Appendix X
Understanding Logarithms Intuitively

Adam A. Smith

Logarithms make a lot of people anxious. A lot of this has to do with the way they're often taught in high school and secondary school: by memorizing all the proper steps, without imparting much deeper meaning. For example, maybe you were once taught to solve problems like this:

$$\log_7 49 = ?$$

If you are like most people, you knew how to solve this years ago, when it was important to pass a math test. You were probably taught how to rearrange this into an exponential equation using the following equivalence:

$$\log_b x = y \iff b^y = x,$$

and from there you solved for the unknown. But without any solid intuition for what this all really means, you were probably just going through a series of memorized steps in order to get the right answer. This isn't helpful: if you just memorize without understanding, you'll forget all about it once it's no longer important.*

The problem is that logarithms *are* important. They are especially crucial in computer science when one wants to analyze two different algorithms, figuring out which one is the more efficient. Thus, the purpose of this work is to give you a more *intuitive* feel for what a logarithm represents, making you more comfortable with what they mean. You'll be able to estimate them in your head, and hopefully this knowledge will stay with you for a long time.

Scales of Ten

The most important thing to know about logarithms is this:

A logarithm represents the *scale* of a number.

Think of all the one-digit numbers, 1 through 9. (For now we're skipping over 0.) Of course these numbers are all different, but they're close enough to each other to be easily comparable. However the two-digit numbers, 10 through 99, are on a totally different scale. They're easily comparable to each

*Maybe it even convinced you that you're "not a math person", even though you had done perfectly fine in math class before. This is a problem, because when people become convinced that they "can't do math", it can limit their career options in the future.

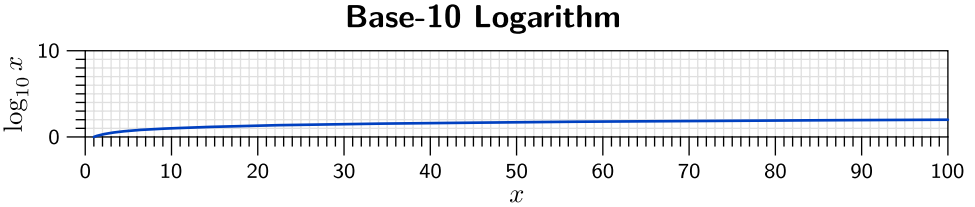
other, but most of them are so much bigger than the one-digit numbers, that it's hard to think of them in quite the same way. If you were drawing them next to each other on a graph, the two-digit numbers would dominate, and you could barely read the one-digit values at all. Next come the three-digit numbers, 100 through 999. And these are on an even higher scale: it's easy to compare them with each other, but harder to compare them with the two-digit numbers and even harder to compare them with the one-digit numbers. They're all on different orders of magnitude.

These scales of size 10 are the base-10 logarithms, which are represented by "$\log_{10}$" in equations. Base-10 logarithms are sometimes called *common logarithms*. That small "10" is the *base*, and it can be any positive number except 1 (though we'll soon see that only a few bases are commonly used). If you have a round number like 1, 10, 100, and so on, the base-10 logarithm of that number is just the number of zeros it has:

$$\begin{aligned} \log_{10} 1 &= 0 & \log_{10} 1000 &= 3 & \log_{10} 1{,}000{,}000 &= 6 \\ \log_{10} 10 &= 1 & \log_{10} 10{,}000 &= 4 & \log_{10} 10{,}000{,}000 &= 7 \\ \log_{10} 100 &= 2 & \log_{10} 100{,}000 &= 5 & \log_{10} 100{,}000{,}000 &= 8 \end{aligned}$$

For a number that lies between two of these round numbers, the logarithm will be some intermediate decimal value (probably one that goes on forever without repeating). So all the one-digit numbers greater than or equal to 1 have a logarithm between 0 and 1. Likewise, the two-digit numbers have a log between 1 and 2, and so on. So if you want to estimate the logarithm of a number, all you have to do is count its digits. For example the number 83,176,000 has eight digits, and therefore its log *must* be between 7 and 8. And since it's a large eight-digit number, the log is closer to 8 than 7. (In fact, the log of this number is approximately 7.92.)

