

Lecture Note 2

Long-Form Proofs

draft of September 29, 2026, by [N. Young](#)

© 2018–2026 [Neal E. Young](#). Text: CC BY-NC-SA 4.0; code: MIT; except where noted. See <https://github.com/nealeyoung/algs101/blob/main/LICENSE>.

Contents

2.1	Introduction	1
2.2	The long-form proof format	3
2.2.1	There are four kinds of proof steps	3
2.2.2	There are four kinds of blocks	4
2.2.3	Checklist and cheat sheet	7
2.3	Proofs about algorithms (a few tips, including use of invariants)	10
2.4	How to write long-form proofs in LaTeX	11
2.5	External sources on proofs for algorithms	13
2.6	Exercises	13
2.6.1	Exercises with solutions	13
2.6.2	Exercises without solutions	18

2.1 Introduction

Why algorithms? Information surrounds us, but much of it is out of our reach. Here’s a simple example. Suppose you want to know the prime factors of the following 186-digit number: 11679847981112819759721399310592745791658017001955007325132913837831330495881519756453703742878526148\8414688067442512219413748768010657572575384986457405973985247465176041951676954461208131403777. Like every positive integer, this one has a unique set of prime factors. In principle, you could find the factors by exhaustive search—after all, there are only finitely many possibilities. But there are about 10^{93} possible factors to check, so that would take too long, even for a computer. In fact, no algorithm is known that can factor large integers in any reasonable amount of time (e.g., the lifetime of the universe).¹ The information is (in some sense) “there”, but it is out of reach, because we don’t know an efficient algorithm to compute it.

This kind of barrier to knowledge is common. For example, Google has a database with details of all the roads in the United States. From that data, Google Maps can quickly compute, say, a shortest possible route from New York City to San Francisco. But what about a shortest possible route that *visits all 50 U.S. capitols in any order*? Nobody knows how to find such a route with a reasonable amount of computation time.

Whether or not we can learn a desired piece of information often depends on whether there is an *efficient algorithm to solve the underlying computational problem*. Further, for many important and easily stated problems, whether the problem has an efficient algorithm *is not currently known*. New algorithms can lead to advances in science, medicine, biology, business, and so on.

Why proofs? Algorithms are important, but can be difficult to reason about. For example, consider the following simple algorithm:

```

input: integer  $N \geq 1$ 
1. while  $N > 1$ :
2.1. if  $N$  is even:  $N \leftarrow N/2$ 
2.2. else:  $N \leftarrow 3 * N + 1$ 
3. print “done”
    
```

In 1937, L. Collatz conjectured that, for every input, this algorithm eventually halts. But even after years of study, nobody knows for sure!² Many simple algorithms are hard to reason about. So, when searching for an algorithm to solve a particular problem, we limit our search to algorithms that we *do* know some way to reason about—that is, algorithms that we can prove do work. For this reason, *proofs drive algorithm design*. (More on this later.)

Do your own pull-ups! Textbooks and lectures generally present algorithms that are already very well studied. The author and reader (or instructor and student) generally *already know* that the algorithms are correct. So textbooks and lectures focus on conveying the main underlying ideas. Listening to such a lecture or reading such a text, it is not hard to get an intuitive sense of the algorithm and why it works. And in many algorithms courses, this is all you would do.

But this experience does not teach you how to design your own algorithms. To find an algorithm that works, you typically must search through a variety of ideas. Ideas that seem promising often

¹No such algorithm is known—indeed, standard encryption schemes rely on this fact. But nobody knows for sure that there isn’t one. (We do know that, if a so-called quantum computer can be built, it would be able to factor efficiently.)

²See https://en.wikipedia.org/wiki/Collatz_conjecture.

fail, sometimes for deviously subtle reasons. To know that an idea works, you must also find an airtight line of reasoning as to why. That is, the search for an algorithm proceeds hand-in-hand along with the search for a proof that it works. Doing this is much harder than following a proof in a lecture or textbook. The difference is like the difference between watching someone else do pull-ups, versus doing them yourself. If you want to build the strength to do pull-ups, it doesn't help to just watch someone else do them. You have to do them yourself! So, what does this entail?

Mind the gap! (Check the details.) When you are developing a new proof, you are usually not quite sure it is right. With experience, you will learn that your intuition has blind spots. Because of this, you have to check the details carefully by writing out your reasoning. Any gap in your *explanation* can hide a gap or error in your *reasoning*. So, when developing a new proof, it is important to explain (and check!) your reasoning step by step, in more detail than you see in a typical textbook or lecture. Also, the more clearly you communicate your reasoning, the more a reader can understand about the line of reasoning you have in mind, so the more useful their feedback can be.

Exploring the logical consequences of an assumption. Suppose we don't know that a given statement is true. We can still assume for the moment that the statement is true, and explore the logical consequences if it is. This kind of imaginative thinking can help figure out what is actually true. It is so common that keeping track of assumptions is one of the main organizational challenges in presenting a proof. Conventionally, this is done via four standard proof patterns, *proof by contradiction*, reasoning by *cases*, and when proving *if then* and *for all* statements. When creating complex proofs, it is useful to know these forms like the back of your hand, so you can keep track of your assumptions and organize your reasoning clearly.

These notes: *long-form proofs*. To learn to develop proofs with enough detail, and to learn the four standard forms for managing assumptions, we'll use *long-form proofs*. A long-form proof is given as a sequence of numbered steps, where each step is small, has a clear meaning, and follows easily from previous steps. The numbering allows each step to be identified unambiguously. (This is useful during the proof to make it easy to tell the reader which previous steps a particular step follows from. It also makes communicating about the proof, including giving feedback on it, easier.)

In long-form proofs, assumptions are explicitly managed by using what we call *blocks*. Each block introduces an assumption, which then holds for the scope of the block. Blocks nest like code blocks in computer programs. There are four block types—one for each of the four standard proof forms mentioned above. Section 2.2 describes the long-form proof format in detail.

Basic terms: *problem, algorithm, proof* To design algorithms, and to reason and communicate clearly about them, it is crucial that the language we use is clear, with a precise and unambiguous meaning. To this end, we use the language of mathematics.

A computational *problem* is defined by specifying what the possible inputs are, and what the output should be for each input. (This means that defining a problem is just defining a function.) We cast our problems in terms of standard mathematical objects such as numbers, strings, trees, graphs, sequence, sets and so on.

