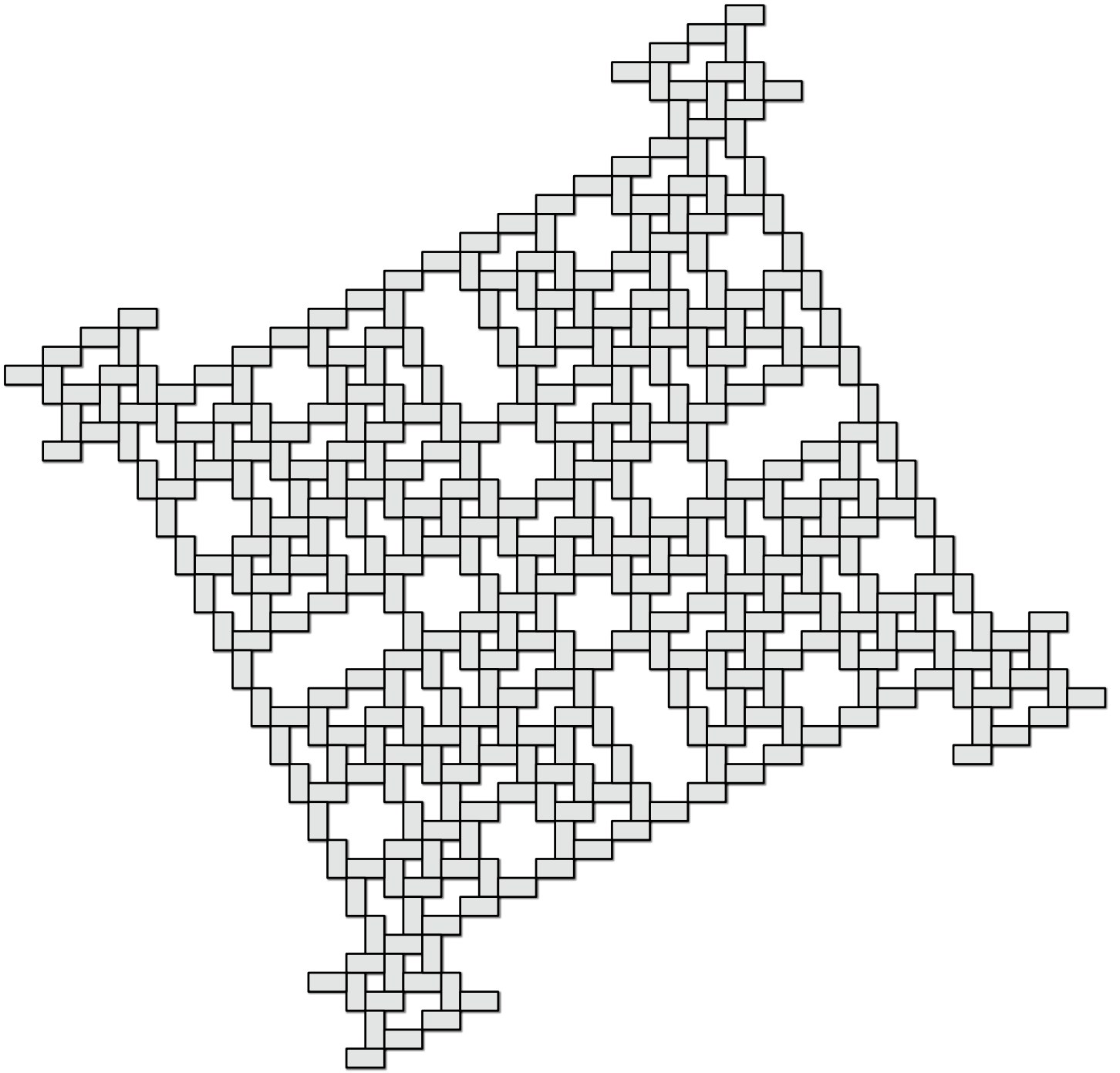


Appendices



*Jeder Genießende meint, dem Baume habe es an der Frucht gelegen;
aber ihm lag am Samen.
[Everyone who enjoys thinks that the fundamental thing about trees is the
fruit,
but in fact it is the seed.]*

— Friedrich Wilhelm Nietzsche,
Vermischte Meinungen und Sprüche [Mixed Opinions and Maxims] (1879)

*In view of all the deadly computer viruses that have been spreading lately,
Weekend Update would like to remind you:
When you link up to another computer,
you're linking up to every computer that that computer has ever linked up to.*
— Dennis Miller, "Saturday Night Live", (c. 1985)

Anything that, in happening, causes itself to happen again, happens again.
— Douglas Adams (2005)

*The Curling Stone slides; and, having slid,
Passes me toward thee on this Icy Grid,
If what's reached is passed for'll Crystals amid,
Th'Stone Reaches thee in its Eternal Skid.*
— Iraj Kalantari (2007)
writing as "Harak A'Myomy (12th century),
translated by Walt Friz De Gradde (1897)"

Proof by Induction

Induction is a method for proving universally quantified propositions—statements about *all* elements of a (usually infinite) set. Induction is also the single most useful tool for reasoning about, developing, and analyzing algorithms. These notes give several examples of inductive proofs, along with a standard boilerplate and some motivation to justify (and help you remember) why induction works.

1 Prime Divisors: Proof by Smallest Counterexample

A **divisor** of a positive integer n is a positive integer p such that the ratio n/p is an integer. The integer 1 is a divisor of every positive integer (because $n/1 = n$), and every integer is a divisor of itself (because $n/n = 1$). A **proper divisor** of n is any divisor of n other than n itself. A positive integer is **prime** if it has *exactly two* divisors, which must be 1 and itself; equivalently; a number is prime if and only if 1 is its only proper divisor. A positive integer is **composite** if it has more than two divisors (or equivalently, more than one proper divisor). The integer 1 is neither prime nor composite, because it has exactly one divisor, namely itself.

Let's prove our first theorem:

Theorem 1. *Every integer greater than 1 has a prime divisor.*

The very first thing that you should notice, after reading just one word of the theorem, is that this theorem is *universally quantified*—it's a statement about *all* the elements of a set, namely, the set of positive integers larger than 1. If we were forced at gunpoint to write this sentence

using fancy logic notation, the first character would be the universal quantifier \forall , pronounced 'for all'. Fortunately, that won't be necessary.

There are only two ways to prove a universally quantified statement: directly or by contradiction. Let's say that again, louder: **There are only two ways to prove a universally quantified statement: directly or by contradiction.** Here are the standard templates for these two methods, applied to Theorem 1:

Direct proof: Let n be an arbitrary integer greater than 1.
 \dots *blah blah blah* \dots
 Thus, n has at least one prime divisor. □

Proof by contradiction: For the sake of argument, assume there is an integer greater than 1 with no prime divisor.
 Let n be an arbitrary integer greater than 1 with no prime divisor.
 \dots *blah blah blah* \dots
 But that's just silly. Our assumption must be incorrect. □

The shaded boxes \dots *blah blah blah* \dots indicate missing proof details (that you will fill in). Most people usually find proofs by contradiction easier to discover than direct proofs, so let's try that first.

Proof by contradiction: For the sake of argument, assume there is an integer greater than 1 with no prime divisor.
 Let n be an arbitrary integer greater than 1 with no prime divisor.
 Since n is a divisor of n , and n has no prime divisors, n cannot be prime.
 Thus, n must have at least one divisor d such that $1 < d < n$.
 Let d be an arbitrary divisor of n such that $1 < d < n$.
 Since n has no prime divisors, d cannot be prime.
 Thus, d has at least one divisor d' such that $1 < d' < d$.
 Let d' be an arbitrary divisor of d such that $1 < d' < d$.
 Because d/d' is an integer, $n/d' = (n/d) \cdot (d/d')$ is also an integer.
 Thus, d' is also a divisor of n .
 Since n has no prime divisors, d' cannot be prime.
 Thus, d' has at least one divisor d_j such that $1 < d_j < d'$.
 Let d_j be an arbitrary divisor of d' such that $1 < d_j < d'$.
 Because d'/d_j is an integer, $n/d_j = (n/d') \cdot (d'/d_j)$ is also an integer.
 Thus, d_j is also a divisor of n .
 Since n has no prime divisors, d_j cannot be prime.
 \dots *blah HELP! blah I'M STUCK IN AN INFINITE LOOP! blah* \dots
 But that's just silly. Our assumption must be incorrect. □

We seem to be stuck in an infinite loop, looking at smaller and smaller divisors $d > d' > d_j > \dots$, none of which are prime. But this loop can't really be infinite. There are only $n - 1$ positive integers smaller than n , so the proof *must* end after *at most* $n - 1$ iterations. But how do we turn this observation into a formal proof? We need a single, self-contained proof for all integers n ; we're not allowed to write longer proofs for bigger integers. The trick is to jump directly to the *smallest* counterexample.

Proof by smallest counterexample: For the sake of argument, assume that there is an integer greater than 1 with no prime divisor.

Let n be **the smallest** integer greater than 1 with no prime divisor.

Since n is a divisor of n , and n has no prime divisors, n cannot be prime.

Thus, n has a divisor d such that $1 < d < n$.

Let d be a divisor of n such that $1 < d < n$.

Because n is the smallest counterexample, d has a prime divisor.

Let p be a prime divisor of d .

Because d/p is an integer, $n/p = (n/d) \cdot (d/p)$ is also an integer.

Thus, p is also a divisor of n .

But this contradicts our assumption that n has no prime divisors!

So our assumption must be incorrect. □

Hooray, our first proof! We're done!

Um... well... no, we're definitely *not* done. That's a first draft up there, not a final polished proof. We don't write proofs just to convince ourselves; proofs are primarily a tool to convince other people. (In particular, 'other people' includes the people grading your homeworks and exams.) And while proofs by contradiction are usually easier to *write*, direct proofs are almost always easier to *read*. So as a service to our audience (and our grade), let's transform our minimal-counterexample proof into a direct proof.

Let's first rewrite the indirect proof slightly, to make the structure more apparent. First, we break the assumption that n is the smallest counterexample into three simpler assumptions: (1) n is an integer greater than 1; (2) n has no prime divisors; and (3) there are no smaller counterexamples. Second, instead of dismissing the possibility that n is prime out of hand, we include an explicit case analysis.

Proof by smallest counterexample: Let n be an arbitrary integer greater than 1.

For the sake of argument, suppose n has no prime divisor.

Assume that every integer k such that $1 < k < n$ has a prime divisor.

There are two cases to consider: Either n is prime, or n is composite.

- Suppose n is prime.
 - Then n is a prime divisor of n .
- Suppose n is composite.
 - Then n has a divisor d such that $1 < d < n$.
 - Let d be a divisor of n such that $1 < d < n$.
 - Because no counterexample is smaller than n** , d has a prime divisor.
 - Let p be a prime divisor of d .
 - Because d/p is an integer, $n/p = (n/d) \cdot (d/p)$ is also an integer.
 - Thus, p is a prime divisor of n .

In each case, we conclude that n has a prime divisor.

But this contradicts our assumption that n has no prime divisors!

So our assumption must be incorrect. □

Now let's look carefully at the structure of this proof. First, we assumed that the statement we want to prove is false. Second, we proved that the statement we want to prove is true. Finally, we concluded from the contradiction that our assumption that the statement we want to prove is false is incorrect, so the statement we want to prove must be true.

But that's just silly. Why do we need the first and third steps? After all, the second step is a proof all by itself! Unfortunately, this redundant style of proof by contradiction is *extremely* common, even in professional papers. Fortunately, it's also very easy to avoid; just remove the first and third steps!

Proof by induction: Let n be an arbitrary integer greater than 1.
Assume that every integer k such that $1 < k < n$ has a prime divisor.
 There are two cases to consider: Either n is prime or n is composite.

- First, suppose n is prime.
 Then n is a prime divisor of n .
- Now suppose n is composite.
 Then n has a divisor d such that $1 < d < n$.
 Let d be a divisor of n such that $1 < d < n$.
Because no counterexample is smaller than n , d has a prime divisor.
 Let p be a prime divisor of d .
 Because d/p is an integer, $n/p = (n/d) \cdot (d/p)$ is also an integer.
 Thus, p is a prime divisor of n .

In both cases, we conclude that n has a prime divisor. □

This style of proof is called *induction*.¹ The assumption that there are no counterexamples smaller than n is called the *induction hypothesis*. The two cases of the proof have different names. The first case, which we argue directly, is called the *base case*. The second case, which actually uses the induction hypothesis, is called the *inductive case*. You may find it helpful to actually label the induction hypothesis, the base case(s), and the inductive case(s) in your proof.

The following point cannot be emphasized enough: The only difference between a proof by induction and a proof by smallest counterexample is the way we write down the argument. The essential structure of the proofs are exactly the same. The core of our original indirect argument is a proof of the following implication for all n :

n has no prime divisor \implies some number smaller than n has no prime divisor.

The core of our direct proof is the following logically equivalent implication:

every number smaller than n has a prime divisor $\implies n$ has a prime divisor

The left side of this implication is just the induction hypothesis.

The proofs we've been playing with have been very careful and explicit; until you're comfortable writing your own proofs, you should be equally careful. A more mature proof-writer might express the same proof more succinctly as follows:

Proof by induction: Let n be an arbitrary integer greater than 1. Assume that every integer k such that $1 < k < n$ has a prime divisor. If n is prime, then n is a prime divisor of n . On the other hand, if n is composite, then n has a proper divisor; call it d . The induction hypothesis implies that d has a prime divisor p . The integer p is also a divisor of n . □

A proof in this more succinct form is still worth full credit, provided the induction hypothesis is written explicitly and the case analysis is obviously exhaustive.

A professional mathematician would write the proof even more tersely:

Proof: Induction. □

And you can write that tersely, too, when you're a professional mathematician.

¹Many authors use the high-falutin' name *the principle of mathematical induction*, to distinguish it from *inductive reasoning*, the informal process by which we conclude that pigs can't whistle, horses can't fly, and NP-hard problems cannot be solved in polynomial time. We already know that every proof is mathematical (and arguably, all mathematics is proof), so as a description of a *proof* technique, the adjective 'mathematical' is simply redundant.

2 The Axiom of Induction

Why does this work? Well, let's step back to the original proof by smallest counterexample. How do we know that a smallest counterexample exists? This seems rather obvious, but in fact, it's impossible to prove without using the following seemingly trivial observation, called the *Well-Ordering Principle*:

Every non-empty set of positive integers has a smallest element.

Every set X of positive integers is the set of counterexamples to some proposition $P(n)$ (specifically, the proposition $n \notin X$). Thus, the Well-Ordering Principle can be rewritten as follows:

*If the proposition $P(n)$ is false for some positive integer n ,
then
the proposition $(P(1) \wedge P(2) \wedge \cdots \wedge P(n-1) \wedge \neg P(n))$ is true for some positive integer n .*

Equivalently, in English:

*If some statement about positive integers has a counterexample,
then
that statement has a smallest counterexample.*

We can write this implication in contrapositive form as follows:

*If the proposition $(P(1) \wedge P(2) \wedge \cdots \wedge P(n-1) \wedge \neg P(n))$ is false for every positive integer n ,
then
the proposition $P(n)$ is true for every positive integer n .*

or less formally,

*If some statement about positive integers has no smallest counterexample,
then
that statement is true for all positive integers.*

Finally, let's rewrite the first half of this statement in a logically equivalent form, by replacing $\neg(p \wedge \neg q)$ with $p \rightarrow q$.

*If the implication $(P(1) \wedge P(2) \wedge \cdots \wedge P(n-1)) \rightarrow P(n)$ is true for every positive integer n ,
then
the proposition $P(n)$ is true for every positive integer n .*

This formulation is usually called the *Axiom of Induction*. In a proof by induction that $P(n)$ holds for all n , the conjunction $(P(1) \wedge P(2) \wedge \cdots \wedge P(n-1))$ is the inductive hypothesis.

A *proof by induction* for the proposition “ $P(n)$ for every positive integer n ” is nothing but a direct proof of the more complex proposition “ $(P(1) \wedge P(2) \wedge \cdots \wedge P(n-1)) \rightarrow P(n)$ for every positive integer n ”. Because it's a direct proof, it *must* start by considering an arbitrary positive integer, which we might as well call n . Then, to prove the implication, we explicitly assume the hypothesis $(P(1) \wedge P(2) \wedge \cdots \wedge P(n-1))$ and then prove the conclusion $P(n)$ for that particular value of n . The proof almost always breaks down into two or more cases, each of which may or may not actually use the inductive hypothesis.

Here is the boilerplate for every induction proof. Read it. Learn it. Use it.

Theorem: $P(n)$ for every positive integer n .

Proof by induction: Let n be an arbitrary positive integer.

Assume inductively that $P(k)$ is true for every positive integer $k < n$.

There are several cases to consider:

- Suppose n is ... *blah blah blah* ...

Then $P(n)$ is true.

- Suppose n is ... *blah blah blah* ...

The inductive hypothesis implies that ... *blah blah blah* ...

Thus, $P(n)$ is true.