Here's the graph of positive base-10 logarithms:



Look at how flat it is—it increases very slowly. Nevertheless, it always goes up: bigger numbers *always* have bigger logarithms. You can also see that it gets flatter and flatter as it goes on; it takes bigger and bigger inputs for each subsequent increase of 1. But it never gets totally flat: if you increase the input, the output is always at least *a little* bigger. And unlike other functions, it doesn't have an *asymptote* that it approaches forever but never reaches. It will reach any value you can think of, eventually.

In the base-10 logarithm, each scale is 10 times bigger than the previous one. Another way to think of it is "Starting from 1, how many times do I

need to multiply by 10 to get to this number?" For example, if we write 83,176,000 in "scientific notation", we get:

$$83{,}176{,}000 = \ 8.3176 \times 10^7.$$

Writing out all the 10s separately, this becomes:

$$83{,}176{,}000 = \ 8.3176 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10.$$

So how many factors of $\times 10$ are there in this number? It's easy to see that there are 7, and then most of an 8th (the 8.3176 is "most of" an 8th $\times 10$). So the log of about 7.92 makes sense. If the one-digit numbers are "in the zeroth scale", and the two-digit numbers are "in the first scale", then we can think of 83,176,000 as being "in the seventh scale, but almost to the eighth scale".

Remember that $\log_{10} 100$ is 2, and $\log_{10} 1000$ is 3. So which number has a log of 2.5, halfway between them?

$$\log_{10} x = 2.5.$$

To solve this, we need to multiply 1 by a factor of $\times 10$, two and a half times. But how do you multiply half a time? The answer is to use a square root:

$$\begin{aligned} x &= 10 \times 10 \times \sqrt{10} \\ x &= \sim 316.23. \end{aligned}$$

(The $\sim$ means "approximately".) This might seem surprising, since ~ 316.23 isn't halfway between 100 and 1000, in the way you usually think of "halfway". But remember, we're talking about scales and multiplicative factors, so ~ 316.23 really *is* halfway between them. When you multiply $100 \times \sqrt{10}$, you get ~ 316.23. And when you multiply that by $\times\sqrt{10}$ again, you get 1000. Multiplicitavely, ~ 316.23 is halfway between 100 and 1000. That is why:

$$\log_{10} \sim 316.23 = 2.5.$$

We can tell a lot about numbers just from their logarithms. For example, let us say that there are two numbers, a and b. And let us say that we don't know the numbers themselves, but we do know their base-10 logs:

$$\begin{aligned} \log_{10} a &= 6.96 \\ \log_{10} b &= 4.26 \end{aligned}$$

What can we deduce about the numbers themselves, before we calculate them?

- a must be bigger, because it has the bigger log.
- In fact, a must be *a lot* bigger, because it has two more digits. (The difference between the logarithms tells us the difference in the number of digits a and b have.)

- We can see that a is a rather high seven-digit number, and b is a low five-digit number.

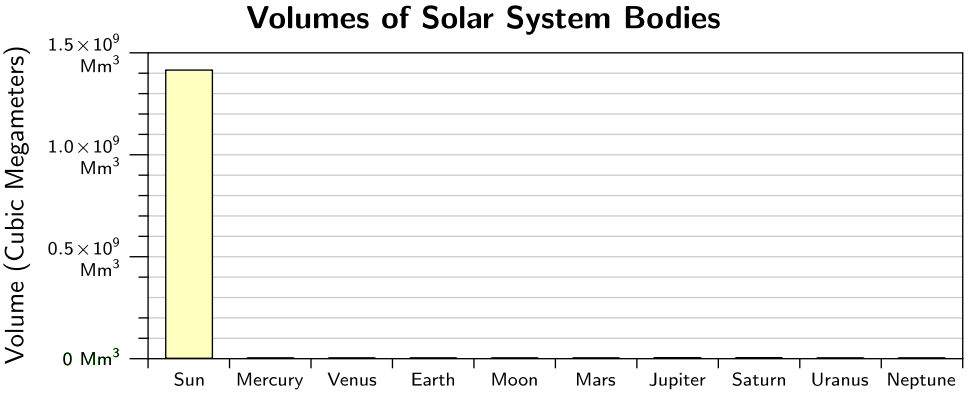
So we really can tell a lot about numbers by just looking at their logarithms, without calculating them. But we can calculate a and b if we choose, by raising 10 to the two logs:

$$a = 10^{6.96} = \sim 9{,}120{,}108$$
$$b = 10^{4.26} = \sim 18{,}197$$

Everything we deduced above was true, and we were able to do so without any difficult calculation.

