

# Solving Recurrences

## The Toolbox

### Two General Methods

Two general purpose methods for solving recurrences are

1. Guess and confirm.

Sometimes investigating a few terms can lead to a guess that can be verified by another method (usually induction).

Other times we can make a *parameterized* guess, use the first few terms to solve for the parameters, and then show that the resulting formula is correct.

**Example.** Suppose we guess that

$$S(n) = \sum_{i=0}^n i = An^2 + Bn + C$$

(a reasonable guess, since we can show that  $S(n) = \Theta(n^2)$ ). Then we can solve for  $A$  and  $B$  and  $C$  using  $S(0) = 0 = C$ ,  $S(1) = 1 = A + B$ , and  $S(2) = 3 = 4A + 2B$ . This gives  $A = 1/2$  and  $B = 1/2$ . Induction can be used to verify that  $S(n) = \frac{n^2}{2} + \frac{n}{2} = \frac{n(n+1)}{2}$ .

2. Unraveling.

**Example.** Suppose  $a(0) = 3$ , and  $a(n) = 2a(n-1) + 1$ . Then

$$\begin{aligned} a(n) &= 2a(n-1) + 1 \\ &= 2[2a(n-2) + 1] + 1 = 2^2a(n-2) + [2 + 1] \\ &= 2^2[2a(n-3) + 1] + [2 + 1] = 2^3a(n-3) + [2^2 + 2 + 1] \\ &= 2^k a(n-k) + [2^{k-1} + \dots + 2^2 + 2 + 1] \\ &= 2^n a(n-n) + [2^{n-1} + \dots + 2^2 + 2 + 1] \\ &= 2^n a(0) + [2^{n-1} + \dots + 2^2 + 2 + 1] \\ &= 2^n \cdot 3 + [2^{n-1} + \dots + 2^2 + 2 + 1] \\ &= 2^n \cdot 3 + \sum_{j=0}^{n-1} 2^j \end{aligned}$$

Since  $\sum_{j=0}^{n-1} 2^j = 2^{n-1} + \dots + 2^2 + 2 + 1 = 2^n - 1$  (geometric series, see below), our unraveling simplifies to

$$a(n) = 3 \cdot 2^n + [2^n - 1]$$

These two methods can be used to find general solutions for classes of recurrences that occur frequently in computer science, especially in recursive code.

### Geometric Sums

Sums of the form  $\sum_{i=0}^n r^i$  are called **geometric sums**, and we can always determine their value:

- If  $r = 1$ , then  $\sum_{i=0}^n r^i = \sum_{i=0}^n 1 = n + 1$ .
- If  $r \neq 1$ , then let  $G(n) = \sum_{i=0}^n r^i$ . Then  $rG(n) - G(n) = r^{n+1} - 1$  (write out the terms and watch for cancellation), so  $G(n) = \frac{r^{n+1} - 1}{r - 1}$ .

These formulas can also be proved by induction.

## Logsmanship

In order to simplify the results we get when working with geometric sums and divide-and-conquer recurrences, we will need to work with logarithms and exponential functions. Most of the identities used are probably familiar to you from other courses. One that may not be is identity

$$a^{\log_b(n)} = n^{\log_b(a)} .$$

This is worth remembering. To see why it is true, consider taking the logarithm (base  $b$ ) of each side.

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

## Divide-and-Conquer

Divide-and-conquer recursion leads to running times that can be expressed using recurrences like

$$f(n) = af(n/b) + g(n) ,$$

where  $a \geq 1$  and  $b > 1$ . Roughly,  $a$  is the number of subproblems processed (number of recursive calls),  $b$  measures the size of the subproblems, and  $g(n)$  is the overhead from the parts of the algorithm not involved in the recursive calls.

The tools needed to analyze this kind of recurrence are **unraveling** and **geometric sums**. Here we will deal with the cases where  $g(n) = Cn^d = O(n^d)$  for constants  $C$  and  $d$ . This provides big O estimates whenever  $g$  is a polynomial.

