the best way to count

Lucilla

introduction

let's talk about the Hindu-Arabic numerals. so deeply ingrained in our lives that some people don't even know them by name,[i.a] so widespread that you'll find them in almost every country on the planet, and so ubiquitous that we often forget they are not numbers – just one of many ways of *writing* numbers.

the Hindu-Arabic numerals are a *positional* numeral system: the value of each digit is affected by its position in the string. historically, this was quite a revolution compared to the existing systems, and it made them one of the most important inventions in the history of mathematics. numbers have no upper limit dictated by the limit of symbols – you can write all infinitely many natural numbers using just ten digits. nor does the length of these strings get out of hand quickly – every additional digit gets you ten times further than the previous one did, and even a number in the millions takes just seven digits to write. moreover, positional notation is *injective*: every natural number has exactly one representation.

because we take all these properties for granted, it's easy to forget that they're not at all self-evident, and how something as fundamental as the way we count can make a huge impact on how fast mathematics can develop and progress.

but what if there was a bug with the Hindu-Arabic numerals? a flaw in their design which, despite all their wonderful properties, prevents them from being the best they could possibly be? well, there is such a bug, filed as bug report number ten: the choice of ten as a base.

being the sum of the first four positive integers, the number ten was sacred to the ancient Greeks,[i.b] although a much more likely reason why we use base ten is because of the number of fingers most people have. but the choice of ten is actually arbitrary, independent of positional notation: the same system would work with any integer base greater than 1. and ever since humans understood their own numeral system, they realized that some bases are mathematically better than other bases – and it turns out ten is not a particularly good choice.

that's why on the educational side of YouTube, you have lots of videos talking about how we should all switch to base twelve, called "dozenal". but why stop there? maybe base twelve is better than base ten, but if we should bother using an alternative numeral system, shouldn't we go for one that's even better?

among educational videos talking about numeral systems, one stands out with a particularly in-depth analysis of what makes certain bases better than others. it's called *a better way to count*, and its author, jan Misali, says that their personal favorite base, a base better than both ten and twelve, is base six, which they call "seximal".[2:46] given that *a better way to count* has over a million views, is over 18 minutes long, and has an entire response video answering some common counterarguments, it seems that seximal is a pretty big deal.

now, there's no reason not to let anyone believe in what they think is best, but just so we're on the same page here, jan Misali is wrong. seximal is not the best base out there. so fasten your mathematical seatbelts, because we're going down a deep rabbit hole – on a quest to discover binary, the *actual* best way to count.

---

binary, or base two, is an extreme case. it's the smallest possible positional system, using only two symbols – say, black and white dots – to represent all the natural numbers. it's also a decomposition of numbers into *additive primes* – powers of two, playing a similar role for addition as ordinary primes for multiplication;[3:43] and it's also a maximally efficient game of *twenty questions*, singling out a number's position on the number line by halving the possible range with each bit.[3:50] binary has appeared in ancient Egyptian mathematics,[i.c] in the 16th-century works of Leibniz,[i.d] in the counting system of the people of Mangareva,[i.e] and in the Arecibo message[i.f] – and, of course, in nearly every modern computer.

but binary isn't just the natural base. no, binary is also the base best suited for human use: by nearly any metric for comparing bases to each other, binary is the one that performs the best of all. and the best thing is: the latter is because of the former. binary is the best base specifically *because* it's the smallest possible base – because 2 is the successor of 1.

in contrast, a base like seximal, Misali's favorite, is indeed a very good base, but its merit is almost entirely due to a mathematical coincidence: 6 contains the first two primes and is adjacent to the next two primes. seximal is a cheater in the world of bases, and a more thorough analysis exposes the shallowness of its tricks and reveals a gaping emptiness underneath. meanwhile, binary shines without the need for coincidences – it plays fair and square, and that fairness pays off in the end.

of course, not everyone thinks the same way, otherwise there would be no reason to be making this video. so since it's meant to be a response to *a better way to count*, let's start by looking at what Misali thinks of binary. the following audio clip and image are from *seximal responses*, which has a section comparing the pros and cons of lots of different bases, including binary. so let's listen to what binary has going for it!

> *base two: binary. binary is the smallest base that actually works... at all. doing simple arithmetic in binary is super easy, and only having two digits means storing binary information on a computer is maximally efficient. that's about it for positives! every other aspect of binary is a downside. numbers get real long real fast and the only terminating ratios are the ones where the denominator is a power of two. oh, by the way, the ratios I'm showing in red here are the ones that are as complex as they hypothetically could be. you know, the really bad ones. and, yes, I'm using underlines instead of overlines for recurring digits.*

that's it? really? does the person who made the most in-depth analysis about base six really think that all aspects of binary can be dismissed in just 30 seconds?

essentially, Misali seems to have a "theory of everything" for positional notation, namely that the merit of bases can be reduced just to the period lengths of small fractions. this can be seen in *seximal responses* as well as the program BASE OFF, which picks an optimal base for the user based on feedback from questions that are... entirely about fractional expansions.[i.g] or look at this segment from a collaboration with Artifexian about designing a number system: they pick one power-of-two base by just straight up listing a bunch of fractional expansions and picking the column with the most "winners"![i.h] ...

the way a base writes simple ratios is important, but it's nowhere even *close* to being the holy grail for comparing different bases to each other. it's as oversimplified as saying that a writing system's merit is entirely tied to the number of strokes needed to write simple words. don't worry, we'll go very in-depth about exactly *why* this analysis is insufficient to fully determine the quality of a base, and probably discover some "fundamental truths" along the way.

so let's take a deep dive, on a journey of mathematical truthmaking, to dissect both binary and seximal, and get to the bottom of positional notation.

chapter zero

binary numbers are long

there's no denying that binary representations of numbers are notoriously long. one hundred takes seven binary digits, one thousand takes ten, and one million takes twenty – far more than their decimal, dozenal, or even seximal counterparts. and this is where most people stop when considering binary as an option for human use. but why? maybe the problem isn't with binary itself, but with a specific *notation* of binary.

our choice of black and white dots for the binary digits is arbitrary – and rather atypical. the most common choice are the Hindu-Arabic numerals, 0 and 1. this is a special case of using a subset of the Hindu-Arabic numerals to notate bases smaller than decimal, like how *a better way to count* notates seximal.

for bases of similar size to decimal, this is a natural and logical choice. but for *binary*, it makes absolutely no sense to do this. the two binary digits only need to be distinct from each other, not from eight other unused symbols. this is the equivalent of using Chinese characters to write English, but by just picking twenty-six characters to substitute for Latin letters.

moreover, research in linguistics has shown that even if some languages are spoken faster than others, most convey information at about the same rate – and the same principle applies to written symbols as well.[0.a] so if binary digits are only worth one bit of information, their shapes can be designed much simpler – and much thinner. say, two vertical bars: low for 0, high for 1.[8:43] *now* the comparisons between big numbers in decimal and in binary seem a whole lot more reasonable.

seximal can't really do the same thing: it's significantly smaller than ten where using the traditional ten digits creates a noticeable disadvantage, but it's also not *that* much smaller that you can invent alternative symbols that fix this issue and are simple enough to subvert the traditional digits.[0.b]

but now our numbers look too similar to a barcode. the digits are too hard to read, and also hard to align in the vertical dimension. fortunately, there's a very easy solution to all of those problems, and to see it, we have to look at a similar problem in decimal.