As described previously in [Lecture Note 1](#), to define an *algorithm*, one specifies exactly how, given any input, the algorithm processes the input to produce the output. The definition should be mathematically precise, and written to be clear and accessible to any competent reader. For this reason, use words, not code. Any competent programmer should be able to translate the definition

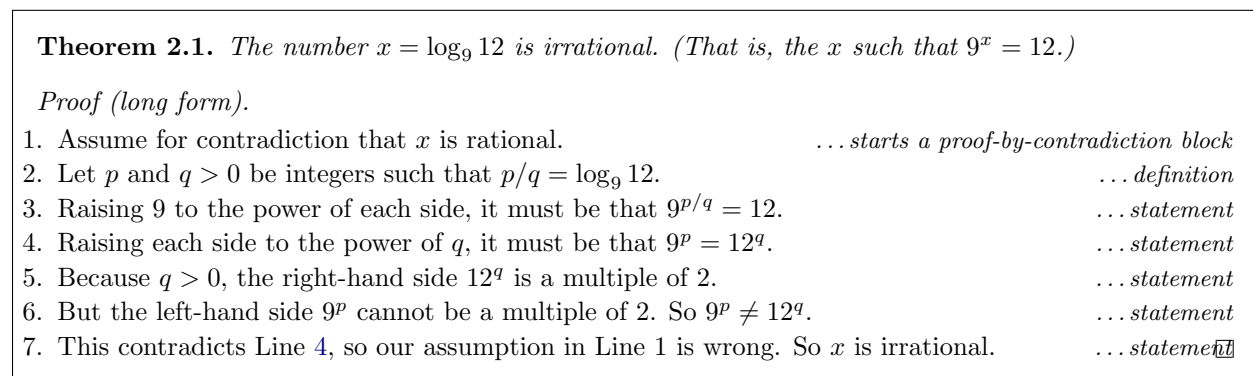


Figure 2.1: A long-form proof with one *proof by contradiction* block.

into code. An algorithm is **correct** (for a given problem) if, for every input, the algorithm produces a correct output.

A **proof** of a fact is a clear explanation, typically in writing, of a complete and correct line of reasoning that establishes that the fact must be true. Typically, the proof is preceded by a clear statement of the fact being proven, e.g. stated as a theorem or lemma. Before you try to write a proof for a fact, you should already have in mind a line of reasoning as to why the fact must be true.³ The reasoning must be complete (covering all cases) and without logical gaps. The goal of the written proof is to communicate your line of reasoning to the reader. The fundamental guiding principle is this: *from what you write, the reader should be able to easily reconstruct your reasoning in detail, and to easily verify that it is correct.*

2.2 The long-form proof format

The long-form proof format uses a particular syntactical form that helps make the details and logical structure of the proof particularly clear. This section defines the format and gives examples. Section 2.6.1 has more examples.

2.2.1 There are four kinds of proof steps

Each long-form proof is given as a sequence of numbered steps. Each step takes up about one line on the page. Figure 2.1 shows an example. There are just four kinds of steps:

1. A *definition* of a term or variable (for example, Step 2 in the figure defines p and q),
2. A *statement of fact*, with a justification (Steps 3–7),
3. A *comment* to guide the reader (the proof above has none),
4. The first line of a *block* (such as in Step 1); per Section 2.2.2, blocks are used to manage assumptions. Each block takes one of four forms:
 - (i) a proof by contradiction,
 - (ii) a proof of an “if then” statement,
 - (iii) a proof by cases,
 - (iv) a proof of a “for all” statement.
5. Some combination of the above (e.g. a definition and a statement; do this rarely, though).

³But of course it is also useful to write partial proofs to guide your thinking.

A **definition** gives a name to some quantity (beware: it must be clear that the quantity does exist). The definition should be mathematically precise, with no ambiguity. The definition holds throughout the remainder of the block in which it occurs. (Just as declaring a variable in a computer program makes the variable available for the rest of the code block in which the declaration occurs.)

Each **statement of fact** states some fact, along with a short justification. *The given fact must follow directly from the facts established in previous steps*, and your anticipated reader should be able to easily see why this is the case. The level of detail you give thus depends on your anticipated reader. For example, the proof in Figure 2.1 assumes the reader is familiar with the definitions of “rational” and “irrational”, and with the basic properties of integers, and it uses such properties without explaining why they hold. When writing up exercises for these notes, you should assume the reader is another student who knows the material in the notes, but who doesn’t yet know anything about the line of reasoning you have in mind.

When you are starting out, if you’re not sure how much detail to give, err on the side of giving more. For the reader, it’s easier to skip unnecessary detail than to figure out what you might have had in mind when details are missing. And gaps (missing details) often hide errors.

As necessary, the statement of fact is accompanied by a justification to guide the reader, along with explicit references to any previous steps that it depends on. (By default, each step depends on the previous step, so if that’s the only dependence you don’t have to mention it.)

2.2.2 There are four kinds of blocks

Long-form proofs can have multiple, possibly nested **blocks**. (The proofs in Figures 2.1 and 2.2 each have just one, though, the main outer block.) Blocks are used to keep track of assumptions made during the proof. *The first line of each block introduces an assumption* (in some cases only implicitly). The assumption is then in effect until the end of the block. That is, within the block we use the assumption as if it were a true fact. We don’t need to know whether the assumption is actually true. Rather, within the block we are imagining that it is true, and exploring what the logical consequences would be if it were. This kind of exploration lets us figure out facts that hold even without any assumptions.

Blocks can be used in long-form proofs in (only) the following four specific ways:

1. Blocks for proof by contradiction. The proof in Figure 2.1 has just one block—its main block, consisting of Steps 1–7. The assumption that starts the block is that the given quantity $\log_9 12$ is rational. The block explores the logical consequences of that assumption, showing that, if the assumption is true, then $9^p = 12^q$ and $9^p \neq 12^q$ (for some p and q). That is, a statement and its negation both hold. This is called a *contradiction*. Since it cannot be true, we deduce that our original assumption (which implies it) must be false. That is, $\log_9 12$ cannot be rational. It must be irrational.

This is called *proof by contradiction*. To do proof by contradiction, use a block whose first line explicitly assumes the opposite of what you want to prove. To signal the reader, in this first line, explicitly use a phrase including the word “contradiction” such as “*Assume for contradiction that...*” Then, in the body of the block, somehow deduce a contradiction—a statement and its negation. If you can do that, you can conclude that the assumption that started the block must be false, and the negation of that assumption must be true. (By the way, proof by contradiction is usually just a convenient way to organize a proof. That is, it is usually possible to give an equivalent “direct” proof without using proof by contradiction.)

2. Blocks to prove “if then” statements. The logical meaning of a statement of the form “if A then B” is that, if A is true, then B must also be true. You can prove such a statement without

Theorem 2.2. *Let $n = 31415923576480135764801$. If $\log_7 n$ is rational, then it is an integer.*

Proof (long form).

1. Assume that $\log_7 n$ is rational. ... starts an “if then” block
2. Let p and q be integers with $q > 0$ such that $p/q = \log_7 n$ definition
3. Raising 7 to the power of each side gives $7^{p/q} = n$ statement
4. Raising each side to the power of q gives $7^p = n^q$.
5. As each integer has a unique prime factorization, this implies that n is a power of 7. ⋮
6. That is, $n = 7^i$ for some integer $i > 0$.
7. Substituting that into the equation in Step 4, by algebra we have

$$\begin{array}{ll}
 7^p = n^q = (7^i)^q = 7^{iq} & \\
 p = iq & \text{(taking the } \log_7 \text{ of each side)} \\
 p/q = i & \text{(dividing each side by } q > 0\text{)} \\
 \log_7 n = i & \text{(using } \log_7 n = p/q\text{).}
 \end{array}$$

8. It follows that $\log_7 n$ is an integer (if it is rational). □

Figure 2.2: A long-form proof with one *if-then* block.

knowing whether or not A is true. Start a block by assuming A, then, under that assumption, show that B must also hold. If you can do that, then B is a logical consequence of A, so, if A is true, then B must also be true. That is, “if A then B”.

