Recurrence Relations

Many algorithms, particularly divide and conquer algorithms, have time complexities which are naturally modeled by recurrence relations.

A recurrence relation is an equation which is defined in terms of itself.

Why are recurrences good things?

1. Many natural functions are easily expressed as recurrences:

$$a_n = a_{n-1} + 1, a_1 = 1 \longrightarrow a_n = n \quad (polynomial)$$

$$a_n = 2a_{n-1}, a_1 = 1 \longrightarrow a_n = 2^{n-1} \quad (exponential)$$

$$a_n = na_{n-1}, a_1 = 1 \longrightarrow a_n = n! \quad (weird\ function)$$

2. It is often easy to find a recurrence as the solution of a counting problem. *Solving* the recurrence can be done for many special cases as we will see, although it is somewhat of an art.

Recursion *is* Mathematical Induction!

In both, we have general and boundary conditions, with the general condition breaking the problem into smaller and smaller pieces.

The *initial* or boundary condition terminate the recursion.

As we will see, induction provides a useful tool to solve recurrences – guess a solution and prove it by induction.

$$T_n = 2T_{n-1} + 1, T_0 = 0$$

n	0	1	2	3	4	5	6	7
T_n	0	1	3	7	15	31	63	127

Guess what the solution is?

Prove $T_n = 2^n - 1$ by induction:

1. Show that the basis is true: $T_0 = 2^0 - 1 = 0$.

2. Now assume true for T_{n-1}.

3. Using this assumption show:

$$T_n = 2T_{n-1} + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$$

∎

Try backsubstituting until you know what is going on

Also known as the iteration method. Plug the recurrence back into itself until you see a pattern.

Example: $T(n) = 3T(\lfloor n/4 \rfloor) + n$.

Try backsubstituting:

$$\begin{aligned} T(n) &= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor) \\ &= n + 3\lfloor n/4 \rfloor + 9(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor)) \\ &= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor) \end{aligned}$$

The $(3/4)^n$ term should now be obvious.

Although there are only $\log_4 n$ terms before we get to $T(1)$, it doesn't hurt to sum them all since this is a fast growing geometric series:

$$T(n) \leq n \sum_{i=0}^{\infty} (\frac{3}{4})^i + \Theta(n^{\log_4 3} \times T(1))$$

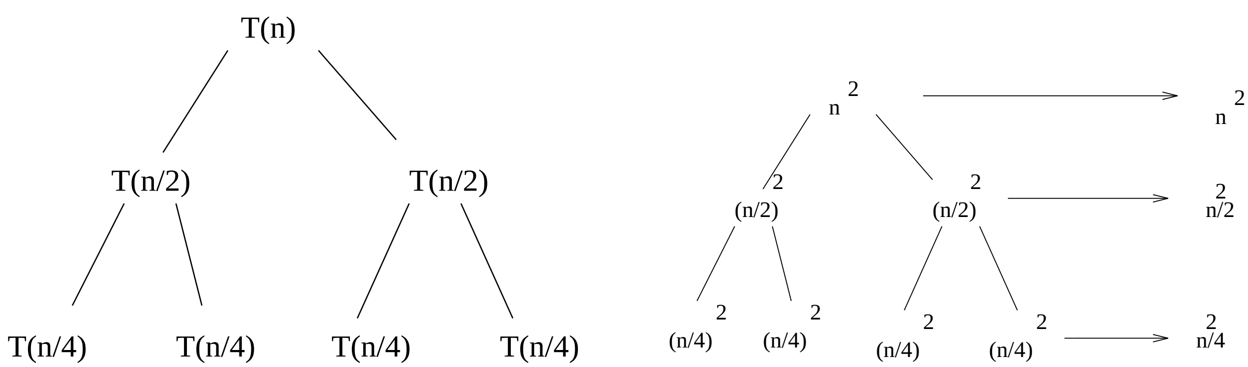
$$T(n) = 4n + o(n) = O(n)$$

Recursion Trees

Drawing a picture of the backsubstitution process gives you a idea of what is going on.

We must keep track of two things – (1) the size of the remaining argument to the recurrence, and (2) the additive stuff to be accumulated during this call.

Example: $T(n) = 2T(n/2) + n^2$



The remaining arguments are on the left, the additive terms on the right.

Although this tree has height $\lg n$, the total sum at each level decreases geometrically, so:

$$T(n) = \sum_{i=0}^{\infty} n^2/2^i = n^2 \sum_{i=0}^{\infty} 1/2^i = \Theta(n^2)$$

The recursion tree framework made this much easier to see than with algebraic backsubstitution.

See if you can use the Master theorem to provide an instant asymptotic solution

The Master Theorem: Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

where we interpret n/b as $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.

3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$, and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Examples of the Master Theorem

Which case of the Master Theorem applies?

- $T(n) = 4T(n/2) + n$

  Reading from the equation, $a = 4$, $b = 2$, and $f(n) = n$.

  Is $n = O(n^{\log_2 4-\epsilon}) = O(n^{2-\epsilon})$?

  Yes, so case 1 applies and $T(n) = \Theta(n^2)$.

- $T(n) = 4T(n/2) + n^2$

  Reading from the equation, $a = 4$, $b = 2$, and $f(n) = n^2$.

  Is $n^2 = O(n^{\log_2 4-\epsilon}) = O(n^{2-\epsilon})$?

  No, if $\epsilon > 0$, but it is true if $\epsilon = 0$, so case 2 applies and $T(n) = \Theta(n^2 \log n)$.

- $T(n) = 4T(n/2) + n^3$

  Reading from the equation, $a = 4$, $b = 2$, and $f(n) = n^3$.

  Is $n^3 = \Omega(n^{\log_2 4+\epsilon}) = \Omega(n^{2+\epsilon})$?

  Yes, for $0 < \epsilon < 1$, so case 3 *might* apply.

  Is $4(n/2)^3 \leq c \cdot n^3$?

  Yes, for $c \geq 1/2$, so there exists a $c < 1$ to satisfy the regularity condition, so case 3 applies and $T(n) = \Theta(n^3)$.