decimal numbers in the thousands and millions are difficult to parse at a glance, so they are usually written with separators, which split the number into digestible, equally sized chunks of digits. in binary, we can do the same thing. join groups of vertical bars with a horizontal bar – usually threes or fours, but you could also do pairs, or unequal groups, or whatever fits.[0.c]

in binary specifically, such groupings can also be treated as units in their own right, like a power-of-two base. triplets of binary digits become digits of base eight; quadruplets become base sixteen, and so on. and it's *these* digits that we can compare to measure number length, or align vertically under each other. such a notation allows us to combine the benefits of all these bases without any of the overhead of converting between them – especially if you can memorize the correspondences.[0.d]

okay, but, are we even still using binary anymore then? aren't we using multiple bases? isn't this cheating?

no, we aren't using any other bases per se. higher bases, with their own set of digits, make the underlying binary no longer explicit: such digits cannot be immediately split into bits and then easily regrouped into another combination of bits.[0.e]

but even if you consider such groups of bits to be entirely equivalent to their corresponding higher-base digits, there's still a difference in how some numbers are represented. consider the number one hundred again: in binary; grouped by threes; and now in octal; and rewritten as triplets of bits.[10:35]

that's right: using power bases means sometimes adding *redundant zeros*.[0.f] and we'll see time and time again that it makes them just plain worse than their non-power counterparts. four is worse than two; one hundred is worse than ten; thirty-six is worse than six.[0.g] even the genetic code is technically more wasteful by being quaternary than if it was binary.[0.h]

finally, one important difference is how human-sized these groupings are. in seximal, the square base is just on the edge of being too large. and base one hundred, compressed decimal, is so big that only Ithkuil dares to use it.[0.i] meanwhile, binary gives us access to at least three or even four higher levels of compression before they become too large to keep track of.

this ability to view the same number at many different scales is one of the biggest and most important perks of binary, one that will keep coming up again and again. binary happens to be tiny enough where such things, often overlooked, are possible, and open up worlds of possibilities.

chapter one

two is a very small number

one thing that matters for a base's efficiency is its size. larger bases can write the same numbers in fewer digits, but their digits also measurably contain more information, so we need to find a compromise between the number of digits and the information cost per digit.

it turns out that one of those factors is a lot more significant than the other. if we compare two similarly sized bases, like decimal and dozenal, then for a long while they'll represent a large percentage of numbers with the exact same number of digits, with only small ranges where the larger base wins by one digit. a permanent difference doesn't happen until a very large number.[1.a]

you can quantify this even further with *radix economy*: an analysis of the long-term efficiency of a base, taking into account both the lengths of numbers in digits and the information content of those digits. since a number's information content is proportional to its logarithm,[1.b] radix economy can be seen as a measure of how close the discrete representations in various bases can come to this ideal. intuitively, smaller bases will perform better here, since they can quantize this information into finer steps.

but there seems to be a misconception about radix economy that, allegedly, base e is the most efficient of all, or among integer bases, base 3.[1.c] this shows up *everywhere* in the discussion of numeral systems, even the Wikipedia article about base 3, and even seximal.net.[1.d]

speaking of which, why is base four "really almost as good as seximal, just a bit smaller", but then binary is so much worse? but we've already established that base four has no advantages whatsoever over binary, it's purely less efficient...

anyway, radix economy. to start comparing different bases for efficiency in writing numbers, we first need a formula for how much information is contained in a given base's representation of a number. to do that, we need to consider a detail we've been ignoring.

so far, we've glossed over the fact that any numeral string in any base can be made longer simply by adding zeros to the left-hand side. this technically makes it ambiguous how many digits such a string actually contains. we can solve this by saying the leftmost digit *cannot* be 0, and we can meaningfully say that this leftmost digit carries slightly less information than all the other digits. so in a decimal number, most digits contain a full decimal digit of information, but the leading digit only contains a *nonary* digit's worth of information, since it only has nine possible values.[1.e]

alright, with that out of the way, we can construct a formula. in a base-b representation of a number n, there are $\lfloor \log_b(n) \rfloor$ digits that can take the entire range of values from 0 inclusive to b exclusive, plus one digit with the slightly smaller range of 1 inclusive to b exclusive. and since the information contained within a digit is proportional to the logarithm of its number of possible values,[1.f] this gives us this formula:

$$\text{cost} = \lfloor \log_b(n) \rfloor \cdot \log(b) + \log(b-1)$$

using this formula, comparing results for different bases, we can come to some conclusions. for similarly sized bases, like two and three, there are regions where one base performs better and regions where the other performs better. the exception is when one base is a power of the other, like two and four: in those cases, the power base is *always* worse.[1.g] no idea what Misali is on about when they say in *seximal responses* that base four is "just a more efficient version of binary": it's clearly the other way around.[1.h]

but we can learn a lot more by dividing the cost by the actual amount of information, the logarithm of the number n itself. after some simplifying, we end up with this neat equation:[1.i]

$$\text{radix economy} = \frac{\lfloor \log_b(n) \rfloor}{\log_b(n)} + \frac{1}{\log_{b-1}(n)}$$

and if we graph this formula to see how it behaves for large values of n, and compare the results for different bases b, we find that base three isn't the most efficient, not by a long shot: binary is.[1.j] in fact, any other base doesn't even come close.

but is this a trick? have we manipulated the definition of radix economy to make binary turn out better? after all, this definition is completely different from the one in the Wikipedia article on radix economy, so what's up with that?[1.k]

the key is that leading digit, carrying slightly less information than the rest. in most bases, this slight decrease in information makes hardly any difference. but in binary, where the only two options for a digit are 0 and 1, excluding 0 means it can *only* be 1. since it can only ever be one value, it contributes *no* information whatsoever.[1.l] it still matters, of course – it contributes to arithmetic, after all – but in terms of actually determining the number's value, it might as well not be there at all. we can meaningfully say the leading bit doesn't count as an actual digit in these strings.[1.m]

this is what makes binary perform so much better than any other base. because it's equal to $1 + 1$, it's the only base b for which $\log(b-1) = 0$.[1.n] and because of the notation we're using, this victory of binary is directly visible in the compactness of numbers – unlike in seximal, where radix economy is just an imaginary concept, to justify that the numbers are in fact *longer*.

so then, where did this misconception with base three come from? well, let's look at the formula shown in the Wikipedia article and see for ourselves:

$$\text{radix economy}^* = \frac{b \cdot \lfloor \log_b(n) + 1 \rfloor}{\log(n)} \approx \frac{b}{\log(b)}$$

firstly, the formula multiplies the number of base-b digits by b itself, not $\log b$, which is wrong: the information contained in an object is proportional to the *logarithm* of the number of its possible states.[1.o] another error is failing to take into account the slightly lower information content of the most significant digit – without it, it becomes a lot harder to see which base is the winner.

but even just looking at the conclusions this formula leads to: not only is base three allegedly the most efficient, but two is *tied* with four as second most efficient, even though we already established that base four is purely a downgrade! it just doesn't add up. base efficiency doesn't have a local minimum around e: it is strictly decreasing.