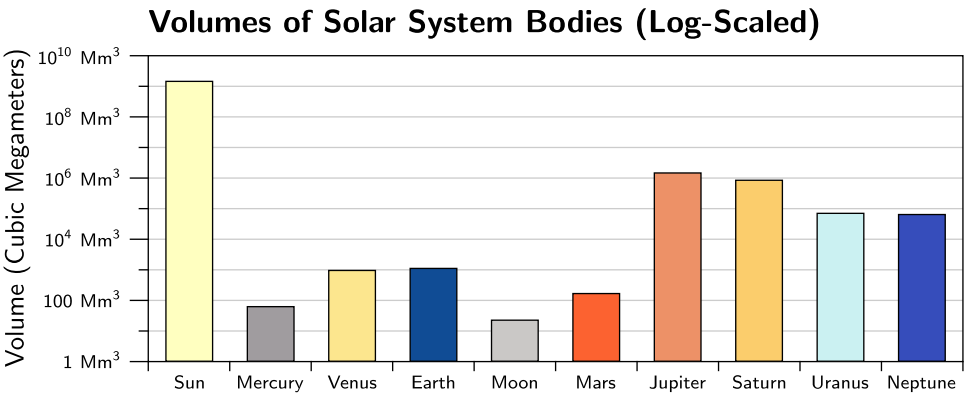
An Interplanetary Example

So why are logarithms useful? Sometimes you might want to compare items that are so different from each other, that it becomes difficult. For example, let us say that we wanted to compare the sizes of the major bodies of the solar system. We want to communicate clearly, so we decide to make a graph of the volumes in cubic megameters (Mm). (A megameter is 1000 km: about the distance from Portland, Oregon to Salt Lake City, Utah, or from Paris, France to Madrid, Spain. It's a good unit to use when talking about sizes of planets.)



The bar graph above shows what happens when we display these values in the traditional way. We can see the problem right away: the Sun is so much bigger than everything else, that it's almost impossible to read any of the other values! Is Saturn bigger or smaller than Neptune? How does the size of Venus compare to the size of the Moon? The only thing that's obvious here is that the Sun is gigantic, but most people already know that. This graph just isn't that useful.

But what if instead make a log-scaled graph, like this:



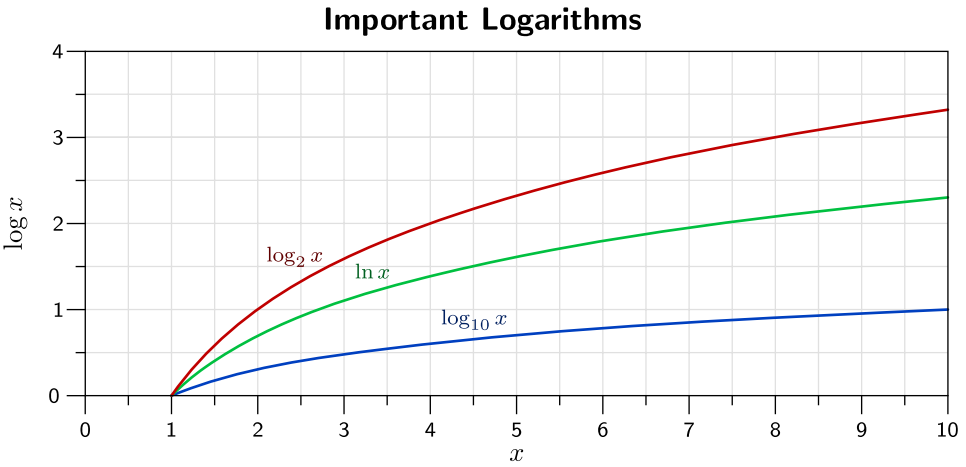
Here, the height of each bar is the logarithm of the value. See how every notch on the left is 10× the value of the one before it? This makes the overall graph much clearer, because it means that we can graph values that differ by quite a lot. Effectively, we're graphing the *scales* of the volumes. The Sun is still clearly the biggest thing, but now we can see better how the planets compare. Jupiter and Saturn are about three scales smaller than the Sun, so they're about 1000× smaller. Uranus and Neptune are one scale smaller still: each one is about one tenth the size of Jupiter or Saturn. And Earth and Venus are two scales smaller still. So therefore Earth is six scales smaller than the Sun—or about a million times smaller! This graph is much more useful and more readable than the one above, thanks to logarithms.

