1.2 Mathematical background

In your chosen profession, it will be necessary to make engineering design decisions. When it comes to programming, you will often have a selection of possible algorithms or data structures; however, when you compare two algorithms, it is not possible to simply say that "algorithm *A* is faster than algorithm *B*". Such a comparison qualitatively compares two values, *e.g.*, $a > b$ or $b > a$. From the Oxford English Dictionary (OED):

qualitative, *a*.	Relating to, connected or concerned with, quality or qualities. Now usually in implied or expressed opposition to quantitative.

Engineering design decisions must be made by choosing a metric, measuring the possible alternatives using that metric, and comparing the two measurements, or *quantities*. Thus, we require a quantitative comparison:

quantitative, *a*.	Relating to, concerned with, quantity or its measurement; ascertaining or expressing quantity.

We will be able to quantitatively compare algorithms and data structures by looking at relatively rates of growth in time requirements and memory requirements as the problem size increases. To do this, we will use mathematics and elementary calculus; however, we will also require a few other tools.

1.2.1 Floor and ceiling functions

The *floor* function maps any real number *x* onto the greatest integer less than or equal to *x*. For example,

$$\lfloor 3.2 \rfloor = \lfloor 3 \rfloor = 3$$
$$\lfloor -5.2 \rfloor = \lfloor -6 \rfloor = -6$$

You can interpret this as *rounding toward negative infinity*.

The ceiling function maps *x* onto the least integer greater than or equal to *x*. For example,

$$\lceil 3.2 \rceil = \lceil 4 \rceil = 4$$
$$\lceil -5.2 \rceil = \lceil -5 \rceil = -5$$

The ceiling function can be interpreted as *rounding toward positive infinity*.

The `cmath` library implements these two functions as

```
double floor( double );      double ceil( double );
```

The justification for the return value being `double` and not a `long` is that a double-precision floating-point number can be as large as 2^{1024} while the largest number that can be represented by `long` is $2^{63} - 1$.

1.2.2 L'Hôpital's rule

If you are attempting to determine the limit

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}$$

but both $\lim_{n\to\infty} f(n)=\infty$ and $\lim_{n\to\infty} g(n)=\infty$, then we can determine the limit by taking the limit of the derivatives

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{f^{(1)}(n)}{g^{(1)}(n)}.$$

This can be repeated as necessary.

Note, that in this course, the kth derivative will always be written as $f^{(k)}(x)$, even if $k = 1$. Thus, we will prefer $f^{(1)}(x)$ and $f^{(2)}(x)$ to $f'(x)$ and $f''(x)$.

1.2.3 Logarithms

Recall that if $n = e^m$, we define $m = \ln(n)$. It is also always true that $e^{\ln(n)} = n$; however, for $\ln(e^n) = n$, it must also be true that n is real.

Because exponentials grow faster than any polynomial, that is,

$$\lim_{n\to\infty}\frac{e^n}{n^d}=\infty$$

for any $d > 0$, it follows that the inverse, the logarithm, must grow more slowly than any polynomial, which we can see by applying l'Hopital's rule once:

$$\lim_{n\to\infty}\frac{\ln(n)}{n^d}=\lim_{n\to\infty}\frac{1/n}{dn^{d-1}}=\frac{1}{d}\lim_{n\to\infty}\frac{1}{n^d}=0.$$

Thus, the logarithm $\ln(n)$ grows more slowly than $\sqrt{n}$ or even $n^{0.01}$.

Another property you may have seen but did not appreciate is that all logarithms are scalar multiples of each other. Recall that

$$\log_b(n)=\frac{\ln(n)}{\ln(b)},$$

and, therefore, $\lg(n) = \log_2(n)$ grows at a factor of $1/\ln(2) \approx 1.4427$ ($\approx 13/9$) faster than $\ln(n)$. For example, Figure 1 shows a plot of $\log_2(n) = \lg(n)$, $\ln(n)$, and $\log_{10}(n)$.