chapter two

most people have twice one less than six fingers

pretty much any argument in favor of any base will bring up finger counting at one point or another. after all, ten fingers is pretty much *the* reason why decimal is the dominant counting system in the world today, so it would be a mistake to ignore it in this analysis.

in decimal, finger counting involves extending all fingers of one hand first, going from zero to five, and then using the other hand to go from five to ten, and then you're out of fingers. isn't this a bit wasteful, though? for all of these, either one hand has no fingers extended or the other hand has all five fingers extended. what if you took advantage of the fact that you can extend or retract each finger independently and counted like this:[18:08]

you most likely already know about binary finger counting – it's nothing new.[2.a] you might argue that it's not that good of a finger counting system in the first place, since it requires you to control each of your fingers independently of one another, which many people can't do. we'll get to that, but binary finger counting also has a deeper point to make, which we'll discover when we analyze methods of finger counting more broadly.

you know how seximal finger counting is presented as being superior to decimal because it uses each hand as a separate digit of a two-digit number, getting you from 0 to 36?[2.b] it's neat, but check this out: you can do the same thing in decimal as well. use the first four fingers of one hand to count from zero to four, then just the thumb for five, then the other fingers again to get from five to nine. you just counted from zero to ten on one hand, and you can use the other hand to get you from 0 to 100: about three times as far as in seximal. this counting method originates from Korea, and is known as *chisanbop*.[2.c]

the main point here is that having a finger counting system isn't just a true-false question. on the one hand, finger counting can be adapted to virtually any base because of how flexible it is; on the other hand, we can analyze them all together from an outside perspective to see how they compare.

binary finger counting, chisanbop, seximal, and decimal are all constructed according to the same principle: the set of fingers is divided into subsets, then each subset is considered a separate digit whose value can range anywhere from zero to all fingers – thus one more than the number of fingers – and then all of those are multiplied together to arrive at the upper limit of numbers that can be represented. the methods we've looked at can be considered to lie on a spectrum in this construction, from binary treating each finger as a separate digit, all the way to decimal treating all fingers together as one big digit.[2.d]

the key insight from these methods is that they essentially rely on choices between addition, on one end of the spectrum, and multiplication, on the other. and since multiplication grows so much faster than addition, the largest possible upper limit is achieved *only* by considering each finger as a separate digit, which corresponds to the binary method.[2.e] the seximal method, presented as so ingenious and elegant, is actually surprisingly *inefficient*, keeping everything additive on the fingers of each hand. it's only barely better than the worst method, the traditional decimal one.[2.f]

there's also a recognizability issue. if you're shown this seximal number, how can you tell if it's dozen three or thirsy two? you can't.[20:32] even if you establish a convention, it's very easy to make a mistake. binary, however, covers roughly the same range on *one* hand as seximal does on *both*. and in this case, the reading is unambiguous – always starting at the thumb as the least significant bit – because a single hand is chiral, while a pair of hands is not.

but there's an even more subtle point at play here. finger counting systems like seximal, or decimal, or even chisanbop, all rely on that magical number five – and while it happens to work out for most people, everyone else is excluded. the binary method, however, works regardless of how many fingers you count on, because it treats each finger as a unit, without the assumption of belonging to a group of five members. binary counting includes everyone – and every alternate history where humans ended up with fewer or more than five fingers in each hand.[2.g]

so if you can't independently control all of your fingers? ...just use only three on each hand, and you can *still* count higher than with the seximal method. add just one finger, and you surpass chisanbop.

there's even more to binary counting than that. it's not just about fingers: binary counting stands out in how it can be implemented on almost anything,[2.h] and it tickles our human brains with how simple, zen-like, mechanical, and... musical it is. that's right, rhythm in many styles of music around the world is binary: we use binary note lengths and binary time signatures, and many pieces of music are in effect binary counters – you can literally practice finger binary on them. this quote from John Nystrom's own proposal of a binary system sums this up perfectly:[2.i]

> *Now, suppose that a musician is requested to divide [their] notes, bars and burdens into fifths or tenths, according to the decimal system, then ask the musician to play a decimal piece of music, and it will sound very much like the decimal system introduced in the shops and markets.*

chapter three
binary arithmetic is easy

one of the most obvious properties of binary is how simple arithmetic becomes, both written out and mental. however, it's easy to be overly reductionist here and jump to conclusions too early, with some arguments even spinning this aspect of binary into a *negative*. so let's have a look at exactly what implications it has on the basic arithmetic operations.

addition & subtraction: the written algorithm for addition requires us to add two numbers digit by digit, carrying if necessary, thus reducing an addition problem to a sequence of one-digit sums. so across different bases, it appears to be a choice between more digits or more effort to add these digits.[3.a] (subtraction follows the same principles, so we'll only be looking at addition here.)

to add using this algorithm, you need to be able to add single digits first. children usually learn this by doing lots of addition problems and internalizing patterns, and anyone switching to a new base would need to do the same. a table of such patterns is a series of diagonal stripes, which grows linearly with the base's size. in binary, however, single digits are as simple as they can get: there is no hurdle to overcome, and you can start adding things instantly.[3.b] a lot of the time, you don't even need to carry at all, and addition is little more than matching up the bits set to 1 in both numbers – a bitwise OR, if you prefer.[3.c]

in fact, addition in binary is so easy that many people argue it's *too* simple, by which they mean it's not faster enough that this outweighs the need to add more digits. for example, Tab Atkins writes:[3.d]

> *[...] our brain can't arbitrarily reallocate "processor cycles" like a computer can, so when we undershoot we're just wasting some of our brain's ability (and, due to the tradeoffs, forcing something else to get harder). So, we know from experience that binary is bad on these tradeoffs – base-2 arithmetic is drastically undershooting our arithmetic abilities.*

this might be right, but only under the assumption that people always do arithmetic digit by digit – which we don't. we aren't quite as fast as computers, but we make up for it by recognizing patterns. in decimal, we don't need to add digit by digit to add $25 + 75$. in binary, you can do the same thing: once you're able to add *pairs* of digits all at once, nothing's stopping you from doing that. triplets? go right ahead.

but then how is that any different from addition in a power-of-two base? the difference is that in higher bases, you can't reduce single-digit addition any further: once you're on that level and you happen to be stuck, tough luck. you have to first overcome that hurdle of adding single digits. meanwhile, in binary, there's a *learning curve* where you get gradually faster and more proficient, instead of a single hurdle you need to overcome right from the start. and whenever you get stuck at a higher level, you're not helpless: you can always fall back to a lower level for one moment.

moreover, power-of-two bases only let you combine digits in one specific way in certain specific positions, when grouping them in more fluid ways could let you add much faster – and in *fewer* steps. seximal addition may make things easier at the digit level at the cost of needing more steps, while dozenal does the opposite; but *binary* addition makes things easier *and* reduces the number of steps *at the same time*.[25:09]

multiplication: here's where things start to get interesting. written multiplication essentially has you multiply each digit of one number with each digit of the other, then add them up. often these products have two digits; in those cases you have to carry, and this time the carry can take a wide range of values.[3.e]

unlike addition, which usually has no systematic way of teaching its prerequisite – adding single digits – multiplication *does* have such a method: the infamous multiplication table.

the decimal multiplication table has 100 entries: one for each of ten rows and ten columns. but we can eliminate the rows for 0 and 1, because those are trivial, and we can eliminate anything below the main diagonal, because natural number multiplication is commutative,[3.f] leaving us with just 36 entries.