Let's look at some cases:

1. If  $g(n) = C$  for some constant  $C$  (i.e.,  $d = 0$ ), then unraveling yields:

$$\begin{aligned} f(n) &= af(n/b) + C \\ &= a[af(n/b^2) + C] + C \\ &= a^2 f(n/b^2) + C[a + 1] \\ &= a^3 f(n/b^3) + C[a^2 + a + 1] \\ &= a^k f(n/b^k) + C[a^{k-1} + a^{k-2} + \dots + a + 1] \\ &= a^{\log_b(n)} f(1) + C \sum_{j=0}^{\log_b(n)-1} a^j \quad (\text{geometric sum}) \end{aligned} \tag{1}$$

- (a) If  $a = 1$  (so the recurrence is  $f(n) = f(n/b) + C$ ), then the geometric sum is easy and (1) becomes

$$\begin{aligned} f(n) &= a^{\log_b(n)} f(1) + C \sum_{j=0}^{\log_b(n)-1} a^j \\ &= 1^{\log_b(n)} f(1) + C \sum_{j=0}^{\log_b(n)-1} 1 \\ &= f(1) + C \log_b(n) = O(\log n) \end{aligned}$$

- (b) If  $a > 1$ , then we can use our knowledge of geometric series and (1) becomes

$$\begin{aligned} f(n) &= a^{\log_b(n)} f(1) + C \sum_{j=0}^{\log_b(n)-1} a^j \\ &= a^{\log_b(n)} f(1) + C \frac{a^{\log_b(n)} - 1}{a - 1} \\ &= a^{\log_b(n)} \left[ f(1) + \frac{C}{a - 1} \right] - \frac{1}{a - 1} \\ &= O(a^{\log_b(n)}) = O(n^{\log_b(a)}) \end{aligned}$$

2. If  $g(n) = Cn^d$  ( $d > 0$ ), then our unraveling is a bit messier, but the same basic ideas work:

$$\begin{aligned}
f(n) &= af(n/b) + g(n) \\
&= a[af(n/b^2) + g(n/b)] + g(n) \\
&= a^2f(n/b^2) + [ag(n/b) + g(n)] \\
&= a^3f(n/b^3) + [a^2g(n/b^2) + ag(n/b) + g(n)] \\
&= a^k f(n/b^k) + \sum_{j=0}^{k-1} a^j g(n/b^j) \\
&= a^{\log_b(n)} f(1) + \sum_{j=0}^{\log_b(n)-1} a^j C \left(\frac{n}{b^j}\right)^d \\
&= a^{\log_b(n)} f(1) + \sum_{j=0}^{\log_b(n)-1} a^j C \left(\frac{n^d}{b^{jd}}\right) \\
&= n^{\log_b(a)} f(1) + Cn^d \sum_{j=0}^{\log_b(n)-1} \left(\frac{a}{b^d}\right)^j \tag{2}
\end{aligned}$$

As before, we have two terms, and once again which of these two terms is larger, depends on the relationship of  $a$ ,  $b$ , and  $d$ .

(a) If  $a = b^d$  (i.e.,  $d = \log_b(a)$ ), then  $\frac{a}{b^d} = 1$ , so the second term in (2)

$$Cn^d \sum_{j=0}^{\log_b(n)-1} 1 = O(n^d \log(n))$$

and dominates the first term.

(b) If  $a < b^d$  (i.e.,  $d > \log_b(a)$ ), then  $n^d > n^{\log_b(a)}$ , so the second term in (2) again dominates. And since  $\frac{a}{b^d} < 1$ , the geometric sum is bounded by a constant, so  $f(n) = O(n^d)$ .