Be careful with log-scaled graphs—they can be misleading if you don't read them carefully. For example, it might look to some people that Earth is about one third the size of the Sun, because its bar is about a third the height. This just isn't true—it's six whole *scales* smaller. Log-scaled graphs should only be used when there's a clear benefit like there is here.

Natural and Binary Logarithms

But wait—aren't there other bases of logarithms? Everything we've done so far deals with factors of 10: each scale is 10 times bigger than the one before it. When you memorized (and probably forgot) how to do logarithms when you were younger, you probably had to do calculations with lots of wacky bases, like base 7 and base 6 and base 11.

Remember, the base of the logarithm determines the size of the scale. But what your old math teacher probably didn't tell you is that most bases just aren't used that much in real scientific and engineering work. You usually don't need to worry about them. In fact, there are only three bases that you will commonly see: common logs (base 10), natural logs (base e), and binary logs (base 2). Here's how they compare to each other:



Notice that the logarithm of 1 is *always* 0, regardless of base. This is because no doubling, $\times$10s, or any other multiplying is needed to attain it. Also, two different logarithms are *always* proportional to each other in the same way. For example, the binary log of some number x will always be $\sim$3.32 times as big as the base-10 logarithm of x. We'll talk more about this later.

Let's talk about these important bases one at a time:

- You've already seen base-10 logarithms like this:

$$\log_{10} 31 \approx 1.49.$$

  Humans like using 10 as a base because it's intutive. A lot of human-scaled measurements use base 10. For example, the prefixes of the metric system are a kind of base-10 logarithm, with milli-, centi-, deci-, and no prefix all referring to different scales that differ by a factor of $\times$10. Also, the Richter scale that used to be used to measure earthquakes is base-10 logarithmic, with a 5.0 earthquake being ten times as strong as a 4.0 earthquake. And the decibel system used for acoustics makes use of the base-10 log.

- People doing calculations in calculus frequently use the number e as the base. This number e (sometimes called "Euler's number"*) is an infinitely-long non-repeating decimal number, approximately equal to 2.71828. Like π, it is a mathematical constant of the universe, that appears in many very important equations and theorems. The base-e logarithm is often called the *natural logarithm* because it is fundamental to many calculus equations and to natural phenomena. It is commonly written in equations as "ln" (or rarely as "$\log_e$").

  If you wish to estimate a natural logarithm, it is the same as asking how many factors of $\times e$ there are in a number. For example,

$$e \times e \times e \approx 20.$$

*Since "Euler" is a German name, it is pronounced much like the English word "oiler".

That is, there are about 3 factors of $\times e$ in 20. Therefore:

$$\ln 20 \approx 3.$$

Calculating natural logs in your head is more difficult than the other common logs, since e is a non-repeating decimal number. Fortunately, they are rare when studying computer science. You will encounter them much more when you study calculus, physics, or chemistry.

- People who work with information theory (like computer scientists) often use a logarithm of base 2, also known as the *binary log*.* Here the scale is 2 rather than 10. Therefore, the binary log asks "Starting from 1, how many times was this number *doubled* in order to reach its current value?" (Equivalently, you may also ask "How many times does this number need to be *halved* in order to get back down to 1?")

  The binary log can be expressed as "$\log_2$" to distinguish it from other logarithms. (It is also sometimes written as "lg".) For example:

$$\log_2 64 = 6.$$

  This is because when 1 is doubled six times, it will be exactly 64.