...but wait. was that really necessary? the size of a multiplication table grows quadratically with the base's size, whether or not you obfuscate the table in this way. quadratic functions grow far faster than linear functions of the same argument, so here, larger bases are punished even more, and smaller bases are rewarded even more. *this* is what matters. shaving off individual entries is more of a rhetorical gesture to make one particular base appear nicer.[3.g]

what about binary, then? binary has four entries, all of which are either 0 or 1 – binary being the only base whose multiplication table consists entirely of one-digit numbers.[3.h] we'll get to the significance of this result in a bit. if you insist on obfuscating the table, then it turns out to have *zero* entries, which means you can say binary is the only base with no multiplication table at all!

to see the implications this has, let's look at the algorithm for written multiplication again. we multiply the top number by every digit of the bottom number in turn. in other bases, the top number is also broken down into digits, so that you're only ever multiplying single digits. but in binary, there's no need to do this. the only possible digits are 0 and 1, and multiplying anything by either of those digits is trivial. if it's a 0, just skip it entirely; if it's a 1, just copy the number at that position. because the binary multiplication table only consists of one-digit numbers, *carrying never happens*. the entire process involves just copying the top number at every position that has a 1 in the bottom number, and then adding those up together. because binary is so small, several layers of complexity are removed entirely from the multiplication algorithm. multiplication in binary is so easy that an algorithm used by both Egyptian scribes and Russian peasants to multiply numbers essentially boils down to converting one or both of them to binary.[3.i]

but it doesn't stop there.[27:49] the simplicity of binary multiplication also lets you factorize some numbers just by looking at them,[3.j] and it completely eliminates the need for doubling and halving – the two most common kinds of scaling. the award for making the most cases of multiplication trivial goes to binary.[3.k]

division: the written algorithm for division, called long division, is notoriously difficult compared to the other three elementary operations. at every step, you have to find the smallest part of the dividend greater than or equal to the divisor, and to write a digit of the answer, you have to estimate how many times it goes into the divisor. then, multiply the divisor with the quotient you just found, and subtract that to obtain a remainder to continue with the next step.

but where are you supposed to know that quotient from? there's no "division table" like there is a multiplication table, and it's not like you can do separate long divisions for those particular steps, either. you just have to guess. and even though the guessing becomes easier, faster, and more intuitive over time, it feels hopeless when you first start out. even Misali admits that this is where seximal fails:[3.l]

> *to do division by hand, you gotta do long division, which is bad in any base, and you shouldn't ever have to use it. there isn't a cool thing about how seximal makes long division easy or anything like that; long division just sucks.*

but hang on... say that again. is long division *really* bad in *every* base?

here's what binary long division looks like: at every step, find the smallest part of the dividend greater than or equal to the divisor, and then you can write a digit of the answer. what digit? well, what digit *can* you write? a 1. the part you have to subtract to obtain the remainder for the next step is always just your original divisor. that's literally all you need to do! the rest of the algorithm is the same as in any other base, but its most difficult part is completely skipped, just because binary is so small. no guessing required.[3.m]

square root: hang on, *square root*? since when has that been a thing? you probably don't know any algorithm for working out square roots, and for good reason: if you thought long division was difficult, well, the square root algorithm is layers of complexity above that. ...in most bases, that is.

to take the square root of a binary number, first group its digits into pairs. one digit of the answer will appear above every *pair* of digits of the input. the algorithm is similar to long division in that we'll be using working values from our input, and subtracting some numbers to obtain new working values, until we reach 0, when the algorithm terminates.[3.n]

to start, we write a 1 above the first pair, and subtract 1 from it. we join on the next pair. now we ask ourselves, if we took the part of the answer found so far, and joined on the digits 01 at the end, is our current working value greater than or equal to this? in our case, yes it is, so we write a 1 in the answer above this digit pair, and we subtract that thing we compared it against. we get a new remainder, and we continue in the same way. join it with the next pair. do the same test again? this time the answer is no, so we write a 0 in the answer and we don't subtract anything. finally, joining on the final pair of digits to the working value, and joining on 01 to the answer we know so far, is the working value greater or equal? yes, so we write a 1. and in fact it's exactly equal, which will mean the algorithm will stop here, because this time our difference will be 0. and just like that, we figured out that 13 is the square root of 169.[30:12]

this same algorithm *does* work in decimal too, but it becomes absolutely incomprehensible; there are just so many steps that completely disappear in the binary variant that it can hardly even be called the same algorithm anymore. for instance, the true-or-false question at each step is now a *scale-of-one-to-ten* question about an entire decimal digit, which is answered by solving a miniature quadratic inequality in your head. also, the algorithm involves multiplying by the base times two and adding a one-digit number, which is reduced to a simple concatenation of 01 in the binary case.[3.o] so even if this algorithm feels like just an arcane party trick, it still shows how binary's arithmetical superpower makes even terrifying operations like square roots feasible.

chapter four

two isn't even divisible by three though

the other big thing that matters about bases is their factors. among other things, factors a base considers important influence divisibility tests – quick ways to check divisibility by certain numbers, without having to do actual division. divisibility tests come in many shapes and sizes, but the most common ones are:

- if the final digit of n shares a factor with b, then n also shares that factor with b.
- if the digit sum of n shares a factor with $b-1$, then n also shares that factor with $b-1$.
- if the *alternating* digit sum of n shares a factor with $b+1$, then n also shares that factor with $b+1$. "alternating sum" means it alternates between adding and subtracting digits.[4.a]

the most well-known decimal divisibility tests are examples of the above. in decimal, if the final digit of a number is a multiple of 5, the entire number is, because 5 is a factor of 10. if the digit sum of a number is divisible by 3, so is the number, because 3 is a factor of 9, which is one less than 10. and if the alternating sum is divisible by 11, so is the number, because 11 is one more than 10.[4.b]

but there's more to it than that. base 10 is also base 100 in disguise, since pairs of decimal digits correspond to entire base-100 digits. so by looking at the last *two* digits, you can check if a number is divisible by 25, which is a factor of 100. and with an alternating sum of *pairs* of digits, you can check if a number is divisible by 101, and so on. so in theory, there is a divisibility test from one of those families for *any number* in every base b, because every number coprime to b will divide $b^n - 1$ or $b^n + 1$ for some n: a consequence of Fermat's Little Theorem.[4.c]

however, there is a limit of practicality. you probably learned that there is no divisibility test for 7, and that's kinda true. 7 is coprime to 10, and it also doesn't divide one less than any small power of 10. it *does*, however, divide 1001, which is one more than 10^3. so in theory, you could test if a number is divisible by 7 by grouping its digits into threes and finding the alternating sum – but then the base cases would still be all the multiples of 7 up to 1000, way beyond the scope of mental arithmetic.[4.d]

factors are the biggest selling point of seximal in *a better way to count*. 6 is divisible by 2 and 3, which are the first two primes, and is *adjacent* to 5 and 7, which are the next two primes. this mathematical coincidence makes seximal astonishingly good at calculations involving the first few primes. and at first glance, this appears to be a big weakness for binary: after all, 2 is a prime number, so only calculations involving 2 are made trivial, at the cost of everything else... right?

