14 Solving Recurrence Relations

Solving recurrence relations is a difficult business and there is no catch all method. However, many relations arising in practice are simple and can be solved with moderate effort.

A few functions. A solution to a recurrence relation is generally given in terms of a function, eg. $f(n) = n \log_2 n$, or a class of similar functions, eg. $T(n) = \mathrm{O}(n \log_2 n)$. It is therefore useful to get a feeling for some of the most common functions that occur. By plotting the graphs, as in Figure 13, we get an initial picture. Here we see a sequence of progressively faster growing functions: constant, logarithmic, linear, and exponential. However,

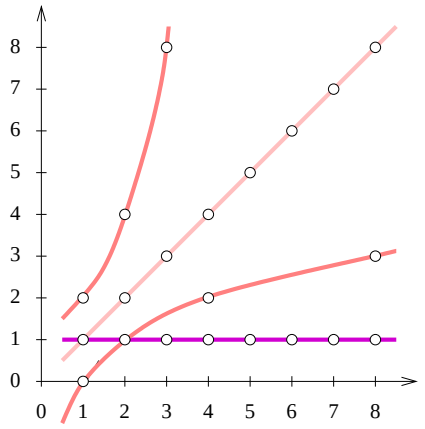

Figure 13: The graphs of a small set of functions, $f(x) = 1$, $f(x) = \log_2 x$, $f(x) = x$, $f(x) = 2^x$.

such plots can be confusing because they depend on the scale. For example, the exponential function, $f(x) = 2^x$, grows a lot faster than the quadratic function, $f(x) = x^2$, but this would not be obvious if we only look at a small portion of the plane like in Figure 13.

Three regimes. In a recurrence relation, we distinguish between the *homogeneous* part, the recursive terms, and the *inhomogeneous* part, the work that occurs. The solution of depends on the relative size of the two, exhibiting qualitatively different behavior if one dominates the other or the two are in balance. Recurrence relations that exhibit this three-regime behavior are so common that it seems worthwhile to study this behavior in more detail. We summarize the findings.

MASTER THEOREM. Let $a \geq 1$ and $b > 1$ be integers and $c \geq 0$ and $d > 0$ real numbers. Let $T(n)$ be defined for integers that are powers of b by

$$T(n) \quad = \quad \begin{cases} aT(\frac{n}{b}) + n^c & \text{if} \quad n > 1 \\ d & \text{if} \quad n = 1. \end{cases}$$

Then we have the following:

- $T(n) = \Theta(n^c)$ if $\log_b a < c$;
- $T(n) = \Theta(n^c \log n)$ if $\log_b a = c$;
- $T(n) = \Theta(n^{\log_b a})$ if $\log_b a > c$.

This behavior can be explained by recalling the formula for a geometric series, $(r^0 + \ldots + r^{n-1})A = \frac{1-r^n}{1-r}A$, and focusing on the magnitude of the constant factor, r. For $0 < r < 1$, the sum is roughly A, the first term, for $r = 1$, the sum is n, the number of terms, and for $r > 1$, the sum is roughly $r^{n-1}A$, the last term.

Let us consider again the recursion tree and, in particular, the total work at its i-th level, starting with $i = 0$ at the root. There are a^i nodes and the work at each node is $(\frac{n}{b^i})^c$. The work at the i-th level is therefore

$$a^i \left(\frac{n}{b^i}\right)^c \quad = \quad n^c \frac{a^i}{b^{ic}}.$$

There are $1 + \log_b n$ levels, and the total work is the sum over the levels. This sum is a geometric series, with factor $r = \frac{a}{b^c}$. It is therefore dominated by the first term if $r < 1$, all terms are the same if $r = 0$, and it is dominated by the last term if $r > 1$. To distinguish between the three cases, we take the logarithm of r, which is negative, zero, positive if $r < 1$, $r = 1$, $r > 1$. It is convenient to take the logarithm to the basis b. This way we get

$$\begin{aligned} \log_b \frac{a}{b^c} \quad &= \quad \log_b a - \log_b b^c \\ &= \quad \log_b a - c. \end{aligned}$$

We have $r < 1$ iff $\log_b a < c$, In which case the dominating term in the series is n^c. We have $r = 1$ iff $\log_b a = c$, in which case the total work is $n^c \log_b n$. We have $r > 1$ iff $\log_b a > c$, in which case the dominating term is $d \cdot a^{\log_b n} = d \cdot n^{\log_b a}$. This explains the three cases in the theorem.