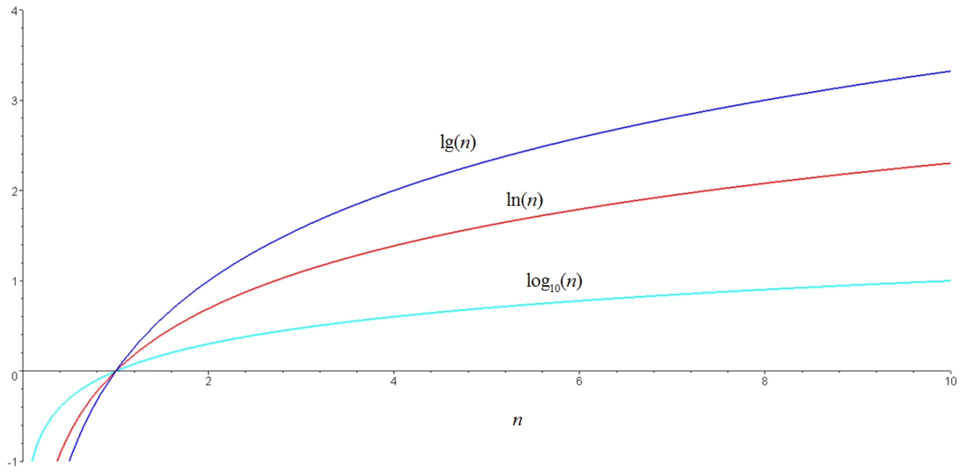

Figure 1. A plot of $\lg(n)$, $\ln(n)$, and $\log_{10}(n)$.

A few notes:

1. The base-2 logarithm, $\log_2(n)$, is written as $\lg(n)$,
2. In most libraries, the natural logarithm, $\ln(n)$, has the signature `double log( double );` , and
3. The *common* logarithm, $\log_{10}(n)$, has the signature `double log10( double );`.

A property of the logarithm that we will repeatedly use is:

$$n^{\log_b(m)} = m^{\log_b(n)}$$

Finally, it is a useful property that $2^{10} = 1024$ is close to 1000:

$$\lg(2^{10}) = 10$$
$$\lg(2^{20}) = 20$$

and therefore

$\lg(10^3) \approx 10$	kilo
$\lg(10^6) \approx 20$	mega
$\lg(10^9) \approx 30$	giga
$\lg(10^{12}) \approx 40$	tera

1.2.4 Arithmetic series

An arithmetic series increases by a constant (usually 1) and very often we will use

$$0+1+2+3+\cdots+n=\sum_{k=0}^{n}k=\frac{n(n+1)}{2}.$$

Proof 1: Add the series twice:

$$\begin{array}{ccccccccccccccc}
 & 1 & + & 2 & + & 3 & + & \cdots & + & n-2 & + & n-1 & + & n & \\
+ & \underline{n} & \underline{+} & \underline{n-1} & \underline{+} & \underline{n-2} & \underline{+} & \underline{\cdots} & \underline{+} & \underline{3} & \underline{+} & \underline{2} & \underline{+} & \underline{1} & \\
 & (n+1) & + & (n+1) & + & (n+1) & + & \cdots & + & (n+1) & + & (n+1) & + & (n+1) & = n(n+1)
\end{array}$$

Having done so, we divide the result by 2. ∎

Proof 2: Using induction, we observe the statement is true for $n = 1$:

$$\sum_{k=0}^{0}k=0 \text{ and } \frac{0\cdot(0+1)}{2}=0.$$

Next, we assume that the statement $\sum_{k=0}^{n}k=\frac{n(n+1)}{2}$ is true for an arbitrary n. Using this assumption, we must now show that the statement is also true for $n + 1$:

$$\begin{aligned}
\sum_{k=0}^{n+1}k &= n+1+\sum_{k=0}^{n}k \\
&= n+1+\frac{n(n+1)}{2} \\
&= \frac{n^2+n+2n+2}{2} \\
&= \frac{n^2+3n+2}{2}=\frac{(n+1)(n+2)}{2}
\end{aligned}$$

Therefore, the statement is true for $n = 0$ and the truth of the statement for n implies the truth of the statement for $n + 1$. Therefore, by the process of mathematical induction, this statement must be true for all $n \geq 0$. ∎

1.2.5 Other polynomial series

It is possible to repeat the above process to prove that the formulas

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6},\ \sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}, \text{ and } \sum_{k=0}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

are all true; however, such precision is often unnecessary. Instead, we can usually rely on reasonable approximations:

$$\sum_{k=0}^{n} k \approx \frac{n^2}{2}$$
$$\sum_{k=0}^{n} k^2 \approx \frac{n^3}{3}$$
$$\sum_{k=0}^{n} k^3 \approx \frac{n^4}{4}$$
$$\sum_{k=0}^{n} k^4 \approx \frac{n^5}{5}$$

In general, we can say that $\sum_{k=0}^{n} k^d \approx \frac{n^{d+1}}{d+1}$. This is actually a very good approximation: we find it by approximating the sum by the corresponding integral:

$$\sum_{k=0}^{n} k^d \approx \int_0^n x^d\ dx = \left.\frac{x^{d+1}}{d+1}\right|_{x=0}^{n} = \frac{n^{d+1}}{d+1} - 0 .$$

The actual sum is the area under the blue piecewise constant function in Figure 2 while the integral is the area of the red region.