Figure 2.2 shows an example. For some particular n , it shows that if $\log_7 n$ is rational, then $\log_7 n$ is an integer. It doesn’t prove whether $\log_7 n$ is rational or not. Rather, it shows that, if we assume that $\log_7 n$ is rational, it follows by logic and algebra that $\log_7 n$ must in fact be an integer.

3. Blocks to prove “for all” statements. A “for all” statement is a statement of the following form, given some set N and predicate Q :

“for all n in N , the predicate $Q(n)$ holds”.

To prove such a statement, use a “for all” block. Start the block with a line such as ⁴

“... 1. Consider an arbitrary n in N .”

This asks the reader to consider a generic element of the set N , referred to by the name n . To let the reader know that you are starting a “for all” block, make sure you include the qualifier “arbitrary” in the first step. Within the block, assuming only that n is in N (and knowing nothing else about n) prove that $Q(n)$ must hold. Suppose you can indeed prove that. Because your reasoning could be applied generically to any particular element of N , it must be that $Q(n)$ holds for every particular n . That is, you can deduce the following statement (outside the block!): *for all n in N , the predicate $Q(n)$ holds*.

For example, note that the proof in Figure 2.2 doesn’t use the specific properties of the particular integer n defined there. So, we can apply the reasoning in that proof to *any* particular positive integer n , to show that if $\log_7 n$ is rational, then it is an integer. In other words, *for all positive integers n , if $\log_7 n$ is rational, then it is an integer*. Figure 2.3 shows the corresponding long-form proof. It uses a “for all” block as described above. The logical structure of the proof mirrors the logical structure of the statement: the main, outer block (Steps 1 through 3) is a “for all” block; the inner block (Block 2) is an “if then” block.

⁴For “for all” blocks, the underlying assumption (not explicitly stated) is that N is non-empty, so n is well-defined.

Theorem 2.3. *For all positive integers n , if $\log_7 n$ is rational, then it is an integer.*

Proof (long form).

1. Consider an arbitrary positive integer n starts a “for all” block
- 2.1. Assume that $\log_7 n$ is rational. ... starts an “if then” block
- 2.2. Let p and q be integers with $q > 0$ such that $p/q = \log_7 n$ definition
- 2.3. Raising 7 to the power of each side gives $7^{p/q} = n$ statement
- 2.4. Raising each side to the power of q gives $7^p = n^q$.
- 2.5. As each integer has a unique prime factorization, this implies that n is a power of 7. ⋮
- 2.6. That is, $n = 7^i$ for some integer $i > 0$.
- 2.7. Substituting that into the equation in Step 2.4, by algebra we have

$$\begin{aligned}
 7^p &= n^q = (7^i)^q = 7^{iq} \\
 p &= iq && \text{(taking the } \log_7 \text{ of each side)} \\
 p/q &= i && \text{(dividing each side by } q > 0) \\
 \log_7 n &= i && \text{(using } \log_7 n = p/q).
 \end{aligned}$$

- 2.8. It follows that $\log_7 n$ is an integer.
3. By Block 2, if $\log_7 n$ is rational, then $\log_7 n$ is an integer. □

Figure 2.3: This proof has a *for-all* block containing an *if-then* block.

4. Blocks for proof by cases. The fourth and final type of block is used for case analysis. Suppose you know that “ C_1 or C_2 ” holds, and you want to show that some proposition X holds. (You may not know which of C_1 or C_2 holds, but you know at least one has to.)

First, consider the case that C_1 holds. Start a block with the assumption that C_1 holds. Under that assumption (within that block) show somehow that X holds. Then you know that, if C_1 holds, then X holds.

Then consider the other case, that C_2 holds. Start a block with the assumption that C_2 holds. Under that assumption (within that block), show somehow that X holds. Then you know that, if C_2 holds, then X holds.

Finally, since you know that C_1 or C_2 holds, and you know that in either case X must hold, you can conclude (without any assumptions) that X holds.

This is the principle of proof by cases. It extends naturally to situations with more than two cases, as long as you know that at least one of case’s assumption must hold. When you start a block for a proof by cases, signal it by starting the statement with a label, such as “Case 1. ...”. You can also use a phrase such as “*Consider the case that ...*” to signal the reader. If necessary, use a comment to the reader before the cases to help the reader understand their structure.

Figure 2.4 shows an example. Here’s the context for the theorem shown there. Assume that, for every pair of people in the world, either the two have met, or they have not. For any given group G of people, if every pair in G has met, call G a *club*. If every pair in G has not met, call G a group of *strangers*. The theorem says that in every group of six, there must be either a group of three who form a club, or a group of three strangers. The proof uses two cases, each with two subcases.

It is interesting to note that in that proof, the part that proves that G is good is constructive. That is, it yields an efficient algorithm, which, given any group G of six people, finds three that are a club or are strangers. (Choose any person x . If x has met at least three of the others, check any pair in those three have met, and so on...)

Theorem 2.4. *Within every group of six people, there exists a club of three or a group of three strangers.*

Proof (long form).

- | | |
|---|------------------------------|
| 1. Let G be an arbitrary group of six people. | ... starts a “for all” block |
| 2. Say that G is <i>good</i> if F includes a club of 3 or a group of 3 strangers. | ... definition |
| 3. Our goal is to prove G is good. | ... comment |
| 4. Fix x to be any one of the six people in G . | ... definition |
| 5. There two cases, each with two subcases: | ... comment |
| 6.1. <u>Case 1:</u> Suppose that, among the five other than x , at least three have met x . | ... start case block |
| 6.2.1. <u>Subcase 1.1:</u> Suppose that no two among those three have met. | ... start case block |
| 6.2.2. Then those three people are a group of strangers, so G is good. | ... statement |
| 6.3.1. <u>Subcase 1.2:</u> Otherwise some pair among those three have met. | ... start case block |
| 6.3.2. That pair, and x , have all met each other, so form a club of three, so G is good. | ... statement |
| 6.4. One of the Subcases 1.1 or 1.2 must hold, and G is good in each, so G is good (in Case 1). | ... stmt |
| 7.1. <u>Case 2:</u> Among the five other than x , at least three have <i>not</i> met x . | ... start case block |
| 7.2.1. <u>Subcase 2.1:</u> Suppose that every pair among those three have met. | ... start case block |
| 7.2.2. Those three are a club, so G is good. | ... statement |
| 7.3.1. <u>Subcase 2.2:</u> Otherwise some pair among those three have not met. | ... start case block |
| 7.3.2. That pair, and x , have all not met, so form a group of three strangers, so G is good. | ... statement |
| 7.4. One of the Subcases 2.1 or 2.2 must hold, and G is good in each, so G is good (in Case 2). | ... stmt |
| 8. One of the Cases 1 or 2 must hold, and G is good in each, so G is good. | ... statement |

Figure 2.4: An example of using blocks for cases. (See the definitions in the text for context.)

Remark. In doing proof by cases, if the reasoning for a case is the same as for a previously proved case, you can just point out that the reasoning is the same, instead of repeating it.