  When working with base-2 logarithms, the powers of two are very important.** The first few powers are:

$2^0 = 1$	$2^3 = 8$	$2^6 = 64$	$2^9 = 512$
$2^1 = 2$	$2^4 = 16$	$2^7 = 128$	$2^{10} = 1024$
$2^2 = 4$	$2^5 = 32$	$2^8 = 256$	

  This means that:

$\log_2 1 = 0$	$\log_2 8 = 3$	$\log_2 64 = 6$	$\log_2 512 = 9$
$\log_2 2 = 1$	$\log_2 16 = 4$	$\log_2 128 = 7$	$\log_2 1024 = 10$
$\log_2 4 = 2$	$\log_2 32 = 5$	$\log_2 256 = 8$	

  Remember, the binary log of a value is the number of doublings needed to attain that value. So if you were to double the number 1 ten times, you would get 1024.

  Notice that 2^{10} (1024) is very close to 1000, so:

$$\log_2 1000 \approx 10.$$

*Musicians also work with binary logs, though most of them don't know it. Each octave in a musical score represents a doubling of frequency. Therefore a score is basically a binary-log-scaled graph of notes' frequencies!

**You do not need to memorize these powers of 2 now. When you work with computers long enough, you will memorize them without even realizing that you've done so.

This useful fact can be used to approximate the binary logarithm of very large numbers. For example, what is the binary logarithm of 16,000,000,000 (sixteen billion)? Separating out the factors of $\times 1000$, we see that:

$$16{,}000{,}000{,}000 = 16 \times 1000 \times 1000 \times 1000.$$

Since we need 4 doublings to get the 16, and we need about 10 doublings to get each of the three 1000s, the total number of doublings (the log of 16,000,000,000) must be:

$$\begin{aligned} \log_2 16{,}000{,}000{,}000 &= 4 + \sim 10 + \sim 10 + \sim 10 \\ &= \sim 34. \end{aligned}$$

In fact, the real answer is ~ 33.90, so our approximation is quite good!

- Sometimes, you'll also see *trinary logarithms* (base 3) when working with information theory. These are a lot like base-2 logarithms, except of course the scale is 3 and the log is a count of *triplings*, not doublings. There is no special way of expressing the trinary log, so they are almost always just "$\log_3$". These logs are quite rare, though.

Analyzing Algorithms

It turns out that logarithms are very important for analyzing and comparing different algorithms to each other. Sometimes, there are two different ways to solve a problem, and both of them will get the right answer. But if one method is much quicker and more efficient than the other, than that method should be used—even if the slower method would eventually solve the problem.

For example, let us say that you are programming some kind of word game in which the players take turns playing cards or tiles with letters on them, in order to spell out words. The computer needs to judge whether or not a certain play is a legitimate word, rejecting nonwords like "AQUR" and "NUMPY". Say you have a big list containing all the allowed words, and you need to check it every time a play is made, to make sure it's allowed. We will call the word we're checking a "query". If a query is "valid", that means that it's in the list. An "invalid" query is not.

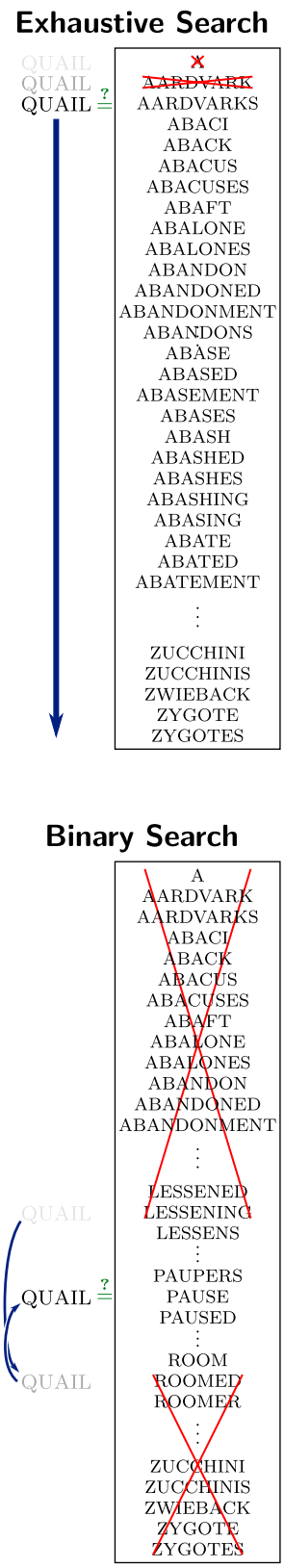
One approach is to do an *exhaustive search*, like is shown on the next page. You just check all the words in the big list one at a time, seeing if you can find the queried word. But this can take a lot of time. If the query is valid, on average you'll have to search through half of them before you find it. But if the query is invalid, you'll have to look through every word in the list and rule them all out one at a time. This search is an example of a *linear* algorithm, because the number of checks it makes is proportional to the number of words in the list. If there were twice as many words, the