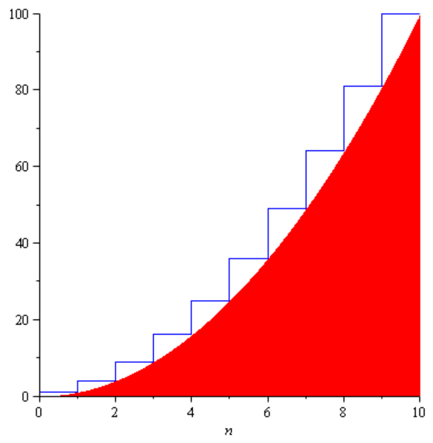

Figure 2. The approximation of $\sum_{k=0}^{n} k^d$ by $\int_0^n x^d\ dx$.

The accumulating error is shown in Figure 3.

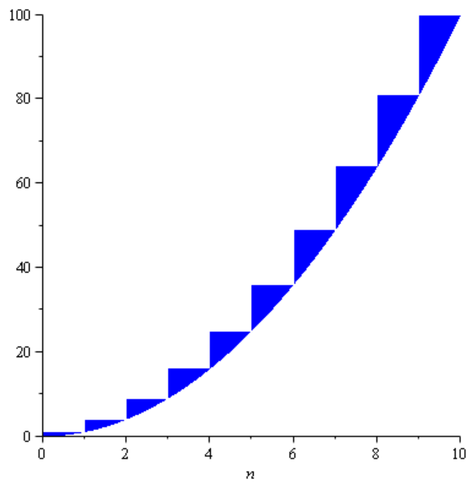

Figure 3. The accumulating error in approximating a sum by an integral.

Fortunately, notice that the error can be shifted into a single column of width 1 and height n^d, as is shown in Figure 4.

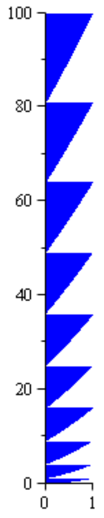

Figure 4. Shifting the errors in Figure 3.

The area of the entire rectangle in Figure 4 is n^d and the part coloured blue is the error $\sum_{k=0}^{n} k^d - \frac{n^{d+1}}{d+1}$.

Now, by inspection, the blue region is half or more of the entire rectangle, and therefore, we have the inequality

$$\frac{n^d}{2} \le \sum_{k=0}^{n} k^d - \frac{n^{d+1}}{d+1} < n^d .$$

Thus, the error grows no faster than n^d but the magnitude of our approximation is $\frac{n^{d+1}}{d+1}$ and therefore the relative error must go to zero as *n* becomes large.

Aside: you may note from the error analysis that it might be reasonable to use the following better approximation:

$$\sum_{k=0}^{n} k^d \approx \frac{n^{d+1}}{d+1} + \frac{n^d}{2}.$$

This is a significantly better approximation (regardless of the value of *d*); however, such accuracy will not be necessary for this class.

You are expected to memorize, if nothing else from this section, the formula and approximation

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2} \text{ and } \sum_{k=0}^{n} k^d \approx \frac{n^{d+1}}{d+1}.$$

1.2.6 Geometric series

The value of a finite geometric series can be evaluated using the formula

$$\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r}.$$

If $|r| < 1$ (whether *r* is real or complex), it is also true that

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}.$$

Proof 1: Multiply both sides by $1 = \frac{1-r}{1-r}$:

$$\begin{aligned}\sum_{k=0}^{n} r^k &= \frac{1-r}{1-r}\sum_{k=0}^{n} r^k \\ &= \frac{(1-r)\sum_{k=0}^{n} r^k}{1-r} \\ &= \frac{\sum_{k=0}^{n} r^k - r\sum_{k=0}^{n} r^k}{1-r}. \\ &= \frac{\sum_{k=0}^{n} r^k - \sum_{k=0}^{n} r^{k+1}}{1-r}\end{aligned}$$

and doing a change-of-index, we get:

$$\begin{aligned}\sum_{k=0}^{n} r^k &= \frac{\sum_{k=0}^{n} r^k - \sum_{k=1}^{n+1} r^k}{1-r} \\ &= \frac{1+\sum_{k=1}^{n} r^k - \left(r^{n+1} + \sum_{k=1}^{n} r^k\right)}{1-r} \\ &= \frac{1-r^{n+1}}{1-r}\end{aligned}$$