we can follow the trail of thought in *a better way to count* and compare seximal to binary by setting up a tier list of prime factors for both – kind of like a certain game show.[34:27] here's what that would look like:

2 divides 6; **S tier**.	2 also divides 2; **S tier**.
3 divides 6; **S tier**.	3 doesn't divide 2 or $2-1$, but it does divide $4-1$. **B tier**.
5 doesn't divide 6, but it does divide $6-1$. **A tier**.	5 doesn't divide 2, $2-1$, $4-1$, or $8-1$, but it does divide $16-1$. **D tier**.
7 doesn't divide 6 or $6-1$, but it does divide $36-1$. **B tier**.	7 doesn't divide 2, $2-1$, or $4-1$, but it does divide $8-1$. **C tier**.
11 divides $6^{10}-1$ and $2^{10}-1$. **ultra-F tier** for both.	

if we look at this tier list, then pretty obviously seximal is *way* better than binary, being either tied or significantly better in every category. so what's the big deal? isn't this the argument that shatters binary in the end, despite everything brought up so far? or maybe this analysis is just superficial?[4.e]

to see why the tier list argument doesn't cover the full picture, we need to understand *why* divisibility tests aren't good sometimes. in almost every base, some numbers, called *cyclic numbers*, are pesky, like 7 in decimal, 5 in dozenal, or 11 in seximal. there is a rigorous definition for "x is a cyclic number in base b", but we won't need it quite yet: for now, we can just think of them as those numbers where the smallest n such that x divides b^n-1 is big. it's conjectured that almost every base has infinitely many cyclic numbers – we'll get to exactly which ones don't, but for now we can assume it's basically all of them. in a way, cyclic numbers are a base's Achilles heel, and that's where we usually say "there's no divisibility test".

but the test for 7 in decimal isn't *inherently* bad; the reason *why* it's bad is because it requires knowing all multiples of 7 up to 1000, which is too many to be feasible. these tests require knowledge of all multiples of some number up to a power of the base; but powers of the base grow exponentially, very quickly exceeding the limits of human arithmetic abilities: *this* is what *actually* makes these tests bad.

if we get caught in this trail of thought, though, it's easy to forget that the base's *size* has a big impact on its first few powers, and this entire argument falls apart for binary. 6^3 and 6^4 may be overwhelming, but 2^3 and 2^4 are perfectly fine. 5 is cyclic in both binary and dozenal, but in dozenal, you need to know the multiples of 5 up to 144, whereas in binary, you need to know them up to... um... 4.[4.f] for most bases, only the base itself and maybe its square are small enough, but for binary, you can go up to the fifth power before things start becoming too big. and so, what doesn't work for seximal and decimal can very well work for binary.

here's a few examples on some actual divisibility tests. the first few odd numbers have this nice sequence of tests that's very easy to learn.

- $3 = 4 - 1$: sum of pairs of bits.
- $5 = 4 + 1$: alternating sum of pairs of bits.
- $7 = 8 - 1$: sum of triplets of bits.
- $9 = 8 + 1$: alternating sum of triplets of bits.

3 even has an alternative test, using the alternating sum of single bits. and in reality it's often even easier, because remember that many binary numbers can be factorized just by looking for repeating patterns.[4.g]

divisibility tests also have implications for things such as primality testing: quickly ruling out numbers that clearly aren't prime. it's true that in seximal, every prime other than the factors of six ends in either 1 or 5 – a fact noted by Leibniz himself.[4.h] but if you allow tests other than just looking at the final digit, then both binary and seximal are tied here, with 11^2 being the first number that "looks prime" but actually isn't.[4.i]

as an aside, what about testing for perfect squares? r/seximal says that, after removing an even number of zeros, the last seximal digit of any perfect square is 1, 3, or 4 – only 3 out of 5 options.[4.j] but in *any* base, it's at most half of all the possible digits rounded up, so this is actually the worst case.[4.k] in binary, on the other hand, after removing an even number of zeros, the last *three* bits of any perfect square are 001.[4.l]

binary also has an unexpected benefit from only having one prime factor: it automatically factors any number into a power of two (the trailing zeros) and an odd number (the remaining bits). these are orthogonal to each other in many ways, and can be used to split up many tasks. for example, to test if 12 divides 72, you can test separately if their power of two parts divide – so if 4 has at most as many trailing zeros as 8 – and if their odd parts divide, so if 3 divides 9.[4.m]

such a factorization is far less useful in bases like seximal, with both 2 and 3 as prime factors, because it lumps them into a single power of six component – as if it was trying, but failing, to juggle both two and three, so it ended up taping them to one another and calling it a day. for example, to test divisibility by 2^3 in seximal, you need to look at a number's last *three* digits, essentially as if you were testing for 6^3. and trailing zeros in multiplication are always additive in binary – not so in seximal, where "2" × "3" = "10". so even though seximal can test both 2 and 3 just by looking at the final digit, while binary needs a digit sum test for 3, in the long run binary has the advantage of simplicity and regularity.[4.n]

so then, from what we've seen, it might appear that binary is tied with seximal when it comes to divisors, both failing at 11 – the first prime that's not adjacent to a power of two, and the first that neither binary nor seximal can easily handle. but there is a tie-breaker to this tie. 11, and similar pesky numbers, hide a deeper mystery. what if binary had a way to perform a divisibility test by *any* arbitrary number? one that does not involve division, and works no matter how annoying the number is?

let's write down the sequence of powers of two modulo 11. another way to phrase it is that we start with 1, and for every next entry, we double the previous, but subtracting 11 if we exceed it. this sequence will go 1, 2, 4, 8, 5, 10, 9, 7, 3, 6, and then 1 again, looping around. this is the "magic sequence" for 11: in a moment you'll understand its namesake. let's take some binary number, say, this one right here. we'll write the magic sequence underneath the bits, starting at the least significant one, like this.[40:45] and now, if we add up all the terms underneath a 1 bit, the result is 11, which means we just verified that 66 is divisible by 11 in binary![4.o] and even though this is most useful for cyclic prime numbers, it actually works with *any* number. that's right: you can test divisibility of any number by *any n whatsoever* by writing the magic sequence for n underneath the number's bits and then adding up all the terms underneath a 1 bit: if the result is divisible by n, then so is the original number.[4.p] in fact, it turns out *every* divisibility test we've looked at so far is some special case of these magic sequences.[4.q]

you *could* do the same test in higher bases, but it becomes significantly more difficult to the point where it's not viable. for one, to generate the magic sequence, you have to write the powers of the *base*, meaning that you'll overflow a lot faster, often *multiple times per entry.* and then on top of that, you have to multiply each term in the magic sequence by the digit it corresponds to, and add up all those results.[4.r]

in seximal, there is just no other way to counter the argument that you can't divide by eleven, other than to bury it away and say that you'll never need to use eleven anyway. but binary has a doable *universal* divisibility test. so while every base has its Achilles heels when it comes to divisibility tests, binary, just due to its simplicity, can fight back and persevere where other bases just completely give up.

