Lecture II
RECURRENCES

Recurrences arise naturally in analyzing the complexity of recursive algorithms and in probabilistic analysis. We introduce some basic techniques for solving recurrences. A recurrence is a recursive relation for a complexity function $T(n)$. Here are two examples:

$$F(n) = F(n-1) + F(n-2) \tag{1}$$

and

$$T(n) = n + 2T(n/2). \tag{2}$$

The reader may recognize the first as the recurrence for Fibonacci numbers, and the second as the complexity of the Merge Sort, described in Lecture 1. These recurrences have the following "separable form":

$$T(n) = g(T(n_1), \ldots, T(n_k)) \tag{3}$$

where $g(x_1, \ldots, x_k)$ is a function in k variables (for some fixed k) and $n_1, \ldots, n_k$ are all strictly less than n.

What does it mean to "solve" recurrences such as equations (1) and (2)? We consider the following acceptable solutions: $F(n) = \Theta(\phi^n)$ where $\phi = (1+\sqrt{5})/2 = 1.618\ldots$ is the golden ratio, and $T(n) = \Theta(n \log n)$. In this course, we generally want to determine the function $T(n)$ in (3) only up to Θ-order. Sometimes, only an upper bound is needed, and we determine $T(n)$ up to its O-order. In special cases, we may be able to derive the exact solution but this may be difficult. A nice benefit of Θ-order solutions is this – all the recurrences we treat can be solved by only elementary methods, without using calculus.

The variable "n" is called the **designated variable** of the recurrence (3). In case there are non-designated variables, these are supposed to be held constant. In mathematics, we usually reserve "n" for natural numbers or perhaps integers. In the above examples, this is the natural intepretation for n. But one of the first steps we take in solving recurrences is to extend reinterpret n (or whatever the designated variable is) to range over the real numbers. For this reason, we may prefer the symbol "x" as our designated variable, since x is often viewed as a real variable.

What does an extension to real numbers mean? In the Fibonacci recurrence (1), what is $F(2.5)$? In Merge Sort (2), what does $T(\pi) = T(3.14159\ldots)$ represent? The short answer is, we don't really care.

In addition to the recurrence (3), we generally need the **boundary conditions** or **initial values** of the function $T(n)$. They give us the values of $T(n)$ *before* the recurrence (3) becomes valid. Without initial values, $T(n)$ is generally under-determined. For our example (1), if n range over natural numbers, then the initial conditions

$$F(0) = 0, \qquad F(1) = 1$$

give rise to the standard Fibonacci numbers, *i.e.*, $F(n)$ is the nth Fibonacci number. Thus $F(2) = 1, F(3) = 2, F(4) = 3$, etc. On the other hand, if we use the initial conditions $F(0) = F(1) = 0$, then the solution is trivial: $F(n) = 0$ for all $n \geq 0$. Thus, our assertion earlier that $F(n) = \Theta(\phi^n)$ is the solution to (1) is not completely true without knowing the initial conditions. On the other hand, $T(n) = O(n \log n)$ can be shown to hold for (2) regardless of the initial conditions.

§1. Simplification

In the real world (as opposed to a classroom situation), when faced with an actual recurrence to be solved, there is usually some simplifications steps to be taken. This section suggests some guidelines. There are three generally applicable simplifications:

- **Default Initial Conditions.** In this book, we normally state recurrence relations without any specific initial conditions. This is deliberate: we expect the student to supply some non-trivial initial conditions. The **default initial condition** has the form

$$T(n) = O(1) \qquad \text{for } \; n \le n_0, \tag{4}$$

  for some n_0. Thus, the recurrence relations is assumed to hold for $n \ge n_0$. The intent is for the student to choose convenient choices for n_0 and the $O(1)$ function. For instance, we may choose $T(n) = C$ for $n \le n_0$, for some constant C. When we derive exact bounds, C and n_0 may be chosen so as to simplify the final form of $T(n)$.

  What is the justification for this approach? It allows us to focus on the recurrence itself rather than the initial conditions. In many cases, this arbitrariness does not affect the asymptotic behavior of the solution. But even if it does, it may not affect the method of solving the recurrence, which is our focus.

- **Extension to Real Functions.** Even if the function $T(n)$ is originally defined for natural numbers n, we will now treat $T(n)$ as a real function (*i.e.*, n is viewed as a real variable), and defined for n sufficiently large. Under the default condition (4), we assume $T(n)$ is define for all $n > n_0$.

- **Convert into a Recurrence Equation.** If we begin with an recurrence inequality such as $T(n) \le g(T(n_1), \ldots, T(n_k))$, we simply treat it as an equality: $T(n) = g(T(n_1), \ldots, T(n_k))$. Our eventual solution for $T(n)$ can only an upper bound. But if we started with $T(n) \ge g(T(n_1), \ldots, T(n_k))$ instead, then the eventual solution should be a lower bound.

Specific Simplications. This depends on the recurrence at hand. Suppose the running time of an algorithm satisfies the following inequality:

$$T(n) \le T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n, \tag{5}$$

for integer $n > 10$. The boundary condition is $T(n) = 3n^2 - 4n + 2$ for $0 < n \le 10$. Such a **recurrence inequation** may arises in some imagined implementation of Merge Sort. Our general simplifications steps already tells us to (1) discard the specific boundary conditions in favor of (4), (2) treat $T(n)$ as a real function, and (3) write the recurrence as a equation.

