Recurrences

When an algorithm contains a recursive call to itself, its running time can often be described by a recurrence. A **recurrence** is an equation or inequality that describes a function in terms of its value on smaller inputs.

For example, the merge sort that we looked at in the first lecture can be expressed as a recurrence:

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T\left(n/2\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

To solve recurrences, we will be focusing on the following methods:

- **Substitution method:** guess a bound and then use mathematical induction to prove our guess correct.
- **Iteration method:** convert the recurrence into a summation and then rely on techniques for bounding summations to solve the recurrence.

1 Substitution Method

There are two parts to the substitution method:

1. Guess the form of the solution.
2. use mathematical induction to find the constants and show the solution works.

For example, let's take the recurrence $T(n) = 2T(\lfloor n/2 \rfloor) + n$, and try to find an upper bound for it. We guess that the solution is $T(n) = O(n \lg n)$. Now we need to prove that $T(n) \leq cn \lg n$ for some $c > 0$. Assuming that this bound holds for $\lfloor n/2 \rfloor$, we substitute into the equation:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n \end{aligned}$$

where the last step holds as long as $c \geq 1$.

We now need to prove that our solution holds for the boundary conditions. Asymptotic notation only requires us to prove for $n \geq n_0$, hence we can choose $n = 2$ and $n = 3$. Substituting in, it is clear that the solution holds for both of these.

2 The Iteration Method

The iteration method does not require guessing the answer, but it may require more algebra than the substitution method. The idea is to expand (iterate) the recurrence and express it as a summation of terms dependent only on n and the initial conditions. Consider our example recurrence again.

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

The first step is expanding the recurrence into a series. A convenient "trick" to do this is to reshuffle the recurrence such that the "work done" part "$+n$" is in front and the recursion part "$2T(n/2)$" at the back.

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

Next expand the recursion part by substituting $n/2$ into $T(n)$.

$$T(n) = n + 2\left(n/2 + 2T(n/4)\right)$$

Simplify.

$$T(n) = n + n + 4T(n/4)$$

Repeat substituting and simplifying until an observable pattern forms.

$$\begin{aligned} T(n) &= n + n + 4\left(n/4 + 2T(n/8)\right) \\ &= n + n + n + 8T(n/8) \\ &= n + n + n + 8\left(n/8 + 2T(n/16)\right) \\ &= n + n + n + n + 16T(n/16) \\ &= n + n + n + n + \ldots + n + T(1) \\ &= n + n + n + n + \ldots + n + c \end{aligned}$$

It has become apparent that the sequence will continue as a series of n additions and stop at $T(1)$ as defined earlier "$T(n) = c, \text{if } n = 1$". The next step is to work out the number of n additions in the series. This turns out to be the number of times n can be divided by 2 until the value is 1. Let k be this number.

$$\begin{aligned} \frac{n}{2^k} &= 1 \\ n &= 2^k \\ k &= \lg n \end{aligned}$$

Knowing that there are $\lg n$ number of n additions in the series, we can solve the recurrence.

$$\begin{aligned} T(n) &= n \lg n + c \\ &= O(n \lg n) \end{aligned}$$

Now, $T(1)$ was chosen because it was defined above that $T(n)$ is a constant if $n = 1$. However, 1 is a constant and if the problem size is a constant, the runtime can also be expected to be some constant! Thus, the series could have ended at $T(2)$ if we wanted to. While this would have changed the number of n additions in the series, the final solution will still be the same. Assume that we stopped the series at $T(c_1)$, and that the runtime of $T(c_1)$ is c_2. Then...

$$\begin{aligned} \frac{n}{2^k} &= c_1 \\ \frac{n}{c_1} &= 2^k \\ k &= \lg \frac{n}{c_1} \\ k &= \lg n - \lg c_1 \end{aligned}$$

Substituting this into the recurrence gives us the same result as before.

$$\begin{aligned} T(n) &= n(\lg n - \lg c_1) + c_2 \\ &= n \lg n - n \lg c_1 + c_2 \\ &= O(n \lg n) \end{aligned}$$