Follow each inner block by its conclusion. For any inner block, the deduction that follows from the reasoning in the block should be stated in the first line following the end of the block (or sequence of blocks for proof by cases). This is true for all of the four block types. For example, in Figure 2.3, just after Block 2 ends, Step 3 states the conclusion implied by the reasoning in the block. In Figure 2.4, just after Blocks 6 and 7 end, Step 8 states the conclusion implied by the reasoning in those blocks.

The main (outer) block should prove the desired fact. The conclusion that follows from the reasoning in the main (outer) block should be the fact that the proof claims to prove.

Lemmas. A lemma is like a “small” theorem. In these notes, we use lemmas within the context of the main proof, to make it modular and therefore easier to understand (just as subroutines are used to make programs more modular). See Problem 2.5 in Section 2.6 for an example.

Section 2.6 has many other examples of long-form proofs for study.

2.2.3 Checklist and cheat sheet

After you’ve written a proof, put it aside for a while, then come back with fresh eyes to check it. Put yourself in the place of the reader, who does not yet know anything about the line of reasoning you have in mind. The high-level criteria is that *from reading just what you wrote, the anticipated reader will be able to easily reconstruct and verify your entire line of reasoning.*

1. The proof is preceded by the statement to be proved (typically as a theorem or lemma).
2. Each step in the proof is either a statement, definition, comment, or the start of a block.
3. Each definition and statement is mathematically precise and well-defined,
4. Each assumption is introduced only in the first line of some block.
5. Each block is of one of the four block types: contradiction, if then, for all, or cases. (A definition does not need its own block!)
6. The line following each inner block (or a series of case blocks) states the fact that follows from the reasoning in the block, according to the block's type.
7. The statement you are trying to prove follows from the main (outer) block.
8. Each name (variable) is defined *before* it is used.
9. Each variable is referred to only within the block in which it is defined.
10. Each statement follows directly by the justification given in that step from the previous steps, either by domain-specific reasoning, basic logic, or via one of the four block structures. The reader should be able to see why easily, without having to think very much, and without having to guess at what you had in mind.
11. The proof doesn't assume that the reader already knows anything specific about the line of reasoning.
12. You *can* assume the reader knows basic facts about the basic structures you are reasoning with (e.g., numbers, trees, graphs, sets, algorithms). You *should* use standard mathematical structures: numbers, tuples, sets, sequences, permutations, graphs, trees, paths, matrices, etc.
13. Resort to formal notation (rather than precise English) only when you can't say what you need to say precisely in words alone. You can define a new term or notation for a concept that you introduce, but avoid this if possible as it increases the cognitive load on the reader.
14. Most importantly, from just reading what you write, the reader should be able to easily reconstruct, and hopefully verify, your entire line of reasoning.

Figure 2.5: A checklist that will help you avoid common errors in proofs.

The main things to check are (i) that each statement and definition is mathematically precise, (ii) that the overall logical structure is correct, (iii) the reasoning is complete (covers all cases), and (iv) that there are no large gaps. Consider each step, and make sure that if asked you could easily explain in more detail exactly what that step says and why it holds. Checking a proof carefully and skeptically is a learned skill.

Figure 2.5 gives checklist that will help you avoid common errors. Figure 2.6 gives a cheat sheet summarizing how to use the four block types.

<u>Proof by cases, given that “A or B” holds</u> ... <u>Case 1.</u> Assume A. ... <i>within this scope, A holds</i> <i>somehow prove C</i> ... - C <u>Case 2.</u> Assume B. ... <i>within this scope, B holds</i> <i>somehow prove C</i> ... - C - By Cases 1 and 2, C holds.	<u>Proof by contradiction</u> ... - Assume for contradiction that B holds. ... <i>within this scope, B holds</i> <i>prove any contradiction</i> ... - contradiction - By Block 2, B is false.
<u>“If then” blocks</u> ... - Assume A. ... <i>within this scope, A holds</i> <i>somehow prove B</i> ... - B - By Block 2, if A, then B. <i>Alternatively, show the contrapositive:</i> “if not B, then not A”.	<u>“For all” blocks</u> ... - Let x be an arbitrary element of S.^a ... <i>within this scope, x is defined</i> <i>somehow prove $P(x)$</i> ... $P(x)$ - By Block 2, every x in S has property $P(x)$. ^aThe definition in Step 2.1 assumes S is non-empty. This assumption is implicitly introduced by the block. For other types of definitions, blocks are not needed.
	<u>Basic logic:</u> (“$\equiv$” means “is equivalent to”) “if A then B” $\equiv$ “if not B then not A” “if A then B” $\equiv$ “B or not A” “not (A and B)” $\equiv$ “(not A) or (not B)” “not (A or B)” $\equiv$ “(not A) and (not B)” “not (not A)” $\equiv$ “A” “not $\forall x \in S. P(x)$” $\equiv$ “$\exists x \in S. \text{not } P(x)$” “not $\exists x \in S. P(x)$” $\equiv$ “$\forall x \in S. \text{not } P(x)$”

Each block introduces an assumption that lasts for the scope of the block. The general principle is that *whenever we want to explore the consequences of some assumption, we begin a new block, with a step that makes that assumption.* The block delimits the scope of that assumption. There are four block types: “proof by contradiction”, “if then”, “for all”, and “cases”. (For the “for all” block, what is the underlying assumption that necessitates a block? That S is not empty!)

Figure 2.6: A cheat sheet summarizing the four block types and how to use them.

2.3 Proofs about algorithms (a few tips, including use of invariants)

Proofs about *algorithms* can be particularly challenging. One reason is that during the execution of an algorithm its state changes. Try to follow these tips when doing proofs about algorithms:

1. When you refer to the state of the algorithm (such as the value of some variable) make sure that it makes clear *exactly what point in time* you have in mind.
2. When you define an object in your proof (i.e., introduce a name for some quantity), by convention, the named object never changes after that—it is static. Be aware of this convention and try to follow it.

It is okay to violate this convention when you are referring in the proof to the name of variable in the algorithm. But (i) if you do this, make sure your wording signals it clearly. For example, say something like this:

1. Consider any execution of the algorithm.
2. Consider the value of the algorithm's variable x just before the first iteration.
- ⋮

And (ii) whenever you refer to the value of x , make sure you make clear exactly *when* during the execution you mean.

3. Learn to use *invariants*. An invariant is a property that is maintained throughout the execution of the algorithm. Invariants are a basic proof technique for reasoning about iterative algorithms. Exercise 2.4 in Section 2.6 gives an example. We'll see more throughout these notes.
4. As a rule, reasoning about processes is generally harder than reasoning about static objects. So, when you can present your proof as reasoning only about static objects, do that. For example, in reasoning about a static object such as a set or a spanning tree, don't present the proof in a way that implicitly or explicitly introduces some new process (e.g. one that builds the object step by step).

Exercise 2.4 in Section 2.6 illustrates the first three points above.

```

\begin{theorem}\label{thm: bad cases}
  For every real number, its square is positive.
  That is,  $\forall x \in \mathbb{R}, x^2 > 0$ .
\end{theorem}

\begin{longFormProof}
  \step Let  $x \in \mathbb{R}$  be an arbitrary real number.

  \begin{block}
    \step \underline{Case 1.} Suppose  $x$  is positive ( $x > 0$ ).
    \step Then  $x^2$  is a product of two positive numbers, so  $x^2$  is positive.
  \end{block}

  \begin{block}
    \step \underline{Case 2.} Suppose  $x$  is negative ( $x < 0$ ).
    \step Then  $x^2$  is a product of two negative numbers, so  $x^2$  is positive.
  \end{block}

  \step In either Case 1 or 2,  $x^2$  is positive, so  $x^2$  is positive.
\end{longFormProof}

```

Figure 2.7: How to use the `theorem`, `longFormProof`, and `block` environments, and the `\step` command. See Exercise 2.7 for the result.