What other simplications might apply here? Let us convert (5) into the following

$$T(n) = 2T(n/2) + n. \tag{6}$$

We have (i) replaced the term "$+6n$" by "$+n$", and (ii) removed the ceiling and floor functions. Step (i) is justified because this does not affect the Θ-order (if this is not clear, then you can always come back to verify this claim). Step (ii) exploits the fact that we now treat $T(n)$ as a real function, so we need not worry about non-integral arguments when we remove the ceiling or floor functions. Also, it does not affect the asymptotic value of $T(n)$ here.

These remarks are hardly supposed to be apparent − they look reasonable and experience may have shown that they are normally harmless. To be sure, you should eventually return to these assumptions to justify[1] them rigorously. Indeed, you can construct counter-examples for which these simplifications affect the Θ-order. Sometimes, removing ceiling functions may not work. Thus, in $T(n) = \sum_{i=1}^{\lceil n/2 \rceil} T(i)$ the upper limit of the sum must be a natural number. Changing the boundary condition may affect the Θ-order of the solution, as we saw in the case of the Fibonacci recurrence (1).

Exercises

[1] Alternatively, armed with a solution for the simplified recurrence, we might now be able to solve the original recurrence with its boundary conditions.

Exercise 1.1: Can you formulate a class of recurrences that is is more general than the "separable case" given by (3)? Give a concrete example of this. //ignore ♢

Exercise 1.2: Show that our above simplifications of the the recurrence (5) (with its initial conditions) cannot affect the asymptotic order of the solution. [Note that you must show that this for ANY choice of a default boundary condition.] ♢

Exercise 1.3: Show counterexamples to the claim that we can replace $\lceil n/2 \rceil$ by $n/2$ in a recurrence without changing the Θ-order of the solution.
(a) Construct a function $g(n)$ that provides a counter example for the followin recurrence: $T(n) = T(\lceil n/2 \rceil) + g(n)$. HINT: make $g(n)$ depend on the parity of n. (b) Construct a different recurrence to provide a counter example.
♢

Exercise 1.4: Construct examples such that the following modifications lead to asymptotically differente solutions.
(a) Removing a ceiling function (say, replace $T(\lceil n/2 \rceil)$ by $T(n/2)$).
(b) Modifying the initial conditions. ♢

Exercise 1.5: Suppose $T(n)$ satisfies a recurrence equation of the form $T(n) = g(T(n_1))$ where $n_1 < n$ and $g(\cdot)$ is an arbitrary function. Assume that the solution of this recurrence with respect to some specific initial conditions (C) yields the solution $T(n) = \Theta(n^k)$ for some constant $k > 0$. Do the following modifications affect the Θ-order of the solution?
(a) The use of our default initial conditions instead of (C).
(b) Using the modified recurrence $T(n) = g(T(n_1+c))$ where c is any real constant, positive or negative, such that $n_1 + c < n$. ♢

Exercise 1.6: Suppose x, n are positive numbers satisfying the following "recurrence" equation,

$$2^x = x^{2n}.$$

Solve for x as a function of n, showing

$$x(n) = [1 + o(1)]2n \log_2(2n).$$

HINT: take logarithms. This is an example of a bootstrapping argument where we use an approximation of $x(n)$ to derive yet a better approximation.

♢

Exercise 1.7: [Ample Domains] Consider the simplification of (5) to (6). Suppose, instead of assuming $T(n)$ to be a real function (so that (6) makes sense for all values of n), we continue to assume n is a natural number. It is easy to see that $T(n)$ is completely defined by (6) iff n is a power of 2. We say that (6) is closed over the set $D_0 := \{2^k : k \in \mathbb{N}\}$ of powers of 2. In general, we say a recurrence is "closed over a set $D \subseteq \mathbb{R}$" if for all $n \in D$, the recurrence for $T(n)$ depends only on smaller values n_i that also belong in D (unless n_i lies within the boundary condition).
(a) Let us call a set $D \subseteq \mathbb{R}$ an "ample set" if, for some $\alpha > 1$, the set $D \cap [n, \alpha \cdot n]$ is non-empty for all $n \in \mathbb{N}$. Here $[n, \alpha n]$ is closed real interval between n and αn. If the solution $T(n)$ is sufficiently "smooth", then knowing the values of $T(n)$ at an ample set D gives us a good approximation to values where $n \notin D$. In this question, our "smoothness assumption" is simply: $T(n)$ *is monotonic*

non-decreasing. Suppose that $T(n) = n^k$ for n ranging over an ample set D. What can you say about $T(n)$ for $n \notin D$? What if $T(n) = c^n$ over D? What if $T(n) = 2^{2^n}$ over D?
(b) Suppose $T(n)$ is recursively expressed in terms of $T(n_1)$ where $n_1 < n$ is the largest prime smaller than n. Is this recurrence defined over an ample set? ♢

Exercise 1.8: Let T, T' be binary trees and $|T|$ denote the number of nodes in T. Define the relation $T \sim T'$ recursively as follows: (BASIS) If $|T| = 0$ or 1 then $|T| = |T'|$. (INDUCTION) If $|T| > 1$ then $|T'| > 1$ and either (i) $T_L \sim T'_L$ and $T_R \sim T'_R$, or (ii) $T_L \sim T'_R$ and $T_R \sim T'_L$. Here T_L and T_R denote the left and right subtrees of T.
(a) Use this to give a recursive algorithm for checking if $T \sim T'$.
(b) Give the recurrence satisfied by the running time $t(n)$ of your algorithm.
(c) Give asymptotic bounds on $t(n)$. ♢

End Exercises

§2. Karasuba Multiplication

Let us see another interesting recurrence that arise in the analysis of an algorithm of Karatsuba.