In each case, we conclude that $P(n)$ is true. □

Some textbooks distinguish between several different types of induction: ‘regular’ induction versus ‘strong’ induction versus ‘complete’ induction versus ‘structural’ induction versus ‘transfinite’ induction versus ‘Noetherian’ induction. Distinguishing between these different types of induction is pointless hairsplitting; I won’t even define them. Every ‘different type’ of induction proof is provably equivalent to a proof by smallest counterexample. (Later we will consider inductive proofs of statements about partially ordered sets other than the positive integers, for which ‘smallest’ has a different meaning, but this difference will prove to be inconsequential.)

3 Stamps and Recursion

Let’s move on to a completely different example.

Theorem 2. *Given an unlimited supply of 5-cent stamps and 7-cent stamps, we can make any amount of postage larger than 23 cents.*

We could prove this by contradiction, using a smallest-counterexample argument, but let’s aim for a direct proof by induction this time. We start by writing down the induction boilerplate, using the standard induction hypothesis: *There is no counterexample smaller than n .*

Proof by induction: Let n be an arbitrary integer greater than 23.

Assume that for any integer k such that $23 < k < n$, we can make k cents in postage.

... *blah blah blah* ...

Thus, we can make n cents in postage. □

How do we fill in the details? One approach is to think about what you would actually do if you really had to make n cents in postage. For example, you might start with a 5-cent stamp, and then try to make $n - 5$ cents in postage. The inductive hypothesis says you can make *any* amount of postage bigger than 23 cents and less than n cents. So if $n - 5 > 23$, then *you already know* that you can make $n - 5$ cents in postage! (You don’t know *how* to make $n - 5$ cents in postage, but so what?)

Let’s write this observation into our proof as two separate cases: either $n > 28$ (where our approach works) or $n \leq 28$ (where we don’t know what to do yet).

Proof by induction: Let n be an arbitrary integer greater than 23.
 Assume that for any integer k such that $23 < k < n$, we can make k cents in postage.
 There are two cases to consider: Either $n > 28$ or $n \leq 28$.

- Suppose $n > 28$.
 Then $23 < n - 5 < n$.
 Thus, **the induction hypothesis** implies that we can make $n - 5$ cents in postage.
 Adding one more 5-cent stamp gives us n cents in postage.
- Now suppose $n \leq 28$.
... blah blah blah ...

In both cases, we can make n cents in postage. □

What do we do in the second case? Fortunately, this case considers only five integers: 24, 25, 26, 27, and 28. There might be a clever way to solve all five cases at once, but why bother? They're small enough that we can find a solution by brute force in less than a minute. To make the proof more readable, I'll unfold the nested cases and list them in increasing order.

Proof by induction: Let n be an arbitrary integer greater than 23.
 Assume that for any integer k such that $23 < k < n$, we can make k cents in postage.
 There are six cases to consider: $n = 24$, $n = 25$, $n = 26$, $n = 27$, $n = 28$, and $n > 28$.

- $24 = 7 + 7 + 5 + 5$
- $25 = 5 + 5 + 5 + 5 + 5$
- $26 = 7 + 7 + 7 + 5$
- $27 = 7 + 5 + 5 + 5 + 5$
- $28 = 7 + 7 + 7 + 7$
- Suppose $n > 28$.
 Then $23 < n - 5 < n$.
 Thus, **the induction hypothesis** implies that we can make $n - 5$ cents in postage.
 Adding one more 5-cent stamp gives us n cents in postage.

In all cases, we can make n cents in postage. □

Voilà! An induction proof! More importantly, we now have a recipe for *discovering* induction proofs.

1. **Write down the boilerplate.** Write down the universal invocation ('Let n be an arbitrary...'), the induction hypothesis, and the conclusion, with enough blank space for the remaining details. Don't be clever. Don't even think. Just write. **This is the easy part.** To emphasize the common structure, the boilerplate will be indicated in green for the rest of this handout.
2. **Think big.** Don't think how to solve the problem all the way down to the ground; you'll only make yourself dizzy. Don't think about piddly little numbers like 1 or 5 or 10^{100} . Instead, think about how to reduce the proof about some *absfoluckingutely ginormous* value of n to a proof about some other number(s) smaller than n . **This is the hard part.**
3. **Look for holes.** Look for cases where your inductive argument breaks down. Solve those cases directly. Don't be clever here; be stupid but thorough.

4. **Rewrite everything.** Your first proof is a rough draft. Rewrite the proof so that your argument is easier for your (unknown?) reader to follow.

The cases in an inductive proof always fall into two categories. Any case that uses the inductive hypothesis is called an *inductive case*. Any case that does not use the inductive hypothesis is called a *base case*. Typically, but *not* always, base cases consider a few small values of n , and the inductive cases consider everything else. Induction proofs are usually clearer if we present the base cases first, but I find it much easier to *discover* the inductive cases first. In other words, I recommend writing induction proofs backwards.

Well-written induction proofs *very* closely resemble well-written recursive programs. We computer scientists use induction primarily to reason about recursion, so maintaining this resemblance is extremely useful—we only have to keep one mental pattern, called ‘induction’ when we’re writing proofs and ‘recursion’ when we’re writing code. Consider the following C and Scheme programs for making n cents in postage:

```
void postage(int n)
{
    assert(n>23);
    switch ($n$)
    {
        case 24: printf("7+7+5+5");    break;
        case 25: printf("5+5+5+5+5"); break;
        case 26: printf("7+7+7+5");   break;
        case 27: printf("7+5+5+5+5"); break;
        case 28: printf("7+7+7+7");   break;
        default:
            postage(n-5);
            printf("+5");
    }
}
```

```
(define (postage n)
  (cond ((= n 24) (5 5 7 7))
        ((= n 25) (5 5 5 5 5))
        ((= n 26) (5 7 7 7))
        ((= n 27) (5 5 5 5 7))
        ((= n 28) (7 7 7 7))
        (> n 28) (cons 5 (postage (- n 5))))))
```

The C program begins by declaring the input parameter (“Let n be an arbitrary integer. . .”) and asserting its range (“. . . greater than 23.”). (Scheme programs don’t have type declarations.) In both languages, the code branches into six cases: five that are solved directly, plus one that is handled by invoking the inductive hypothesis recursively.

4 More on Prime Divisors

Before we move on to different examples, let’s prove another fact about prime numbers:

Theorem 3. *Every positive integer is a product of prime numbers.*

First, let's write down the boilerplate. Hey! I saw that! You were *thinking*, weren't you? Stop that this instant! Don't make me turn the car around. **First** we write down the boilerplate.

Proof by induction: Let n be an arbitrary positive integer.
 Assume that any positive integer $k < n$ is a product of prime numbers.
 There are **some** cases to consider:
 ... blah blah blah ...
 Thus, n is a product of prime numbers. □

Now let's think about how you would actually factor a positive integer n into primes. There are a couple of different options here. One possibility is to find a prime divisor p of n , as guaranteed by Theorem 1, and recursively factor the integer n/p . This argument works as long as $n \geq 2$, but what about $n = 1$? The answer is simple: 1 is the product of the **empty** set of primes. What else could it be?

Proof by induction: Let n be an arbitrary positive integer.
 Assume that any positive integer $k < n$ is a product of prime numbers.
 There are two cases to consider: either $n = 1$ or $n \geq 2$.

- If $n = 1$, then n is the product of the elements of the empty set, each of which is prime, green, sparkly, vanilla, and hemophagic.
- Suppose $n > 1$. Let p be a prime divisor of n , as guaranteed by Theorem 2. The inductive hypothesis implies that the positive integer n/p is a product of primes, and clearly $n = (n/p) \cdot p$.

In both cases, n is a product of prime numbers. □

But an even simpler method is to factor n into any two proper divisors, and recursively handle them both. This method works as long as n is composite, since otherwise there is no way to factor n into smaller integers. Thus, we need to consider prime numbers separately, as well as the special case 1.

Proof by induction: Let n be an arbitrary positive integer.
 Assume that any positive integer $k < n$ is a product of prime numbers.
 There are three cases to consider: either $n = 1$, n is prime, or n is composite.

- If $n = 1$, then n is the product of the elements of the empty set, each of which is prime, red, broody, chocolate, and lycanthropic.
- If n is prime, then n is the product of one prime number, namely n .
- Suppose n is composite. Let d be any proper divisor of n (guaranteed by the definition of 'composite'), and let $m = n/d$. Since both d and m are positive integers smaller than n , the inductive hypothesis implies that d and m are both products of prime numbers. We clearly have $n = d \cdot m$.

In both cases, n is a product of prime numbers. □

5 Summations

Here's an easy one.

Theorem 4. $\sum_{i=0}^n 3^i = \frac{3^{n+1} - 1}{2}$ for every non-negative integer n .

First let's write down the induction boilerplate, which empty space for the details we'll fill in later.

Proof by induction: Let n be an arbitrary non-negative integer.

Assume inductively that $\sum_{i=0}^k 3^i = \frac{3^{k+1} - 1}{2}$ for every non-negative integer $k < n$.

There are some number of cases to consider:

... blah blah blah ...

We conclude that $\sum_{i=0}^n 3^i = \frac{3^{n+1} - 1}{2}$. □

Now imagine you are part of an infinitely long assembly line of mathematical provers, each assigned to a particular non-negative integer. Your task is to prove this theorem for the integer 8675310. The regulations of the Mathematical Provers Union require you not to think about any other integer but your own. The assembly line starts with the Senior Master Prover, who proves the theorem for the case $n = 0$. Next is the Assistant Senior Master Prover, who proves the theorem for $n = 1$. After him is the Assistant Assistant Senior Master Prover, who proves the theorem for $n = 2$. Then the Assistant Assistant Assistant Senior Master Prover proves the theorem for $n = 3$. As the work proceeds, you start to get more and more bored. You attempt strike up a conversation with Jenny, the prover to your left, but she ignores you, preferring to focus on the proof. Eventually, you fall into a deep, dreamless sleep. An undetermined time later, Jenny wakes you up by shouting, "Hey, doofus! It's your turn!" As you look around, bleary-eyed, you realize that Jenny and everyone to your left has finished their proofs, and that everyone is waiting for you to finish yours. What do you do?

What you do, after wiping the drool off your chin, is stop and think for a moment about what you're trying to prove. What does that \sum notation actually mean? Intuitively, we can expand the notation as follows:

$$\sum_{i=0}^{8675310} 3^i = 3^0 + 3^1 + \dots + 3^{8675309} + 3^{8675310}.$$

Notice that this expression also contains the summation that Jenny just finished proving something about:

$$\sum_{i=0}^{8675309} 3^i = 3^0 + 3^1 + \dots + 3^{8675308} + 3^{8675309}.$$

Putting these two expressions together gives us the following identity:

$$\sum_{i=0}^{8675310} 3^i = \sum_{i=0}^{8675309} 3^i + 3^{8675310}$$

In fact, this recursive identity is the *definition* of \sum . Jenny just proved that the summation on the right is equal to $(3^{8675310} - 1)/2$, so we can plug that into the right side of our equation:

$$\sum_{i=0}^{8675310} 3^i = \sum_{i=0}^{8675309} 3^i + 3^{8675310} = \frac{3^{8675310} - 1}{2} + 3^{8675310}.$$

And it's all downhill from here. After a little bit of algebra, you simplify the right side of this equation to $(3^{8675311} - 1)/2$, wake up the prover to your right, and start planning your well-earned vacation.

Let's insert this argument into our boilerplate, only using a generic 'big' integer n instead of the specific integer 8675310:

Proof by induction: Let n be an arbitrary non-negative integer.

Assume inductively that $\sum_{i=0}^k 3^i = \frac{3^{k+1}-1}{2}$ for every non-negative integer $k < n$.

There are two cases to consider: Either n is big or n is small.

- If n is big, then

$$\begin{aligned} \sum_{i=0}^n 3^i &= \sum_{i=0}^{n-1} 3^i + 3^n && \text{[definition of } \sum \text{]} \\ &= \frac{3^n - 1}{2} + 3^n && \text{[induction hypothesis, with } k = n - 1 \text{]} \\ &= \frac{3^{n+1} - 1}{2} && \text{[algebra]} \end{aligned}$$
- On the other hand, if n is small, then ... blah blah blah ...

In both cases, we conclude that $\sum_{i=0}^n 3^i = \frac{3^{n+1}-1}{2}$. □

Now, how big is 'big', and what do we do when n is 'small'? To answer the first question, let's look at where our existing inductive argument breaks down. In order to apply the induction hypothesis when $k = n - 1$, the integer $n - 1$ must be non-negative; equivalently, n must be positive. But that's the only assumption we need: **The only case we missed is $n = 0$** . Fortunately, this case is easy to handle directly.

Proof by induction: Let n be an arbitrary non-negative integer.

Assume inductively that $\sum_{i=0}^k 3^i = \frac{3^{k+1}-1}{2}$ for every non-negative integer $k < n$.

There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- If $n = 0$, then $\sum_{i=0}^n 3^i = 3^0 = 1$, and $\frac{3^{n+1}-1}{2} = \frac{3^1-1}{2} = 1$.
- On the other hand, if $n \geq 1$, then

$$\begin{aligned} \sum_{i=0}^n 3^i &= \sum_{i=0}^{n-1} 3^i + 3^n && \text{[definition of } \sum \text{]} \\ &= \frac{3^n - 1}{2} + 3^n && \text{[induction hypothesis, with } k = n - 1 \text{]} \\ &= \frac{3^{n+1} - 1}{2} && \text{[algebra]} \end{aligned}$$

In both cases, we conclude that $\sum_{i=0}^n 3^i = \frac{3^{n+1}-1}{2}$. □

Here is the same proof, written more tersely; the non-standard symbol $\stackrel{IH}{=}$ indicates the use of the induction hypothesis.

Proof by induction: Let n be an arbitrary non-negative integer, and assume inductively that $\sum_{i=0}^k 3^i = (3^{k+1} - 1)/2$ for every non-negative integer $k < n$. The base case $n = 0$ is trivial, and for any $n \geq 1$, we have

$$\sum_{i=0}^n 3^i = \sum_{i=0}^{n-1} 3^i + 3^n \stackrel{IH}{=} \frac{3^n - 1}{2} + 3^n = \frac{3^{n+1} - 1}{2}.$$

□

This is not the only way to prove this theorem by induction; here is another:

Proof by induction: Let n be an arbitrary non-negative integer, and assume inductively that $\sum_{i=0}^k 3^i = (3^{k+1} - 1)/2$ for every non-negative integer $k < n$. The base case $n = 0$ is trivial, and for any $n \geq 1$, we have

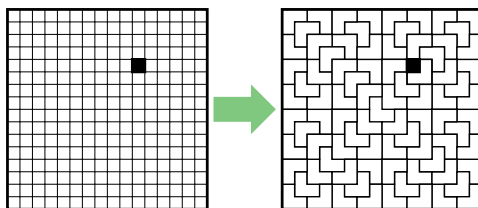
$$\sum_{i=0}^n 3^i = 3^0 + \sum_{i=1}^n 3^i = 3^0 + 3 \cdot \sum_{i=0}^{n-1} 3^i \stackrel{IH}{=} 3^0 + 3 \cdot \frac{3^n - 1}{2} = \frac{3^{n+1} - 1}{2}.$$

□

In the remainder of these notes, I'll give several more examples of induction proofs. In some cases, I give multiple proofs for the same theorem. Unlike the earlier examples, I will not describe the thought process that lead to the proof; in each case, I followed the basic outline on page 7.

6 Tiling with Triominos

The next theorem is about *tiling* a square checkerboard with *triominos*. A triomino is a shape composed of three squares meeting in an L-shape. Our goal is to cover as much of a $2^n \times 2^n$ grid with triominos as possible, without any two triominos overlapping, and with all triominos inside the square. We can't cover every square in the grid—the number of squares is 4^n , which is not a multiple of 3—but we can cover all but one square. In fact, as the next theorem shows, we can choose *any* square to be the one we don't want to cover.



Almost tiling a 16×16 checkerboard with triominos.

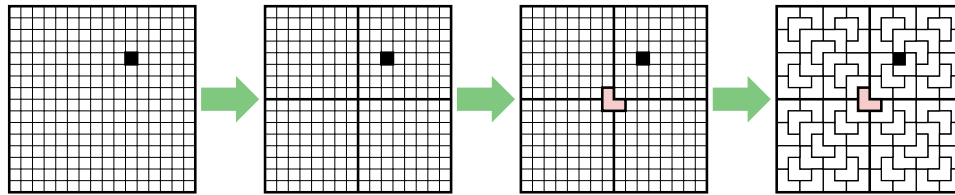
Theorem 5. For any non-negative integer n , the $2^n \times 2^n$ checkerboard with any square removed can be tiled using L-shaped triominos.

Here are two inductive proofs for this theorem, one ‘top down’, the other ‘bottom up’.

Proof by top-down induction: Let n be an arbitrary non-negative integer. Assume that for any non-negative integer $k < n$, the $2^k \times 2^k$ grid with any square removed can be tiled using triominos. There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- The $2^0 \times 2^0$ grid has a single square, so removing one square leaves nothing, which we can tile with zero triominos.
- Suppose $n \geq 1$. In this case, the $2^n \times 2^n$ grid can be divided into four smaller $2^{n-1} \times 2^{n-1}$ grids. Without loss of generality, suppose the deleted square is in the upper right quarter. With a single L-shaped triomino at the center of the board, we can cover one square in each of the other three quadrants. The induction hypothesis implies that we can tile each of the quadrants, minus one square.