There are extensions of this result that discuss the cases in which n is not a lower of b, we have floors and ceilings in the relation, a and b are not integers, etc. The general behavior of the solution remains the same.

Using induction. Once we know (or feel) what the solution to a recurrence relation is, we can often use induction to verify. Here is a particular relation defined for integers that are powers of 4:

$$T(n) \quad = \quad \begin{cases} T(\frac{n}{2}) + T(\frac{n}{4}) + n & \text{if} \quad n > 1 \\ 1 & \text{if} \quad n = 1. \end{cases}$$

To get a feeling for the solution, we group nodes with equal work together. We get n once, $\frac{n}{2}$ once, $\frac{n}{4}$ twice, $\frac{n}{8}$ three times, $\frac{n}{16}$ five times, etc. These are the Fibonacci numbers, which grow exponentially, with basis equal to the golden ratio, which is roughly 1.6. On the other hand, the work shrinks exponentially, with basis 2. Hence, we have a geometric series with factor roughly 0.8, which is less than one. The dominating term is therefore the first, and we would guess that the solution is some constant times n. We can prove this by induction.

CLAIM. There exists a positive constant c such that $T(n) \leq cn$.

PROOF. For $n = 1$, we have $T(1) = 1$. Hence, the claimed inequality is true provided $c \geq 1$. Using the strong form of Mathematical Induction, we get

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + n \\ &= c\frac{n}{2} + c\frac{n}{4} + n \\ &= \left(\frac{3c}{4} + 1\right) n. \end{aligned}$$

This is at most cn provided $\frac{3c}{4} + 1 \leq c$ or, equivalently, $c \geq 4$. □

The inductive proof not only verified what we thought might be the case, but it also gave us the smallest constant, $c = 4$, for which $T(n) \leq cn$ is true.

Finding the median. Similar recurrence relations arise in practice. A classic example is an algorithm for finding the k-smallest of an unsorted set of n items. We assume the items are all different. A particularly interesting case is the middle item, which is called the *median*. For odd n, this is the k-smallest with $k = \frac{n+1}{2}$. For even n, we set k equal to either the floor or the ceiling of $\frac{n+1}{2}$. The algorithm takes four steps to find the k-smallest item.

STEP 1. Partition the set into groups of size 5 and find the median in each group.

STEP 2. Find the median of the medians.

STEP 3. Split the set into S, the items smaller than the median of the medians, and L, the items larger than the median of the medians.

STEP 4. Let $s = |S|$. If $s < k-1$ then return the $(k-s)$-smallest item in L. If $s = k-1$ then return the median of the medians. if $s > k-1$ then return the k-smallest item in S.

The algorithm is recursive, computing the median of roughly $\frac{n}{5}$ medians in Step 2, and then computing an item either in L or in S. To prove that the algorithm terminates, we need to show that the sets considered recursively get strictly smaller. This is easy as long as n is large but tricky for small n. We ignore these difficulties.

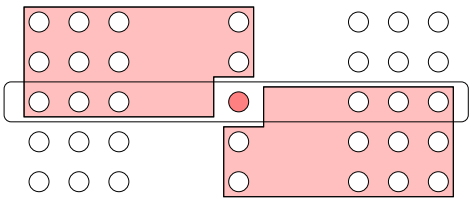

Figure 14: The upper left shaded region consists of items smaller than the median of the medians. Symmetrically, the lower right shaded region consists of items larger than the median of the medians. Both contain about three tenth of all items.

To get a handle on the running time, we need to estimate how much smaller than n the sets S and L are. Consider Figure 14. In one iteration of the algorithm, we eliminate either all items smaller or all items larger than the median of the medians. The number of such items is at least the number in one of the two shaded regions, each containing roughly $\frac{3n}{10}$ items. Hence, the recurrence relation describing the running time of the algorithm is

$$T(n) \quad = \quad \begin{cases} T(\frac{7n}{10}) + T(\frac{n}{5}) + n & \text{if} \quad n > n_0 \\ n_0 & \text{if} \quad n \leq n_0, \end{cases}$$

for some large enough constant n_0. Since $\frac{7}{10} + \frac{1}{5}$ is strictly less than one, we guess that the solution to this recurrence relation is again $\mathrm{O}(n)$. This can be verified by induction.