Logarithms and Exponentials

\[ \left\lfloor \log_a(r) \right\rfloor = \text{Starting with 1, the minimum number of times that we must multiply by } a \text{ to get an integer } \geq r. \]

**Example:** \[ \left\lfloor \log_2(9) \right\rfloor = \left\lfloor \log_2(16) \right\rfloor = 4 \]

\[ \left\lceil \log_a(r) \right\rceil = \text{Starting with 1, the maximum number of times that we can multiply by } a \text{ to get an integer } \leq r. \]

**For example:** \[ \left\lceil \log_2(8) \right\rceil = \left\lceil \log_2(15) \right\rceil = 3 \]
Generalization of Logs and Exps to the Continuous Case

Facts:

• For any real numbers $a>0$ and $x$ such that $a\neq 1$:
  \[ \log_a(a^x) = a^{\log_a(x)} = x \]

• For any real numbers $a, x, y > 0$ such that $a\neq 1$:
  \[ \log_a(1/x) = -\log_a(x), \quad x \neq 0 \]
  \[ \log_a(xy) = \log_a(x) + \log_a(y) \]
  \[ \log_a(x^y) = y \log_a(x) \]
(facts about logarithms and exponentials continued)

• For any real numbers $a, b, x > 0$ such that $a, b \neq 1$:

$$\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$$

and

$$a^x = b^{x/\log_a(b)}$$
(facts about logarithms and exponentials continued)

• For any real numbers $a>0$ and $x$:
  
  $a^{-x} = 1/a^x$
  
  $a^{x+y} = a^x a^y$
  
  $a^{xy} = (a^x)^y$

• For any real numbers $a>0$ and $x$, and any integer $i>0$:
  
  $a^{(x/i)} = i\sqrt[n]{a^x} = \left(i\sqrt[n]{a}\right)^x$
Logarithms and Exponentials Versus Polynomials

Logarithm base does not change asymptotic complexity.

**Theorem:** For any integer \( k \geq 2 \), \( \log_k(n) \) is \( \Theta(\log_2(n)) \).

**Proof:**

Since \( \log_k(n) \leq \log_2(n) \), \( \log_k(n) \) is \( O(\log_2(n)) \).

Since \( \log_k(n) = \log_2(n)/\log_2(k) \), \( \log_k(n) \) is \( \Omega(\log_2(n)) \);
that is, choose \( c = 1/\log_k(2) \) in the definition of \( \Omega \).

***Because logarithm base does not affect asymptotic complexity, we often omit it; for example, we may say a function is \( O(n\log(n)) \) rather than \( O(n\log_2(n)) \).
Logarithms are smaller than roots.

**Theorem:** For any real number $r > 0$, $\log_2(n)$ is $O(n^r)$.

**Proof:**

Since $\log_2(n) \leq n$ for all $n \geq 1$, $\log_2(n) = (1/r)\log_2(n') \leq (1/r)n^r$. Hence, $a=1$ and $b \geq (1/r)$ satisfies the definition of $O$. 
A polynomial is $O$ of its first term.

**Theorem:** For any integer $k \geq 0$, $k^{th}$ degree polynomial $P(n)$ is $O(n^k)$.

**Proof:**

Let $m$ be the absolute value of the largest coefficient. Then each of the $k+1$ terms of $P(n)$ is less than $mn^k$. Hence, any $a \geq 0$ and $b \geq (k+1)m$ satisfies the definition of $O$.

**Note:** In fact, for any $\varepsilon > 0$, there is a constant $a$ such that for all $n > a$,

$$P(n) < (1 + \varepsilon)hn^k,$$

where $h$ is the coefficient of the high order term.
Exponentials are larger than polynomials.

**Theorem:** For any integer $k \geq 0$, $2^n$ is $\Omega(n^k)$.

**Proof:**

$$2^n = 2^{n\log_2(n)/\log_2(n)} = \left(2^\log_2(n)\right)^{n/\log_2(n)} = n^{n/\log_2(n)}$$

Hence, any $c>0$ satisfies the definition of $\Omega$. For example, if we let $c=1$ then $2^n \geq n^k$ for all $n$ such that $n/\log_2(n) \geq k$. 

- 8-
The constant used for an exponential affects asymptotic complexity.

**Theorem:** For any real numbers $0 < x < y$, $y^n$ is not $O(x^n)$.

**Proof:**

Since

$$y^n = \left(\frac{y}{x}x\right)^n = \left(\frac{y}{x}\right)^n x^n$$

there can be no $a$ and $b$ that satisfy the definition of $O$; because $y/x > 1$ and hence $(y/x)^n$ is greater than any constant for large $n$. 


Examples of Asymptotic Relationships with Logs and Exponentials

A. $n^2$ and $n^{100}$ are both $O(2^n)$

We can re-write:

$$2n = 2^{n \log_2(n)/\log_2(n)} = n^{n/\log_2(n)}$$

For $n^2$, since for $n \geq 4$, $n/\log_2(n) \geq 2$, we can use $a=4$ and $b=1$ in the definition of $O$.