We learn a fairly non-trivial algorithm in high school, namely a method to multiply two integers. Given positive integers X, Y, we want to compute Z which is their produce XY. Usually we think of X, Y in decimal notation, but the algorithm works equally well for binary notation. We assume binary notation for simplicity. In any case, if X and Y are at most n digits each, then the high school algorithm clearly takes $\Theta(n^2)$ time. Can we improve on this?

Assume X and Y has length exactly n where n is a power of 2 (we can padd with 0's if necessary). Let us split up X into a high-order half X_1 and low-order half X_0. Thus

$$X = X_0 + 2^{n/2} X_1$$

where X_0, X_1 are $n/2$-bit numbers. Similarly,

$$Y = Y_0 + 2^{n/2} Y_1.$$

Then

$$\begin{aligned} Z &= (X_0 + 2^{n/2} X_1)(Y_0 + 2^{n/2} Y_1) \\ &= X_0 Y_0 + 2^{n/2}(X_1 Y_0 + X_0 Y_1) + 2^n X_1 Y_1 \\ &= Z_0 + 2^{n/2} Z_1 + 2^n Z_1, \end{aligned}$$

where $Z_0 = X_0 Y_0$, etc. Clearly, each of these Z_i's have at most $2n$ bits. Now, if we compute the 4 products

$$X_0 Y_0, X_1 Y_0, X_0 Y_1, X_1 Y_1$$

recursively, then we can put them together ("conquer step") in $O(n)$ time. To see this, we must make an observation: in binary notation, multiplying any number X by 2^k (for any positive integer k) takes $O(k)$ time, independent of X. We can view this as a matter of shifting left by k, or by appending a string of k zeros to X.

Hence, if $T(n)$ is the time to multiply two n-bit numbers, we obtain the recurrence

$$T(n) \leq 4T(n/2) + Cn \tag{7}$$

for some $C > 1$. Given our simplification suggestions, we immediately rewrite this as

$$T(n) = 3T(n/2) + n.$$

It turns out that this recurrence has solution $T(n) = \Theta(n^2)$, so we have not really improved on the high-school method.

Karatsuba observed that we can proceed as follows: we can compute $Z_0 = X_0Y_0$ and $Z_2 = X_1Y_1$ first. Then we can compute Z_1 using the formula

$$Z_1 = (X_0 + X_1)(Y_0 + Y_1) - Z_0 - Z_2.$$

Thus Z_1 can be computed with one recursive multiplication plus some addition $O(n)$ work. From Z_0, Z_1, Z_2, we can again obtain Z in $O(n)$ time. This gives us the **Karatsuba recurrence**,

$$T(n) = 3T(n/2) + n. \tag{8}$$

We shall show that $T(n) = \Theta(n^\alpha)$ where $\alpha = \lg 3 = 1.58\cdots$. This is clearly an improvement of the high school method.

The recurrences (2), (7) and (8) are all instances of the Master recurrence

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

where $a > 0$ and $b > 1$ are constants and f any reasonably smooth function. We shall solve this recurrence rather completely.

Exercises

Exercise 2.1: (a) Suppose you are told that the running time of an algorithm is $O(n^\alpha)$ for some $\alpha > 0$. How can you estimate α by timing your algorithm on various input sizes? Assume you can find a "best fit" straight line to a set of points.
(b) In the programming language `Java` you can multiply numbers of arbitrarily large size. Determine the constant α for the multiplication algorithm in your version of `Java`. $\diamondsuit$

End Exercises

§3. Rote Method

We are going to introduce two "direct methods" methods for solving recurrences: rote method and induction. They are "direct" as opposed to other transformational methods which we will introduce later. Although fairly straightforward, these direct methods do call for some creativity (educated guesses). We begin with the rote method, as it appears to require somewhat less guess work.

Expand, Guess, Verify, Stop. The "rote method" may be thought of as the method of repeated expansion of a recurrence. Actually, this is only the first of 4 distinct stages. After several expansion steps, you guess the general term in the growing summation. Next, you verify your guess by natural induction. Finally, we must terminate the process by choosing a base of induction. The creative part of this process lies in the guessing step.

Using our above example (equation (6)),

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 4T(n/4) + n + n \\ &= 8T(n/8) + n + n + n \end{aligned}$$

At this point, we may guess that the $(i-1)$st step of this expansion yields

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

for a general i. To verify our guess, we expand the guessed formula one more time,

$$\begin{aligned} T(n) &= 2^i[2T(n/2^{i+1}) + n/2^i] + in \\ &= 2^{i+1}T(n/2^{i+1}) + (i+1)n. \end{aligned}$$

Thus the formula is verified for $i+1$. We must now stop this induction process. It is natural to choose $i = \log_2 n$ if n is a power of 2. Curiously, notice that even though we now intepréte n as a real number, it is not at all obvious what it means to extend i to a non-integral number! What does it mean to expand the recurrence $3/2$ times? So, restricting i to a natural number, we now choose

$$i = 1, \ldots, \lfloor \log_2 n \rfloor.$$

With this choice of i, we see that $1 \le n/2^i < 2$. Hence we may choose the initial condition to be

$$T(n) = 0, \qquad \text{for } n < 2.$$

This yields the *exact* solution

$$T(n) = n \lfloor \log_2 n \rfloor.$$

If the boundary condition is not for us to choose, we can easily adjust for this although the solution may not be as elegant.