(c) Finally, if  $a > b^d$  (i.e.,  $d < \log_b(a)$ ), then (2) becomes

$$\begin{aligned}
f(n) &= n^{\log_b(a)} f(1) + Cn^d \frac{\left(\frac{a}{b^d}\right)^{\log_b(n)} - 1}{\frac{a}{b^d} - 1} \\
&= n^{\log_b(a)} f(1) + \left( Cn^d \left(\frac{a}{b^d}\right)^{\log_b(n)} \boxed{\frac{1}{\frac{a}{b^d} - 1}} \right) + \left( Cn^d \boxed{\frac{-1}{\frac{a}{b^d} - 1}} \right) \\
&= n^{\log_b(a)} \boxed{f(1)} + n^d \left(\frac{a}{b^d}\right)^{\log_b(n)} \boxed{\frac{C}{\frac{a}{b^d} - 1}} + \boxed{\frac{-C}{\frac{a}{b^d} - 1}} n^d \\
&= n^{\log_b(a)} \boxed{f(1)} + n^d \frac{a^{\log_b(n)}}{b^{d \log_b(n)}} \boxed{\frac{C}{\frac{a}{b^d} - 1}} + \boxed{\frac{-C}{\frac{a}{b^d} - 1}} n^d \\
&= n^{\log_b(a)} \boxed{f(1)} + n^d \frac{n^{\log_b(a)}}{n^d} \boxed{\frac{C}{\frac{a}{b^d} - 1}} + \boxed{\frac{-C}{\frac{a}{b^d} - 1}} n^d \\
&= O(n^{\log_b(a)}) + O(n^{\log_b(a)}) + O(n^d) = O(n^{\log_b(a)})
\end{aligned}$$

(Notice that the boxed quantities are constants.)

The results above can be summarized as follows:

If $f(n) = af(n/b) + g(n)$ , and $g(n) = Cn^d$ , then $f(n) =$	$\begin{cases} O(\log n) & \text{if } d = 0 \text{ and } a = 1 \\ O(n^{\log_b(a)}) & \text{if } d = 0 \text{ and } a > 1 \\ O(n^d) & \text{if } d > 0 \text{ and } d > \log_b(a) \\ O(n^d \log n) & \text{if } d > 0 \text{ and } d = \log_b(a) \\ O(n^{\log_b(a)}) & \text{if } d > 0 \text{ and } d < \log_b(a) \end{cases}$
--	--

Things to notice:

1.  $C$  does not occur in any of the expressions on the right hand side, so a big O analysis is not sensitive to the constant  $C$ , and we could replace the left hand side with

$$f(n) = af(n/b) + O(n^d).$$

2. The first two cases look just like instances of the last three with  $d = 0$ .

### Examples

1. The running time of a binary search algorithm satisfies  $t(n) = t(n/2) + O(1)$ , since we split the array into two halves and recursively check only one and the overhead cost for the split is just a little bit of arithmetic that takes the same amount of time regardless of the size of the array. By the results above  $t(n) = O(\log n)$ .
2. The running time of MergeSort satisfies  $t(n) = 2t(n/2) + O(n)$ , since we split the array into two halves and recursively sort each one and then have to merge the results (at a fixed cost *per item merged*). By the results above (this is a case where  $a = 2 > 1$  and  $d = \log_b(a) = \log_2(2)$ ):

$$t(n) = O(n \log n).$$

## Linear (Homogeneous) Recurrences

There are general methods for solving recurrences of the form

$$a(n) = c_1a(n-1) + c_2a(n-2) + \dots + c_k a(n-k) + f(n),$$

where each of the  $c_i$  is a constant. If  $f(n) = 0$ , then this is a *linear homogeneous recurrence relation* (with constant coefficients). If  $f(n) \neq 0$ , then this is a *linear non-homogeneous recurrence relation* (with constant coefficients).

We developed methods for solving the homogeneous case of degree 1 or 2. (Higher degree examples are done in a very similar way. A new wrinkle enters if we do non-homogeneous examples, but straightforward methods exist if  $f(n)$  is a polynomial or exponential function.)

For degree 1, simply unravel. (In fact, we can handle some simple functions  $f(n)$  in this case, too, as we did in the unraveling example above.)

For degree 2, we use the method of *parameterized guess* and confirm. It seems reasonable that a solution to

$$a(n) = c_1a(n-1) + c_2a(n-2) \tag{3}$$

will have exponential growth. So let's

guess that  $a(n) = \alpha r^n$  for some  $C$  and  $r$ .