In both cases, we conclude that the $2^n \times 2^n$ grid with any square removed can be tiled with triominos. □

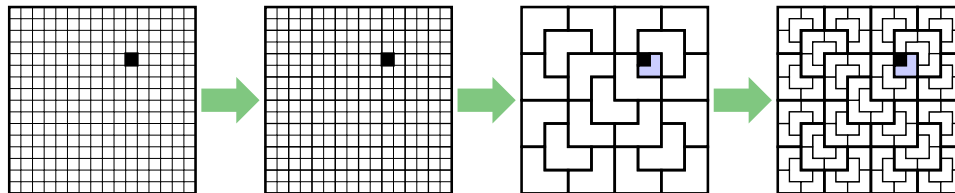


Top-down inductive proof of Theorem 4.

Proof by bottom-up induction: Let n be an arbitrary non-negative integer. Assume that for any non-negative integer $k < n$, the $2^k \times 2^k$ grid with any square removed can be tiled using triominos. There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- The $2^0 \times 2^0$ grid has a single square, so removing one square leaves nothing, which we can tile with zero triominos.
- Suppose $n \geq 1$. Then by clustering the squares into 2×2 blocks, we can transform any $2^n \times 2^n$ grid into a $2^{n-1} \times 2^{n-1}$ grid. Suppose square (i, j) has been removed from the $2^n \times 2^n$ grid. The induction hypothesis implies that the $2^{n-1} \times 2^{n-1}$ grid with block $(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor)$ removed can be tiled with double-size triominos. Each double-size triomino can be tiled with four smaller triominos, and block $(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor)$ with square (i, j) removed is another triomino.

In both cases, we conclude that the $2^n \times 2^n$ grid with any square removed can be tiled with triominos. □



Second proof of Theorem 4.

7 Binary Numbers Exist

Theorem 6. Every non-negative integer can be written as the sum of distinct powers of 2.

Intuitively, this theorem states that every number can be represented in binary. (That's not a proof, by the way; it's just a restatement of the theorem.) I'll present *four* distinct inductive proofs for this theorem. The first two are standard, by-the-book induction proofs.

Proof by top-down induction: Let n be an arbitrary non-negative integer. Assume that any non-negative integer less than n can be written as the sum of distinct powers of 2. There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- The base case $n = 0$ is trivial—the elements of the empty set are distinct and sum to zero.
- Suppose $n \geq 1$. Let k be the largest integer such that $2^k \leq n$, and let $m = n - 2^k$. Observe that $m < 2^{k+1} - 2^k = 2^k$. Because $0 \leq m < n$, the inductive hypothesis implies that m can be written as the sum of distinct powers of 2. Moreover, in the summation for m , each power of 2 is at most m , and therefore less than 2^k . Thus, $m + 2^k$ is the sum of distinct powers of 2.

In either case, we conclude that n can be written as the sum of distinct powers of 2. \square

Proof by bottom-up induction: Let n be an arbitrary non-negative integer. Assume that any non-negative integer less than n can be written as the sum of distinct powers of 2. There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- The base case $n = 0$ is trivial—the elements of the empty set are distinct and sum to zero.
- Suppose $n \geq 1$, and let $m = \lfloor n/2 \rfloor$. Because $0 \leq m < n$, the inductive hypothesis implies that m can be written as the sum of distinct powers of 2. Thus, $2m$ can also be written as the sum of distinct powers of 2, each of which is greater than 2^0 . If n is even, then $n = 2m$ and we are done; otherwise, $n = 2m + 2^0$ is the the sum of distinct powers of 2.

In either case, we conclude that n can be written as the sum of distinct powers of 2. \square

The third proof deviates slightly from the induction boilerplate. At the top level, this proof doesn't actually use induction at all! However, a key step requires its own (straightforward) inductive proof.

Proof by algorithm: Let n be an arbitrary non-negative integer. Let S be a multiset containing n copies of 2^0 . Modify S by running the following algorithm:

```

while  $S$  has more than one copy of any element  $2^i$ 
  Remove two copies of  $2^i$  from  $S$ 
  Insert one copy of  $2^{i+1}$  into  $S$ 

```

Each iteration of this algorithm reduces the cardinality of S by 1, so the algorithm must eventually halt. When the algorithm halts, the elements of S are distinct. We claim that just after each iteration of the while loop, the elements of S sum to n .

Proof by induction: Consider an arbitrary iteration of the loop. Assume inductively that just after each previous iteration, the elements of S sum to n . Before any iterations of the loop, the elements of S sum to n by definition. The induction hypothesis implies that just before the current iteration begins, the elements of S sum to n . The loop replaces two copies of some number 2^i with their sum 2^{i+1} , leaving the total sum of S unchanged. Thus, when the iteration ends, the elements of S sum to n . \square

Thus, when the algorithm halts, the elements of S are distinct powers of 2 that sum to n . We conclude that n can be written as the sum of distinct powers of 2. \square

The fourth proof uses so-called ‘weak’ induction, where the inductive hypothesis can only be applied at $n - 1$. Not surprisingly, tying all but one hand behind our backs makes the resulting proof longer, more complicated, and harder to read. It doesn’t help that the algorithm used in the proof is overly specific. Nevertheless, this is the first approach that occurs to most students who have not truly accepted the Recursion Fairy into their hearts.

Proof by baby-step induction: Let n be an arbitrary non-negative integer. Assume that any non-negative integer less than n can be written as the sum of distinct powers of 2. There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- The base case $n = 0$ is trivial—the elements of the empty set are distinct and sum to zero.
- Suppose $n \geq 1$. The inductive hypothesis implies that $n - 1$ can be written as the sum of distinct powers of 2. Thus, n can be written as the sum of powers of 2, which are distinct except possibly for two copies of 2^0 . Let S be this multiset of powers of 2.

Now consider the following algorithm:

```

i ← 0
while S has more than one copy of 2i
    Remove two copies of 2i from S
    Insert one copy of 2i+1 into S
    i ← i + 1
```

Each iteration of this algorithm reduces the cardinality of S by 1, so the algorithm must eventually halt. We claim that for every non-negative integer i , the following invariants are satisfied after the i th iteration of the while loop (or before the algorithm starts if $i = 0$):

- The elements of S sum to n .

Proof by induction: Let i be an arbitrary non-negative integer. Assume that for any non-negative integer $j \leq i$, after the j th iteration of the while loop, the elements of S sum to n . If $i = 0$, the elements of S sum to n by definition of S . Otherwise, the induction hypothesis implies that just *before* the i th iteration, the elements of S sum to n ; the i th iteration replaces two copies of 2^i with 2^{i+1} , leaving the sum unchanged. \square

- The elements in S are distinct, except possibly for two copies of 2^i .

Proof by induction: Let i be an arbitrary non-negative integer. Assume that for any non-negative integer $j \leq i$, after the j th iteration of the while loop, the elements of S are distinct except possibly for two copies of 2^j . If $i = 0$, the invariant holds by definition of S . So suppose $i > 0$. The induction hypothesis implies that just *before* the i th iteration, the elements of S are distinct except possibly for two copies of 2^i . If there are two copies of 2^i , the algorithm replaces them both with 2^{i+1} , and the invariant is established; otherwise, the algorithm halts, and the invariant is again established. \square

The second invariant implies that when the algorithm halts, the elements of S are distinct.

In either case, we conclude that n can be written as the sum of distinct powers of 2. \square

Repeat after me: “Doctor! Doctor! It hurts when I do this!”

8 Irrational Numbers Exist

Theorem 7. $\sqrt{2}$ is irrational.

Proof: I will prove that $p^2 \neq 2q^2$ (and thus $p/q \neq \sqrt{2}$) for all positive integers p and q .

Let p and q be arbitrary positive integers. Assume that for any positive integers $i < p$ and $j < q$, we have $i^2 \neq 2j^2$. Let $i = \lfloor p/2 \rfloor$ and $j = \lfloor q/2 \rfloor$. There are three cases to consider:

- Suppose p is odd. Then $p^2 = (2i + 1)^2 = 4i^2 + 4i + 1$ is odd, but $2q^2$ is even.
- Suppose p is even and q is odd. Then $p^2 = 4i^2$ is divisible by 4, but $2q^2 = 2(2j + 1)^2 = 4(2j^2 + 2j) + 2$ is not divisible by 4.
- Finally, suppose p and q are both even. The induction hypothesis implies that $i^2 \neq 2j^2$. Thus, $p^2 = 4i^2 \neq 8j^2 = 2q^2$.

In every case, we conclude that $p^2 \neq 2q^2$. □

This proof is usually presented as a proof by *infinite descent*, which is just another form of proof by smallest counterexample. Notice that the induction hypothesis assumed that *both* p and q were as small as possible. Notice also that the ‘base cases’ included every pair of integers p and q where at least one of the integers is odd.

9 Fibonacci Parity

The *Fibonacci numbers* 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... are recursively defined as follows:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2 \end{cases}$$

Theorem 8. For all non-negative integers n , F_n is even if and only if n is divisible by 3.

Proof: Let n be an arbitrary non-negative integer. Assume that for all non-negative integers $k < n$, F_k is even if and only if k is divisible by 3. There are three cases to consider: $n = 0$, $n = 1$, and $n \geq 2$.

- If $n = 0$, then n is divisible by 3, and $F_n = 0$ is even.
- If $n = 1$, then n is not divisible by 3, and $F_n = 1$ is odd.
- If $n \geq 2$, there are two subcases to consider: Either n is divisible by 3, or it isn't.
 - Suppose n is divisible by 3. Then neither $n - 1$ nor $n - 2$ is divisible by 3. Thus, the inductive hypothesis implies that both F_{n-1} and F_{n-2} are odd. So F_n is the sum of two odd numbers, and is therefore even.
 - Suppose n is not divisible by 3. Then exactly one of the numbers $n - 1$ and $n - 2$ is divisible by 3. Thus, the inductive hypothesis implies that exactly one of the numbers F_{n-1} and F_{n-2} is even, and the other is odd. So F_n is the sum of an even number and an odd number, and is therefore odd.

In all cases, F_n is even if and only if n is divisible by 3. □

10 Recursive Functions

Theorem 9. Suppose the function $F: \mathbb{N} \rightarrow \mathbb{N}$ is defined recursively by setting $F(0) = 0$ and $F(n) = 1 + F(\lfloor n/2 \rfloor)$ for every positive integer n . Then for every positive integer n , we have $F(n) = 1 + \lfloor \log_2 n \rfloor$.

Proof: Let n be an arbitrary positive integer. Assume that $F(k) = 1 + \lfloor \log_2 k \rfloor$ for every positive integer $k < n$. There are two cases to consider: Either $n = 1$ or $n \geq 2$.

- Suppose $n = 1$. Then $F(n) = F(1) = 1 + F(\lfloor 1/2 \rfloor) = 1 + F(0) = 1$ and $1 + \lfloor \log_2 n \rfloor = 1 + \lfloor \log_2 1 \rfloor = 1 + \lfloor 0 \rfloor = 1$.
- Suppose $n \geq 2$. Because $1 \leq \lfloor n/2 \rfloor < n$, the induction hypothesis implies that $F(\lfloor n/2 \rfloor) = 1 + \lfloor \log_2 \lfloor n/2 \rfloor \rfloor$. The definition of $F(n)$ now implies that $F(n) = 1 + F(\lfloor n/2 \rfloor) = 2 + \lfloor \log_2 \lfloor n/2 \rfloor \rfloor$.

Now there are two subcases to consider: n is either even or odd.

- If n is even, then $\lfloor n/2 \rfloor = n/2$, which implies

$$\begin{aligned} F(n) &= 2 + \lfloor \log_2 \lfloor n/2 \rfloor \rfloor \\ &= 2 + \lfloor \log_2 (n/2) \rfloor \\ &= 2 + \lfloor (\log_2 n) - 1 \rfloor \\ &= 2 + \lfloor \log_2 n \rfloor - 1 \\ &= 1 + \lfloor \log_2 n \rfloor. \end{aligned}$$

- If n is odd, then $\lfloor n/2 \rfloor = (n-1)/2$, which implies

$$\begin{aligned} F(n) &= 2 + \lfloor \log_2 \lfloor n/2 \rfloor \rfloor \\ &= 2 + \lfloor \log_2 ((n-1)/2) \rfloor \\ &= 1 + \lfloor \log_2 (n-1) \rfloor \\ &= 1 + \lfloor \log_2 n \rfloor \end{aligned}$$

by the algebra in the even case. Because $n > 1$ and n is odd, n cannot be a power of 2; thus, $\lfloor \log_2 n \rfloor = \lfloor \log_2 (n-1) \rfloor$.

In all cases, we conclude that $F(n) = 1 + \lfloor \log_2 n \rfloor$. □

11 Trees

Recall that a *tree* is a connected undirected graph with no cycles. A *subtree* of a tree T is a connected subgraph of T ; a *proper subtree* is any tree except T itself.

Theorem 10. In every tree, the number of vertices is one more than the number of edges.

This one is actually pretty easy to prove directly from the definition of ‘tree’: a connected acyclic graph.

Proof: Let T be an arbitrary tree. Choose an arbitrary vertex v of T to be the root, and direct every edge of T outward from v . Because T is connected, every node except v has at least one edge directed into it. Because T is acyclic, every node has at most one edge directed into it, and no edge is directed into v . Thus, for every node $x \neq v$, there is exactly one edge directed into x . We conclude that the number of edges is one less than the number of nodes. □

But we can prove this theorem by induction as well, in several different ways. Each inductive proof is structured around a different recursive definition of ‘tree’. First, a tree is either a single node, or two trees joined by an edge.

Proof: Let T be an arbitrary tree. Assume that in any proper subtree of T , the number of vertices is one more than the number of edges. There are two cases to consider: Either T has one vertex, or T has more than one vertex.

- If T has one vertex, then it has no edges.
- Suppose T has more than one vertex. Because T is connected, every pair of vertices is joined by a path. Thus, T must contain at least one edge. Let e be an arbitrary edge of T , and consider the graph $T \setminus e$ obtained by deleting e from T .

Because T is acyclic, there is no path in $T \setminus e$ between the endpoints of e . Thus, T has at least two connected components. On the other hand, because T is connected, $T \setminus e$ has at most two connected components. Thus, $T \setminus e$ has exactly two connected components; call them A and B .

Because T is acyclic, subgraphs A and B are also acyclic. Thus, A and B are subtrees of T , and therefore the induction hypothesis implies that $|E(A)| = |V(A)| - 1$ and $|E(B)| = |V(B)| - 1$.

Because A and B do not share any vertices or edges, we have $|V(T)| = |V(A)| + |V(B)|$ and $|E(T)| = |E(A)| + |E(B)| + 1$.

Simple algebra now implies that $|E(T)| = |V(T)| - 1$.

In both cases, we conclude that the number of vertices in T is one more than the number of edges in T . \square

Second, a tree is a single node connected by edges to a finite set of trees.

Proof: Let T be an arbitrary tree. Assume that in any proper subtree of T , the number of vertices is one more than the number of edges. There are two cases to consider: Either T has one vertex, or T has more than one vertex.

- If T has one vertex, then it has no edges.
- Suppose T has more than one vertex. Let v be an arbitrary vertex of T , and let d be the degree of v . Delete v and all its incident edges from T to obtain a new graph G . This graph has exactly d connected components; call them G_1, G_2, \dots, G_d . Because T is acyclic, every subgraph of T is acyclic. Thus, every subgraph G_i is a proper subtree of G . So the induction hypothesis implies that $|E(G_i)| = |V(G_i)| - 1$ for each i . We conclude that

$$|E(T)| = d + \sum_{i=1}^d |E(G_i)| = d + \sum_{i=1}^d (|V(G_i)| - 1) = \sum_{i=1}^d |V(G_i)| = |V(T)| - 1.$$

In both cases, we conclude that the number of vertices in T is one more than the number of edges in T . \square

But you should *never* attempt to argue like this:

Not a Proof: The theorem is clearly true for the 1-node tree. So let T be an arbitrary tree with at least two nodes. Assume inductively that the number of vertices in T is one more than the number of edges in T . Suppose we add one more leaf to T to get a new tree T' . This new tree has one more vertex than T and one more edge than T . Thus, the number of vertices in T' is one more than the number of edges in T' . \square

This is not a proof. Every sentence is true, and the connecting logic is correct, but it does not imply the theorem, because it doesn't *explicitly* consider *all possible* trees. Why should the reader believe that their favorite tree can be recursively constructed by adding leaves to a 1-node tree? It's *true*, of course, but that argument doesn't *prove* it. Remember: **There are only two ways to prove any universally quantified statement:** Directly ("Let T be an arbitrary tree. . .") or by contradiction ("Suppose some tree T doesn't. . .").

Here is a *correct* inductive proof using the same underlying idea. In this proof, I don't have to prove that the proof considers arbitrary trees; it says so right there on the first line! As usual, the proof very strongly resembles a recursive algorithm, including a subroutine to find a leaf.

Proof: Let T be an arbitrary tree. Assume that in any proper subtree of T , the number of vertices is one more than the number of edges. There are two cases to consider: Either T has one vertex, or T has more than one vertex.

- If T has one vertex, then it has no edges.
- Otherwise, T must have at least one vertex of degree 1, otherwise known as a leaf.