The appearance of the floor function in this solution makes $T(n)$ non-continuous whenever n is a power of 2. This can be avoided if we exploit our great freedom in specifying boundary conditions. Let us now assume that $T(n) = n\log_2 n$ for $1 \le n < 2$. We may let $T(n) = 0$ for $n < 1$. Then the above proof shows that $T(n) = n\log_2 n$ for $n \ge 1$

To summarize, the rote method consists of (E) expansions steps, (G) guessing of a general formula, (V) verification of the formula, and (S) choice of stopping point. It should be pointed out that this method may not always work. Although you can always perform expansions, you may be stuck at the next step trying to guess a formula. But when the method works, it gives you the exact solution. It should probably be the first method to try.

Exercises

Exercise 3.1: No credit work: Rote learning is much discredited and so we would like a more dignified name for this method. We can call this the 4-Fold Path or the the "EGVS Method" (Expand, Guess, Verify, Stop). In a humorous vein, what can EGVS stand for? ♢

Exercise 3.2: Use the Rote Method to solve the following recurrences
(a) $T(n) = n + 8T(n/2)$.
(b) $T(n) = n + 16T(n/4)$.

(c) Can you generalize your results from (a) and (b) to solving recurrences of the form $T(n) = n + aT(n/b)$ for $b > 1$ and $a > 0$? ♢

Exercise 3.3: Consider equation (6). Fix any $k > 1$. Show by induction that $T(n) = \mathcal{O}(n^k)$. Which part of your argument suggests to you that this solution is not tight? ♢

Exercise 3.4: Give the exact solution for $T(n) = 2T(n/2) + n$ for $n > 1$ and $T(n) = 0$ for $n \leq 1$. ♢

Exercise 3.5: Consider the recurrence $T(n) = n + 10T(n/3)$. Suppose we want to show $T(n) = \mathcal{O}(n^3)$.
(a) Attempting to prove by real induction, students often begin with a statement such as "Using the default initial conditions, we may assume that there is some $C > 0$ and $n_0 > 0$ such that $T(n) \leq Cn^3$ for all $n < n_0$". What is wrong with this statement?
(b) Give a correct proof by real induction.
(c) Suppose $T(n) = n + 10T((n + K)/2)$ for some constant K. How does your proof in (b) change? ♢

Exercise 3.6: Let $T(n) = 2T(\frac{n}{2} + c) + n$ for some $c > 0$.
(a) By choosing suitable initial conditions, prove the following bounds on $T(n)$ by induction, and *not* by any other method:
(a.1) $T(n) \leq D(n - 2c)\lg(n - 2c)$ for some $D > 1$. Is there a smallest D that depends only on c? Explain. Similarly, show $T(n) \geq D'(n - 2c)\lg(n - 2c)$ for some $D' > 0$.
(a.2) $T(n) = n \lg n - o(n)$.
(a.3) $T(n) = n \lg n + \Theta(n)$.
(b) Obtain the exact solution to $T(n)$.
(c) Use your solution to (b) to explain your answers to (a). ♢

Exercise 3.7: Consider recurrences of the form

$$T(n) = (T(n-1))^2 + g(n). \tag{9}$$

In this exercise, we restrict n to natural numbers and use explicit boundary conditions. (a) Show that the number of binary trees of height at most n is given by this recurrence with $g(n) = 1$ and the boundary condition $T(1) = 1$. Show that this particular case of (9) has solution

$$T(n) = \left\lfloor k^{2^n} \right\rfloor. \tag{10}$$

(b) Show that the number of Boolean functions on n variables is given by (9) with $g(n) = 0$ and $T(1) = 2$. Solve this.
NOTE: Aho and Sloane (1973) investigate the recurrence (9). ♢

End Exercises

§4. Real Induction

The student should be familiar with **natural induction**, a method of proof based on induction over natural numbers. In brief, natural induction is this. Suppose $P(n)$ is a predicate about the natural number n. For example, $P(n)$ might be "There is a prime number between n and $n+10$ inclusive". When $n = 200$, we may verify[2] that $P(200)$ is false. In general, a **predicate** is a function of its arguments (in this case, n is the only argument) such that for each instance of its arguments, the result is a proposition. A **proposition** is either true or false. Suppose we want to prove the following proposition based on $P(n)$.

$$\text{For all } n \in \mathbb{N}, P(n) \text{ holds.} \tag{11}$$

Here is the usual proof using "natural induction": first we show that $P(0)$ holds. Then we show that if $P(k)$ holds for all $k = 0, 1, \ldots, n-1$, then $P(n)$ holds. Finally, we simply invoke the **principle of natural induction** to deduce the statement (11).

We now introduce **real induction**, which can be used as a method for solving recurrences. It has many similarities to natural induction. Suppose we wish to solve a recurrence equation for $T(x)$ where $T(x)$ is viewed as a real function. We can use induction to first show that

$$T(x) = \mathcal{O}(f(x)), \tag{12}$$

and use another induction to prove that $T(x) = \Omega(g(x))$. So to use this method, you need to guess appropriate $f(x)$ and $g(x)$. If successful, and $f(x) = \Theta(g(x))$, then we have a Θ-order bound on $T(x)$. If the bounds are not tight, you can further refine by trying a smaller $f(x)$ and/or a larger $g(x)$.