chapter five

binary fractions glow red, and red is bad

even though positional notation was originally only invented for natural numbers, extending it to include *negative* powers of the base allows us to notate fractions as well, by introducing a new symbol called the *radix point*. in this notation, rational numbers fall into two categories. some have a terminating expansion, if they can be multiplied by some power of the base to yield an integer; the others have an infinite expansion, but eventually fall into a periodic pattern that repeats forever: these are *recurring fractions*. it might not be obvious at first, but every rational number in every base is either terminating or recurring – this follows from Fermat's Little Theorem plus the formula for the sum of a geometric series.[5.a]

as before, decimal provides the most well-known examples. 1/2 is written as .5, because it's equal to 5/10, and 1/5 is written as .2, because it's equal to 2/10. fractions with denominators containing primes other than 2 or 5 will be recurring in decimal, such as 1/6, written as .1666..., with infinitely many sixes afterwards – "point one rep six", if you will. traditionally, to notate a recurring fraction, the *entire* recurring segment is marked, but that doesn't make a lot of sense: it's simpler to just mark where it *starts*.

and that's the approach we'll be using for our binary notation. we will be using this symbol for the radix point – a little below the baseline – and this symbol for the... "recurring point". also, if they both occur in the same position, the radix point is omitted. so 1/2 is .1, 1/3 is r01, and 1/6 is .0r01.[43:40]

there are a few simple patterns that show up in every base b: .1 always means $1/b$, r1 always means $1/(b-1)$, and if we denote the base's largest digit by Z, then r0Z always means $1/(b+1)$. for example, in decimal, .1 means 1/10, r1 means 1/9, and r09 means 1/11.

these patterns occur because these fractional expansions are closely related to the divisibility tests we looked at earlier. 1/9 has a period of 1 because 9 divides $10-1$, and 1/11 has a period of 2 because 11 divides 10^2-1. another connection is to cyclic numbers. if k is cyclic in base b, then the period of $1/k$ will attain its maximum in base b: *this* is what a cyclic number means. for example, 5 is a cyclic number in both binary and dozenal, and in both of those 1/5 has its longest possible period, which is 4 digits.[5.b]

in *a better way to count*, these links are taken to the extreme, and the two are assumed to be the same thing. as a result, it puts *way* too much emphasis on how different bases write the reciprocals of the positive integers. again, it seems like there's this underlying assumption that the period lengths of these representations can say *everything* about the merit of a base.

once again, to expose why such a logical leap is incorrect, we can follow it to its conclusions. let's look at the expansions of some simple fractions, up to 1/12, in both binary and seximal. of these, in binary, 1/3, 1/5, 1/9, and 1/11 are all cyclic, being the longest they could possibly be, and 1/6, 1/10, and 1/12 also share their period lengths. they're all marked in red. meanwhile, in seximal, only 1/11 is cyclic, and its period is the same length as for binary, while all other fractions in this range are much shorter.[5.c]

judging by this table, seximal is clearly superior to binary. but check this out: let's add base 4 into the mix. suddenly, things are looking fishy. base 4 has *no* ratios that glow red, no matter how far you go, because it has no cyclic numbers. that's because base 4 is just a compression of the information already present in base 2 into pairs. and since all cyclic numbers in binary correspond to even periods,[5.d] their length is halved when represented in base-4 digits, so it'll never reach its maximum. the same is true for all other *square* bases: those are precisely the ones that have finitely many cyclic numbers, possibly none at all.[5.e]

but we already know that doesn't mean base 4 is better than base 2. those two bases convey *exactly* the same information, only base 4 does it slightly worse because it requires grouping bits into pairs. which actually is also visible in this picture: note how 1/7 is 3 digits in base 2, and also 3 digits in base 4 – that's because grouping a period of 3 bits into pairs doesn't make the period length any shorter. this problem shows up everywhere in power bases, here it is yet again: enforcing one particular grouping. we know for certain that base 4 is worse than binary, and yet according to this table, it looks not only better than binary, but better than *seximal* – since seximal has infinitely many cyclic numbers, whereas base 4 has none.

the error is, again, that something like 4 binary digits in the representation of 1/5 are treated as *inherently* bad just because it's a number of digits corresponding to a cyclic number. but just think about it for a moment: are the 4 binary digits for 1/5 *really* just as bad as the 4 *dozenal* digits for 1/5? is r0011 *really* just as bad as r2497? and if base 4 is better than binary, why isn't niftimal better than seximal? both of those are just their respective lower base, but with number lengths rounded up to the next even number.[5.f]

what we *actually* need to be looking at is the number of digits weighted by the logarithm of the base itself – there's no reason to do that when talking about radix economy and then turn it off for fractions.[5.g] this eliminates the bias that presents base 4 as better, since it accounts for the fact that each base 4 digit is worth two binary digits. the failure to take this factor into consideration is the single biggest error that permeates nearly all discussion of fractions in seximal. it's present in the comparison of various bases on seximal.net.[48:07] it's present in BASE OFF, which will never recommend binary to you if base 4 is also an option.[48:10] and it's present in the Artifexian video, using a metric that depicts binary as the worst power-of-two base – here's the correct version, if you're curious.[48:15]

if we compare with this metric in mind, all of a sudden the proportions change dramatically. base 4 is now visibly worse than binary – always either worse or equal – and *seximal* is better than binary in this range only for 1/5, 1/6, and 1/9.[5.h] note for example that even though 1/7 in binary contains *three* digits, those three digits together actually cost *less* than the *two* digits required to write 1/7 in seximal. in fact, any time a cyclic number occurs in any higher base, binary is guaranteed to be more efficient.[5.i]

but of *course* binary's score approaches seximal if we give it such a huge advantage: doesn't it just mean seximal is so good that its scores are still similar despite being three times bigger? well, there's even more to this fraction argument than that. as *a better way to count* correctly established, the true deal with fractions isn't that some contain more information than others, but that some are easier to re-derive from scratch than others.

in decimal, if for some reason you forget how to write 1/4, it's not that difficult to figure out from scratch. since 4 is a factor of 100, all you need to do is divide 100 by 4 and write the result shifted 2 decimal places to the right: .25. on the other hand, if you forget how to write 1/7, you're basically out of luck. being able to not only figure out that 7 is a factor of 999,999, but then also dividing them in your head and finding that the quotient is 142,857, isn't something that anyone would expect you to be able to do.

in *binary*, on the other hand, you can derive almost any fraction almost instantly. check it out:

- 1/2, 1/4, and 1/8 are .1, .01, and .001, respectively.
- $3 = 4 - 1$, so 1/3 is r1 in base 4, and 1/6 and 1/12 are just a half and a quarter of that respectively, so they just add more zeros at the start.
- $5 = 4 + 1$, so 1/5 is r0Z in base 4, and 1/10 is just half of that, so just add a zero at the start.
- $7 = 8 - 1$, so 1/7 is r1 in base 8.
- $9 = 8 + 1$, so 1/9 is r0Z in base 8.

that's *all* the ratios in the range we've looked at, except for 1/11, derived from scratch in a matter of seconds. in particular, binary's first three cyclic numbers – 3, 5, and 9 – no longer look that bad anymore: they're all of the form $2^n + 1$, and they have periods of length $2 \times n$, where the left half is all zeros and the right half is all ones. they're not at all pesky like typical cyclic numbers are.