Then

$$\alpha r^n = c_1 \alpha r^{n-1} + c_2 \alpha r^{n-2},$$

so

$$r^2 = c_1 r + c_2.$$

This tells us how to find  $r$ :  $r$  must be a solution to the polynomial equation  $r^2 - c_1 r - c_2 = 0$ . This polynomial in  $r$  is called the *characteristic polynomial* for the recurrence relation. By factoring or by the quadratic formula, such equations are easily solved and will in general have two solutions (we'll call them  $r_1$  and  $r_2$ ), although it is possible to have a "double root" (in which case  $r_1 = r_2$  and the characteristic polynomial factors as  $(r - r_1)(r - r_2)$ ).

So far we know that for any constant  $\alpha$ ,  $\alpha r^n$  is a solution to the recurrence relation in (3), but we don't know if those are the only solutions. In fact they are not, we need a slightly more general guess. Our guess will be that the solution is a combination of solutions of the form above (using both solutions to the characteristic polynomial).

### The general method

The solutions to a linear homogeneous recurrence relation of degree two can be solved by the method of parameterized guessing using the following guesses:

1. If the characteristic polynomial has two solutions, then  $a(n) = \alpha_1 r_1^n + \alpha_2 r_2^n$ , where  $r_1$  and  $r_2$  are the two solutions of  $r^2 = c_1 r + c_2$  [i.e., of  $0 = r^2 - c_1 r - c_2$ ],
2. If the characteristic polynomial has one solution (a double root):  $a(n) = \alpha_1 r^n + \alpha_2 n r^n$ , where  $r$  is the only solution of  $r^2 = c_1 r + c_2$  [i.e., of  $0 = r^2 - c_1 r - c_2$ ].

It turns out that these guesses are always correct, and it is not terribly difficult to find the parameters involved. We can find the roots of the *characteristic polynomial*  $0 = r^2 - c_1 r - c_2$  by factoring or by the quadratic formula. We can then find  $\alpha_1$  and  $\alpha_2$  by comparing with the first two terms of the sequence ( $a(0)$  and  $a(1)$ ).

### Examples

1.  $a(n) = a(n-1) + 2a(n-2)$ ;  $a(0) = 5$ ;  $a(1) = 4$ .

This is linear homogeneous of degree 2. The characteristic polynomial is  $r^2 - r - 2 = (r-2)(r+1)$  so the two roots are 2 and -1.  $\alpha_1 2^0 + \alpha_2 (-1)^0 = \alpha_1 + \alpha_2 = a(0) = 5$ , and  $\alpha_1 2^1 + \alpha_2 (-1)^1 = 2\alpha_1 - \alpha_2 = a(1) = 4$ . Solving for  $\alpha_1$  and  $\alpha_2$  shows that  $\alpha_1 = 3$  and  $\alpha_2 = 2$ , so

$$a(n) = 3 \cdot 2^n + 2(-1)^n = 3 \cdot 2^n \pm 2.$$

(Whether we add or subtract 2 depends on the parity of  $n$ .)

2.  $a(n) = 6a(n-1) - 9a(n-2)$ ;  $a(0) = 2$ ;  $a(1) = 9$ .

This is linear homogeneous of degree 2. The characteristic polynomial is  $r^2 - 6r + 9 = (r-3)(r-3)$  so there is only one root (3). This means that  $a(n) = \alpha_1 3^n + \alpha_2 n 3^n$  for some  $\alpha_1$  and  $\alpha_2$ .  $\alpha_1 3^0 + \alpha_2 0(3)^0 = \alpha_1 = a(0) = 2$ , and  $\alpha_1 3^1 + \alpha_2 (1)(3)^1 = 2(3) + \alpha_2 (1)(3) = a(1) = 9$ . So  $\alpha_2 = 1$ , and

$$a(n) = 2 \cdot 3^n + n 3^n.$$