So, the question reduces to how one proves a statement of the form

$$\text{For all } x \in \mathbb{R}, P(x) \text{ holds,} \tag{13}$$

where $P(x)$ is a predicate on real numbers. In (12) above, $P(x)$ is the predicate "if $x \geq x_0$ then $T(x) \leq C \cdot f(x)$)" where x_0 and $C > 0$ are real constants. The answer lies in the "principle of real induction". We now formulate a version of this principle.

Theorem 1 (Principle of Real Induction) *Let $P(x)$ be a predicate on real numbers x. Suppose there exists real numbers x_1 and $\delta > 0$ such that*

Real Basis: *For all $x < x_1$, $P(x)$ holds.*

Real Induction Hypothesis: *For all $y \geq x_1$, if $(\forall x < y - \delta)[P(x)$ holds$]$ then $P(y)$ holds.*

Then for all x, $P(x)$ holds.

Proof. Define the function $t(x) = \lfloor \frac{x - x_1}{\delta} \rfloor$ and $Q(n)$ be the predicate

$$(\forall x)[t(x) < n \Rightarrow P(x)\text{holds}].$$

Clearly, $Q(0)$ holds by the Real Basis assumption. Suppose $Q(n)$ holds for some natural number n. Consider y such that $t(y) = n$. Then for all $z < y - \delta$,

$$t(z) \leq \lfloor (y - \delta - x_1)/\delta \rfloor = t(y) - 1 = n - 1.$$

But $t(z) < n$ and $Q(n)$ implies $P(z)$. Now the Real Induction Hypothesis tells us that $P(y)$ holds. This fact, plus $Q(n)$, is equivalent to $Q(n+1)$. By the principle of natural induction, $Q(n)$ holds for all n. But this is equivalent to saying $P(x)$ holds for all x. **Q.E.D.**

[2] The smallest n such that $P(n)$ is true is $n = 114$.

Example: Suppose that $T(x)$ satisfies the recurrence

$$T(x) = x^5 + T(x/a) + T(x/b) + T(x/c) \tag{14}$$

where $a^{-5} + b^{-5} + c^{-5} = k_0 < 1$ and $a \ge b \ge c > 1$. Let $P(x)$ be the assertion[3]

$$x \ge x_0 \Rightarrow T(x) \le Kx^5,$$

for suitable $x_0 > 0$ and $K > 0$. By our default initial condition, there is some $C > 0$ and $x_0 > 0$ such that

$$T(x) \le C$$

for all $x \le x_0$. Let $x_1 = \max\left\{x_0, \frac{x_0}{C-1}\right\}$. Clearly, we may choose $K = K(C, x_1)$ such that $T(x) \le Kx^5$ for all $0 < x < x_1$. Let $\delta = x_0/c$. Suppose $y \ge x_1$. The **Real Induction Hypothesis for** x is that all $y < x - \delta$, $P(y)$ holds. We need to infer that $P(x)$ holds. To see this,

$$\begin{aligned} T(x) &= x^5 + T(x/a) + T(x/b) + T(x/c) \\ &\le y^5 + K \cdot (x/a)^5 + K \cdot (x/b)^5 + K \cdot (x/c)^5, \qquad \text{(since } x/c \le x - \delta, x(1-1/c) \ge \delta, x \ge \delta) \\ &= x^5(1 + K \cdot k_0) \\ &\le Kx^5 \end{aligned}$$

where the last inequality is true provided we ensure that our original choice of K further satisfies $1+K\cdot k_0 \le K$ or $K \ge 1/(1-k_0)$.

In a similar vein, we can prove a matching lower bound on $T(x)$ using real induction. In summary, the induction method has two basic steps: (i) first guess an upper bound (and a lower bound if desired), and (ii) verify the guess by induction.

As this example shows, the direct application of the principle of real induction is tedious, as we have to keep track of the constants δ and x_1. Instead, we give a simple rule which is easy to use:

Theorem 2 *Assume the real recurrence for $T(n)$ is separable with the form given in (3). Let the parameters $n_1, n_2, \ldots, n_k$ be of the form n/b for some $b > 1$ or $n - c$ for some constant $c > 0$. Let $P(t, n)$ be a predicate on real numbers t, n. Suppose that for sufficiently large $n_1, \ldots, n_k$, the following holds*

$$P(T(n_1), n_1), \ldots, P(T(n_1), n_k) \Rightarrow P(T(n), n).$$

Then for some default boundary condition, we may conclude that $P(T(n), n)$ holds for n large enough,

Summary. The guessing of a bounding function is generally more creative than verifying the guess. One phenomenon that arises is that one often has to introduce a strong induction hypothesis than the actual result aimed for. For instance, to prove that $T(x) = O(x \log x)$, we may need to guess that $T(x) = Cx \log x + Dx$ for some $C, D > 0$.

A comparison of the rote method with the induction method is in order: in general, the rote method is a more accurate tool than the induction method. It can often give an exact solution, not just asymptotic bounds. On the other hand, it seems clear that we can always use the induction method even when the rote method fails. This is illustrated in the section below with a recurrence whose solution is $T(x) = \Theta(\lg^2 x)$. Another thing to notice is that natural induction lurks behind both rote and induction methods.

[3] Note that we need the condition "$x \ge x_0 > 0$" since x^5 is not even defined for $x \le 0$.

In the rest of this lecture, we indicate other systematic pathways, influenced by the lecture notes of Mishra and Siegel [7], and the books of Knuth [5], Greene and Knuth [3]. See also Purdom and Brown [9] and the survey of Lueker [6].

