University of Nevada, Las Vegas Computer Science 477/677 Fall 2022

This material, and much more, can be found at `https://en.wikipedia.org/wiki/Big_O_notation`

Complexity Classes of Functions

In our course, we deal primarily with positive-valued increasing functions. If g is such a function, we define *complexity classes* $O(g(n))$, $\Omega(g(n))$, and $\Theta(g(n))$, which are classes of functions which are *asymptotically bounded* by g. Here is the precise definition.

1. $O(g(n))$ is the set of all functions f such that for some N and some positive constant C, $f(n) \leq C \cdot g(n)$ for all $n \geq N$.
2. $\Omega(g(n))$ is the set of all functions f such that for some N and some positive constant C, $f(n) \geq C \cdot g(n)$ for all $n \geq N$.
3. $\Theta(g(n))$ is the set of all functions f such that for some N and some positive constants C_1 and C_2 $C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$ for all $n \geq N$.

Thus $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$.

Notation. We use the equal sign to denote membership in a complexity class. For example, we write $3n + \log n + 5 = O(n)$ to say that the function $f(n) = 3n + \log n + 5$ is a member of the set $O(n)$.

We also use an equal sign to denote inclusion of complexity classes. For example, we write $O(n) = O(n^2)$ to indicate that $O(n)$ is a subset of $O(n^2)$. But this "equality" is not symmetric: "$O(n^2) = O(n)$" is false.

There are infinitely many complexity classes of functions. Most importantly for this semester:

1. Constant: $f(n) = O(1)$.
2. logarithmic: $f(n) = O(\log n)$
3. polylogarithmic: $f(n)$ is a constant power of the logarithm, such as $O(\log^2 n)$, which is how we write the square of $\log n$.
4. polynomial: $f(n) = O(n^k)$, where k is positive constant.
5. exponential: $f(n) = O(n^{g(n)})$ where g is a polynomial function.

Additional classes that will arise occasionally:

6. $O(\log \log n)$: grows slower than logarithm.
7. $O(\log^* n)$, the iterated logarithm: grows much slower than than $\log \log$.
8. $O(\alpha(n))$, the inverse Ackermann function. It's unbounded, but grows slower than anything you can possibly imagine! Yet, it arises naturally in some problems.

Logarithms

Among the kids in my high school, one question was, "What is a one-word definition of a logarithm?" The answer is "exponent." The base b logarithm of a positive number x, written $\log_b x$, is the exponent in the equation $b^{\log_b x} = x$. (The logarithm of a negative number or zero is undefined.)

When we write $\log n$ without specifying the base, what do we mean?

1. In engineering or physical science, the base is assumed to be 10. Thus, for example, $\log 1 = 0$, $\log 10 = 1$, $\log 1000 = 4$ and $\log 0.00001 = -5$.
2. In mathematics, the logarithm is assumed to be the natural logarithm, which is the logarithm base e, where $e = 2.71828\ldots$ We write $\ln x$ for the natural logarithm of x.
3. In computer science, we normally assume that the base of the logarithm is 2. Thus for example $\log 8 = 3$ and $\log 1024 = 10$.
4. In asymptotic complexity analysis, which is a major part of CS477/677, the choice of base is irrelevant; that is, $\log_b x$ is in the same asymptotic complexity class for all $b > 1$.

Here are some facts about logarithms (any base):

9. $\log 1 = 0$
10. $\log xy = \log x + \log y$
11. $\log \frac{x}{y} \log x - \log y$
12. $\log x^k = k \log x$

Finally, we relate logarithms of different bases:

13. $\log_a x = \log_b x * \log_a b = \log_b x / \log_b a$
14. $\log n$ grows more slowly than n, or any polynomial function. That is: $\lim_{n\to\infty} \frac{\log n}{n} = 0$

What about summation of logarithms?

15. $\sum_{i=1}^{n} \log i = \log 1 + \log 2 + \ + \log n = \Theta(n \log n)$. (Regardless of base.) There are two ways to see this. The first is by using calculus, taking the antiderivative of the natural logarithm.

    The non-calculus method is to observe that $\log(n/2)$ is roughly the middle term of the series. The sum of the series is at least $\log(n/2) + \cdots + \log n$, which has at least $n/2$ terms each at least $\log(n/2) = \log n - \log 2 = \log n - 1$. Thus $\sum_{i=1}^{n} \geq \sum_{i=n/2}^{n} \log i \geq \frac{n}{2}(\log n - 1) = (n \log n)$.

16. From Equation 7 we obtain a lower bound on any comparison based algorithm for sorting. Any algorithm which sorts n items where all branches in the code are by comparisons of items takes $\Omega((n \log n)$ steps in the worst case. This is because there are $n!$ permutations of n items, and any sorting algorithm, which inverts permutations, must distinguish those permutations. The flow chart (which is a binary tree for comparison based algorithm) must therfore have at least n! leaves, and hence height at least $\log_2 n!$. The number of comparisons in the longest path to a leaf is at least $\log n!$, which is $\log(1 \cdot 2 \cdot 3 \cdots n) = \Theta(n \log n)$.