Proof: Consider a walk through the graph T that starts at an arbitrary vertex and continues as long as possible without repeating any edge. The walk can never visit the same vertex more than once, because T is acyclic. Whenever the walk visits a vertex of degree at least 2, it can continue further, because that vertex has at least one unvisited edge. But the walk must eventually end, because T is finite. Thus, the walk must eventually reach a vertex of degree 1. \square

Let ℓ be an arbitrary leaf of T , and let T' be the tree obtained by deleting ℓ from T . Then we have the identity

$$|E(T)| = |E(T')| + 1 = |V(T')| = |V(T)| - 1,$$

where the first and third equalities follow from the definition of T' , and the second equality follows from the inductive hypothesis.

In both cases, we conclude that the number of vertices in T is one more than the number of edges in T . \square

Exercises

1. Prove that given an unlimited supply of 6-cent coins, 10-cent coins, and 15-cent coins, one can make any amount of change larger than 29 cents.

2. Prove that $\sum_{i=0}^n r^i = \frac{1-r^{n+1}}{1-r}$ for every non-negative integer n and every real number $r \neq 1$.

3. Prove that $\left(\sum_{i=0}^n i\right)^2 = \sum_{i=0}^n i^3$ for every non-negative integer n .

4. Recall the standard recursive definition of the Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, and

$F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Prove the following identities for all non-negative integers n and m .

(a) $\sum_{i=0}^n F_i = F_{n+2} - 1$

(b) $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

* (c) If n is an integer multiple of m , then F_n is an integer multiple of F_m .

5. Prove that every integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example:

$$42 = 3^4 - 3^3 - 3^2 - 3^1$$

$$25 = 3^3 - 3^1 + 3^0$$

$$17 = 3^3 - 3^2 - 3^0$$

6. Prove that every integer (positive, negative, or zero) can be written in the form $\sum_i (-2)^i$, where the exponents i are distinct non-negative integers. For example:

$$42 = (-2)^6 + (-2)^5 + (-2)^4 + (-2)^0$$

$$25 = (-2)^6 + (-2)^5 + (-2)^3 + (-2)^0$$

$$17 = (-2)^4 + (-2)^0$$

7. (a) Prove that every non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number F_i appears in the sum, it appears exactly once, and its neighbors F_{i-1} and F_{i+1} do not appear at all. For example:

$$17 = F_7 + F_4 + F_2$$

$$42 = F_9 + F_6$$

$$54 = F_9 + F_7 + F_5 + F_3$$

- (b) Prove that every positive integer can be written as the sum of distinct Fibonacci numbers *with no consecutive gaps*. That is, for any index $i \geq 1$, if the consecutive Fibonacci numbers F_i or F_{i+1} do not appear in the sum, then no larger Fibonacci number F_j with $j > i$ appears in the sum. In particular, the sum *must* include either F_1 or F_2 . For example:

$$16 = F_6 + F_5 + F_3 + F_2$$

$$42 = F_8 + F_7 + F_5 + F_3 + F_1$$

$$54 = F_8 + F_7 + F_6 + F_5 + F_4 + F_3 + F_2 + F_1$$

- (c) The Fibonacci sequence can be extended backward to negative indices by rearranging the defining recurrence: $F_n = F_{n+2} - F_{n+1}$. Here are the first several negative-index Fibonacci numbers:

n	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
F_n	-55	34	-21	13	-8	5	-3	2	-1	1

Prove that $F_{-n} = (-1)^{n+1}F_n$.

- * (d) Prove that *every* integer—positive, negative, or zero—can be written as the sum of distinct, non-consecutive Fibonacci numbers *with negative indices*. For example:

$$\begin{aligned} 17 &= F_{-7} + F_{-5} + F_{-2} \\ -42 &= F_{-10} + F_{-7} \\ 54 &= F_{-9} + F_{-7} + F_{-5} + F_{-3} + F_{-1}. \end{aligned}$$

8. Consider the following game played with a finite number of identical coins, which are arranged into *stacks*. Each coin belongs to exactly one stack. Let n_i denote the number of coins in stack i . In each turn, you must make one of the following moves:

- For some i and j such that $n_j \leq n_i - 2$, move one coin from stack i to stack j .
- Move one coin from any stack into a new stack.
- Find a stack containing only one coin, and remove that coin from the game.

The game ends when all coins are gone. For example, the following sequence of turns describes a complete game; each vector lists the number of coins in each non-empty stack:

$$\begin{aligned} \langle 4, 2, 1 \rangle &\implies \langle 4, 1, 1, 1 \rangle \implies \langle 3, 2, 1, 1 \rangle \implies \langle 2, 2, 2, 1 \rangle \implies \langle 2, 2, 1, 1, 1 \rangle \\ &\implies \langle 2, 1, 1, 1, 1, 1 \rangle \implies \langle 2, 1, 1, 1, 1 \rangle \implies \langle 2, 1, 1, 1 \rangle \implies \langle 2, 1, 1 \rangle \\ &\implies \langle 2, 1 \rangle \implies \langle 2 \rangle \implies \langle 1, 1 \rangle \implies \langle 1 \rangle \implies \langle \rangle \end{aligned}$$

- (a) Prove that this game ends after a finite number of turns.
- (b) What are the minimum and maximum number of turns in a game, if we start with a single stack of n coins? Prove your answers are correct.
- (c) Now suppose each time you remove a coin from a stack, you must place *two* coins onto smaller stacks. In each turn, you must make one of the following moves:
- For some indices i , j , and k such that $n_j \leq n_i - 2$ and $n_k \leq n_i - 2$ and $j \neq k$, remove a coin from stack i , add a coin to stack j , and add a coin to stack k .
 - For some i and j such that $n_j \leq n_i - 2$, remove a coin from stack i , add a coin to stack j , and create a new stack with one coin.
 - Remove one coin from any stack and create two new stacks, each with one coin.
 - Find a stack containing only one coin, and remove that coin from the game.

For example, the following sequence of turns describes a complete game:

$$\begin{aligned} \langle 4, 2, 1 \rangle &\implies \langle 3, 3, 2 \rangle \implies \langle 3, 2, 2, 1, 1 \rangle \implies \langle 3, 2, 2, 1 \rangle \implies \langle 3, 2, 2 \rangle \implies \langle 3, 2, 1, 1, 1 \rangle \\ &\implies \langle 2, 2, 2, 2, 1 \rangle \implies \langle 2, 2, 2, 2 \rangle \implies \langle 2, 2, 2, 1, 1, 1 \rangle \implies \langle 2, 2, 2, 1, 1 \rangle \\ &\implies \langle 2, 2, 2, 1 \rangle \implies \langle 2, 2, 2 \rangle \implies \langle 2, 2, 1, 1, 1 \rangle \implies \langle 2, 2, 1, 1 \rangle \implies \langle 2, 2, 1 \rangle \\ &\implies \langle 2, 2 \rangle \implies \langle 2, 1, 1, 1 \rangle \implies \langle 1, 1, 1, 1, 1, 1 \rangle \implies \langle 1, 1, 1, 1, 1 \rangle \implies \langle 1, 1, 1, 1 \rangle \\ &\implies \langle 1, 1, 1 \rangle \implies \langle 1, 1 \rangle \implies \langle 1 \rangle \implies \langle \rangle. \end{aligned}$$

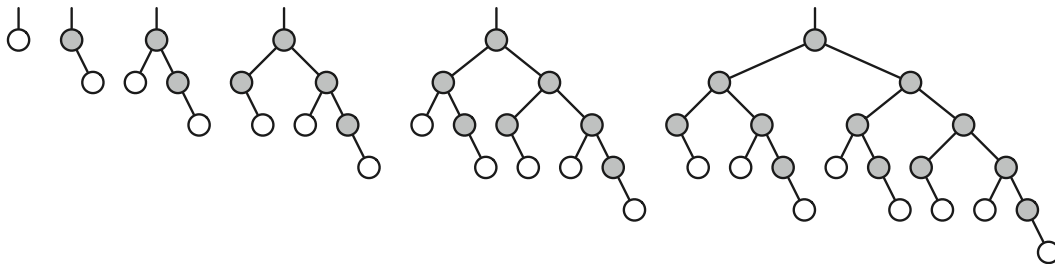
Prove that this modified game still ends after a finite number of turns.

- (d) What are the minimum and maximum number of turns in this modified game, starting with a single stack of n coins? Prove your answers are correct.

9. (a) Prove that $|A \times B| = |A| \times |B|$ for all finite sets A and B .
 (b) Prove that for all *non-empty* finite sets A and B , there are exactly $|B|^{|A|}$ functions from A to B .

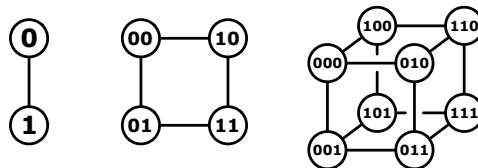
10. Recall that a binary tree is *full* if every node has either two children (an internal node) or no children (a leaf). Give at least *four different* proofs of the following fact: *In any full binary tree, the number of leaves is exactly one more than the number of internal nodes.*

11. The n th *Fibonacci binary tree* \mathcal{F}_n is defined recursively as follows:
 - \mathcal{F}_1 is a single root node with no children.
 - For all $n \geq 2$, \mathcal{F}_n is obtained from \mathcal{F}_{n-1} by adding a right child to every leaf and adding a left child to every node that has only one child.
 - (a) Prove that the number of leaves in \mathcal{F}_n is precisely the n th Fibonacci number: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$.
 - (b) How many nodes does \mathcal{F}_n have? Give an exact, closed-form answer in terms of Fibonacci numbers, and prove your answer is correct.
 - (c) Prove that for all $n \geq 2$, the right subtree of \mathcal{F}_n is a copy of \mathcal{F}_{n-1} .
 - (d) Prove that for all $n \geq 3$, the left subtree of \mathcal{F}_n is a copy of \mathcal{F}_{n-2} .



The first six Fibonacci binary trees. In each tree \mathcal{F}_n , the subtree of gray nodes is \mathcal{F}_{n-1} .

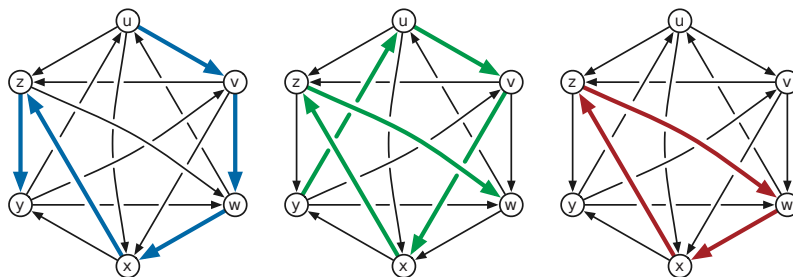
12. The d -dimensional hypercube is the graph defined as follows. There are 2^d vertices, each labeled with a different string of d bits. Two vertices are joined by an edge if and only if their labels differ in exactly one bit.



The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

Recall that a Hamiltonian cycle is a closed walk that visits each vertex in a graph exactly once. Prove that for every integer $d \geq 2$, the d -dimensional hypercube has a Hamiltonian cycle.

13. A **tournament** is a directed graph with exactly one directed edge between each pair of vertices. That is, for any vertices v and w , a tournament contains either an edge $v \rightarrow w$ or an edge $w \rightarrow v$, but not both. A **Hamiltonian path** in a directed graph G is a directed path that visits every vertex of G exactly once.
- (a) Prove that every tournament contains a Hamiltonian path.
- (b) Prove that every tournament contains either *exactly one* Hamiltonian path or a directed cycle of length three.



A tournament with two Hamiltonian paths $u \rightarrow v \rightarrow w \rightarrow x \rightarrow z \rightarrow y$ and $y \rightarrow u \rightarrow v \rightarrow x \rightarrow z \rightarrow w$ and a directed triangle $w \rightarrow x \rightarrow z \rightarrow w$.

14. Scientists recently discovered a planet, tentatively named “Ygdrasil”, that is inhabited by a bizarre species called “nertices” (singular “nertex”). All nertices trace their ancestry back to a particular nertex named Rudy. Rudy is still quite alive, as is every one of his many descendants. Nertices reproduce asexually; every nertex has exactly one parent (except Rudy, who sprang forth fully formed from the planet’s core). There are three types of nertices—red, green, and blue. The color of each nertex is correlated exactly with the number and color of its children, as follows:

- Each red nertex has two children, exactly one of which is green.
- Each green nertex has exactly one child, which is not green.
- Blue nertices have no children.

In each of the following problems, let R , G , and B respectively denote the number of red, green, and blue nertices on Ygdrasil.

- (a) Prove that $B = R + 1$.
- (b) Prove that either $G = R$ or $G = B$.
- (c) Prove that $G = B$ if and only if Rudy is green.
15. **Well-formed formulas** (wffs) are defined recursively as follows:

- T is a wff.
- F is a wff.
- Any proposition variable is a wff.
- If X is a wff, then $(\neg X)$ is also a wff.
- If X and Y are wffs, then $(X \wedge Y)$ is also a wff.

- If X and Y are wffs, then $(X \vee Y)$ is also a wff.

We say that a formula is in **De Morgan normal form** if it satisfies the following conditions. (“De Morgan normal form” is not standard terminology; I just made it up.)

- Every negation in the formula is applied to a variable, not to a more complicated subformula.
- Either the entire formula is T , or the formula does not contain T .
- Either the entire formula is F , or the formula does not contain F .

Prove that for every wff, there is a logically equivalent wff in De Morgan normal form. For example, the well-formed formula

$$(\neg((p \wedge q) \vee \neg r)) \wedge (\neg(p \vee \neg r) \wedge q)$$

is logically equivalent to the following wff in De Morgan normal form:

$$(((\neg p \vee \neg q) \wedge r)) \wedge ((\neg p \wedge r) \wedge q)$$

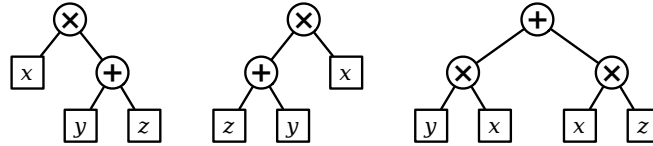
16. A **polynomial** is a function $f : \mathbb{R} \rightarrow \mathbb{R}$ of the form $f(x) = \sum_{i=0}^d a_i x^i$ for some non-negative integer d (called the degree) and some real numbers a_0, a_1, \dots, a_d (called the coefficients).
- Prove that the sum of two polynomials is a polynomial.
 - Prove that the product of two polynomials is a polynomial.
 - Prove that the composition $f(g(x))$ of two polynomials $f(x)$ and $g(x)$ is a polynomial.
 - Prove that the derivative f' of a polynomial f is a polynomial, using **only** the following facts:
 - Constant rule: If f is constant, then f' is identically zero.
 - Sum rule: $(f + g)' = f' + g'$.
 - Product rule: $(f \cdot g)' = f' \cdot g + f \cdot g'$.

- *17. An **arithmetic expression tree** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are $+$ and \times . Different leaves may or may not represent different variables.

Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any $+$ -node is the sum of the values of its children. (2) The value of any \times -node is the product of the values of its children.

Two arithmetic expression trees are **equivalent** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots. An arithmetic expression tree is in **normal form** if the parent of every $+$ -node (if any) is another $+$ -node.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form. [Hint: This is harder than it looks.]



Three equivalent expression trees. Only the third is in normal form.

- *18. A **Gaussian integer** is a complex number of the form $x + yi$, where x and y are integers. Prove that any Gaussian integer can be expressed as the sum of distinct powers of the complex number $\alpha = -1 + i$. For example:

$$\begin{aligned}
 4 &= 16 + (-8 - 8i) + 8i + (-4) &= \alpha^8 + \alpha^7 + \alpha^6 + \alpha^4 \\
 -8 &= (-8 - 8i) + 8i &= \alpha^7 + \alpha^6 \\
 15i &= (-16 + 16i) + 16 + (-2i) + (-1 + i) + 1 &= \alpha^9 + \alpha^8 + \alpha^2 + \alpha^1 + \alpha^0 \\
 1 + 6i &= (8i) + (-2i) + 1 &= \alpha^6 + \alpha^2 + \alpha^0 \\
 2 - 3i &= (4 - 4i) + (-4) + (2 + 2i) + (-2i) + (-1 + i) + 1 &= \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^0 \\
 -4 + 2i &= (-16 + 16i) + 16 + (-8 - 8i) + (4 - 4i) + (-2i) &= \alpha^9 + \alpha^8 + \alpha^7 + \alpha^5 + \alpha^2
 \end{aligned}$$

The following list of values may be helpful:

$$\begin{array}{llll}
 \alpha^0 = 1 & \alpha^4 = -4 & \alpha^8 = 16 & \alpha^{12} = -64 \\
 \alpha^1 = -1 + i & \alpha^5 = 4 - 4i & \alpha^9 = -16 + 16i & \alpha^{13} = 64 - 64i \\
 \alpha^2 = -2i & \alpha^6 = 8i & \alpha^{10} = -32i & \alpha^{14} = 128i \\
 \alpha^3 = 2 + 2i & \alpha^7 = -8 - 8i & \alpha^{11} = 32 + 32i & \alpha^{15} = -128 - 128i
 \end{array}$$

[Hint: How do you write $-2 - i$?]

- *19. *Lazy binary* is a variant of standard binary notation for representing natural numbers where we allow each “bit” to take on one of three values: 0, 1, or 2. Lazy binary notation is defined inductively as follows.