For $n^{100}$, since for $n \geq 1024$, $n/\log_2(n) \geq 100$, we can use $a=1024$ and $b=1$ in the definition of $O$. 
(examples of asymptotic relationships with logs and exponentials, continued)

**B.** $n^2 + \log_2(n)^3$ is $O(n^2)$

Since for $n \geq 8$, $\log_2(n)^3 < n^2$, we can use $a=8$ and $b=2$ in the definition of $O$. 
(examples of asymptotic relationships with logs and exponentials, continued)

**C.** \( n(\log_2(n))^2 \) is \( O(n^2) \)

The functions \( \log_2(n) \) and \( n^{1/2} \) cross at \( n=2 \) and \( n=4 \), and then never again, and hence we can use \( a=4 \) and \( b=1 \) in the big O definition.

Or, without using any calculus, since \( \log_2(x) < x \) for \( x \geq 1 \),

\[
n \log_2(n)^2 = n(2\log_2(n^{1/2}))^2 < 4n^2
\]

and hence \( a=1 \) and \( b=4 \) suffices.
Example: Binary Search

**Problem:** Given part of an array $A[a] \ldots A[b]$ that is already arranged in increasing order, and a value $x$, find an $i$, $a \leq i \leq b$, such that $x = A[i]$, or determine that no such $i$ exists.

**Binary search algorithm:** Compute the midpoint $m$ between $a$ and $b$, check whether $x \leq A[m]$, and repeat this process on the appropriate half of the array.

\[ \text{while } a < b \text{ do begin} \]
\[ m := \lfloor (a+b)/2 \rfloor \]
\[ \text{if } x \leq A[m] \text{ then } b := m \text{ else } a := m+1 \]
\[ \text{end} \]
\[ \text{if } x = A[a] \text{ then } \{x \text{ is at position } a\} \text{ else } \{x \text{ is not in } A\} \]
(binary search continued)

**Complexity of binary search:**

**Time:** Let $n$ denote the number of elements. To simplify our analysis, let $N$ be the smallest power of 2 that is $\geq n$ and imagine that the input is padded with copies of a special $+\infty$ value so that the number of elements is exactly $N$ (i.e., the initial value of $b-a+1$ is exactly $N$). Then since each iteration of the main loop halves $b-a+1$, the number of iterations is $\leq \log_2(N) = \lceil \log_2(n) \rceil$, and hence the algorithm is $O(\log(n))$ since the time for each iteration is $O(1)$.

**Space:** $O(1)$ space in addition to the $O(n)$ space used by $A$.

**Observation:** Binary search is similar to how one typically finds a word in a dictionary or a phone book. Open the book in the middle, if past the word, open the book somewhere in the left half, or if before the word, open the book somewhere in the right half, ...
Example: Binary Numbers

Idea: Binary integers (i.e., base 2 integers) use only the digits ("bits") 0 and 1; for example the integers zero through ten are written in binary as:

\[
0, \ 1, \ 10, \ 11, \ 100, \ 101, \ 110, \ 111, \ 1000, \ 1001, \ 1010
\]

Base ten versus binary:

Base ten integers express a quantity as a sum of each digit times the corresponding power of ten, that is if the digits of a number \( x \) written in base ten are \( t_k t_{k-1} \ldots t_1 t_0 \), then:

\[
x = t_k 10^k + t_{k-1} 10^{k-1} + t_{k-2} 10^{k-2} + \cdots + t_1 10 + t_0
\]

Similarly, if the digits (i.e., bits) in the binary representation of \( x \) are \( b_k b_{k-1} \ldots b_1 b_0 \), then:

\[
x = b_k 2^k + b_{k-1} 2^{k-1} + b_{k-2} 2^{k-2} + \cdots + b_1 2 + b_0
\]
Real numbers in binary:
To the right of the decimal point, base ten sums powers of 1/10 whereas base 2 (binary) sums powers of 1/2. For example:

\[
\begin{align*}
1/2 &= .5 \text{ base ten} = .1 \text{ binary} \\
1/8 &= .125 \text{ base ten} = .001 \text{ binary} \\
11/16 &= .6875 \text{ base ten} = .1011 \text{ binary}
\end{align*}
\]

*** A real number represented in binary with a finite number of bits can be represented in base 10 with a finite number of digits; the reverse is not always true.
Arithmetic with binary numbers:

Idea: Work in the same way as base 10.

For example, to add two binary numbers, just add each column and if the result is a two bit number (10 or 11), then carry the leading 1 to the next column; in this example, the carry bits are written in italics on top:

\[
27 + 25 = 52 \quad \Rightarrow \quad \begin{array}{cc}
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]
The number of bits in a binary number:

Assuming that the leftmost bit must be 1, it can be shown:

- The number of bits needed for \( n \) distinct numbers is \( \lceil \log_2(n) \rceil \); that is, for \( n \) a power of two, 0, 1, ..., \( n-1 \) can use \( \log_2(n) \) bits, but we need \( \log_2(n)+1 \) bits for \( n \).

- For \( n \geq 1 \), least power of two that is \( \geq n \) uses \( \lceil \log_2(n+1) \rceil \) bits.

- The number of bits to represent a binary integer \( n \geq 0 \) is \( \lfloor \log_2(n) \rfloor +1 \). For example, it takes 3 bits to represent 7 (111) and 4 bits to represent 8 (1000).

- If \( n > 1 \) is a binary integer of \( i \) bits, then \( 2n \) has \( i+1 \) bits and \( \lfloor n/2 \rfloor \) has \( i-1 \) bits.

- If \( n > 1 \) is a binary integer of \( i \) bits and \( m > 1 \) is a binary integer of \( j \) bits, then \( n \times m \) has at least \( (i \times j) - 1 \) bits and at most \( i \times j \) bits.