search would take twice as long. If there were three times as many words, it would take three times as long. And so on. It's not very efficient.

One better way to do this task is to use a *binary search*, which can rule out many words simultaneously. All it needs is for the list to be presorted alphabetically. It starts by checking the middle word in the list (instead of the first one) and uses that to rule out half the list. If the middle word comes before the query, the query cannot be in the first half. Likewise if the middle word comes after the query, it cannot be in the second half. (If the middle word happens to be the right word, the search is very short indeed.) Keep doing this with all the words that haven't been ruled out, until you either find the query or prove that it can't be in the list. Thus each query can rule out half of the remaining list!

So let's say that a player has just played "QUAIL", and we want to make sure that it's a valid play. We'll start by comparing that to the word in the middle of the list, which happens to be "LESSENING". "QUAIL" comes after "LESSENING", so every word before "LESSENING" can be ruled out. So either "QUAIL" is an invalid word, or it's in the second half. (We don't know yet.) Next we compare it with the word that is in the middle of the remaining list: "ROOMED". "QUAIL" comes before "ROOMED", so if QUAIL is valid, it must be in the first half of what remains. The entire second half (that is, the fourth quarter of the whole list) can be ruled out. We've already eliminated three quarters of the whole list, after only comparing "QUAIL" to two words! This process is kept up, until "QUAIL" is found.

Every time we check a word, we can rule out half the remaining words in the list. So how many checks do you need to make for a list of n words? That's the same as asking how many times do you need to halve n to get down to 1. The answer is (of course) $\log_2 n$. The amount of time it'll take is proportional to the log of n, which means that if you were to double the size of the list, the process of checking a query would only get slightly longer. We say that binary search is a logarithmic algorithm. It



is much more efficient than a linear algorithm like exhaustive search. Many other computer algorithms depend on a doubling or halving factor like this, and so their analysis will involve binary logs. Thus, understanding logs can help you pick out the fastest method to solve a problem.

Negative Logarithms

So far, we've been concentrating on the positive numbers. But not every logarithm is positive—so it's natural to ask what the relation is between negative numbers and logarithms.

First, we usually don't allow the base or the input of a logarithm to be negative. We start running into the same problem as when taking the square root of a negative number: the result is often an imaginary or complex number. So we're not going to worry about those cases here.

But if you take the logarithm of a positive number less than 1, the result will be negative. This makes sense: 1 has a logarithm of 0, so smaller numbers must have logs less than 0. Negative logarithms represent smaller and smaller scales. For base-10 logs:

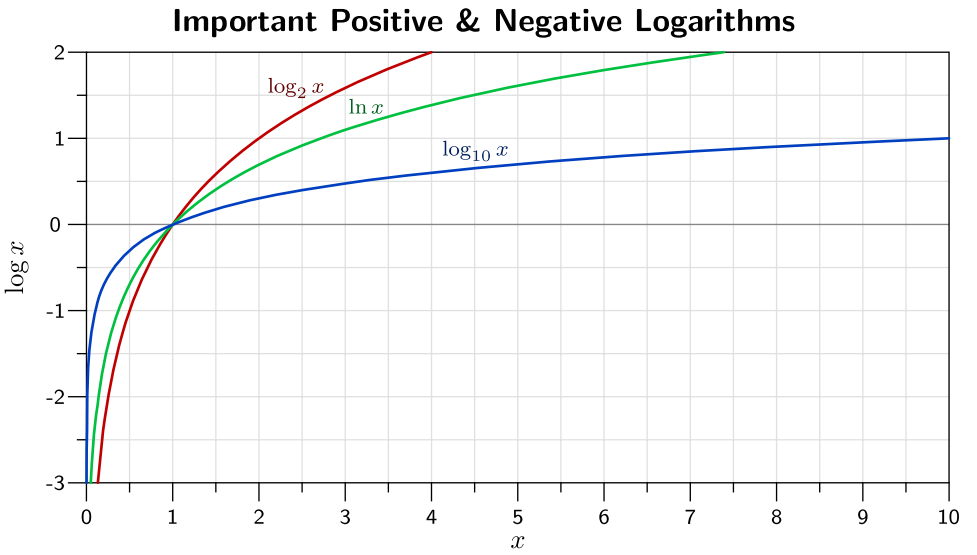
$$\begin{array}{lll} \log_{10} 1 = 0 & \log_{10} 0.001 = -3 & \log_{10} 0.000001 = -6 \\ \log_{10} 0.1 = -1 & \log_{10} 0.0001 = -4 & \log_{10} 0.0000001 = -7 \\ \log_{10} 0.01 = -2 & \log_{10} 0.00001 = -5 & \log_{10} 0.00000001 = -8 \end{array}$$