- The lazy binary representation of zero is 0.
- Given the lazy binary representation of any non-negative integer n , we can construct the lazy binary representation of $n + 1$ as follows:
 - (a) increment the rightmost digit;
 - (b) if any digit is equal to 2, replace the rightmost 2 with 0 and increment the digit immediately to its left.

Here are the first several natural numbers in lazy binary notation:

0, 1, 10, 11, 20, 101, 110, 111, 120, 201, 210, 1011, 1020, 1101, 1110, 1111, 1120, 1201, 1210, 2011, 2020, 2101, 2110, 10111, 10120, 10201, 10210, 11011, 11020, 11101, 11110, 11111, 11120, 11201, 11210, 12011, 12020, 12101, 12110, 20111, 20120, 20201, 20210, 21011, 21020, 21101, 21110, 101111, 101120, 101201, 101210, 102011, 102020, . . .

- (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.
- (b) Prove that for any natural number N , the sum of the digits of the lazy binary representation of N is exactly $\lfloor \lg(N + 1) \rfloor$.

- ★20. Consider the following recursively defined sequence of rational numbers:

$$R_0 = 0$$

$$R_n = \frac{1}{2[R_{n-1}] - R_{n-1} + 1} \quad \text{for all } n \geq 1$$

The first several elements of this sequence are

$$0, 1, \frac{1}{2}, 2, \frac{1}{3}, \frac{3}{2}, \frac{2}{3}, 3, \frac{1}{4}, \frac{4}{3}, \frac{3}{5}, \frac{5}{2}, \frac{2}{5}, \frac{5}{3}, \frac{3}{4}, 4, \frac{1}{5}, \dots$$

Prove that every non-negative rational number appears in this sequence exactly once.

21. Let $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ be an **arbitrary** (not necessarily continuous) function such that

- $f(x) > 0$ for all $x > 0$, and
- $f(x) = \pi f(x/\sqrt{2})$ for all $x > 1$.

Prove by induction that $f(x) = \Theta(x)$ (as $x \rightarrow \infty$). Yes, this is induction over the real numbers.

- *22. There is a natural generalization of induction to the real numbers that is familiar to analysts but relatively unknown in computer science. The precise formulation given below is was proposed independently by Hathaway² and Clark³ fairly recently, but the idea dates back to at least to the 1920s. Recall that there are four types of intervals for any real numbers a and z :

- The **open** interval $(\mathbf{a}, \mathbf{z}) := \{t \in \mathbb{R} \mid a \leq t < z\}$,
- The **half-open** intervals $[\mathbf{a}, \mathbf{z}) := \{t \in \mathbb{R} \mid a \leq t < z\}$ and $(\mathbf{a}, \mathbf{z}] := \{t \in \mathbb{R} \mid a < t \leq z\}$
- The **closed** interval $[\mathbf{a}, \mathbf{z}] := \{t \in \mathbb{R} \mid a \leq t \leq z\}$.

Theorem 11 (Continuous Induction). Fix a closed interval $[a, z] \subset \mathbb{R}$. Suppose some subset $S \subseteq [a, z]$ has following properties:

- (a) $a \in S$.
- (b) If $a \leq s < z$ and $s \in S$, then $[s, u] \subseteq S$ for some $u > s$.
- (c) If $a \leq s \leq z$ and $[a, s] \subseteq S$, then $s \in S$.

Then $S = [a, z]$.

²Dan Hathaway. Using continuity induction. *College Math. J.* 42:229–231, 2011.

³Pete L. Clark. The instructor's guide to real induction. [arXiv:1208.0973](https://arxiv.org/abs/1208.0973).

Proof: For the sake of argument, let S be a proper subset of $[a, b]$. Let $T = [a, z] \setminus S$. Because \bar{S} is bounded but non-empty, it has a greatest lower bound $\ell \in [a, z]$. More explicitly, ℓ be the largest real number such that $\ell \leq t$ for all $t \in T$. There are three cases to consider:

- Suppose $\ell = a$. Condition (a) and (b) imply that $[a, u] \in S$ for some $u > a$. But then we have $\ell = a < u \leq t$ for all $t \in T$, contradicting the fact that ℓ is the *greatest* lower bound of T .
- Suppose $\ell > a$ and $\ell \in S$. If $\ell = z$, then $S = [a, z]$, contradicting our initial assumption. Otherwise, by condition (b), we have $[\ell, u] \subseteq S$ for some $u > \ell$, again contradicting the fact that ℓ is the *greatest* lower bound of T .
- Finally, suppose $\ell > a$ and $\ell \in \bar{S}$. Because no element of T is smaller than ℓ , we have $[a, \ell] \subseteq S$. But then condition (c) implies that $\ell \in S$, and we have a contradiction.

In all cases, we have a contradiction. □

Continuous induction hinges on the **axiom of completeness**—every non-empty set of positive real numbers has a greatest lower bound—just as standard induction requires the **well-ordering principle**—every non-empty set of positive integers has a smallest element. Thus, continuous induction cannot be used to prove properties of *rational* numbers, because the greatest lower bound of a set of rational numbers need not be rational.

Fix real numbers $a \leq z$. Recall that a function $f : [a, z] \rightarrow \mathbb{R}$ is **continuous** if it satisfies the following condition: for any $t \in [a, z]$ and any $\varepsilon > 0$, there is some $\delta > 0$ such that for all $u \in [a, z]$ with $|t - u| \leq \delta$, we have $|f(t) - f(u)| \leq \varepsilon$. Prove the following theorems using continuous induction.

- (a) **Connectedness:** *There is no continuous function from $[a, z]$ to the set $\{0, 1\}$.*
- (b) **Intermediate Value Theorem:** *For any continuous function $f : [a, z] \rightarrow \mathbb{R} \setminus \{0\}$, if $f(a) > 0$, then $f(t) > 0$ for all $a \leq t \leq z$.*
- (c) **Extreme Value Theorem:** *Any continuous function $f : [a, z] \rightarrow \mathbb{R}$ attains its maximum value; that is, there is some $t \in [a, z]$ such that $f(t) \geq f(u)$ for all $u \in [a, z]$.*
- ★(d) **The Heine-Borel Theorem:** *The interval $[a, z]$ is compact.*

This one requires some expansion.

- A set $X \subseteq \mathbb{R}$ is **open** if every point in X lies inside an open interval contained in X .
- An **open cover** of $[a, z]$ is a (possibly uncountably infinite) family $\mathcal{U} = \{U_i \mid i \in I\}$ of open sets U_i such that $[a, z] \subseteq \bigcup_{i \in I} U_i$.
- A **subcover** of \mathcal{U} is a subset $\mathcal{V} \subseteq \mathcal{U}$ that is also a cover of $[a, z]$.
- A cover \mathcal{U} is **finite** if it contains a finite number of open sets.
- Finally, a set $X \subseteq \mathbb{R}$ is **compact** if every open cover of X has a finite subcover.

The Heine-Borel theorem is one of the most fundamental results in real analysis, and the proof usually requires several pages. But the continuous-induction proof is shorter than the list of definitions!

Change is certain. Peace is followed by disturbances; departure of evil men by their return. Such recurrences should not constitute occasions for sadness but realities for awareness, so that one may be happy in the interim.

— *I Ching [The Book of Changes]* (c. 1100 BC)

To endure the idea of the recurrence one needs: freedom from morality; new means against the fact of pain (pain conceived as a tool, as the father of pleasure; there is no cumulative consciousness of displeasure); the enjoyment of all kinds of uncertainty, experimentalism, as a counterweight to this extreme fatalism; abolition of the concept of necessity; abolition of the "will"; abolition of "knowledge-in-itself."

— Friedrich Nietzsche *The Will to Power* (1884)
[translated by Walter Kaufmann]

Wil Wheaton: *Embrace the dark side!*

Sheldon: *That's not even from your franchise!*

— "The Wheaton Recurrence", *Bing Bang Theory*, April 12, 2010

Solving Recurrences

1 Introduction

A **recurrence** is a recursive description of a function, or in other words, a description of a function in terms of itself. Like all recursive structures, a recurrence consists of one or more *base cases* and one or more *recursive cases*. Each of these cases is an equation or inequality, with some function value $f(n)$ on the left side. The base cases give explicit values for a (typically finite, typically small) subset of the possible values of n . The recursive cases relate the function value $f(n)$ to function value $f(k)$ for one or more integers $k < n$; typically, each recursive case applies to an infinite number of possible values of n .

For example, the following recurrence (written in two different but standard ways) describes the identity function $f(n) = n$:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ f(n-1) + 1 & \text{otherwise} \end{cases} \quad \begin{matrix} f(0) = 0 \\ f(n) = f(n-1) + 1 \text{ for all } n > 0 \end{matrix}$$

In both presentations, the first line is the only base case, and the second line is the only recursive case. The same function can satisfy *many* different recurrences; for example, both of the following recurrences also describe the identity function:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) & \text{otherwise} \end{cases} \quad f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 2 \cdot f(n/2) & \text{if } n \text{ is even and } n > 0 \\ f(n-1) + 1 & \text{if } n \text{ is odd} \end{cases}$$

We say that a particular function **satisfies** a recurrence, or is the **solution** to a recurrence, if each of the statements in the recurrence is true. Most recurrences—at least, those that we will encounter in this class—have a solution; moreover, if every case of the recurrence is an equation, that solution is unique. Specifically, if we transform the recursive formula into a recursive *algorithm*, the solution to the recurrence is the function computed by that algorithm!

Recurrences arise naturally in the analysis of algorithms, especially recursive algorithms. In many cases, we can express the running time of an algorithm as a recurrence, where the recursive cases of the recurrence correspond exactly to the recursive cases of the algorithm. Recurrences are also useful tools for solving counting problems—How many objects of a particular kind exist?

By itself, a recurrence is not a satisfying description of the running time of an algorithm or a bound on the number of widgets. Instead, we need a **closed-form** solution to the recurrence; this is a *non-recursive* description of a function that satisfies the recurrence. For recurrence *equations*, we sometimes prefer an *exact* closed-form solution, but such a solution may not exist, or may be too complex to be useful. Thus, for most recurrences, especially those arising in algorithm analysis, we are satisfied with an *asymptotic* solution of the form $\Theta(g(n))$, for some explicit (non-recursive) function $g(n)$.

For recursive *inequalities*, we prefer a **tight** solution; this is a function that would still satisfy the recurrence if all the inequalities were replaced with the corresponding equations. Again, exactly tight solutions may not exist, or may be too complex to be useful, in which case we seek either a looser bound or an asymptotic solution of the form $O(g(n))$ or $\Omega(g(n))$.

2 The Ultimate Method: Guess and Confirm

Ultimately, there is only one fail-safe method to solve *any* recurrence:

Guess the answer, and then prove it correct by induction.

Later sections of these notes describe techniques to generate guesses that are guaranteed to be correct, provided you use them correctly. But if you're faced with a recurrence that doesn't seem to fit any of these methods, or if you've forgotten how those techniques work, don't despair! If you guess a closed-form solution and then try to verify your guess inductively, usually either the proof will succeed, in which case you're done, or the proof will fail, in which case *your failure will help you refine your guess*. Where you get your initial guess is utterly irrelevant¹—from a classmate, from a textbook, on the web, from the answer to a different problem, scrawled on a bathroom wall in Siebel, included in a care package from your mom, dictated by the machine elves, whatever. If you can prove that the answer is correct, then it's correct!

2.1 Tower of Hanoi

The classical Tower of Hanoi problem gives us the recurrence $T(n) = 2T(n-1) + 1$ with base case $T(0) = 0$. Just looking at the recurrence we can guess that $T(n)$ is something like 2^n . If we write out the first few values of $T(n)$, we discover that they are each one less than a power of two.

$$T(0) = 0, \quad T(1) = 1, \quad T(2) = 3, \quad T(3) = 7, \quad T(4) = 15, \quad T(5) = 31, \quad T(6) = 63, \quad \dots,$$

It looks like $T(n) = 2^n - 1$ might be the right answer. Let's check.

$$\begin{aligned} T(0) &= 0 = 2^0 - 1 \quad \checkmark \\ T(n) &= 2T(n-1) + 1 \\ &= 2(2^{n-1} - 1) + 1 && \text{[induction hypothesis]} \\ &= 2^n - 1 \quad \checkmark && \text{[algebra]} \end{aligned}$$

¹... except of course during exams, where you aren't supposed to use *any* outside sources

We were right! Hooray, we're done!

Another way we can guess the solution is by *unrolling* the recurrence, by substituting it into itself:

$$\begin{aligned}
 T(n) &= 2T(n-1) + 1 \\
 &= 2(2T(n-2) + 1) + 1 \\
 &= 4T(n-2) + 3 \\
 &= 4(2T(n-3) + 1) + 3 \\
 &= 8T(n-3) + 7 \\
 &= \dots
 \end{aligned}$$

It looks like unrolling the initial Hanoi recurrence k times, for any non-negative integer k , will give us the new recurrence $T(n) = 2^k T(n-k) + (2^k - 1)$. Let's prove this by induction:

$$\begin{aligned}
 T(n) &= 2T(n-1) + 1 \quad \checkmark && [k = 0, \text{ by definition}] \\
 T(n) &= 2^{k-1}T(n-(k-1)) + (2^{k-1} - 1) && [\text{inductive hypothesis}] \\
 &= 2^{k-1}(2T(n-k) + 1) + (2^{k-1} - 1) && [\text{initial recurrence for } T(n-(k-1))] \\
 &= 2^k T(n-k) + (2^k - 1) \quad \checkmark && [\text{algebra}]
 \end{aligned}$$

Our guess was correct! In particular, unrolling the recurrence n times give us the recurrence $T(n) = 2^n T(0) + (2^n - 1)$. Plugging in the base case $T(0) = 0$ give us the closed-form solution $T(n) = 2^n - 1$.

2.2 Fibonacci numbers

Let's try a less trivial example: the Fibonacci numbers $F_n = F_{n-1} + F_{n-2}$ with base cases $F_0 = 0$ and $F_1 = 1$. There is no obvious pattern in the first several values (aside from the recurrence itself), but we can reasonably guess that F_n is exponential in n . Let's try to prove inductively that $F_n \leq \alpha \cdot c^n$ for some constants $\alpha > 0$ and $c > 1$ and see how far we get.

$$\begin{aligned}
 F_n &= F_{n-1} + F_{n-2} \\
 &\leq \alpha \cdot c^{n-1} + \alpha \cdot c^{n-2} && [\text{"induction hypothesis"}] \\
 &\leq \alpha \cdot c^n \text{ ???}
 \end{aligned}$$

The last inequality is satisfied if $c^n \geq c^{n-1} + c^{n-2}$, or more simply, if $c^2 - c - 1 \geq 0$. The smallest value of c that works is $\phi = (1 + \sqrt{5})/2 \approx 1.618034$; the other root of the quadratic equation has smaller absolute value, so we can ignore it.

So we have *most* of an inductive proof that $F_n \leq \alpha \cdot \phi^n$ for *some* constant α . All that we're missing are the base cases, which (we can easily guess) must determine the value of the coefficient α . We quickly compute

$$\frac{F_0}{\phi^0} = \frac{0}{1} = 0 \quad \text{and} \quad \frac{F_1}{\phi^1} = \frac{1}{\phi} \approx 0.618034 > 0,$$

so the base cases of our induction proof are correct as long as $\alpha \geq 1/\phi$. It follows that $F_n \leq \phi^{n-1}$ for all $n \geq 0$.

What about a matching lower bound? Essentially the same inductive proof implies that $F_n \geq \beta \cdot \phi^n$ for some constant β , but the only value of β that works for *all* n is the trivial $\beta = 0$!

We could try to find some lower-order term that makes the base case non-trivial, but an easier approach is to recall that asymptotic $\Omega()$ bounds only have to work for *sufficiently large* n . So let's ignore the trivial base case $F_0 = 0$ and assume that $F_2 = 1$ is a base case instead. Some more easy calculation gives us

$$\frac{F_2}{\phi^2} = \frac{1}{\phi^2} \approx 0.381966 < \frac{1}{\phi}.$$

Thus, the new base cases of our induction proof are correct as long as $\beta \leq 1/\phi^2$, which implies that $F_n \geq \phi^{n-2}$ for all $n \geq 1$.

Putting the upper and lower bounds together, we obtain the tight asymptotic bound $F_n = \Theta(\phi^n)$. It is possible to get a more exact solution by speculatively refining and conforming our current bounds, but it's not easy. Fortunately, if we really need it, we can get an exact solution using the *annihilator* method, which we'll see later in these notes.

2.3 Mergesort

Mergesort is a classical recursive divide-and-conquer algorithm for sorting an array. The algorithm splits the array in half, recursively sorts the two halves, and then merges the two sorted subarrays into the final sorted array.

```

MERGESORT(A[1..n]):
  if (n > 1)
    m ← ⌊n/2⌋
    MERGESORT(A[1..m])
    MERGESORT(A[m+1..n])
    MERGE(A[1..n], m)

```

```

MERGE(A[1..n], m):
  i ← 1; j ← m + 1
  for k ← 1 to n
    if j > n
      B[k] ← A[i]; i ← i + 1
    else if i > m
      B[k] ← A[j]; j ← j + 1
    else if A[i] < A[j]
      B[k] ← A[i]; i ← i + 1
    else
      B[k] ← A[j]; j ← j + 1
  for k ← 1 to n
    A[k] ← B[k]

```

Let $T(n)$ denote the worst-case running time of MERGESORT when the input array has size n . The MERGE subroutine clearly runs in $\Theta(n)$ time, so the function $T(n)$ satisfies the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{otherwise.} \end{cases}$$

For now, let's consider the special case where n is a power of 2; this assumption allows us to take the floors and ceilings out of the recurrence. (We'll see how to deal with the floors and ceilings later; the short version is that they don't matter.)