▮

Proof 2: By induction, we note the formula is correct for *n* = 1:

$$\sum_{k=0}^{0} r^k = r^0 = 1 = \frac{1-r^{0+1}}{1-r}.$$

Assume that the formula is correct for an arbitrary *n*. In that case, we have

$$\begin{aligned}\sum_{k=0}^{n+1} r^k &= r^{n+1} + \sum_{k=0}^{n} r^k \\ &= r^{n+1} + \frac{1-r^{n+1}}{1-r} \\ &= \frac{r^{n+1}(1-r)+1-r^{n+1}}{1-r} \\ &= \frac{r^{n+1}-r^{n+2}+1-r^{n+1}}{1-r} \\ &= \frac{1-r^{n+2}}{1-r}\end{aligned}$$

Therefore, the statement is true for *n* = 0 and the truth of the statement for *n* implies the truth of the statement for *n* + 1. Therefore, by the process of mathematical induction, this statement must be true for all $n \geq 0$. ▮

Common geometric series that we will see in this class are when *r* = ½ and *r* = 2:

$$\sum_{k=0}^{n}\left(\frac{1}{2}\right)^k = \frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}} = 2-2^{-n}$$

$$\sum_{k=0}^{\infty}\left(\frac{1}{2}\right)^k = \frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}} = 2$$

$$\sum_{k=0}^{n} 2^k = \frac{1-2^{n+1}}{1-2} = 2^{n+1}-1$$

1.2.7 Recurrence relations

If the terms in a sequence are given by an explicit formula, it is possible to calculate any term in the sequence. For example, if $x_n = \frac{1}{n}$ then $x_{40} = 0.025$.

How much time, however, is it to perform a binary search on a sorted list? The algorithm is:

1. Check the middle entry of an array—if that is the object we're looking for, return *true*,
2. If the object is less than the middle entry, do a binary search on the left half, otherwise
3. The object must be greater than the middle entry, so do a binary search on the right half.

If the *half* that we are searching is ever empty, we return *false*.

This is a recursive description of the binary search algorithm. We can describe a recursive algorithm using a recurrence relation; for example, the following are all recurrence relations—the last being the recurrence relation describing the number of calls to a binary search:

$$\begin{array}{llll} x_1 = 1 & x_1 = 1 & x_1 = 1 & x_0 = 1 \\ x_n = x_{n-1} + 2 & x_n = 2x_{n-1} + n & x_n = x_{n-1} + x_{n-2} & x_n = x_{n/2} + 1 \end{array}$$

Before we go on, we will introduce an alternative notation. In general, we will use recurrence relations to describe the run times or memory usage of data structures and algorithms. Consequently, we usually denote these quantities as $T(n)$ and $Mem(n)$. Therefore, our recurrence relations will usually be written using functional notation:

$$\begin{array}{llll} f(1) = 1 & f(1) = 1 & f(1) = 1 & f(0) = 1 \\ f(n) = f(n-1) + 2 & f(n) = 2f(n-1) + n & f(n) = f(n-1) + f(n-2) & f(n) = f(n/2) + 1 \end{array}$$

We would like to take these and find explicit formulas. In the first two cases, this is relatively easy:

$$\begin{array}{l} f(1) = 1 \\ f(n) = f(n-1) + 2 \end{array} \quad \Rightarrow \quad f(n) = 2n - 1$$

and, with the help of Maple,

$$\begin{array}{l} f(1) = 1 \\ f(n) = 2x_{n-1} + n \end{array} \quad \Rightarrow \quad f(n) = 2^{n+1} - n - 1.$$

The last two recurrence relations are slightly more complex. Beyond the scope of this course, it is possible to show that

$$\begin{array}{l} f(1)=1 \\ f(n)=f(n-1)+f(n-2) \end{array} \quad \Rightarrow \quad f(n)=\frac{2+\phi}{5}\phi^n+\frac{3-\phi}{5}\phi^{-n}$$

where $\phi=\frac{\sqrt{5}+1}{2}\approx 1.6180$ is the golden ratio.