So to go a higher scale, we have to multiply by 10. But to go a scale lower, we have to divide by 10.

Binary logs are similar. Each step down means that we have to halve one more time:

$$\begin{array}{llll} \log_2 1 = 0 & \log_2 \frac{1}{8} = -3 & \log_2 \frac{1}{64} = -6 & \log_2 \frac{1}{512} = -9 \\ \log_2 \frac{1}{2} = -1 & \log_2 \frac{1}{16} = -4 & \log_2 \frac{1}{128} = -7 & \log_2 \frac{1}{1024} = -10 \\ \log_2 \frac{1}{4} = -2 & \log_2 \frac{1}{32} = -5 & \log_2 \frac{1}{256} = -8 & \end{array}$$

Here's the graph of base-10, natural, and binary logarithms including some negative logs:



No matter which base we're using, the logarithms get steeper and steeper as the input approaches 0. But you can't really take the logarithm of 0 itself—0 just isn't on any scale, so the answer isn't defined. However, you can say that the log *approaches* negative infinity as x *approaches* 0. The log of any infinitesimally small number will be a very negative number.

Let's try to estimate the log of a number less than 1. What is the binary log of $\frac{1}{3}$? We know that $\frac{1}{3}$ is between $\frac{1}{4}$ and $\frac{1}{2}$, whose logs we already know. If we take the logs of all the numbers in this relation, we can find upper and lower bounds on the log of $\frac{1}{3}$:

$$\begin{array}{ccccc} \frac{1}{4} & < & \frac{1}{3} & < & \frac{1}{2} \\ \log_2 \frac{1}{4} & < & \log_2 \frac{1}{3} & < & \log_2 \frac{1}{2} \\ -2 & < & \log_2 \frac{1}{3} & < & -1 \end{array}$$

Therefore, we know that the binary log of $\frac{1}{3}$ must be between -2 and -1. (Indeed, it is ~ -1.58—roughly halfway between them.)

Logarithm Identities

Finally, let's work through some common logarithm identities. These are the equations that you might see when studying logarithms in a more formal way. After each identity, we'll explain why it is true on an intuitive level.

The first identity can be used to relate different logarithms with bases b and d to each other:

$$\boxed{\log_b d = \frac{1}{\log_d b}}.$$

What does this mean? For now, imagine that $b = 2$ and $d = 10$. Remember that the binary log is counting the number of doublings, and the base-10

log is counting the number of $\times 10$ factors, to make up some number x. But how many doublings are there in one $\times 10$ factor? That's just the binary log of 10:

$$\log_2 10 \approx 3.32.$$

In other words, multiplying something by 10 is the same thing as doubling it about 3.32 times. Multiplying a number by 100 is the same as doubling it about 6.64 times. The two logarithms are very closely related. We can also go the other way:

$$\log_{10} 2 \approx 0.301.$$

This means that doubling something is the same thing as multiplying it by 10 about 0.301 times. And those numbers $\log_2 10$ (~ 3.32) and $\log_{10} 2$ (~ 0.301) must be reciprocals of each other, since they represent the "exchange rate" between the two logarithms.