Because the recurrence itself is given only asymptotically—in terms of $\Theta()$ expressions—we can't hope for anything but an asymptotic solution. So we can safely simplify the recurrence further by removing the Θ 's; any asymptotic solution to the simplified recurrence will also satisfy the original recurrence. (This simplification is actually important for another reason; if we kept the asymptotic expressions, we might be tempted to simplify them inappropriately.)

Our simplified recurrence now looks like this:

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

To guess at a solution, let's try unrolling the recurrence.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(2T(n/4) + n/2) + n \\ &= 4T(n/4) + 2n \\ &= 8T(n/8) + 3n = \dots \end{aligned}$$

It looks like $T(n)$ satisfies the recurrence $T(n) = 2^k T(n/2^k) + kn$ for any positive integer k . Let's verify this by induction.

$$T(n) = 2T(n/2) + n = 2^1 T(n/2^1) + 1 \cdot n \quad \checkmark \quad [k = 1, \text{ given recurrence}]$$

$$T(n) = 2^{k-1} T(n/2^{k-1}) + (k-1)n \quad [\text{inductive hypothesis}]$$

$$= 2^{k-1} (2T(n/2^k) + n/2^{k-1}) + (k-1)n \quad [\text{substitution}]$$

$$= 2^k T(n/2^k) + kn \quad \checkmark \quad [\text{algebra}]$$

Our guess was right! The recurrence becomes trivial when $n/2^k = 1$, or equivalently, when $k = \log_2 n$:

$$T(n) = nT(1) + n \log_2 n = n \log_2 n + n.$$

Finally, we have to put back the Θ 's we stripped off; our final closed-form solution is $T(n) = \Theta(n \log n)$.

2.4 An uglier divide-and-conquer example

Consider the divide-and-conquer recurrence $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$. This doesn't fit into the form required by the Master Theorem (which we'll see below), but it still sort of resembles the Mergesort recurrence—the total size of the subproblems at the first level of recursion is n —so let's *guess* that $T(n) = O(n \log n)$, and then try to prove that our guess is correct. (We could also attack this recurrence by unrolling, but let's see how far just guessing will take us.)

Let's start by trying to prove an upper bound $T(n) \leq a n \lg n$ for all sufficiently large n and some constant a to be determined later:

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\leq \sqrt{n} \cdot a \sqrt{n} \lg \sqrt{n} + n && [\text{induction hypothesis}] \\ &= (a/2)n \lg n + n && [\text{algebra}] \\ &\leq a n \lg n \quad \checkmark && [\text{algebra}] \end{aligned}$$

The last inequality assumes only that $1 \leq (a/2) \log n$, or equivalently, that $n \geq 2^{2/a}$. In other words, the induction proof is correct if n is sufficiently large. So we were right!

But before you break out the champagne, what about the multiplicative constant a ? The proof worked for any constant a , no matter how small. This strongly suggests that our upper bound $T(n) = O(n \log n)$ is not tight. Indeed, if we try to prove a matching lower bound $T(n) \geq b n \log n$ for sufficiently large n , we run into trouble.

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\geq \sqrt{n} \cdot b \sqrt{n} \log \sqrt{n} + n && [\text{induction hypothesis}] \\ &= (b/2)n \log n + n \\ &\not\geq b n \log n \end{aligned}$$

The last inequality would be correct only if $1 > (b/2) \log n$, but that inequality is false for large values of n , no matter which constant b we choose.

Okay, so $\Theta(n \log n)$ is too big. How about $\Theta(n)$? The lower bound is easy to prove directly:

$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n \geq n \checkmark$$

But an inductive proof of the upper bound fails.

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\leq \sqrt{n} \cdot a \sqrt{n} + n && \text{[induction hypothesis]} \\ &= (a + 1)n && \text{[algebra]} \\ &\not\leq an \end{aligned}$$

Hmmm. So what's bigger than n and smaller than $n \lg n$? How about $n \sqrt{\lg n}$?

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \leq \sqrt{n} \cdot a \sqrt{n} \sqrt{\lg \sqrt{n}} + n && \text{[induction hypothesis]} \\ &= (a/\sqrt{2})n \sqrt{\lg n} + n && \text{[algebra]} \\ &\leq an \sqrt{\lg n} \quad \text{for large enough } n \checkmark \end{aligned}$$

Okay, the upper bound checks out; how about the lower bound?

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \geq \sqrt{n} \cdot b \sqrt{n} \sqrt{\lg \sqrt{n}} + n && \text{[induction hypothesis]} \\ &= (b/\sqrt{2})n \sqrt{\lg n} + n && \text{[algebra]} \\ &\not\geq bn \sqrt{\lg n} \end{aligned}$$

No, the last step doesn't work. So $\Theta(n \sqrt{\lg n})$ doesn't work.

Okay... what else is between n and $n \lg n$? How about $n \lg \lg n$?

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \leq \sqrt{n} \cdot a \sqrt{n} \lg \lg \sqrt{n} + n && \text{[induction hypothesis]} \\ &= an \lg \lg n - an + n && \text{[algebra]} \\ &\leq an \lg \lg n \quad \text{if } a \geq 1 \checkmark \end{aligned}$$

Hey look at that! For once, our upper bound proof requires a constraint on the hidden constant a . This is a good indication that we've found the right answer. Let's try the lower bound:

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \geq \sqrt{n} \cdot b \sqrt{n} \lg \lg \sqrt{n} + n && \text{[induction hypothesis]} \\ &= bn \lg \lg n - bn + n && \text{[algebra]} \\ &\geq bn \lg \lg n \quad \text{if } b \leq 1 \checkmark \end{aligned}$$

Hey, it worked! We have most of an inductive proof that $T(n) \leq an \lg \lg n$ for any $a \geq 1$ and most of an inductive proof that $T(n) \geq bn \lg \lg n$ for any $b \leq 1$. Technically, we're still missing the base cases in both proofs, but we can be fairly confident at this point that $T(n) = \Theta(n \log \log n)$.

3 Divide and Conquer Recurrences (Recursion Trees)

Many divide and conquer algorithms give us running-time recurrences of the form

$$\boxed{T(n) = aT(n/b) + f(n)} \tag{1}$$

where a and b are constants and $f(n)$ is some other function. There is a simple and general technique for solving many recurrences in this and similar forms, using a *recursion tree*. The root of the recursion tree is a box containing the value $f(n)$; the root has a children, each of which is the root of a (recursively defined) recursion tree for the function $T(n/b)$.

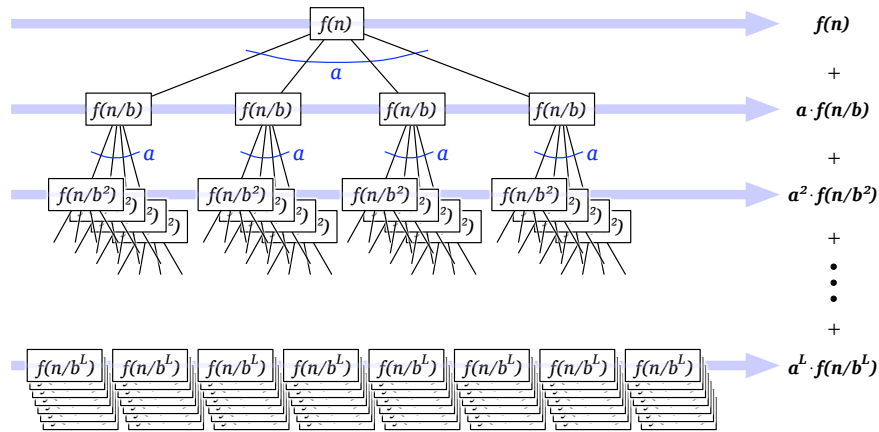
Equivalently, a recursion tree is a complete a -ary tree where each node at depth i contains the value $f(n/b^i)$. The recursion stops when we get to the base case(s) of the recurrence. Because we're only looking for asymptotic bounds, the exact base case doesn't matter; we can safely assume that $T(1) = \Theta(1)$, or even that $T(n) = \Theta(1)$ for all $n \leq 10^{100}$. I'll also assume for simplicity that n is an integral power of b ; we'll see how to avoid this assumption later (but to summarize: it doesn't matter).

Now $T(n)$ is just the sum of all values stored in the recursion tree. For each i , the i th level of the tree contains a^i nodes, each with value $f(n/b^i)$. Thus,

$$T(n) = \sum_{i=0}^L a^i f(n/b^i) \tag{\Sigma}$$

where L is the depth of the recursion tree. We easily see that $L = \log_b n$, because $n/b^L = 1$. The base case $f(1) = \Theta(1)$ implies that the last non-zero term in the summation is $\Theta(a^L) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$.

For most divide-and-conquer recurrences, the level-by-level sum (Σ) is a *geometric series*—each term is a constant factor larger or smaller than the previous term. In this case, only the largest term in the geometric series matters; all of the other terms are swallowed up by the $\Theta(\cdot)$ notation.



A recursion tree for the recurrence $T(n) = aT(n/b) + f(n)$

Here are several examples of the recursion-tree technique in action:

- **Mergesort (simplified):** $T(n) = 2T(n/2) + n$

There are 2^i nodes at level i , each with value $n/2^i$, so every term in the level-by-level sum (Σ) is the same:

$$T(n) = \sum_{i=0}^L n.$$

The recursion tree has $L = \log_2 n$ levels, so $T(n) = \Theta(n \log n)$.

- **Randomized selection:** $T(n) = T(3n/4) + n$

The recursion tree is a single path. The node at depth i has value $(3/4)^i n$, so the level-by-level sum (Σ) is a decreasing geometric series:

$$T(n) = \sum_{i=0}^L (3/4)^i n.$$

This geometric series is dominated by its initial term n , so $T(n) = \Theta(n)$. The recursion tree has $L = \log_{4/3} n$ levels, but so what?

- **Karatsuba's multiplication algorithm:** $T(n) = 3T(n/2) + n$

There are 3^i nodes at depth i , each with value $n/2^i$, so the level-by-level sum (Σ) is an increasing geometric series:

$$T(n) = \sum_{i=0}^L (3/2)^i n.$$

This geometric series is dominated by its final term $(3/2)^L n$. Each leaf contributes 1 to this term; thus, the final term is equal to the number of leaves in the tree! The recursion tree has $L = \log_2 n$ levels, and therefore $3^{\log_2 n} = n^{\log_2 3}$ leaves, so $T(n) = \Theta(n^{\log_2 3})$.

- $T(n) = 2T(n/2) + n/\lg n$

The sum of all the nodes in the i th level is $n/(\lg n - i)$. This implies that the depth of the tree is at most $\lg n - 1$. The level sums are neither constant nor a geometric series, so we just have to evaluate the overall sum directly.

Recall (or if you're seeing this for the first time: Behold!) that the n th *harmonic number* H_n is the sum of the reciprocals of the first n positive integers:

$$H_n := \sum_{i=1}^n \frac{1}{i}$$

It's not hard to show that $H_n = \Theta(\log n)$; in fact, we have the stronger inequalities $\ln(n+1) \leq H_n \leq \ln n + 1$.

$$T(n) = \sum_{i=0}^{\lg n - 1} \frac{n}{\lg n - i} = \sum_{j=1}^{\lg n} \frac{n}{j} = nH_{\lg n} = \Theta(n \lg \lg n)$$

- $T(n) = 4T(n/2) + n \lg n$

There are 4^i nodes at each level i , each with value $(n/2^i) \lg(n/2^i) = (n/2^i)(\lg n - i)$; again, the depth of the tree is at most $\lg n - 1$. We have the following summation:

$$T(n) = \sum_{i=0}^{\lg n - 1} n2^i (\lg n - i)$$

We can simplify this sum by substituting $j = \lg n - i$:

$$T(n) = \sum_{j=i}^{\lg n} n2^{\lg n - j} j = \sum_{j=i}^{\lg n} \frac{n2^j}{2^j} = n^2 \sum_{j=i}^{\lg n} \frac{j}{2^j} = \Theta(n^2)$$

The last step uses the fact that $\sum_{i=1}^{\infty} j/2^j = 2$. Although this is not quite a geometric series, it is still dominated by its largest term.

- **Ugly divide and conquer:** $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$

We solved this recurrence earlier by guessing the right answer and verifying, but we can use recursion trees to get the correct answer directly. The *degree* of the nodes in the recursion tree is no longer constant, so we have to be a bit more careful, but the same basic technique still applies. It's not hard to see that the nodes in any level sum to n . The depth L satisfies the identity $n^{2^{-L}} = 2$ (we can't get all the way down to 1 by taking square roots), so $L = \lg \lg n$ and $T(n) = \Theta(n \lg \lg n)$.

- **Randomized quicksort:** $T(n) = T(3n/4) + T(n/4) + n$

This recurrence isn't in the standard form described earlier, but we can still solve it using recursion trees. Now nodes in the same level of the recursion tree have different values, and different leaves are at different levels. However, the nodes in any *complete* level (that is, above any of the leaves) sum to n . Moreover, every leaf in the recursion tree has depth between $\log_4 n$ and $\log_{4/3} n$. To derive an upper bound, we overestimate $T(n)$ by ignoring the base cases and extending the tree downward to the level of the *deepest* leaf. Similarly, to derive a lower bound, we overestimate $T(n)$ by counting only nodes in the tree up to the level of the *shallowest* leaf. These observations give us the upper and lower bounds $n \log_4 n \leq T(n) \leq n \log_{4/3} n$. Since these bounds differ by only a constant factor, we have $T(n) = \Theta(n \log n)$.

- **Deterministic selection:** $T(n) = T(n/5) + T(7n/10) + n$

Again, we have a lopsided recursion tree. If we look only at complete levels of the tree, we find that the level sums form a descending geometric series $T(n) = n + 9n/10 + 81n/100 + \dots$. We can get an upper bound by ignoring the base cases entirely and growing the tree out to infinity, and we can get a lower bound by only counting nodes in complete levels. Either way, the geometric series is dominated by its largest term, so $T(n) = \Theta(n)$.

- **Randomized search trees:** $T(n) = \frac{1}{4}T(n/4) + \frac{3}{4}T(3n/4) + 1$

This looks like a divide-and-conquer recurrence, but what does it mean to have a quarter of a child? The right approach is to imagine that each node in the recursion tree has a *weight* in addition to its value. Alternately, we get a standard recursion tree again if we add a second real parameter to the recurrence, defining $T(n) = T(n, 1)$, where

$$T(n, \alpha) = T(n/4, \alpha/4) + T(3n/4, 3\alpha/4) + \alpha.$$

In each complete level of the tree, the (weighted) node values sum to exactly 1. The leaves of the recursion tree are at different levels, but all between $\log_4 n$ and $\log_{4/3} n$. So we have upper and lower bounds $\log_4 n \leq T(n) \leq \log_{4/3} n$, which differ by only a constant factor, so $T(n) = \Theta(\log n)$.

- **Ham-sandwich trees:** $T(n) = T(n/2) + T(n/4) + 1$

Again, we have a lopsided recursion tree. If we only look at complete levels, we find that the level sums form an *ascending* geometric series $T(n) = 1 + 2 + 4 + \dots$, so the solution

is dominated by the number of leaves. The recursion tree has $\log_4 n$ complete levels, so there are more than $2^{\log_4 n} = n^{\log_4 2} = \sqrt{n}$; on the other hand, every leaf has depth at most $\log_2 n$, so the total number of leaves is at most $2^{\log_2 n} = n$. Unfortunately, the crude bounds $\sqrt{n} \ll T(n) \ll n$ are the best we can derive using the techniques we know so far!

The following theorem completely describes the solution for any divide-and-conquer recurrence in the ‘standard form’ $T(n) = aT(n/b) + f(n)$, where a and b are constants and $f(n)$ is a polynomial. This theorem allows us to bypass recursion trees for “standard” divide-and-conquer recurrences, but many people (including Jeff) find it harder to even remember the statement of the theorem than to use the more powerful and general recursion-tree technique. Your mileage may vary.

The Master Theorem. *The recurrence $T(n) = aT(n/b) + f(n)$ can be solved as follows.*

- If $a f(n/b) = \kappa f(n)$ for some constant $\kappa < 1$, then $T(n) = \Theta(f(n))$.
- If $a f(n/b) = K f(n)$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$.
- If $a f(n/b) = f(n)$, then $T(n) = \Theta(f(n) \log_b n)$.
- If none of these three cases apply, you’re on your own.

Proof: If $f(n)$ is a constant factor larger than $a f(b/n)$, then by induction, the level sums define a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$.

If $f(n)$ is a constant factor smaller than $a f(b/n)$, then by induction, the level sums define an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $\Theta(n^{\log_b a})$.