2.4 How to write long-form proofs in LaTeX

Instructions for getting started with LaTeX are in [Appendix B of the book](#). This section assumes you've read them. This one is about how to use LaTeX (and the macros in the homework templates) to write long-form proofs.

Within the body of the document, you can state a theorem or lemma using the `theorem` or `lemma` environments. LaTeX will automatically number the theorem or lemma for you. Follow your theorem or lemma statement by a long-form proof, using the `longFormProof` environment. With that, start each step with a `\step` command. To introduce an inner block, use the `block` environment. Figure 2.7 illustrates these environments and the `\step` command.

Optionally, you can use a `\label` command to record the theorem or lemma number, using a name of your choice. Then you can recall the number by using the `\ref` command. Specifically, `\ref{thm: bad cases}` will reproduce the number (assuming you use the same name as you gave to `\label`). You can use `\label` and `\ref` to label and refer to any type of numbered object in your document, such as a proof step or block. When the numbers change, LaTeX automatically updates the references. (It may take two iterations of running LaTeX for the references to fully update.)

To introduce a lemma inside a long form proof, use the `lemma` environment to state the lemma, then use the `longFormSubProof` environment for the proof of the lemma. An example is shown in Figure 2.8.

```

\begin{theorem}
  Any finite, connected graph  $G$  that has Property  $A$  also has Property  $B$ .
\end{theorem}

\begin{longFormProof}[First long-form proof.]
  \step Let  $G$  be an arbitrary finite, connected graph having Property  $A$ .

  \step Before we give the proof, we prove the following utility lemma:

  \begin{lemma}\label{lemma: high low}
    Let  $v$  be any vertex in  $G$  that has a neighbor  $x$  with larger weight
    ( $w(x) > w(v)$ ). Then  $v$  has a neighbor  $y$  with a smaller weight ( $w(y) < w(v)$ ).
  \end{lemma}

  \begin{longFormSubProof}[Proof of lemma.]

    \step ...

  \end{longFormSubProof}

  \begin{block} \label{A}
    \step Now assume for contradiction that  $G$  does not have Property  $B$ .

    \step ...

    \step So, by Lemma~\ref{lemma: high low} ...

    \step This is a contradiction (as  $G$  is finite).
  \end{block}

  \step By Block~\ref{A},  $G$  must have Property  $B$ .

\end{longFormProof}

```

Figure 2.8: Making a lemma with the `lemma` and `longFormSubProof` environments. See the solution to Problem 2.5 in Section 2.6 for the output.

2.5 External sources on proofs for algorithms

- [LLM](#) Chapter 1 (proofs by contradiction, cases, proofs of “if then” statements, but nothing about “for all” statements), Chapter 6 (invariants), Chapter 5 (induction).

Those topics are also covered in the videos of the first three lectures here:

<https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010/video-lectures/>

- [MF](#) Chapters 2 (Logic), 3 (Proofs), and 11 (Algorithms)
- More practice proof problems (marked “P”) are at <https://www.cs.ucr.edu/~neal/puzzles>.
- *Book of Proof*, by Hammack, Parts II and III (about 70 pages with examples from discrete math). <http://www.people.vcu.edu/~rhammack/BookOfProof/>
- Long-form proofs are similar in spirit to Lamport’s “structured proofs,” which you can read about here: <https://www.microsoft.com/en-us/research/publication/write-21st-century-proof/>.

2.6 Exercises

2.6.1 Exercises with solutions

Exercise 2.1. (*Long-form proof*) Convert the following short-form proof into long form.

Theorem 2.5. *There are infinitely many primes.*

The proof will use that, if a is a common factor of integers b and c , then a is a factor of $c - b$.

Proof (short form). Suppose for contradiction that there are only finitely many primes: $p_1, p_2, \dots, p_n$. Consider the product P of all the primes. Let $p_i > 1$ be some particular prime in the prime factorization of $P + 1$. Then p_i is a factor of both P and $P + 1$, so it is a factor of $(P + 1) - P = 1$, which is impossible. $\square$

Solution. Here’s the proof in long form.

Proof (long form).

1. Suppose for contradiction that there are only finitely many primes.
2. Let $p_1, p_2, \dots, p_n$ be a list of all the primes (larger than 1).
3. Let $P = \prod_{i=1}^n p_i$ be their product. (Note $P > 1$ so $n \geq 1$.)
4. Fix p_i to be any particular prime in the prime factorization of $P + 1$.
5. Then p_i evenly divides P , and also evenly divides $P + 1$.
6. So p_i evenly divides their difference, $(P + 1) - P$, which equals 1.
7. But this is a contradiction, as 1 has no factor other than 1. $\square$

Step 4 defines p_i to be a specific number (one that we know exists at that point) for use later in the proof. Step 4 does not open a new “for all” block—we would only use a “for all” block there to prove something about *all* primes in the prime factorization of $P + 1$.

Exercise 2.2. (*Long-form proof*) In the kingdom of Platonica:

- (P1) Every native is either a knight or a knave (and nobody is both).
- (P2) Every statement a knight makes is true.
- (P3) Every statement a knave makes is false.

Convert the short-form proof of the following theorem into long form:

Theorem 2.6. *Given P1–P3, any person who states “I am a knave” is not a native of Platonica.*

Proof (short form). Assume for contradiction that X is a native of Platonica. If X is a knight, then (by P2) X ’s statement “I am a knave” must be true, so X is a knight and a knave, contradicting P1. Otherwise (by P2) X is a knave, so (by P3) X ’s statement is false, so X is not a knave (contradicting that X is a knave). $\square$

Solution. Here is the long-form proof.

Proof (long form).

1. Let X be an arbitrary person who states “I am a knave”.
- 2.1. Suppose for contradiction that X is a native of Platonica.
- 2.2. By P1, X is either a knight or a knave. We consider each case separately.
- 2.3.1. Case 1. First consider the case that X is a knight.
- 2.3.2. By P2, X ’s statement “I am a knave” is true, so X is a knave.
- 2.3.3. By this and P1, X is not a knight.
- 2.3.4. This is a contradiction (as X is a knight).
- 2.4.1. Case 2. Otherwise, by P1, X is a knave.
- 2.4.2. By P3, X ’s statement “I am a knave” is false.
- 2.4.3. That is, X is not a knave.
- 2.4.4. This is a contradiction (as X is a knave).
- 2.5. By P1, Case 1 or 2 must hold, and each case leads to a contradiction, so we have a contradiction.
3. By Block 2, X is not a native of Platonica. $\square$

Bonus problem: Could we replace (P1) by “(P1) Every native is a knight, a knave, or both”?