in seximal, many of those are about as easy. however, 1/8 requires us to divide 6^3 by 2^3, which amounts to calculating 3^3 – which, if you're new to seximal, isn't brain dead trivial. and 1/10 involves the product of a prime in the base with a prime outside of the base, and is also somewhat tricky to compute.

this connects very well to a point about terminating expansions. yes, in seximal, any fractions with powers of two *or* three in the denominator will terminate, whereas in binary it has to be just powers of two – but that might not be such a bad thing. after all, *almost all* fractional expansions in *any* base will be recurring,[5.j] so in the long run a base should face its recurring expansions honorably, rather than trying to flee from them as much as it can. the compatible denominators for seximal might be common among the first few numbers, but if you mark them for a bigger range, you can see how quickly they become extremely rare.[51:37] also, in a base divisible by two prime factors, like seximal, we've seen how the reciprocals of the powers of one prime need some effort to derive, and mixing primes inside and outside the base leads to problems as well. in binary, both of those are avoided – this is because, again, we can handle any number simply by factoring it into a power of two component and an odd component.[5.k]

there's even more to see, though.[52:07] one comment on *a better way to count* talks about how balanced dozenal can allegedly combine the power of dozenal factors with the simplicity of seximal fractions, saying in particular that fifths are two recurring digits and sevenths are three.[5.l] how can that be? after all, balanced dozenal is still just dozenal, so how can it have fractional expansions that are half as long? looking at the website reveals that, as expected, the claim is false – at least partially. 1/7 in balanced dozenal indeed has *six* recurring digits, it's just that half of them are the negatives of the other half. still, that might seem astonishing at first... until you notice something else.

take a look at binary again. the first three cyclic numbers are all zeros followed by all ones, but other cyclic numbers also follow a generalized pattern: the left half is the bit negation of the right half. binary does the same halving thing as balanced bases do.[5.m] in fact, not just binary: *every base* does this – this is known as Midy's theorem.[5.n] but did you ever notice that in decimal 1/7, the left half is the nines' complement of the right half? did you notice it in dozenal 1/5? in seximal 1/11? just because of how noticeable it is in binary specifically, it probably took you all the way until binary to notice it ever existed in the first place.[5.o]

but let's get back to the star of our show: the elusive 1/11. the number that both seximal and binary shun so much... or *do* they?

remember the "magic sequence" thing from divisibility tests? let's bring up the magic sequence for 11 one more time: 1, 2, 4, 8, 5, 10, 9, 7, 3, 6, 1. now, between every pair of numbers, let's mark where the numbers are ascending and where they're descending, and write a 0 at every increase and a 1 at every decrease. and because the magic sequence loops indefinitely, so does our sequence of bits, and... hang on, this *is* 1/11 in binary![5.p] just like that, without doing any cumbersome division by huge numbers in our heads, we figured out how to write 1/11 in binary using the magic sequence for 11.[5.q]

if we *really* wanted to do this in higher bases, we *could*, but we'd run into two issues. the first is one we already know: to generate the magic sequence, we need to multiply numbers by the *base* at each step. but the second issue is a lot bigger. remember how the numbers in magic sequences in higher bases grow so fast that overflow can occur multiple times? well, get this: in higher bases, to determine the digit at a particular position, you have to know *how many times* overflow has occurred from one position to the next. in other words, you have to do not just a straightforward modulo calculation, but straight up *division*, at each step of the magic sequence. once again, a method for which binary is simple enough completely falls apart for higher bases.[5.r]

magic sequences are an extremely powerful tool in binary. they're like a number's DNA sequence: they're trivial to compute, and they unlock all sorts of things that would require far more complex methods in higher bases – so much that perhaps learning *them* would be the binary equivalent of learning the multiplication table. so while seximal handles the first few primes astonishingly well, with higher primes, it doesn't even give you the option. meanwhile, binary gives *all* the numbers equal treatment. seximal may win a sprint, but binary wins the marathon.

chapter six

how do I say these numbers with my mouth

at the end of the day, counting systems don't exist in a vacuum: they exist within languages. this topic is intentionally at the very end: to demonstrate that binary is a big deal even in the absence of a system of number names, rather than introduce one right at the start – and also because even here, there's a surprising amount of depth in how words for numbers work.

at first, binary seems like a nightmare for pronouncing numbers: surely all this "one zero one zero" nonsense would drive you crazy. but most counting systems don't just read off all the digits of a number in sequence. instead, the most common approach seems to be to follow each nonzero digit with a word for the power of the base that it represents – so that, for instance, "three two five" becomes "three hundred two ten five".[55:56] even though this seems more costly, it has several advantages, such as letting you know a number's rough size right away, and being shorter for numbers with many zeros.[6.a]

if we were to follow this method, what we'd need is words for each digit in the base, and words for every power of the base. but, again, it still seems that binary is the worst base to design such a counting system for. its powers are the most dense, so you need lots of words to cover the same range where you'd get by with far fewer words in other bases.

but we've overlooked something crucial. take a step back and look at decimal. do we *really* have words for every power of ten? we call 10^2 "hundred" and 10^3 "thousand". but what about 10^4? 10^5? the next named power of ten is 10^6, "million". and in general, we only give powers of 10^3 unique names, and fill in the gaps with a kind of sub-base of three.[6.b]

but why stop there? you could name powers of 10^3 up to 10^9, and then start only naming powers of 10^9. in fact, you can do it smarter: instead of naming *three* powers before skipping, you can do just *two*. in other words, the only powers of ten you really need to name are those where the exponent is a *power of two*. this was the concept behind the Chinese long scale,[6.c] or Donald Knuth's -yllion system.[6.d]

so in binary, we only need to name... the double powers of two. the super-powers of two. the meta-powers of two. whatever you wanna call them. and it turns out that naming only these powers makes it so we actually need *fewer* names in the long run than larger bases: decimal, for instance, would need sixteen number names to exceed a googol, while binary only needs ten.

if we analyze the number of distinct words necessary up to a number n, we can see why that's the case. a higher base needs more digit names than a lower base, with a disadvantage equal to their difference. if we name *all* the powers of each base, then the higher base more than makes up for this initial disadvantage. that's because the number of power names required to get to n is proportional to the logarithm of n, so the advantage of using a higher base at n is equal to the difference of the logarithms of n in the two bases, and will eventually – for a large enough value of n – always outweigh the initial disadvantage. however, if we only name *power-of-two* powers of each base, *now* the number of power names up to n is proportional to the *double* logarithm of n. and now the higher base is no longer *increasingly* better for bigger values of n; it now only has a *constant* advantage, being the difference of the double logarithms of the bases. but linear

functions grow way faster than logarithms, let alone *double* logarithms; so the disadvantage from needing more digit names will always outweigh the advantage of needing one or two fewer power names. surprisingly, efficiency again increases as the base decreases, reaching its maximum for binary.[6.e]

right, but what should the names for the double powers of two be? well, here's the best part. we don't need to invent any – computer scientists have already done it for us: *two*, *four*, *hex*, *byte*, *short*, *int*, *long*, perhaps *overlong*, and finally, 2 to the power of byte could be *byteplex*.[6.f] just these few names, most of which are one syllable, are enough to reach the number of particles in the visible universe.[6.g]

in binary specifically, we get an additional unexpected advantage from having the exponents themselves follow a binary pattern: orders of magnitude become much easier to use. to represent *any* power of two, just list the double powers of two under its bits and pronounce the ones underneath a 1 bit, in ascending order. in decimal, the exponents aren't themselves decimal, but a kind of base 3; neither are they seximal in seximal, but a kind of base 4. meanwhile, in binary, these two systems are one and the same.