Finally, if $a f(b/n) = f(n)$, then by induction, each of the $L + 1$ terms in the sum is equal to $f(n)$, and the recursion tree has depth $L = \Theta(\log_b n)$. \square

*4 The Nuclear Bomb

Finally, let me describe *without proof* a powerful generalization of the recursion tree method, first published by Lebanese researchers Mohamad Akra and Louay Bazzi in 1998. Consider a general divide-and-conquer recurrence of the form

$$T(n) = \sum_{i=1}^k a_i T(n/b_i) + f(n),$$

where k is a constant, $a_i > 0$ and $b_i > 1$ are constants for all i , and $f(n) = \Omega(n^c)$ and $f(n) = O(n^d)$ for some constants $0 < c \leq d$. (As usual, we assume the standard base case $T(\Theta(1)) = \Theta(1)$.) Akra and Bazzi prove that this recurrence has the closed-form asymptotic solution

$$T(n) = \Theta\left(n^\rho \left(1 + \int_1^n \frac{f(u)}{u^{\rho+1}} du\right)\right),$$

where ρ is the unique real solution to the equation

$$\sum_{i=1}^k a_i / b_i^\rho = 1.$$

In particular, the Akra-Bazzi theorem immediately implies the following form of the Master Theorem:

$$T(n) = aT(n/b) + n^c \implies T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } c < \log_b a - \varepsilon \\ \Theta(n^c \log n) & \text{if } c = \log_b a \\ \Theta(n^c) & \text{if } c > \log_b a + \varepsilon \end{cases}$$

The Akra-Bazzi theorem does not require that the parameters a_i and b_i are integers, or even rationals; on the other hand, even when all parameters are integers, the characteristic equation $\sum_i a_i/b_i^\rho = 1$ may have no analytical solution.

Here are a few examples of recurrences that are difficult (or impossible) for recursion trees, but have easy solutions using the Akra-Bazzi theorem.

- **Randomized quicksort:** $T(n) = T(3n/4) + T(n/4) + n$

The equation $(3/4)^\rho + (1/4)^\rho = 1$ has the unique solution $\rho = 1$, and therefore

$$T(n) = \Theta\left(n\left(1 + \int_1^n \frac{1}{u} du\right)\right) = O(n \log n).$$

- **Deterministic selection:** $T(n) = T(n/5) + T(7n/10) + n$

The equation $(1/5)^\rho + (7/10)^\rho = 1$ has no analytical solution. However, we easily observe that $(1/5)^x + (7/10)^x$ is a decreasing function of x , and therefore $0 < \rho < 1$. Thus, we have

$$\int_1^n \frac{f(u)}{u^{\rho+1}} du = \int_1^n u^{-\rho} du = \frac{u^{1-\rho}}{1-\rho} \Big|_{u=1}^n = \frac{n^{1-\rho} - 1}{1-\rho} = \Theta(n^{1-\rho}),$$

and therefore

$$T(n) = \Theta(n^\rho \cdot (1 + \Theta(n^{1-\rho}))) = \Theta(n).$$

(A bit of numerical computation gives the approximate value $\rho \approx 0.83978$, but why bother?)

- **Randomized search trees:** $T(n) = \frac{1}{4}T(n/4) + \frac{3}{4}T(3n/4) + 1$

The equation $\frac{1}{4}(\frac{1}{4})^\rho + \frac{3}{4}(\frac{3}{4})^\rho = 1$ has the unique solution $\rho = 0$, and therefore

$$T(n) = \Theta\left(1 + \int_1^n \frac{1}{u} du\right) = \Theta(\log n).$$

- **Ham-sandwich trees:** $T(n) = T(n/2) + T(n/4) + 1$. Recall that we could only prove the very weak bounds $\sqrt{n} \ll T(n) \ll n$ using recursion trees. The equation $(1/2)^\rho + (1/4)^\rho = 1$ has the unique solution $\rho = \log_2((1 + \sqrt{5})/2) \approx 0.69424$, which can be obtained by setting $x = 2^\rho$ and solving for x . Thus, we have

$$\int_1^n \frac{1}{u^{\rho+1}} du = \frac{u^{-\rho}}{-\rho} \Big|_{u=1}^n = \frac{1 - n^{-\rho}}{\rho} = \Theta(1)$$

and therefore

$$T(n) = \Theta(n^\rho (1 + \Theta(1))) = \Theta(n^{\lg \phi}).$$

The Akra-Bazzi method is that it can solve *almost* any divide-and-conquer recurrence with just a few lines of calculation. (There are a few nasty exceptions like $T(n) = \sqrt{n}T(\sqrt{n}) + n$ where we have to fall back on recursion trees.) On the other hand, the steps appear to be magic, which makes the method hard to remember, and for most divide-and-conquer recurrences that arise in practice, there are much simpler solution techniques.

*5 Linear Recurrences (Annihilators)

Another common class of recurrences, called *linear* recurrences, arises in the context of recursive backtracking algorithms and counting problems. These recurrences express each function value $f(n)$ as a *linear* combination of a small number of nearby values $f(n-1), f(n-2), f(n-3), \dots$. The Fibonacci recurrence is a typical example:

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

It turns out that the solution to *any* linear recurrence is a simple combination of polynomial and exponential functions in n . For example, we can verify by induction that the linear recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \text{ or } n = 2 \\ 3T(n-1) - 8T(n-2) + 4T(n-3) & \text{otherwise} \end{cases}$$

has the closed-form solution $T(n) = (n-3)2^n + 4$. First we check the base cases:

$$T(0) = (0-3)2^0 + 4 = 1 \quad \checkmark$$

$$T(1) = (1-3)2^1 + 4 = 0 \quad \checkmark$$

$$T(2) = (2-3)2^2 + 4 = 0 \quad \checkmark$$

And now the recursive case:

$$\begin{aligned} T(n) &= 3T(n-1) - 8T(n-2) + 4T(n-3) \\ &= 3((n-4)2^{n-1} + 4) - 8((n-5)2^{n-2} + 4) + 4((n-6)2^{n-3} + 4) \\ &= \left(\frac{3}{2} - \frac{8}{4} + \frac{4}{8}\right)n \cdot 2^n - \left(\frac{12}{2} - \frac{40}{4} + \frac{24}{8}\right)2^n + (2-8+4) \cdot 4 \\ &= (n-3) \cdot 2^n + 4 \quad \checkmark \end{aligned}$$

But how could we have possibly come up with that solution? In this section, I'll describe a general method for solving linear recurrences that's arguably easier than the induction proof!

5.1 Operators

Our technique for solving linear recurrences relies on the theory of *operators*. Operators are higher-order functions, which take one or more functions as input and produce different functions as output. For example, your first two semesters of calculus focus almost exclusively on the *differential* and *integral* operators $\frac{d}{dx}$ and $\int dx$. All the operators we will need are combinations of three elementary building blocks:

- **Sum:** $(f + g)(n) := f(n) + g(n)$
- **Scale:** $(\alpha \cdot f)(n) := \alpha \cdot (f(n))$
- **Shift:** $(Ef)(n) := f(n + 1)$

The shift and scale operators are *linear*, which means they can be distributed over sums; for example, for any functions f , g , and h , we have $E(f - 3(g - h)) = Ef + (-3)Eg + 3Eh$.

We can combine these building blocks to obtain more complex *compound* operators. For example, the compound operator $E - 2$ is defined by setting $(E - 2)f := Ef + (-2)f$ for any function f . We can also apply the shift operator twice: $(E(Ef))(n) = f(n + 2)$; we write usually E^2f as a synonym for $E(Ef)$. More generally, for any positive integer k , the operator E^k shifts its argument k times: $E^k f(n) = f(n + k)$. Similarly, $(E - 2)^2$ is shorthand for the operator $(E - 2)(E - 2)$, which applies $(E - 2)$ twice.

For example, here are the results of applying different operators to the exponential function $f(n) = 2^n$:

$$\begin{aligned}
 2f(n) &= 2 \cdot 2^n = 2^{n+1} \\
 3f(n) &= 3 \cdot 2^n \\
 Ef(n) &= 2^{n+1} \\
 E^2f(n) &= 2^{n+2} \\
 (E - 2)f(n) &= Ef(n) - 2f(n) = 2^{n+1} - 2^{n+1} = 0 \\
 (E^2 - 1)f(n) &= E^2f(n) - f(n) = 2^{n+2} - 2^n = 3 \cdot 2^n
 \end{aligned}$$

These compound operators can be manipulated exactly as though they were polynomials over the “variable” E . In particular, we can factor compound operators into “products” of simpler operators, which can be applied in any order. For example, the compound operators $E^2 - 3E + 2$ and $(E - 1)(E - 2)$ are equivalent:

$$\text{Let } g(n) := (E - 2)f(n) = f(n + 1) - 2f(n).$$

$$\begin{aligned}
 \text{Then } (E - 1)(E - 2)f(n) &= (E - 1)g(n) \\
 &= g(n + 1) - g(n) \\
 &= (f(n + 2) - 2f(n + 1)) - (f(n + 1) - 2f(n)) \\
 &= f(n + 2) - 3f(n + 1) + 2f(n) \\
 &= (E^2 - 3E + 2)f(n). \quad \checkmark
 \end{aligned}$$

It is an easy exercise to confirm that $E^2 - 3E + 2$ is also equivalent to the operator $(E - 2)(E - 1)$.

The following table summarizes everything we need to remember about operators.

Operator	Definition
addition	$(f + g)(n) := f(n) + g(n)$
subtraction	$(f - g)(n) := f(n) - g(n)$
multiplication	$(\alpha \cdot f)(n) := \alpha \cdot (f(n))$
shift	$Ef(n) := f(n + 1)$
k -fold shift	$E^k f(n) := f(n + k)$
composition	$(X + Y)f := Xf + Yf$ $(X - Y)f := Xf - Yf$ $XYf := X(Yf) = Y(Xf)$
distribution	$X(f + g) = Xf + Xg$

5.2 Annihilators

An **annihilator** of a function f is any nontrivial operator that transforms f into the zero function. (We can trivially annihilate any function by multiplying it by zero, so as a technical matter, we do not consider the zero operator to be an annihilator.) Every compound operator we consider annihilates a specific class of functions; conversely, every function composed of polynomial and exponential functions has a unique (minimal) annihilator.

We have already seen that the operator $(E - 2)$ annihilates the function 2^n . It's not hard to see that the operator $(E - c)$ annihilates the function $\alpha \cdot c^n$, for any constants c and α . More generally, the operator $(E - c)$ annihilates the function a^n if and only if $c = a$:

$$(E - c)a^n = E a^n - c \cdot a^n = a^{n+1} - c \cdot a^n = (a - c)a^n.$$

Thus, $(E - 2)$ is essentially the *only* annihilator of the function 2^n .

What about the function $2^n + 3^n$? The operator $(E - 2)$ annihilates the function 2^n , but leaves the function 3^n unchanged. Similarly, $(E - 3)$ annihilates 3^n while *negating* the function 2^n . But if we apply *both* operators, we annihilate both terms:

$$\begin{aligned} (E - 2)(2^n + 3^n) &= E(2^n + 3^n) - 2(2^n + 3^n) \\ &= (2^{n+1} + 3^{n+1}) - (2^{n+1} + 2 \cdot 3^n) = 3^n \\ \implies (E - 3)(E - 2)(2^n + 3^n) &= (E - 3)3^n = 0 \end{aligned}$$

In general, for any integers $a \neq b$, the operator $(E - a)(E - b) = (E - b)(E - a) = (E^2 - (a + b)E + ab)$ annihilates any function of the form $\alpha a^n + \beta b^n$, but nothing else.

What about the operator $(E - a)(E - a) = (E - a)^2$? It turns out that this operator annihilates all functions of the form $(\alpha n + \beta)a^n$:

$$\begin{aligned} (E - a)((\alpha n + \beta)a^n) &= (\alpha(n + 1) + \beta)a^{n+1} - a(\alpha n + \beta)a^n \\ &= \alpha a^{n+1} \\ \implies (E - a)^2((\alpha n + \beta)a^n) &= (E - a)(\alpha a^{n+1}) = 0 \end{aligned}$$

More generally, the operator $(E - a)^d$ annihilates all functions of the form $p(n) \cdot a^n$, where $p(n)$ is a polynomial of degree at most $d - 1$. For example, $(E - 1)^3$ annihilates any polynomial of degree at most 2.

The following table summarizes everything we need to remember about annihilators.

Operator	Functions annihilated
$E - 1$	α
$E - a$	αa^n
$(E - a)(E - b)$	$\alpha a^n + \beta b^n$ [if $a \neq b$]
$(E - a_0)(E - a_1) \cdots (E - a_k)$	$\sum_{i=0}^k \alpha_i a_i^n$ [if a_i distinct]
$(E - 1)^2$	$\alpha n + \beta$
$(E - a)^2$	$(\alpha n + \beta)a^n$
$(E - a)^2(E - b)$	$(\alpha n + \beta)a^b + \gamma b^n$ [if $a \neq b$]
$(E - a)^d$	$(\sum_{i=0}^{d-1} \alpha_i n^i)a^n$
If X annihilates f , then X also annihilates $E f$.	
If X annihilates both f and g , then X also annihilates $f \pm g$.	
If X annihilates f , then X also annihilates αf , for any constant α .	
If X annihilates f and Y annihilates g , then XY annihilates $f \pm g$.	

5.3 Annihilating Recurrences

Given a linear recurrence for a function, it's easy to extract an annihilator for that function. For many recurrences, we only need to rewrite the recurrence in operator notation. Once we have an annihilator, we can factor it into operators of the form $(E - c)$; the table on the previous page then gives us a generic solution with some unknown coefficients. If we are given explicit base cases, we can determine the coefficients by examining a few small cases; in general, this involves solving a small system of linear equations. If the base cases are not specified, the generic solution almost always gives us an asymptotic solution. Here is the technique step by step:

1. Write the recurrence in operator form
2. Extract an annihilator for the recurrence
3. Factor the annihilator (if necessary)
4. Extract the *generic solution* from the annihilator
5. Solve for coefficients using base cases (if known)

Here are several examples of the technique in action:

- $r(n) = 5r(n - 1)$, where $r(0) = 3$.

1. We can write the recurrence in operator form as follows:

$$r(n) = 5r(n - 1) \implies r(n + 1) - 5r(n) = 0 \implies (E - 5)r(n) = 0.$$

2. We immediately see that $(E - 5)$ annihilates the function $r(n)$.
3. The annihilator $(E - 5)$ is already factored.
4. Consulting the annihilator table on the previous page, we find the generic solution $r(n) = \alpha 5^n$ for some constant α .
5. The base case $r(0) = 3$ implies that $\alpha = 3$.

We conclude that $r(n) = 3 \cdot 5^n$. We can easily verify this closed-form solution by induction:

$$\begin{aligned} r(0) &= 3 \cdot 5^0 = 3 \quad \checkmark && \text{[definition]} \\ r(n) &= 5r(n - 1) && \text{[definition]} \\ &= 5 \cdot (3 \cdot 5^{n-1}) && \text{[induction hypothesis]} \\ &= 5^n \cdot 3 \quad \checkmark && \text{[algebra]} \end{aligned}$$

- **Fibonacci numbers:** $F(n) = F(n - 1) + F(n - 2)$, where $F(0) = 0$ and $F(1) = 1$.

1. We can rewrite the recurrence as $(E^2 - E - 1)F(n) = 0$.
2. The operator $E^2 - E - 1$ clearly annihilates $F(n)$.
3. The quadratic formula implies that the annihilator $E^2 - E - 1$ factors into $(E - \phi)(E - \hat{\phi})$, where $\phi = (1 + \sqrt{5})/2 \approx 1.618034$ is the golden ratio and $\hat{\phi} = (1 - \sqrt{5})/2 = 1 - \phi = -1/\phi \approx -0.618034$.
4. The annihilator implies that $F(n) = \alpha \phi^n + \hat{\alpha} \hat{\phi}^n$ for some unknown constants α and $\hat{\alpha}$.

5. The base cases give us two equations in two unknowns:

$$F(0) = 0 = \alpha + \hat{\alpha}$$

$$F(1) = 1 = \alpha\phi + \hat{\alpha}\hat{\phi}$$

Solving this system of equations gives us $\alpha = 1/(2\phi - 1) = 1/\sqrt{5}$ and $\hat{\alpha} = -1/\sqrt{5}$.

We conclude with the following exact closed form for the n th Fibonacci number:

$$F(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

With all the square roots in this formula, it's quite amazing that Fibonacci numbers are integers. However, if we do all the math correctly, all the square roots cancel out when i is an integer. (In fact, this is pretty easy to prove using the binomial theorem.)

- **Towers of Hanoi:** $T(n) = 2T(n-1) + 1$, where $T(0) = 0$. This is our first example of a *non-homogeneous* recurrence, which means the recurrence has one or more non-recursive terms.

1. We can rewrite the recurrence as $(E - 2)T(n) = 1$.
2. The operator $(E - 2)$ doesn't quite annihilate the function; it leaves a *residue* of 1. But we can annihilate the residue by applying the operator $(E - 1)$. Thus, the compound operator $(E - 1)(E - 2)$ annihilates the function.
3. The annihilator is already factored.
4. The annihilator table gives us the generic solution $T(n) = \alpha 2^n + \beta$ for some unknown constants α and β .
5. The base cases give us $T(0) = 0 = \alpha 2^0 + \beta$ and $T(1) = 1 = \alpha 2^1 + \beta$. Solving this system of equations, we find that $\alpha = 1$ and $\beta = -1$.