Exercises

Exercise 4.1: Give another proof of theorem 1, by using contradiction. ♢

Exercise 4.2: Suppose $T(x) = 3T(x/2) + x$. Show by real induction that $T(x) = \Theta(x^{\lg 3})$. ♢

Exercise 4.3: Let us introduce the "multiterm master recurrence",

$$T(x) = f(x) + \sum_{i=1}^{k} a_i T\left(\frac{x}{b_i}\right)$$

where $k \geq 1$, $a_i > 0$ and $b_1 > b_2 > \cdots > b_k > 1$ $(i = 1, \ldots, k)$. Suppose $f(x) = \mathcal{O}(x^\alpha)$ for some constant $\alpha > 0$ and $\sum_{i=1}^{k} a_i/b_i^\alpha = c_0 < 1$. Prove that $T(x) = \mathcal{O}(x^\alpha)$. ♢

End Exercises

§5. Basic sums

Consider the recurrence $T(n) = T(n-1) + n$. By rote method, this has the "solution"

$$T(n) = \sum_{i=1}^{n} i,$$

assuming $T(0) = 0$. But the RHS of this equation involves an **open sum**, meaning that the number of summands is unbounded as a function of n. We do not accept this "answer" even though it is perfectly accurate.

What Does It Mean to Solve a Recurrence? Actually, you may have noticed that the open sum above is well-known and is equal to

$$\binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2).$$

We would be perfectly happy with the answer "$T(n) = \Theta(n^2)$". In general, one can *always* express a separable recurrence equation of $T(n)$ as an open sum, by rote expansion. We do not regard this as acceptable. Hence, we are really only interested in solutions which can be written as a **closed sum** or **product**, meaning that the number of terms (i.e., summands or factors) is independent of n. Moreover, each term must be composed of "familiar" functions.

Familiar Functions. So we conclude that "solving a recurrence" is relative to the form of solution we allow. This we interpret to mean a closed sum of "familiar" functions. For our purposes, the functions considered familiar include

$$\text{polynomials } f(n) = n^k, \text{ logarithms } f(n) = \log n, \text{ and exponentials } f(n) = c^n \ (c > 0).$$

Functions such as factorials $n!$, binomial coefficients $\binom{n}{k}$ and harmonic numbers H_n (see below) are tightly bounded by familiar functions, and are therefore considered familiar. Finally, we have a rule saying that *the sum, product and functional composition of familiar functions are considered familiar*. Thus $\log^k n$, $\log\log n$, $n + 2\log n$ and $n^n \log n$ are familiar.

In addition to the above list of functions, two very slow growing functions arise naturally in algorithmic analysis. These are the log-star function $\log^* x$ and the inverse Ackermann function $\alpha(n)$ (see Lecture XII). We will consider them familiar, although functional compositions involving them are only familiar in a very technical sense!

We refer the reader to the Appendix A in this lecture for basic properties of the exponential and logarithm function. In this section, we present some basic closed form summations.

Arithmetic series. The basic arithmetic series is

$$\begin{aligned} S_n &:= \sum_{i=1}^{n} i \\ &= \binom{n+1}{2}. \end{aligned} \tag{15}$$

In proof,

$$2S_n = \sum_{i=1}^{n} i + \sum_{i=1}^{n} (n+1-i) = \sum_{i=1}^{n} (n+1) = n(n+1).$$

More generally, for fixed $k \geq 1$, we have the "arithmetic series of order k",

$$S_n^k := \sum_{i=1}^{n} i^k = \Theta(n^{k+1}). \tag{16}$$

In proof, we have

$$n^{k+1} > S_n^k > \sum_{i=\lceil n/2 \rceil)}^{n} (n/2)^k \geq (n/2)^{k+1}.$$

For more precise bounds, we bound S_n^k by integrals,

$$\frac{n^{k+1}}{k+1} = \int_0^n x^k dx < S_n^k < \int_1^{n+1} x^k dx = \frac{(n+1)^{k+1} - 1}{k+1},$$

yielding

$$S_n^k = \frac{n^{k+1}}{k+1} + \mathcal{O}_k(n^k). \tag{17}$$

Geometric series. For $x \neq 1$ and $n \geq 1$,

$$\begin{aligned} S_n(x) &:= \sum_{i=0}^{n-1} x^i \\ &= \frac{x^n - 1}{x - 1}. \end{aligned} \tag{18}$$

In proof, note that $xS_n(x) - S_n(x) = x^n - 1$. When $n \to \infty$, we get the series

$$\begin{aligned} S_\infty(x) &:= \sum_{i=0}^{\infty} x^i \\ &= \begin{cases} \infty & \text{if } x \geq 1 \\ \uparrow \text{(undefined)} & \text{if } x \leq -1 \\ \frac{1}{1-x} & \text{if } |x| < 1. \end{cases} \end{aligned}$$

One of the simplest infinite series is $\sum_{i=0}^{\infty} x^i$. It also has a very simple closed form solution,

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \tag{19}$$

I call $\sum_{i=0}^{\infty} x^i$. the "mother of series" because, from the, we can derive many other solutions for series, including finite series. In fact, for $|x| < 1$, we can derive equation (18) by plugging equation (19) into

$$S_n(x) = S_\infty(x) - x^n S_\infty(x) = (1 - x^n)S_\infty(x).$$

By differentiating both sides of the mother series with respect to x, we get:

$$\begin{aligned} \frac{1}{(1-x)^2} &= \sum_{i=1}^{\infty} ix^{i-1} \\ \frac{x}{(1-x)^2} &= \sum_{i=1}^{\infty} ix^i \end{aligned} \tag{20}$$