okay, how do we convert to and from this system? to pronounce a number, break it up at its largest double power of two, and represent the left and right sides recursively, with the name of the double power of two between them. parsing a number works the other way: find the largest double power of two in your word string, and your number is equal to the left side times that double power of two, plus the right side. both of those algorithms may seem arcane – they involve recursion, after all[6.h] – but they're more for completeness; in reality there's a clear pattern to these number names, one that's very easy to pick up on.[1:00:17]

as it stands, there still is one problem with this system: many names for small numbers are pretty long. but there's an easy way to fix that: we already have names for such numbers, so we can carefully reintroduce them in places. *three* is a good choice, as a shortening of the very common combination *two one*, essentially making the system quaternary (wait a second. wasn't base 4 supposed to be bad? not in this case; read the footnote if you're curious).[6.i] we can also give some small powers of two their own names even if they're not double powers of two: *eight* comes to mind.[6.j] as does eight squared,[6.k] since there already is a word for that very number in common parlance.[1:00:53]

now, sit back and enjoy, and watch some names of numbers scroll by.

chapter seven

Mersenne primes, Fermat primes, and the truth about two and six

this is it. we've hit rock bottom. at every step we've confronted more aspects of the way we count, and it seems there isn't much more to see. everything, from size to arithmetic, from counting to factors, points to the conclusion that being the smallest base is a strength; but to reach that conclusion, we had to untangle assumptions, break conventions and think twice on almost every occasion.

counting in binary requires relearning a lot of what we know about counting, but for a reward that's well worth it. and maybe this is why we see so many proposals of a dozenal or hexadecimal system, but hardly anyone has had the audacity to propose a binary one. and we've barely scratched the surface. we missed binary tally marks,[7.a] balanced ternary,[7.b] two's complement notation,[7.c] a binary monetary system,[7.d] binary timekeeping,[7.e] the dot notation in music,[7.f] and so much more.

this final chapter is not a reprise of all the points we've made so far as much as it is a philosophical exploration of how binary permeates the fabric of existence itself. it's not a case for binary being efficient and useful as much as it is a case for binary being fundamental and natural. if you don't care much for questions like these, then there's nothing more beyond this point, but maybe you can stick around anyway.

---

the Mersenne primes are a famous subset of primes all one less than some power of two. remarkably, the exponent of this power of two must itself be prime; if it was composite, say, 6, we could immediately find a repeating pattern in the binary expansion of $2^6 - 1$ and therefore factorize it. the predecessors of the powers of any other base can never be prime, because they would all repeat a digit greater than 1, and so every such number is divisible at least by that digit.[7.g]

another famous subset are the Fermat primes, all one *more* than a power of two; this time, the exponent must itself be a power of two. otherwise, it would have an "odd part", and you could find some alternating sum test which its binary expansion would pass by making the two ones cancel out. this time, other bases can generate Fermat primes – 37 is a seximal Fermat prime, 101 a decimal Fermat prime – but no matter the base, the *exponent* must be a power of specifically two.[7.h]

of course, the powers of two occur throughout mathematics with astonishing frequency[7.i] – but more meaningfully, they do so for the same reason that binary triumphs over all other bases: because 2 is the smallest integer greater than 1.

six, on the other hand, may have lots of spectacular properties: it's an antiprime, a triangular number, a perfect number, and many more all at once – but most of those are no more than arithmetical curiosities, nuggets of trivia from a book on recreational mathematics; and certainly having no connection to the value of seximal as a base.

so, yes, six may be a perfect number, but *all* the currently known perfect numbers are the product of a power of two and one less than the next power of two; and while no power of two is a perfect number, *all* the powers of two are the *only* currently known *almost*-perfect numbers.

six may be the integer part of the true circle constant.[7.j] but two is the integer part of the exponential constant e, and this is more than just a coincidence: the same role that e^x has in standard infinitesimal calculus is played by 2^x in "Newton's calculus" of discrete steps of change.[7.k]

six may be a convenient number to count up to, but so is *every* power of two, because of how much they reduce counting to just a feeling of rhythm. six may be a better base for orders of magnitude than ten, but two is even better, giving an estimate of size that's often good enough on its own, without a mantissa.

six may be an antiprime, but so is two. it may be a superior highly composite number, but so is two. both two and six are the only antiprimes that are also primorials, but two is the only antiprime that's also... a prime.

with how ubiquitous the powers of two are – there's even a whole *game* about them[7.l] – they may very well be the most important integer sequence. even Misali seems to agree, as the powers of two are the *very first* entry on the list of things worth memorizing if switching to seximal.[7.m]

two is so important that many languages have an entire grammatical number for it: the dual.[1:07:15] no, wait, *this* one.[7.n] and remind me what number Toki Pona can count up to? not six, that's for sure.[1:07:19]

perhaps cells divide into two because 2 is the smallest integer greater than 1 – because duplication is the least effort that produces growth. chromosomes and DNA base strands are paired because the smallest number of spares you can have is one – "duplication" and "copying" are basically synonyms.

perhaps electrons and positrons form a pair because 2 is the smallest integer greater than 1 – if there was only one, there'd be no conservation of charge; any number greater than 1 works, of which 2 is the smallest. duality is the simplest possible system of opposites – maybe that's why it's so common.

binary counting is fundamental to rhythm,[1:08:00] but that's not its only musical aspect: it turns out *pitch* is binary too. raising a note by an octave doubles its frequency, so octave equivalence, such a big part of our music perception that we give octave-equivalent notes the same names, is just the equivalence of binary representations that differ by at most a bit shift. so you could say our ears have been counting in binary this whole time. perhaps that too is the case because 2 is the smallest integer greater than 1 – because $2:1$ is the simplest integer ratio after $1:1$.[7.o]

the wonders of binary could go on, but this story about binary ends here. maybe you're convinced that binary is the best way to count. maybe not. maybe you're not sure yet, but you came out knowing more than you did before.

numeral systems are important to nerds, but what's more important is coming to your own conclusions instead of accepting one person's opinion as fact. in the end, none of this really matters, because no numeral system is better than any other at depicting numbers as the abstract entities they really are, and choosing one over another is just an artifact of our human desire to compete, to improve, to optimize.

but in a way, that's also what makes this discussion so valuable and beautiful. if we could refer to numbers directly, no numeral system would ever exist, and maybe we'd never discover the deep and wonderful math behind it all. there'd never be an exchange of opinions or a trip down rabbit holes. it's our own limitations that make us strong, and understanding that is what really counts.[7.p] ...in binary, that is.[1:11:48]

the best way to count
a video essay in response to *a better way to count* by jan Misali

concept, script, programming, visuals, animations, music, etc. by Lucilla
narrated by kepe
additional segments narrated by Addy

seven great gross four gross two dozen eleven words
thirsy four nif nine slides
MCIX lines of code
one cute fraud