We conclude that $T(n) = 2^n - 1$.

For the remaining examples, I won't explicitly enumerate the steps in the solution.

- **Height-balanced trees:** $H(n) = H(n-1) + H(n-2) + 1$, where $H(-1) = 0$ and $H(0) = 1$. (Yes, we're starting at -1 instead of 0. So what?)

We can rewrite the recurrence as $(E^2 - E - 1)H = 1$. The residue 1 is annihilated by $(E - 1)$, so the compound operator $(E - 1)(E^2 - E - 1)$ annihilates the recurrence. This operator factors into $(E - 1)(E - \phi)(E - \hat{\phi})$, where $\phi = (1 + \sqrt{5})/2$ and $\hat{\phi} = (1 - \sqrt{5})/2$. Thus, we get the generic solution $H(n) = \alpha \cdot \phi^n + \beta + \gamma \cdot \hat{\phi}^n$, for some unknown constants α, β, γ that satisfy the following system of equations:

$$H(-1) = 0 = \alpha\phi^{-1} + \beta + \gamma\hat{\phi}^{-1} = \alpha/\phi + \beta - \gamma/\hat{\phi}$$

$$H(0) = 1 = \alpha\phi^0 + \beta + \gamma\hat{\phi}^0 = \alpha + \beta + \gamma$$

$$H(1) = 2 = \alpha\phi^1 + \beta + \gamma\hat{\phi}^1 = \alpha\phi + \beta + \gamma\hat{\phi}$$

Solving this system (using Cramer's rule or Gaussian elimination), we find that $\alpha = (\sqrt{5} + 2)/\sqrt{5}$, $\beta = -1$, and $\gamma = (\sqrt{5} - 2)/\sqrt{5}$. We conclude that

$$H(n) = \frac{\sqrt{5} + 2}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n - 1 + \frac{\sqrt{5} - 2}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^n.$$

- $T(n) = 3T(n-1) - 8T(n-2) + 4T(n-3)$, where $T(0) = 1$, $T(1) = 0$, and $T(2) = 0$. This was our original example of a linear recurrence.

We can rewrite the recurrence as $(E^3 - 3E^2 + 8E - 4)T = 0$, so we immediately have an annihilator $E^3 - 3E^2 + 8E - 4$. Using high-school algebra, we can factor the annihilator into $(E - 2)^2(E - 1)$, which implies the generic solution $T(n) = \alpha n 2^n + \beta 2^n + \gamma$. The constants α , β , and γ are determined by the base cases:

$$\begin{aligned} T(0) &= 1 = \alpha \cdot 0 \cdot 2^0 + \beta 2^0 + \gamma = \beta + \gamma \\ T(1) &= 0 = \alpha \cdot 1 \cdot 2^1 + \beta 2^1 + \gamma = 2\alpha + 2\beta + \gamma \\ T(2) &= 0 = \alpha \cdot 2 \cdot 2^2 + \beta 2^2 + \gamma = 8\alpha + 4\beta + \gamma \end{aligned}$$

Solving this system of equations, we find that $\alpha = 1$, $\beta = -3$, and $\gamma = 4$, so $T(n) = (n - 3)2^n + 4$.

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

We can rewrite the recurrence as $(E^2 - E - 2)T(n) = E^2(2^n - n^2)$. Notice that we had to shift up the non-recursive parts of the recurrence when we expressed it in this form. The operator $(E - 2)(E - 1)^3$ annihilates the residue $2^n - n^2$, and therefore also annihilates the shifted residue $E^2(2^n - n^2)$. Thus, the operator $(E - 2)(E - 1)^3(E^2 - E - 2)$ annihilates the entire recurrence. We can factor the quadratic factor into $(E - 2)(E + 1)$, so the annihilator factors into $(E - 2)^2(E - 1)^3(E + 1)$. So the generic solution is $T(n) = \alpha n 2^n + \beta 2^n + \gamma n^2 + \delta n + \varepsilon + \eta(-1)^n$. The coefficients α , β , γ , δ , ε , η satisfy a system of six equations determined by the first six function values $T(0)$ through $T(5)$. For almost² every set of base cases, we have $\alpha \neq 0$, which implies that $T(n) = \Theta(n 2^n)$.

For a more detailed explanation of the annihilator method, see George Lueker, Some techniques for solving recurrences, *ACM Computing Surveys* 12(4):419-436, 1980.

6 Transformations

Sometimes we encounter recurrences that don't fit the structures required for recursion trees or annihilators. In many of those cases, we can transform the recurrence into a more familiar form, by defining a new function in terms of the one we want to solve. There are many different kinds of transformations, but these three are probably the most useful:

- **Domain transformation:** Define a new function $S(n) = T(f(n))$ with a simpler recurrence, for some simple function f .
- **Range transformation:** Define a new function $S(n) = f(T(n))$ with a simpler recurrence, for some simple function f .
- **Difference transformation:** Simplify the recurrence for $T(n)$ by considering the difference function $\Delta T(n) = T(n) - T(n-1)$.

Here are some examples of these transformations in action.

²In fact, the only possible solutions with $\alpha = 0$ have the form $-2^{n-1} - n^2/2 - 5n/2 + \eta(-1)^n$ for some constant η .

- **Unsimplified Mergesort:** $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$

When n is a power of 2, we can simplify the mergesort recurrence to $T(n) = 2T(n/2) + \Theta(n)$, which has the solution $T(n) = \Theta(n \log n)$. Unfortunately, for other values of n , this simplified recurrence is incorrect. When n is odd, then the recurrence calls for us to sort a fractional number of elements! Worse yet, if n is not a power of 2, we will *never* reach the base case $T(1) = 1$.

So we really need to solve the original recurrence. We have no hope of getting an *exact* solution, even if we ignore the $\Theta()$ in the recurrence; the floors and ceilings will eventually kill us. But we can derive a tight asymptotic solution using a domain transformation—we can rewrite the function $T(n)$ as a nested function $S(f(n))$, where $f(n)$ is a simple function and the function $S()$ has a simpler recurrence.

First let's overestimate the time bound, once by pretending that the two subproblem sizes are equal, and again to eliminate the ceiling:

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

Now we define a new function $S(n) = T(n + \alpha)$, where α is an unknown constant, chosen so that $S(n)$ satisfies the Master-Theorem-ready recurrence $S(n) \leq 2S(n/2) + O(n)$. To figure out the correct value of α , we compare two versions of the recurrence for the function $T(n + \alpha)$:

$$\begin{aligned} S(n) \leq 2S(n/2) + O(n) &\implies T(n + \alpha) \leq 2T(n/2 + \alpha) + O(n) \\ T(n) \leq 2T(n/2 + 1) + n &\implies T(n + \alpha) \leq 2T((n + \alpha)/2 + 1) + n + \alpha \end{aligned}$$

For these two recurrences to be equal, we need $n/2 + \alpha = (n + \alpha)/2 + 1$, which implies that $\alpha = 2$. The Master Theorem now tells us that $S(n) = O(n \log n)$, so

$$T(n) = S(n - 2) = O((n - 2) \log(n - 2)) = O(n \log n).$$

A similar argument implies the matching lower bound $T(n) = \Omega(n \log n)$. So $T(n) = \Theta(n \log n)$ after all, just as though we had ignored the floors and ceilings from the beginning!

Domain transformations are useful for removing floors, ceilings, and lower order terms from the arguments of any recurrence that otherwise looks like it ought to fit either the Master Theorem or the recursion tree method. But now that we know this, we don't need to bother grinding through the actual gory details!

- **Ham-Sandwich Trees:** $T(n) = T(n/2) + T(n/4) + 1$

As we saw earlier, the recursion tree method only gives us the uselessly loose bounds $\sqrt{n} \ll T(n) \ll n$ for this recurrence, and the recurrence is in the wrong form for annihilators. The authors who discovered ham-sandwich trees (Yes, this is a real data structure!) solved this recurrence by guessing the solution and giving a complicated induction proof. We got a tight solution using the Akra-Bazzi method, but who can remember that?

In fact, a simple domain transformation allows us to solve the recurrence in just a few lines. We define a new function $t(k) = T(2^k)$, which satisfies the simpler linear recurrence $t(k) = t(k - 1) + t(k - 2) + 1$. This recurrence should immediately remind you of Fibonacci

numbers. Sure enough, the annihilator method implies the solution $t(k) = \Theta(\phi^k)$, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio. We conclude that

$$T(n) = t(\lg n) = \Theta(\phi^{\lg n}) = \Theta(n^{\lg \phi}) \approx \Theta(n^{0.69424}).$$

This is the same solution we obtained earlier using the Akra-Bazzi theorem.

Many other divide-and-conquer recurrences can be similarly transformed into linear recurrences and then solved with annihilators. Consider once more the simplified mergesort recurrence $T(n) = 2T(n/2) + n$. The function $t(k) = T(2^k)$ satisfies the recurrence $t(k) = 2t(k-1) + 2^k$. The annihilator method gives us the generic solution $t(k) = \Theta(k \cdot 2^k)$, which implies that $T(n) = t(\lg n) = \Theta(n \log n)$, just as we expected.

On the other hand, for some recurrences like $T(n) = T(n/3) + T(2n/3) + n$, the recursion tree method gives an easy solution, but there's no way to transform the recurrence into a form where we can apply the annihilator method directly.³

- **Random Binary Search Trees:** $T(n) = \frac{1}{4}T(n/4) + \frac{3}{4}T(3n/4) + 1$

This looks like a divide-and-conquer recurrence, so we might be tempted to apply recursion trees, but what does it mean to have a quarter of a child? If we're not comfortable with weighted recursion trees (or the Akra-Bazzi theorem), we can instead apply the following range transformation. The function $U(n) = n \cdot T(n)$ satisfies the more palatable recurrence $U(n) = U(n/4) + U(3n/4) + n$. As we've already seen, recursion trees imply that $U(n) = \Theta(n \log n)$, which immediately implies that $T(n) = \Theta(\log n)$.

- **Randomized Quicksort:** $T(n) = \frac{2}{n} \sum_{k=0}^{n-1} T(k) + n$

This is our first example of a *full history* recurrence; each function value $T(n)$ is defined in terms of *all* previous function values $T(k)$ with $k < n$. Before we can apply any of our existing techniques, we need to convert this recurrence into an equivalent *limited history* form by shifting and subtracting away common terms. To make this step slightly easier, we first multiply both sides of the recurrence by n to get rid of the fractions.

$$\begin{aligned}
 n \cdot T(n) &= 2 \sum_{k=0}^{n-1} T(k) + n^2 && \text{[multiply both sides by } n\text{]} \\
 (n-1) \cdot T(n-1) &= 2 \sum_{k=0}^{n-2} T(k) + (n-1)^2 && \text{[shift]} \\
 nT(n) - (n-1)T(n-1) &= 2T(n-1) + 2n - 1 && \text{[subtract]} \\
 T(n) &= \frac{n+1}{n} T(n-1) + 2 - \frac{1}{n} && \text{[simplify]}
 \end{aligned}$$

³However, we can still get a solution via functional transformations as follows. The function $t(k) = T((3/2)^k)$ satisfies the recurrence $t(n) = t(n-1) + t(n-\lambda) + (3/2)^k$, where $\lambda = \log_{3/2} 3 = 2.709511 \dots$. The *characteristic function* for this recurrence is $(r^\lambda - r^{\lambda-1} - 1)(r - 3/2)$, which has a double root at $r = 3/2$ and nowhere else. Thus, $t(k) = \Theta(k(3/2)^k)$, which implies that $T(n) = t(\log_{3/2} n) = \Theta(n \log n)$. This line of reasoning is the core of the Akra-Bazzi method.

We can solve this limited-history recurrence using another functional transformation. We define a new function $t(n) = T(n)/(n + 1)$, which satisfies the simpler recurrence

$$t(n) = t(n - 1) + \frac{2}{n + 1} - \frac{1}{n(n + 1)},$$

which we can easily unroll into a summation. If we only want an asymptotic solution, we can simplify the final recurrence to $t(n) = t(n - 1) + \Theta(1/n)$, which unrolls into a very familiar summation:

$$t(n) = \sum_{i=1}^n \Theta(1/i) = \Theta(H_n) = \Theta(\log n).$$

Finally, substituting $T(n) = (n + 1)t(n)$ gives us a solution to the original recurrence: $T(n) = \Theta(n \log n)$.

Exercises

- For each of the following recurrences, first **guess** an exact closed-form solution, and then prove your guess is correct. You are free to use any method you want to make your guess—unrolling the recurrence, writing out the first several values, induction proof template, recursion trees, annihilators, transformations, ‘It looks like that other one’, whatever—but please describe your method. All functions are from the non-negative integers to the reals. If it simplifies your solutions, express them in terms of Fibonacci numbers F_n , harmonic numbers H_n , binomial coefficients $\binom{n}{k}$, factorials $n!$, and/or the floor and ceiling functions $\lfloor x \rfloor$ and $\lceil x \rceil$.
 - $A(n) = A(n - 1) + 1$, where $A(0) = 0$.
 - $B(n) = \begin{cases} 0 & \text{if } n < 5 \\ B(n - 5) + 2 & \text{otherwise} \end{cases}$
 - $C(n) = C(n - 1) + 2n - 1$, where $C(0) = 0$.
 - $D(n) = D(n - 1) + \binom{n}{2}$, where $D(0) = 0$.
 - $E(n) = E(n - 1) + 2^n$, where $E(0) = 0$.
 - $F(n) = 3 \cdot F(n - 1)$, where $F(0) = 1$.
 - $G(n) = \frac{G(n-1)}{G(n-2)}$, where $G(0) = 1$ and $G(1) = 2$. [Hint: This is easier than it looks.]
 - $H(n) = H(n - 1) + 1/n$, where $H(0) = 0$.
 - $I(n) = I(n - 2) + 3/n$, where $I(0) = I(1) = 0$. [Hint: Consider even and odd n separately.]
 - $J(n) = J(n - 1)^2$, where $J(0) = 2$.
 - $K(n) = K(\lfloor n/2 \rfloor) + 1$, where $K(0) = 0$.
 - $L(n) = L(n - 1) + L(n - 2)$, where $L(0) = 2$ and $L(1) = 1$. [Hint: Write the solution in terms of Fibonacci numbers.]
 - $M(n) = M(n - 1) \cdot M(n - 2)$, where $M(0) = 2$ and $M(1) = 1$. [Hint: Write the solution in terms of Fibonacci numbers.]

$$(n) N(n) = 1 + \sum_{k=1}^n (N(k-1) + N(n-k)), \text{ where } N(0) = 1.$$

$$(p) P(n) = \sum_{k=0}^{n-1} (k \cdot P(k-1)), \text{ where } P(0) = 1.$$

$$(q) Q(n) = \frac{1}{2-Q(n-1)}, \text{ where } Q(0) = 0.$$

$$(r) R(n) = \max_{1 \leq k \leq n} \{R(k-1) + R(n-k) + n\}$$

$$(s) S(n) = \max_{1 \leq k \leq n} \{S(k-1) + S(n-k) + 1\}$$

$$(t) T(n) = \min_{1 \leq k \leq n} \{T(k-1) + T(n-k) + n\}$$

$$(u) U(n) = \min_{1 \leq k \leq n} \{U(k-1) + U(n-k) + 1\}$$

$$(v) V(n) = \max_{n/3 \leq k \leq 2n/3} \{V(k-1) + V(n-k) + n\}$$

2. Use recursion trees or the Akra-Bazzi theorem to solve each of the following recurrences.

$$(a) A(n) = 2A(n/4) + \sqrt{n}$$

$$(b) B(n) = 2B(n/4) + n$$

$$(c) C(n) = 2C(n/4) + n^2$$

$$(d) D(n) = 3D(n/3) + \sqrt{n}$$

$$(e) E(n) = 3E(n/3) + n$$

$$(f) F(n) = 3F(n/3) + n^2$$

$$(g) G(n) = 4G(n/2) + \sqrt{n}$$

$$(h) H(n) = 4H(n/2) + n$$

$$(i) I(n) = 4I(n/2) + n^2$$

$$(j) J(n) = J(n/2) + J(n/3) + J(n/6) + n$$

$$(k) K(n) = K(n/2) + K(n/3) + K(n/6) + n^2$$

$$(l) L(n) = L(n/15) + L(n/10) + 2L(n/6) + \sqrt{n}$$

$$*(m) M(n) = 2M(n/3) + 2M(2n/3) + n$$

$$(n) N(n) = \sqrt{2n}N(\sqrt{2n}) + \sqrt{n}$$

$$(p) P(n) = \sqrt{2n}P(\sqrt{2n}) + n$$

$$(q) Q(n) = \sqrt{2n}Q(\sqrt{2n}) + n^2$$

$$(r) R(n) = R(n-3) + 8^n \text{ — Don't use annihilators!}$$

$$(s) S(n) = 2S(n-2) + 4^n \text{ — Don't use annihilators!}$$

$$(t) T(n) = 4T(n-1) + 2^n \text{ — Don't use annihilators!}$$

3. Make up a bunch of linear recurrences and then solve them using annihilators.
4. Solve the following recurrences, using any tricks at your disposal.

(a) $T(n) = \sum_{i=1}^{\lg n} T(n/2^i) + n$ *[Hint: Assume n is a power of 2.]*

(b) More to come. . .