x^2 & + & f_3 x^3 & + & f_4 x^4 & + & f_5 x^5 & + \ldots \\ xF(x) & = & & f_0 x & + & f_1 x^2 & + & f_2 x^3 & + & f_3 x^4 & + & f_4 x^5 & + \ldots \\ x^2F(x) & = & & & & f_0 x^2 & + & f_1 x^3 & + & f_2 x^4 & + & f_3 x^5 & + \ldots \end{array}$$

You might think that this was not an obvious thing to do, but look what we can now do:

If we add up the second and third rows (term-by-term) and subtract from the first row, most of the terms cancel out. And in fact, we see that

$$F(x)(1 - x - x^2) = x$$

so

$$F(x) = \frac{x}{1 - x - x^2} \tag{3}$$

So we actually know precisely what the function F is. Now we will use this information to get a formula for the coefficients $\{f_n\}$; and that's what we're looking for.

The basic fact we will use in doing this is the geometric series:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots = \sum_{n=0}^{\infty} x^n$$

(which we know converges for $|x| < 1$).

Now the right-hand side of our basic identity (3) is not of the form $\frac{1}{1-<\text{anything}>}$. So we would like to transform it somehow so we get one or more expressions like that out of it.

The denominator is a quadratic polynomial. We can certainly factor it into something of the form $-(x-x_0)(x-x_1)$. However, because of where we are heading, it is more convenient to factor it into the form $(1-\alpha x)(1-\beta x)$. So let us set

$$(1-\alpha x)(1-\beta x) = 1 - x - x^2$$

and see what α and β must be[5]. Just by multiplying out the left-hand side, we see that

$$\alpha + \beta = 1$$
$$\alpha\beta = -1$$

So from the first equation, $\beta = 1-\alpha$, and then from the second equation,

$$\alpha(1-\alpha) = -1$$

or equivalently,

$$\alpha^2 - \alpha - 1 = 0$$

Of course we can solve this equation; we get

$$\alpha = \frac{1 \pm \sqrt{5}}{2}$$

and in fact the two solutions of this add up to 1, so that one is α and the other is β. (It doesn't matter which one is which.) So let us say that

$$\alpha = \frac{1+\sqrt{5}}{2}$$
$$\beta = \frac{1-\sqrt{5}}{2}$$

So now we know that

$$F(x) = \frac{x}{1-x-x^2} = \frac{x}{(1-\alpha x)(1-\beta x)}$$

And now we can further use a "partial fractions decomposition"—we can find numbers A and B such that

$$\frac{x}{(1-\alpha x)(1-\beta x)} = \frac{A}{1-\alpha x} + \frac{B}{1-\beta x}$$

In fact, just adding the two terms on the right-hand side shows that this will be true provided

$$A(1-\beta x) + B(1-\alpha x) = x$$

[5]Some students get intimidated by this kind of algebraic manipulation. But that's all it is—it's just high school algebra. You have to get used to performing computations like this. You should study this until you understand exactly how it works, and so that you could explain it to someone else—for instance, to your younger brother or sister.

which gives us two equations:

$$A + B = 0$$
$$A\beta + B\alpha = -1$$

So from the first equation we have $B = -A$, and we already know that $\beta = 1 - \alpha$. Making these substitutions, we get

$$A(1 - \alpha) - A\alpha = -1$$

That is,

$$A - A\alpha - A\alpha = -1$$

and so

$$A(1 - 2\alpha) = -1$$

Now $1 - 2\alpha = -\sqrt{5}$. So we have

$$A = \frac{1}{\sqrt{5}}$$

and (from the equation $A + B = 0$),

$$B = -A = -\frac{1}{\sqrt{5}}$$

Now we can put this all together. We have

$$\begin{aligned} F(x) &= \frac{A}{1 - \alpha x} + \frac{B}{1 - \beta x} \\ &= A \sum_{n=0}^{\infty} \alpha^n x^n + B \sum_{n=0}^{\infty} \beta^n x^n \\ &= \frac{1}{\sqrt{5}} \sum_{n=0}^{\infty} (\alpha^n - \beta^n) x^n \end{aligned}$$

and since the coefficients of F are by definition the Fibonacci numbers, we see that

$$f_n = \frac{1}{\sqrt{5}}(\alpha^n - \beta^n) = \frac{1}{\sqrt{5}} \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right)$$

and we are done.