The above boxed equation simply makes it clear that this is true no matter what b and d are. So long as they're valid logarithmic bases, you can find this "exchange rate", and the two rates must be reciprocals of one another.

Furthermore, we can take advantage of this to calculate binary logarithms on calculators that can only do base-10 logarithms. This is expressed in our next important equation:

$$\boxed{\log_b x = \log_d x \div \log_d b}.$$

That is, a base-b logarithm of some number is equal to the base-d logarithm of that same number, divided by the base-d logarithm of b. Let us say you wanted to calculate the binary log of 256, but your calculator only has base-10 logs. Not a problem! Here are the steps you can do:

1. Start by calculating the base-10 log of 256, which is ~ 2.41. So there are ~ 2.41 factors of $\times 10$ in 256.

2. Now divide that number by $\log_{10} 2$, which is ~ 0.301.

Why do we divide by $\log_{10} 2$? Well, really we'd like to multiply by $\log_2 10$ (~ 3.32), because we know that each $\times 10$ is "worth" about 3.32 doublings. But our calculator doesn't do binary logs, so we can't easily calculate that number. Fortunately, we can calculate its reciprocal: $\log_{10} 2$. So instead of multiplying by $\log_2 10$, we divide by $\log_{10} 2$. It'll get us the exact same answer, like so:

$$\begin{aligned}\log_2 64 &= \log_{10} 64 \div \log_{10} 2 \\ &= \sim 2.41 \div \sim 0.301 \\ &= 8.\end{aligned}$$

This could easily be done on a calculator that does base-10 logs. Of course, the above boxed formula works no matter what b and d are. But you will

find this most useful when you're trying to calculate binary logarithms, when your equipment won't let you do it directly.

Next:

$$\boxed{\log_b xy = \log_b x + \log_b y}.$$

All this is saying is that the number of $\times b$ factors in the product xy is just equal to the number of $\times b$ factors in x plus the number of $\times b$ factors in y. For example, what is $\log_2 4096$? Remember, that means how many doublings you must do to attain 4096. First, notice that:

$$4096 = 1024 \times 4.$$

And we already know that you need 10 doublings to get to 1024, and 2 doublings to get to 4. Therefore, the total number of doublings to get to 4096 must be 12:

$$\begin{aligned}\log_2 4096 &= \log_2(1024 \times 4)\\ &= \log_2 1024 + \log_2 4\\ &= 10 + 2\\ &= 12.\end{aligned}$$

And of course, this is true no matter what base you are using.

A related formula is:

$$\boxed{\log_b \frac{x}{y} = \log_b x - \log_b y}$$

This one just goes in the opposite direction. What if you're trying to calculate $\log_2 512$, but you don't have it memorized? But you remember that $\log_2 1024$ is 10, and 512 is half of 1024. So you can think of 512 this way: start at 1 and double it ten times to get up to 1024, but then halve it once to get back down to 512. Mathematically, this is:

$$\begin{aligned}\log_2 512 &= \log_2(\tfrac{1024}{2})\\ &= \log_2 1024 - \log_2 2\\ &= 10 - 1\\ &= 9.\end{aligned}$$

This identity is also related:

$$\boxed{\log_b x^p = p \times \log_b x}.$$

If you have some number x raised to a power p, that means that you're multiplying together p different instances of x. For example:

$$x^5 = x \times x \times x \times x \times x.$$

Each of those instances of x must have the exact same factors. So if you know the logarithm of x, the logarithm of x^p must be p times as great. For example, what is the binary log of 8^5, which is 32,768?

$$\begin{aligned}\log_2 32{,}768 &= \log_2 8^5 \\ &= 5 \times \log_2 8 \\ &= 5 \times 3 \\ &= 15.\end{aligned}$$

All this really means is that 8^5 is five different factors of $\times 8$, and each individual $\times 8$ is 3 doublings. Therefore, 8^5 must be $5 \times 3 = 15$ doublings.

Conclusion

Logarithms can be hard when you're first getting used to them. They're a different kind of math, working with scalings of numbers rather than directly with the numbers themselves. However, you can be at ease with them once you develop an intuitive sense for the way they work. Hopefully this appendix has helped with that, and will let you go forward and use them with more confidence.