In the last case, there is no closed-form solution, but it can be shown that

$$\begin{array}{l} f(1)=1 \\ f(n)=f\left(\left\lfloor\frac{n}{2}\right\rfloor\right)+1 \end{array} \quad \Rightarrow \quad f(n)\approx \ln(n).$$

We will investigate such recurrence relations in the topic on divide-and-conquer algorithms.

1.2.8 Weighted averages

Given n objects, $x_1, x_2, \ldots, x_n$, the average of these numbers is

$$\frac{x_1+x_2+x_3+\cdots+x_n}{n}.$$

If, however, we have a sequence of coefficients $c_1, c_2, \ldots, c_n$ such that $c_1 + c_2 + \cdots + c_n = 1$, we can also calculate the *weighted average*

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n.$$

For example, we can approximate an integral from a to c by taking a weighted average of the values of the functions at a, b, and c where b is the midpoint, as is shown in Figure X.

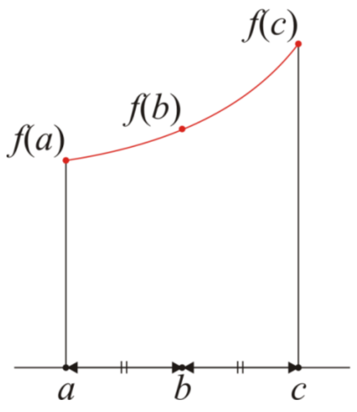

Figure X. The integral $\int_a^c f(x)\,dx$.

We could take the average of these three points and multiply the result by the width $c - a$,

$$\tfrac{1}{3}\left(f(a) + f(b) + f(c)\right)(c-a);$$

however, a better solution is to use the weighted average

$$\left(\tfrac{1}{6} f(a) + \tfrac{2}{3} f(b) + \tfrac{1}{6} f(c)\right)(c-a).$$

For example, $\int_0^2 \cos(x)dx = \sin(2) \approx 0.9093$. Approximating this with the weighted average yields

$$\left(\tfrac{1}{6}\cos(0) + \tfrac{2}{3}\cos(1) + \tfrac{1}{6}\cos(2)\right)2 \approx 0.9150$$

while the simple average is significantly less accurate:

$$\frac{\cos(0)+\cos(1)+\cos(2)}{3}2 \approx 0.7494.$$

1.2.9 Combinations

Given *n* distinct items, one may ask

"How many ways can you choose *k* items?"

or, equivalently,

"How many ways can you combine *k* items from *n* items?"

For example, given the set {1, 2, 3, 4, 5, 6}, you can choose three items in 20 different ways:
{1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 2, 6}, {1, 3, 4}, {1, 3, 5}, {1, 3, 6}, {1, 4, 5}, {1, 4, 6}, {1, 5, 6},
{2, 3, 4}, {2, 3, 5}, {2, 3, 6}, {2, 4, 5}, {2, 4, 6}, {2, 5, 6}, {3, 4, 5}, {3, 4, 6}, {3, 5, 6}, {4, 5, 6}

The general formula for this is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

and this is read "*n* choose *k*". Usually, we will be interested in the specific case where $k = 2$:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}.$$

For example, given the set {1, 2, 3, 4, 5, 6}, you can choose two items in 15 different ways:

{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}
{2, 3}, {2, 4}, {2, 5}, {2, 6}
{3, 4}, {3, 5}, {3, 6}
{4, 5}, {4, 6}
{5, 6}

You may have seen these numbers in polynomial expansions:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} \text{ ;}$$

for example,

$$\begin{aligned}
(x+y)^4 &= \sum_{k=0}^{4} \binom{4}{k} x^k y^{4-k} \\
&= \binom{4}{0} y^4 + \binom{4}{1} xy^3 + \binom{4}{2} x^2y^2 + \binom{4}{3} x^3y + \binom{4}{4} x^4 \\
&= y^4 + 4xy^3 + 6x^2y^2 + 4x^3y + x^4
\end{aligned}$$

You may also have seen these numbers in Pascal's triangle:

$$\begin{array}{ccccccccc}
 & & & & \binom{0}{0} & & & & \\
 & & & \binom{1}{0} & & \binom{1}{1} & & & \\
 & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & \\
 & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & \\
\binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4}
\end{array}$$

$$\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 1 & & 1 & & & \\
 & & 1 & & 2 & & 1 & & \\
 & 1 & & 3 & & 3 & & 1 & \\
1 & & 4 & & 6 & & 4 & & 1
\end{array}$$