This process can be repeated to yield formulas for $\sum_{i=0}^{\infty} i^k x^i$, for any integer $k \geq 2$. Differentiating both sides of equation (18), we obtain the finite summation analogue:

$$\begin{aligned} \sum_{i=1}^{n-1} ix^{i-1} &= \frac{(n-1)x^n - nx^{n-1} + 1}{(x-1)^2}, \\ \sum_{i=1}^{n-1} ix^i &= \frac{(n-1)x^{n+1} - nx^n + x}{(x-1)^2}, \end{aligned} \tag{21}$$

$$\tag{22}$$

Combining the infinite and finite summation formulas, equations (20) and (21), we finally obtain

$$\sum_{i=n}^{\infty} ix^i = \frac{nx^n - (n-1)x^{n+1}}{(1-x)^2}. \tag{23}$$

We may verify by induction that these formulas actually hold for all $x \neq 1$. In general, for any $k \geq 0$, we obtain formulas for the **geometric series of order** k:

$$\sum_{i=1}^{n-1} i^k x^i. \tag{24}$$

The infinite series have finite values only when $|x| < 1$.

Harmonic series.

$$\begin{aligned} H_n &:= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\ &= \ln n + \Theta(1). \end{aligned}$$

Note that ln is the natural logarithm (appendix A). This is easy to see using calculus,

$$H_n < 1 + \int_1^n \frac{dx}{x} < 1 + H_n.$$

But $\int_1^n \frac{dx}{x} = \ln n$. This proves that $H_n = \ln n + g(n)$ where $0 < g(n) < 1$. More precise estimates for $g(n)$ are known: $g(n) = \gamma + (2n)^{-1} + \mathcal{O}(n^{-2})$ where $\gamma = 0.577...$ is Euler's constant.

For any real $\alpha \geq 1$, we can define the sum

$$H_n^{(\alpha)} := \sum_{i=0}^{n} \frac{1}{i^\alpha}.$$

Thus $H_n^{(1)}$ is just H_n. If we let $n = \infty$, the sum $H_\infty^{(\alpha)}$ is bounded for $\alpha > 1$; it is clearly unbounded for $\alpha = 1$ since $\ln n$ is unbounded. The sum is just the value of the Riemann zeta function at α. For instance, $H_\infty^{(2)} = \pi^2/6$.

Stirling's Approximation. So far, we have treated open sums. If we have an open product such as the factorial function $n!$, we can convert it into an open sum by taking logarithms. This method of estimating an open product may not give as tight a bound as we wish (why?). For the factorial function, there is a family of more direct bounds that are collectively called **Stirling's approximation**. The following Stirling approximation is from Robbins (1955) and it should be committed to memory:

$$n! = \left(\frac{n}{e}\right)^n \sqrt{2\pi n}\, e^{\alpha_n}$$

where

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}.$$

Sometimes, the alternative bound $\alpha_n > (12n)^{-1} - (360n^3)^{-1}$ is useful [2]. Up to Θ-order, we may prefer to simplify the above bound to

$$n! = \Theta\left(\left(\frac{n}{e}\right)^{n+\frac{1}{2}}\right).$$

Binomial theorem.

$$\begin{aligned} (1+x)^n &= 1 + nx + \frac{n(n-1)}{2}x^2 + \cdots + x^n \\ &= \sum_{i=0}^{n} \binom{n}{i} x^i. \end{aligned}$$

In general, the binomial function $\binom{x}{i}$ is defined for all real x and integer i:

$$\binom{x}{i} = \begin{cases} 0 & \text{if } i < 0 \\ 1 & \text{if } i = 0 \\ \frac{x(x-1)\cdots(x-i+1)}{i(i-1)\cdots 2\cdots 1} & \text{if } i > 0. \end{cases}$$

The binomial theorem can be viewed as an application of Taylor's expansion for a function $f(x)$ at $x = a$:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots$$

where $f^{(n)}(x) = \frac{d^n f}{dx^n}$. This expansion is defined provided all derivatives of f exist and the series converges. Applied to $f(x) = (1+x)^p$ for any real p at $x = 0$, we get

$$\begin{aligned}(1+x)^p &= 1 + px + \frac{p(p-1)}{2!}x^2 + \frac{p(p-1)(p-2)}{3!}x^3 + \cdots \\ &= \sum_{i \ge 0} \binom{p}{i} x^i.\end{aligned}$$

Exercises

Exercise 5.1: Solve the recurrence $T(x) = \frac{1}{x} + T(x-1)$ for all $x > 1$. ◊

Exercise 5.2: Let $c > 0$ be any real constant.
(a) Prove that $H_n = o(n^c)$. HINT: first let $c = 1$ and sum the first $\sqrt{n}$ terms of H_n/n.
(b) Show that $\ln(n+c) - \ln n = \mathcal{O}(c/n)$.
(c) Show that $|H_{x+c} - H_x| = \mathcal{O}(c/n)$ where H_x is the generalized Harmonic function.
(d) Bound the sum $\sum_{i=1+\lfloor c \rfloor}^{n} \frac{1}{i(i-c)}$. ◊

