Logarithms

If $a^x = N$ then $x \equiv \log_a N$, and we have the very useful identity $a^{\log_a N} = N$. Letting $x = a^b$ and $y = a^c$, we get the following laws of logarithms:

logarithm rule	**corresponding algebra**	**comments**
$\log_a(xy) = \log_a x + \log_a y$	$a^b a^c = a^{b+c}$	multiplication adds exponents
$\log_a(x^p) = p \log_a x$	$(a^b)^p = a^{pb}$	exponentiation multiplies exponents
$\log_a(x+y)$	$a^b + a^c$	does not (in general) simplify

Notice that $a^{\log_a N} = N = b^{\log_b N}$. By taking $\log_b$ of both sides and rearranging, we get the change-of-base formula, which allows us to compute $\log_a$ of a number even if we only know how to do $\log_b$:

$$\log_a N = \frac{\log_b N}{\log_b a}$$

Note that $\log_b a$ is just some constant, a fact that will be useful when discussing big-O notation.

Writing Numbers In Any Base

We can expand any number as the sum of contributions of each of its digits:

$$\begin{aligned} 2107_{10} &= (2 \cdot 1000) + (1 \cdot 100) + (0 \cdot 10) + (7 \cdot 1) \\ &= (2 \cdot 10^3) + (1 \cdot 10^2) + (0 \cdot 10^1) + (7 \cdot 10^0) \end{aligned}$$

The same could be done in any other base, e.g., base-2:

$$\begin{aligned} 2107_{10} =& (1 \cdot 2048) + (0 \cdot 1024) + (0 \cdot 512) + (0 \cdot 256) + (0 \cdot 128) + (0 \cdot 64) + \\ & (1 \cdot 32) + (1 \cdot 16) + (1 \cdot 8) + (0 \cdot 4) + (1 \cdot 2) + (1 \cdot 1) \\ =& (1 \cdot 2^{11}) + (0 \cdot 2^{10}) + (0 \cdot 2^9) + (0 \cdot 2^8) + (0 \cdot 2^7) + (0 \cdot 2^6) + \\ & (1 \cdot 2^5) + (1 \cdot 2^4) + (1 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0) \end{aligned}$$

which gives us $2107_{10} = 100000111011_2$. It is important to realize that the exponent pattern continues to the right of the radix (e.g., decimal or binary) point:

$$\begin{aligned} 4.75_{10} =& (1 \cdot 4) + (0 \cdot 2) + (0 \cdot 1) + (1 \cdot 0.5) + (1 \cdot 0.25) \\ =& (1 \cdot 2^2) + (0 \cdot 2^1) + (0 \cdot 2^0) + (1 \cdot 2^{-1}) + (1 \cdot 2^{-2}) \\ =& 100.11_2 \end{aligned}$$

The general case of base-r is shown in Figure 1.

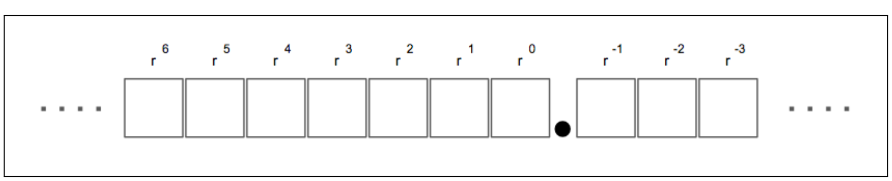

Figure 1: Base-r

Important Number Bases In Computer Science

You should get comfortable with bases 2 (binary), 8 (octal), 10 (decimal), and 16 (hexadecimal). This means writing values in each of them, converting values between them, and (especially in binary) how to do basic math (addition, subtraction, multiplication, division).

	base					**base**			
value	**2**	**8**	**10**	**16**	**value**	**2**	**8**	**10**	**16**
0	0	0	0	0	11	1011	13	11	B
1	1	1	1	1	12	1100	14	12	C
2	10	2	2	2	13	1101	15	13	D
3	11	3	3	3	14	1110	16	14	E
4	100	4	4	4	15	1111	17	15	F
5	101	5	5	5	16	10000	20	16	10
6	110	6	6	6	17	10001	21	17	11
7	111	7	7	7	18	10010	22	18	12
8	1000	10	8	8	19	10011	23	19	13
9	1001	11	9	9	20	10100	24	20	14
10	1010	12	10	A	21	10101	25	21	15