Exercise 2.3. (*Big-O definition*) Give a long-form proof of the following theorem:

Theorem 2.7 (KT Chapter 2 Fact 2.4, Page 39). *Suppose $f(n)$, $g(n)$, and $h(n)$ are non-decreasing, non-negative functions and that $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Then $f(n) + g(n) = O(h(n))$.*

Solution.

Proof (long form).

1. Consider any such functions f , g , and h .
2. Let constants $c_1, n_1 > 0$ be such that $(\forall n \geq n_1) f(n) \leq c_1 h(n)$. (These exist by def’n of big-O.)
3. Likewise, let constants $c_2, n_2 > 0$ be such that $(\forall n \geq n_2) g(n) \leq c_2 h(n)$.
4. Fix $n_3 = \max\{n_1, n_2\}$ and $c_3 = c_1 + c_2$.
5. By algebra and the previous steps, then, $(\forall n \geq n_3) f(n) + g(n) \leq c_3 h(n)$.
6. Therefore, by the definition of big-O, $f(n) + g(n) = O(h(n))$. $\square$

Exercise 2.4. (*Invariants*) Here's a puzzle to illustrate the first three points from Section 2.3, including the use of an invariant.

Elizabeth has a secret hiding place where she presently has 35 coins, 38 baseball cards, and 39 pieces of candy. Her younger brother John has his own secret hiding place, but that does not prevent him from taking things from Elizabeth's (her place isn't as secret as she thinks). John always takes two items at a time, each of a different kind so that the number won't decrease too quickly, and he always adds one item of the third kind. On day, some time later, Elizabeth is shocked to find that all the items in her hiding place were of the same kind. Is it possible that it is all candy?

Solution.

Theorem 2.8. *It is not possible that it is all candy.*

Proof (long form).

1. Consider an arbitrary finite sequence, X , of such actions by John.
2. We will prove that, that throughout the sequence, the following invariant holds:
invariant: *the number of coins plus the number of cards is odd.*
3. At the start, the invariant is true because the number of coins plus the number of cards is $35+38$.
4. We will show that each action that John takes preserves the invariant:
 - 5.1. Consider any action in the sequence, where the invariant holds just before the action.
 - 5.2. That is, just before the action the number of coins plus the number of cards is odd.
 - 5.3.1. Case 1. Suppose John takes a coin and a card, and puts back a candy.
 - 5.3.2. Then the number of coins plus cards decreases by two, so remains odd.
 - 5.4.1. Case 2. Suppose John takes a coin and a candy, and puts back a card.
 - 5.4.2. Then the number of coins plus cards doesn't change, so remains odd.
 - 5.5.1. Case 3. Otherwise John takes a card and a candy, and puts back a coin.
 - 5.5.2. Then the number of coins plus cards doesn't change, so remains odd.
 - 5.6. One of the cases holds, and in each case the number of coins plus cards remains odd.
 - 5.7. So the invariant holds just after the action.
6. By Block 5, for each action in X , if the invariant holds just before it, then it holds just after it.
7. As observed in Step 3, the invariant holds initially.
8. So by induction the invariant holds throughout the sequence.
9. In particular, it holds at the end, so at the end the number of coins plus cards is odd.
10. So at the end, the number of coins plus cards is not zero. □

Exercise 2.5. (*Complex long-form proof*) Let $G = (V, E)$ be any graph, where each vertex $v \in V$ has a numeric weight $w(v) \in \mathbb{R}$. Consider the following two properties:

Property A: For every vertex v in the graph, if v has neighbors, then the weight $w(v)$ of v equals the average of v 's neighbor's weights.

Property B: All vertices in G have the same weight.

Give a long-form proof of the following theorem:

Theorem 2.9. *Any finite, connected graph G that has Property A also has Property B.*

Solution. We give two correct long-form proofs. Exercise 2.6 gives an incorrect proof by induction.

First long-form proof.

1. Let G be an arbitrary finite, connected graph having Property A .
2. Before we give the proof, we prove the following utility lemma:

Lemma 2.1. *Let v be any vertex in G that has a neighbor x with larger weight ($w(x) > w(v)$). Then v has a neighbor y with a smaller weight ($w(y) < w(v)$).*

Proof of lemma.

- 3.1. Let v be an arbitrary vertex as in the lemma (so G has a neighbor x with larger weight).
- 3.2.1. Assume for contradiction that v 's neighbors all have weight at least $w(v)$.
- 3.2.2. Then, since v also has at least one neighbor whose weight is strictly larger, the average of v 's neighbors' weights is strictly larger than $w(v)$.
- 3.2.3. This is a contradiction (as G has Property A .)
- 3.3. By Block 3.2, vertex v has some neighbor with weight less than $w(v)$, proving the lemma. $\square$
- 4.1. Now assume for contradiction that G does not have Property B .
- 4.2. Let u and v be two vertices of different weight (they exist because G doesn't have Property B).
- 4.3. Consider any path from u to v in G (it exists as G is connected).
- 4.4. The path must have two adjacent vertices with different weights. (Otherwise, by transitivity, all vertices along the path would have the same weight, but u and v have different weights.)
- 4.5. Of these two adjacent vertices with different weights, let v_1 be the one with smaller weight.
- 4.6. Then v_1 has a neighbor with larger weight.
- 4.7. So, by Lemma 2.1 applied to v_1 , v_1 has some neighbor v_2 with smaller weight ($w(v_1) > w(v_2)$).
- 4.8. So, by Lemma 2.1 applied to v_2 , v_2 has some neighbor v_3 with smaller weight ($w(v_2) > w(v_3)$).
- 4.9. Inductively, there is an infinite path $v_1, v_2, v_3, \dots$ with strictly decreasing vertex weights.
- 4.10. Since the weights strictly decrease along the path, no vertex in G occurs twice on the path.
- 4.11. So G has infinitely many vertices.
- 4.12. This is a contradiction (as G is finite).
5. By Block 4, G must have Property B . $\square$

Here's the same proof in short form.

Same proof, short form. Let G be any finite connected graph with Property A . We use the following observation: *If a vertex v in G has a neighbor x with larger weight ($w(x) > w(v)$) then v also has a neighbor y with smaller weight ($w(y) < w(v)$).* (Otherwise the weight of v would not equal the average of its neighbors' weights.) Suppose for contradiction that G doesn't have Property B . So it has two vertices with different weights. There is a path between these two vertices, along which there must be two neighbors whose weights differ. The smaller-weight neighbor, say v_1 (having a larger-weight neighbor) must also (by the observation) have a smaller-weight neighbor, say v_2 . In turn, v_2 (with larger-weight neighbor v_1) has a smaller-weight neighbor v_3 . Inductively, G contains an infinite path along which the vertex-weights strictly decrease, contradicting that G is finite. $\square$

Here's the second long-form proof.

Second long-form proof.

1. Let G be an arbitrary finite, connected graph having Property A .

2. Let $W = \min_{v \in V} w(v)$ be the minimum vertex weight. (W is well-defined as G is finite.)

We will use the following utility lemma:

Lemma 2.2. *If v is any vertex of weight W , then each of v 's neighbors has weight W .*

Proof of lemma.

