1 Asymptotics

This topic is very useful in Algorithms: when comparing the efficiency of two algorithms, one concentrates on a given resource (time, space, communication, energy consumption, etc.) and tries to estimate the amount of resources needed (usually in the worst case) for inputs of length $n$ as a function of $n$. We want a measure that allows us to say ‘$f(n)$ is no more than $g(n)$’.

Simply saying $f(n) \leq g(n)$ is not enough:

- maybe $f(n)$ runs on a device that is 20 times faster than the one $g(n)$ runs on
- maybe $g(n)$ is only small on a few small inputs (for example, one could set up a lookup table instead of a computation)

This motivates the following

**Definition 1.** Let $f()$, $g()$ be real valued. $f(n) = O(g(n))$ (‘$f$ is Big Oh of $g$’) iff

$$\exists C > 0, \ n_0 \in \mathbb{N} : \forall n \geq n_0 \ |f(n)| \leq C|g(n)|$$

We formalize the notion of ‘greater than’ similarly:
Definition 2. \( f(n) = \Omega(g(n)) \) (‘\( f \) is Big Omega of \( g \)’) iff 
\[ \exists C > 0, \ n_0 \in \mathbb{N} : \forall n \geq n_0 \ |f(n)| \geq C|g(n)| \]

Note that \( f = O(g) \) iff \( g = \Omega(f) \)

We also define

Definition 3. \( f(n) = \Theta(g(n)) \) (‘\( f \) is Big Theta of \( g \)’) iff 
\[ f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)) \]

Note that this last definition the constants \( C \) and \( n_0 \) used for Big Oh and for Big Omega are different.

The definitions above are the ones used in Rosen and in Algorithms books. Mathematicians typically use the same notation when dealing with infinite series (and with real functions.) This is the approach in Babai’s notes.

The ‘translation’ between the notations is to look at the sequence of values \( f(0), f(1), \cdots, f(n) \cdots \) as a sequence of reals. It is an easy exercise that the definition of \( f = O(g) \) is equivalently
\[ \lim_{n \to \infty} \frac{|a_n|}{|b_n|} \leq C \text{ for some positive } C \] – in other words, the limit is bounded (with the special case that \( 0/0 \) is bounded).

‘Asymptotic’ refers to the idea that the inequalities eventually hold for sufficiently large \( n \). The Computer Science justification is that one is interested in resources for large problems, or, more precisely, in the growth rate of resources needed.

Also, for the Computer Science applications above, the functions \( f() \) and \( g() \) in the definition are nonnegative, so \( g = |g| \).

There will be homework problems to make these concepts more real. They will consist of proofs that require that you find \( C \) and \( n_0 \) in the definitions above, or to prove that no such numbers exist. For example, it should be clear that \( 20n^2 = O(n^2) \) (take \( n_0 = 1, \ C = 20 \)), that \( n^2 = O(n^3) \) but not \( n^3 = O(n^2) \), etc.

1.1 Asymptotic equivalence

Definition 4 (Asymptotic Equivalence). Let \( a_n, b_n \) be sequences of real numbers. We write \( a_n \sim b_n \) iff \( \lim_{n \to \infty} \frac{|a_n|}{|b_n|} = 1 \).

This means that for large enough \( n \) the two sequences are multiplicatively close. In particular, if \( b_n \) is easier to manipulate or to understand, we can use
it instead of $a_n$ with very little loss in precision. The following are important equivalences (which we will not prove):

$$n! \sim \sqrt{(2\pi n)(\frac{n}{e})^n}$$

If $\pi(n)$ denotes the number of primes less than $n$, then

$$\pi(n) \sim \frac{n}{\ln n}$$

This is another important notion, but we do not have time to work with it.

### 1.2 Big Oh notation warning

The equality sign in the definitions above is misleading (in Programming Languages lingo, there is a type mismatch. $f$ is a function, while $O(g)$ isn’t: it is a collection of functions.) It would be more accurate to write $f \in O(g)$, but that is not what the convention is. However, you should be careful with doing things like $O(g) + O(h)$ (especially because some actually make sense) – but you have to prove such properties.

An illustrative example:

### 1.3 MergeSort Runtime

We have seen previously the MergeSort algorithm:

- to sort an array of size $n$ (assume for the moment that $n = 2^k$ for some $k$)
  
  Sort the first half $A[1..n/2]$

  Sort the second half $A[n/2 + 1..n]$

  Merge the two sorted arrays of size $n/2$ each.

We can bound the runtime of the algorithm as follows: if the runtime of MergeSort on an array of size $n$ is bounded by $T(n)$, then, since the merge step takes linear time, and the time of each recursive call is bounded by $T(n/2)$, we get the inequality

$$T(n) \leq 2T(n/2) + n$$
We are ignoring constants here...

By Using $T(1) = 1$ (for the cost of the call), we get a bound by treating the inequality as an equality, getting the recurrence relation

$$T(n) = 2T(n/2) + n$$

As we saw, one can solve such a recurrence relation by expanding the expression:

$$T(n) = 2T(n/2) + n = 2[2T(\frac{n/2^2}{2}) + n/2] + n$$

(where we used the formula for input $n/2$)

$$T(n) = 2^2T(n/2^n) + 2n = 2^2[2T(\frac{n/2^2}{2}) + n/2^2] + 2n = 2^3T(n/2^3) + 3n$$

We see that this procedure yields

$$T(n) = 2^iT(n/2^i) + in$$

(prove it by induction!)

When $i = k$, we have $n/2^k = 1$, so $T(n) = 2^kT(1) + kn$, or

$$T(n) = n + kn$$

(since $T(1) = 1$ and $2^k = n$), and since $k = \log_2 n$ we have

$$T(n) = n + n \log_2 n$$

Exercise: Relating to asymptotics just studied-

$T(n) = O(n \log n)$ since $\log_2 n = O(\ln n) = O(\log_{10} n)$, and $n = O(n \log n)$

(prove these!)

## 2 Solutions

The most useful method is to guess a solution – if we are right, it may be easy to show it is correct, using (strong) induction. We have seen this technique before–for example if we have the relation

$$a_n = 2a_{n-1}$$

with $a_0 = 1$

If we guess that the solution is $a_n = 2^n$, we can prove our guess easily. (Do it!)

Many times iteration works – we have seen the method in analyzing MergeSort: one keeps applying the recurrence to values smaller than $n$, expanding each of them, until we see a pattern – which we prove correct by
induction— and eventually we get a formula (which we also prove by induction to be correct.)