Conversions between binary and octal (or hexadecimal) are easy, since each octal (or hexadecimal) digit maps to exactly 3 (or 4) binary digits, starting from the radix point. For example,

$$\begin{aligned} 1111010110_2 =& 1\ 111\ 010\ 110_2 = 1726_8 \\ =& \ 11\ 1101\ 0110_2 = 3D6_{16} \end{aligned}$$

Converting between decimal and any of the other bases is usually done only approximately. Make sure you remember that:

$$2^{10} = 1024 \approx 1000 = 10^3$$

since that will be very handy. Also, $\log_{10}(2) \approx 0.3$ and $\log_2(10) \approx 3.33$.

Note that any value written in base-r shifts one place to the right when divided by r.

Sequences, Series, Sums, and Products

You should be comfortable with sum and product notations:

$$\sum_{i=L}^{H} f(i) = f(L) + f(L+1) + f(L+2) + \cdots + f(H-2) + f(H-1) + f(H)$$
$$\prod_{i=L}^{H} f(i) = f(L) \cdot f(L+1) \cdot f(L+2) \cdots f(H-2) \cdot f(H-1) \cdot f(H)$$

for instance:

$$\sum_{x=4}^{9} 2x = (2 \cdot 4) + (2 \cdot 5) + (2 \cdot 6) + (2 \cdot 7) + (2 \cdot 8) + (2 \cdot 9)$$
$$\prod_{k=-2}^{2} (k-1)^2 = (-2-1)^2 \cdot (-1-1)^2 \cdot (0-1)^2 \cdot (1-1)^2 \cdot (2-1)^2$$

We can define factorial this way:

$$N! = \prod_{k=1}^{N} k = 1 \cdot 2 \cdots (N-1) \cdot N$$

It is easy to prove that:

$$1 = 0.999\bar{9}_{10} = 0.111\bar{1}_2$$

which is handy because it makes following sum easy to remember:

$$\sum_{x=0}^{\infty} \frac{1}{2^x} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = 2$$

Another commonly occuring sum, especially when dealing with trees, is:

$$2^{N-1} + 2^{N-2} + \cdots + 4 + 2 + 1 = 2^N - 1$$

which is easy to remember if you think about binary numbers. Choosing $N = 5$ gives:

$$11111_2 = 100000_2 - 1$$

When a series of independent decisions is made, the number of possible outcomes is the product of the number of choices for each decision:

$$(\text{\# possible outcomes}) = \prod_{i=1}^{\text{\# decisions}} (\text{\# choices for decision } i)$$

Since we can choose to store one of two different values (0 or 1) in a bit, this result tells us that the number of values V we can store in B bits is:

$$V = 2^B$$

Asymptotic Growth and Big-O Notation

We say that $f(x) = O(g(x))$ if there exist constants c and n_0 such that $f(n) \leq c \cdot g(n)$ for any $n \geq n_0$.

Essentially, this means that $g(x)$ eventually grows at least as fast as $f(x)$.

Looking at Figure 2, we can see that $x = O(x^2)$. It is also the case that $x = O(x \log x)$, and even $x = O(x)$. Because constants are absorbed by the big-O notation, $x = O(4x)$ and $4x = O(x)$. But $x \neq O(\log x)$, since $\log x$ will never "catch up" to x.

The equals sign in big-O notation is unfortunately quite deceptive, since equality is not really implied. It would be better to think in terms of set membership: $f(x) = O(g(x))$ really means that $f(x)$ is a member of the set of functions which do not grow more quickly than $g(x)$.

Using this idea, we can rank different growth rates as follows:

$$O(1) \subsetneq O(\log x) \subsetneq O(\sqrt{x}) \subsetneq O(x) \subsetneq O(x \log x) \subsetneq O(x^2) \subsetneq O(3^x) \subsetneq O(5^x)$$

where $\subsetneq$ means "is a strict subset of".