- 3.1. Let v be an arbitrary vertex with weight W .
- 3.2. Each of v 's neighbors has weight *at least* W (by definition of W).
- 3.3.1. Assume for contradiction that v has a neighbor of weight strictly more than W .
- 3.3.2. Then the average of the neighbors' weights is strictly more than W .
- 3.3.3. This is a contradiction (as v has weight W and G has Property A).
- 3.4. By Block 3.3, each of v 's neighbors has weight at most W .
- 3.5. By this and Line 3.2, each of v 's neighbors has weight W . □
4. Let v_1 be any vertex of weight W . (Vertex v_1 exists by definition of W .)
5. By the lemma, all neighbors of v_1 have weight W .
6. Applying the lemma to each neighbor of v_1 , all of the neighbors' neighbors also have weight W .
7. Continuing inductively, all vertices reachable from v_1 have weight W .
8. Since G is connected, all vertices are reachable from v_1 , so all vertices have weight W .
9. So G has Property B. □

Exercise 2.6. (*Incorrect induction*) Here's a third proof for the theorem in Problem 2.5. This one has an error. Find the error. Specifically, identify the first step of the proof that asserts a statement that does not follow from the previous steps and the definitions. Explain why the statement does not follow.

Theorem 2.9. *Any finite, connected graph G that has Property A also has Property B.*

Third long-form proof (INCORRECT! FIND THE ERROR!).

1. The proof is by induction on the size of G .
2. Base case: G has two vertices. Then it is easy to verify that Property A implies Property B.
- 3.1. For the induction step, for any $n \geq 2$, assume the theorem holds for all graphs with n vertices.
- 3.2.1. Let G be an arbitrary connected $(n + 1)$ -vertex graph having property A.
- 3.2.2. Let v_1 be any vertex in G , and let G' be obtained from G by deleting v_1 and its edges.
- 3.2.3. G' has Property A, because G does.
- 3.2.4. So, by the inductive assumption, G' has Property B.
- 3.2.5. That is, all vertices in G' have equal weight, say W .
- 3.2.6. Now consider v_1 in G . Each neighbor of v_1 is in G' , so has weight W .
- 3.2.7. Since G has Property A, then, v_1 also has weight W .
- 3.2.8. So all vertices in G have weight W . That is, G has Property B.
- 3.3. By Block 3.2, the theorem holds for all graphs with $n + 1$ vertices.
4. By Line 2, the theorem holds for any graph with two vertices. By Block 3, for every $n \geq 2$, it holds for any graph with $n + 1$ vertices, as long as it holds for every graph with n vertices. Inductively, it holds for all graphs with $n \geq 2$ vertices. □

Solution. The first error is in Step 3.2.3. Nothing established so far implies that G' has Property A: deleting v_1 changes the neighborhoods (and so the neighbor averages) of v_1 's neighbors, so Property A for G does not, by itself, give Property A for G' . (For finite graphs the statement

happens to be true, but only as a consequence of the theorem we are trying to prove, so using it here is circular.) Another error is in the following step: G' does not have to be connected, so the theorem does not apply to G' (however this one can be fixed).

Bonus question. Suppose $G = (V, E)$ is an infinite connected graph with vertex weights and Property A . If all the vertex weights are non-negative, must G also have Property B ? (The answer is no. Can you find an example?)

2.6.2 Exercises without solutions

Exercise 2.7. (*Proof by cases*) Consider the following long-form proof. Note that “positive” means “strictly greater than zero”, while “negative” means “strictly less than zero.”

Theorem 2.10 (False!). *For every real number, its square is positive. That is, $\forall x \in \mathbb{R}, x^2 > 0$.*

Proof (long form). Find the error!

1. Let $x \in \mathbb{R}$ be an arbitrary real number.
- 2.1. Case 1. Suppose x is positive ($x > 0$).
- 2.2. Then x^2 is a product of two positive numbers, so x^2 is positive.
- 3.1. Case 2. Suppose x is negative ($x < 0$).
- 3.2. Then x^2 is a product of two negative numbers, so x^2 is positive.
4. In either Case 1 or 2, x^2 is positive, so x^2 is positive. □

Here is the same proof in short form.

Proof (short form). The proof is by cases. If x is positive, then x^2 is a product of two positive numbers, so is positive. If x is negative, then x^2 is a product of two negative numbers, so is positive. In either case, x^2 is positive. So x^2 is positive. □

Find the error. What is the first step of the long-form proof that makes an assertion that is not necessarily true? In what situation does that assertion fail?

Exercise 2.8. (*Proof practice—red/blue/round/square*)

Complete the proof of the following theorem.

Theorem 2.11. *In a set S of objects, each is either red or blue, and each is either round or square. S has at least one red object, at least one blue object, at least one round object, and at least one square object. Then S has two objects that differ both in color and in shape.*

Proof (long form).

1. Let S be the set of objects in question.
- 2.1. Case 1. Suppose all red objects in S are the same shape.
- 2.2. Assume that the red objects are all square (the case when they are all round is symmetric).
- 2.3. ...
- 2.4. So, in Case 1, there exist two objects that differ both in color and in shape.
- 3.1. Case 2. Otherwise S has a red-round object and a red-square object.
- 3.2. ...
- 3.3. So, in Case 2, there exist two objects that differ both in color and in shape.
4. Case 1 or Case 2 must hold, and in either case the theorem holds, so the theorem holds. □

Exercise 2.9. (*Understanding induction*) Find the error in the following proof by induction.

Theorem 2.12. *In any finite set of marbles, all the marbles are the same color.*

Proof (long form). Where's the error?

1. Define $P(n) \equiv$ “in any set S of n marbles, the marbles are the same color.”
2. We show by induction on n that $P(n)$ holds for all integers $n \geq 0$.
3. First we consider the base cases, $n \in \{0, 1\}$.
4. In any set of zero or one marbles, all marbles are trivially the same color. I.e., $P(0)$ and $P(1)$ hold.
- 5.1. Next we prove the inductive step. Fix an arbitrary integer $n \geq 2$.
- 5.2.1. Assume that $P(n - 1)$ holds. (We'll show that $P(n)$ holds.)
- 5.2.2.1. Let $S = \{m_1, m_2, \dots, m_n\}$ be an arbitrary set of n marbles.
- 5.2.2.2. Let set $S_1 = \{m_1, m_2, \dots, m_{n-1}\}$ contain the first $n - 1$ marbles in S .
- 5.2.2.3. Let set $S_2 = \{m_2, m_3, \dots, m_n\}$ contain the last $n - 1$ marbles in S .
- 5.2.2.4. S_1 has size $n - 1$, and $P(n - 1)$ holds (Step 5.2.1), so all marbles in S_1 are the same color.
- 5.2.2.5. Likewise all marbles in S_2 are the same color.
- 5.2.2.6. Marble m_2 is in S_1 , so by Step 5.2.2.4, each marble in S_1 is the same color as m_2 .
- 5.2.2.7. Marble m_2 is also in S_2 , so by Step 5.2.2.5, each marble in S_2 is the same color as m_2 .
- 5.2.2.8. Each marble in S is in S_1 or S_2 , so (by the previous two steps) is the same color as m_2 .
- 5.2.2.9. That is, all marbles in S are all the same color.
- 5.2.3. By Block 5.2.2, $P(n)$ holds.
- 5.3. By Block 5.2, if $P(n - 1)$ holds, then $P(n)$ holds.
6. By Block 5, for any $n \geq 2$, if $P(n - 1)$ holds, then $P(n)$ holds.
7. By Step 4, $P(0)$ and $P(1)$ hold.
8. By the preceding two steps, and induction, $P(n)$ holds for all integers $n \geq 0$.
9. That is, within any finite set of marbles, all the marbles are the same color. □

For comparison, here's the proof in short form.

Proof (short form). We show by induction on $n \geq 0$ that, within any set S of n marbles, all the marbles are the same color. For the base cases $n \in \{0, 1\}$ this is true because S contains at most one marble. Consider any set S of $n \geq 2$ marbles. Assume the inductive hypothesis holds for any set of $n - 1$ marbles. Order the n marbles in S arbitrarily. Let S_1 contain the first $n - 1$ marbles in S . Let S_2 contain the last $n - 1$ marbles in S . By the inductive assumption, within S_1 all marbles are the same color. Likewise, within S_2 all marbles are the same color. Since S_1 and S_2 both contain marble m_2 , all marbles in $S = S_1 \cup S_2$ are the same color. □

Exercise 2.10. (*Proof practice—rationality (Problem 1.13 of LLM (2018))*)

Complete the proof of the following theorem.

Theorem 2.13. *Let*

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1} + x^m$$

be an arbitrary polynomial with integer coefficients, whose highest-degree term has coefficient 1. Then any rational root of $p(x)$ is an integer.

Proof (long form).

1. Let p be an arbitrary polynomial as in the theorem, and let x be any rational root of p .
2. Fix integers i and j with $j > 0$ such that $x = i/j$.
3. ...
4. So x is an integer. □

Exercise 2.11. (*Proof practice, greedy algorithms—partition the integers*)

The positive integers are partitioned into several subsets $A_1, A_2, \dots, A_n$ such that, for each $i \in \{1, 2, \dots, n\}$ and x in A_i , $2x$ is not in A_i . What is the minimum possible value of n ?

Complete the proof of the following theorem.

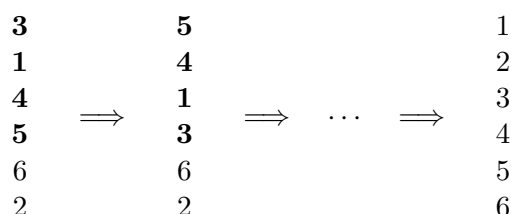
Theorem 2.14. *The minimum possible value of n is 2.*

Proof (long form).

1. For any such partition n is at least 2. (Otherwise 1 and 2 would be in A_1 .)
2. Define $A_1 = \dots$ and $A_2 = \dots$
3. To finish we show that A_1, A_2 is a partition with the claimed property... □

Bonus question: what if you want neither $2x$ nor $3x$ to be in A_i with x ?

Exercise 2.12. (*Reasoning about processes—flipping disks upper bound*) You have n discs labeled $1, 2, 3, \dots, n$ (not necessarily in that order), stacked one on top of each other (sort of like Towers of Hanoi). The basic operation you can do on the stack is the following: take some disc, and all the discs above it, and flip over that part of the stack. For example, you can take the stack on the left, and by applying the basic operation starting at disc 5, you can get the stack in the middle:



The goal is to get to the stack on the right, in which the discs are ordered. Give a long-form proof of the following theorem.

Theorem 2.15. *For any initial stack of n discs, there is a sequence of at most $2n$ basic operations that ends with the discs stacked in order (1 on top, 2 next, ..., n on the bottom).*

Exercise 2.13. (*Reasoning about processes—flipping disks lower bound*) Give a long-form proof of the following theorem, for the problem defined in Exercise 2.12.

Theorem 2.16. *For any $n \geq 1$, there is some initial way to stack the n discs such that any sequence of basic operations that ends with the discs stacked in order has at least $n - 1$ operations.*

(For every n , define a particular initial stacking of the n discs, then prove that there is no sequence of less than $n - 1$ operations that orders them.)

Exercise 2.14. (*Proof practice—boys and girls around a table*) Complete the proof of the following theorem. (*Hint:* make sure your reasoning is complete and covers all possible arrangements. It's not enough to observe that one particular kind of arrangement fails.)

Theorem 2.17. *Fifty chairs are arranged evenly spaced around a round table. For any seating arrangement of 25 boys and 25 girls (placing one boy or girl in each chair), at least one person has a girl on each side.*

Proof (long form).

1. Suppose for contradiction that there is an arrangement where nobody sits between two girls.
2. ...
3. This is a contradiction, so some person sits between two girls. □

Exercise 2.15. (*Proof practice—inclusion/exclusion for reals*) ★ Prove the following theorem.

Theorem 2.18. *Define $f(x, y, z) = |x| + |y| + |z| - |x + y| - |x + z| - |y + z| + |x + y + z|$. Every triple x, y, z of real numbers satisfies the inequality $f(x, y, z) \geq 0$.*

Exercise 2.16. (*Proof practice—handshakes after the party*) ★

Setup: $N + 1$ couples (where $N \geq 0$) attend a dinner. At the end of the dinner, one person (Alice) asks each other person to write down the number of distinct people with whom he or she shook hands at the dinner. Surprisingly, all numbers $0, 1, 2, \dots, 2N$ are written down. Assuming that no person shook hands with their partner, how many people did Alice shake hands with at the dinner? (*Hint:* one person shook hands with $2N$ people. What about that person's partner?)

Complete the proof of the following theorem.

Theorem 2.19. *Alice shook hands with N people.*

Proof (long form).

1. We prove by induction on N that, in any such scenario, Alice shakes hands with N people.
2. In the base case, $N = 0$, ...
- 3.1. For the induction step, consider any $n \geq 1$, and assume that the statement holds for any such scenario with $N = n - 1$ couples. We show that the statement holds for any such party scenario with $N = n$ couples.
- 3.2. ...
4. ... □

Exercise 2.17. (*Induction—Hanoi lower bound*) ★ Check out the towers of hanoi puzzle (https://en.wikipedia.org/wiki/Tower_of_Hanoi), then complete the proof of the following theorem.

Theorem 2.20. *Any legal sequence of moves (which never puts any disc on top of a smaller disk, and moves N discs from one peg to another) takes at least $2^N - 1$ moves.*

Proof (long form).

1. The proof is by induction on $N \in \{1, 2, 3, \dots\}$.
2. The base case $N = 1$ holds because moving one disc requires at least $2^1 - 1 = 1$ move.
- 3.1. For the induction step, consider any $n \geq 1$, and assume that the theorem holds for $N = n$.
- 3.2. We show that the theorem holds for $N = n + 1$...
4. ... □

Exercise 2.18. (*Proof practice—all pairs with different sums*) ★★

Complete the proof of the following theorem.

Theorem 2.21. *A subset S of $\{1, 2, \dots, 100\}$ has the following property: for every quadruple u, v, w, x of distinct numbers in S , the sum of u and v differs from the sum of w and x . Then the size of S is at most fifteen.*

Proof (long form).

1. Consider any such subset S .
2. ...
3. Therefore the size of S is at most fifteen. □

For more problems, see <https://www.cs.ucr.edu/~neal/puzzles> (those marked “P”).