There are a few things to note:

- We didn't say which base of logarithm we're using, and it doesn't matter. The change-of-base formula says that converting between bases is the same as multiplying by a constant, and that constant is just absorbed by the big-O notation.
- Even though the base doesn't matter for $\log x$, it *does* matter for exponentials.
- Technically, $\log x = O(x^c)$ for any $c > 0$, so *any* polynomial in x (including *any* root of x) will eventually dominate $\log x$.
- Any exponential c^x with $c > 1$ will eventually dominate any polynomial in x.

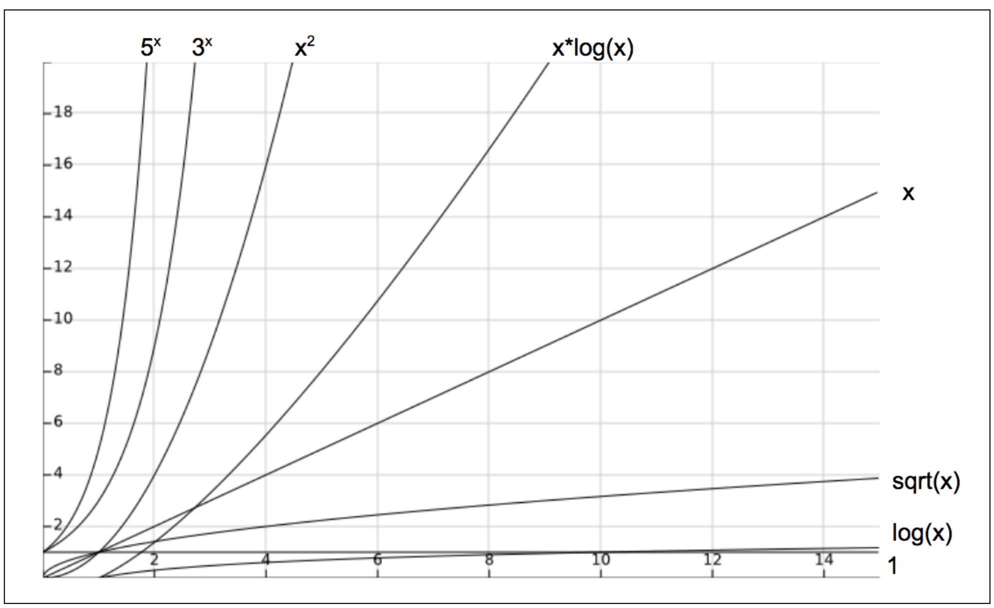

Figure 2: Asymptotic growth

Digits, Logarithms, and Trees

We can think as $\log_r N$ as:

- (roughly) the number of base-r digits needed to write N
- (roughly) the number of times we can divide N by r before the result is between zero and one
- (roughly) the height of a balanced r-ary tree with N nodes

Let's consider the value 24 and base-2 in detail.

We already know that the number V of values that can be stored in B bits is $V = 2^B$, so:

$$B = \log_2 V$$

In our specific case, $B = \log_2 24 \approx 4.6$, so 5 bits are needed to store the value 24:

$$24_{10} = 11000_2$$

Note that this approximation is off by one bit when $N = 2^k$ exactly.

If we start dividing by 2, we get:

$$\begin{aligned} 24_{10} &= 11000.00000_2 \\ 24_{10}/2 &= 01100.00000_2 \\ 24_{10}/4 &= 00110.00000_2 \\ 24_{10}/8 &= 00011.00000_2 \\ 24_{10}/16 &= 00001.10000_2 \\ 24_{10}/32 &= 00000.11000_2 \end{aligned}$$

where the last result is in the interval $[0, 1]$.

If we start building a (nearly) balanced binary tree with $N/2 = 12$ leaf nodes we get Figure 3, a tree with a height of 5. Notice that each layer has (roughly) half the nodes of the layer beneath it.

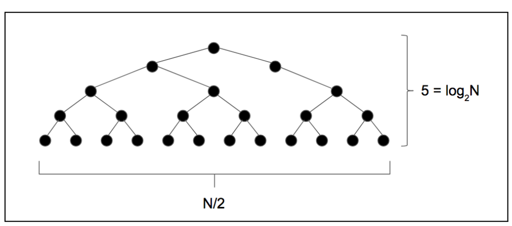

Figure 3: A (roughly) balanced binary tree with $N = 24$ nodes.