Exercise 5.3:
(a) Give the exact value of $\sum_{i=2}^{n} \frac{1}{i(i-1)}$. HINT: use partial fraction decomposition of $\frac{1}{i(i-1)}$.
(b) Conclude that $H_\infty^{(2)}$ is bounded.
(c) Give the asymptotic value of $\sum_{i=1}^{n} \frac{1}{i(n-i)}$. ◊

Exercise 5.4: The goal is to give tight bounds for $H_n^{(2)} := \sum_{i=1}^{n} \frac{1}{i^2}$ (cf. (a) in previous exercise).
(a) Let $S(n) = \sum_{i=2}^{n} \frac{1}{(i-1)(i+1)}$. Find the exact bound for $S(n)$.
(b) Let $G(n) = S(n) - H_n^{(2)} + 1$. Now $\gamma' = G(\infty)$ is a real constant,

$$\gamma' = \frac{1}{1 \cdot 3 \cdot 4} + \frac{1}{2 \cdot 4 \cdot 9} + \frac{1}{3 \cdot 5 \cdot 16} + \cdots + \frac{1}{(i-1) \cdot (i+1) \cdot i^2} + \cdots.$$

Show that $G(n) = \gamma' - \theta(n^{-3})$.
(c) Give an approximate expression for $H_n^{(2)}$ (involving γ') that is accurate to $\mathcal{O}(n^{-3})$. Note that γ' plays a role similar to Euler's constant γ for harmonic numbers.
(d) What can you say about γ', given that $H_\infty^{(2)} = \pi^2/6$? Use a calculator (and a suitable approximation for π) to compute γ' to 6 significant digits. ◊

Exercise 5.5: Solve the recurrence $T(n) = 5T(n-1) + n$. ◊

Exercise 5.6: Solve exactly (choose your own initial conditions):
(a) $T(n) = 1 + \frac{n+1}{n} T(n-1)$.
(b) $T(n) = 1 + \frac{n+2}{n} T(n-1)$. HINT: compare previous exercise (a). ◊

Exercise 5.7: Show that $\sum_{i=1}^{n} H_i = (n+1)H_n - n$. More generally,

$$\sum_{i=1}^{n} \binom{i}{m} H_i = \binom{n+1}{m+1} \left[H_{n+1} - \frac{1}{m+1} \right].$$

$\diamondsuit$

Exercise 5.8: Give a recurrence for S_n^k (see (16)) in terms of S_n^i, for $i < k$. Solve exactly for S_n^4. $\diamondsuit$

Exercise 5.9: Derive the formula for the "geometric series of order 2", $k = 2$ in (24). $\diamondsuit$

Exercise 5.10: (a) Use Stirling's approximation to give an estimate of the exponent E in the expression $2^E = \binom{2n}{n}$.
(b) (Feller) Show $\binom{2n}{n} = \sum_{k=0}^{n} \binom{n}{k}^2$. $\diamondsuit$

§6. Standard form and Summation Techniques

We try to reduce all recurrences to the following **standard form**:

$$t(n) = t(n-1) + f(n). \tag{25}$$

Let us assume that the recurrence is valid for integers $n \geq 1$. Thus

$$t(i) - t(i-1) = f(i), \quad (i = 1, \ldots, n).$$

Adding these n equations together, all but two terms on the left-hand side cancel, leaving us $t(n) - t(0) = \sum_{i=1}^{n} f(i)$. (We say the left-hand side is a "telescoping sum", and this trick is known as "telescopy".) Choosing the convenient initial condition $t(0) = 0$, we obtain

$$t(n) = \sum_{i=1}^{n} f(i). \tag{26}$$

If this open sum has the form of one of the basic sums in the previous section, we are done! For instance, in bubble sort, we obtain a standard form recurrence:

$$t(n) = t(n-1) + n.$$

Choosing the initial condition $t(0) = 0$, we obtain the exact solution $t(n) = \sum_{i=1}^{n} i = \binom{n+1}{2}$.

Let us consider what is to be done if the open sum (26) does not have one of the basic forms. Here are some simple techniques that are useful in the analysis of algorithms. Assume

$$f(i) > 0$$

in (26). Two situations appear very often:

Polynomial Type: The terms $f(i)$ **increase polynomially** in i. By this, we mean that the $f(i)$'s are increasing and

$$f(i) = \mathcal{O}(f(i/2)).$$

E.g.,

$$\sum_{i=1}^{n} i^3, \qquad \sum_{i=1}^{n} i \log i, \qquad \sum_{i=1}^{n} \log i \quad . \tag{27}$$

Exponential Type: The terms $f(i)$ **grow exponentially** in i. There are two possibilities: (a) $f(i)$ grows exponentially large:

$$(\exists C > 1)\left[f(i) \geq C \cdot f(i-1)\right].$$

(b) Or $f(i)$ grows exponentially small:

$$(\exists c < 1)\left[f(i) \leq c \cdot f(i-1)\right].$$

E.g.,

$$\sum_{i=1}^{n} 2^i, \qquad \sum_{i=1}^{n} 2^{-i}, \qquad \sum_{i=1}^{n} i^5 2^{2^i}, \qquad \sum_{i=1}^{n} i! \quad . \tag{28}$$

Summation Rules: Let $S_n = \sum_{i=1}^{n} f(n)$.

1. *If S_n is a polynomial type summation, replace* each term *by its "largest term" $f(n)$. Hence $S_n = \Theta(nf(n))$.* Example: For $k > 0$,

$$\sum_{i=1}^{n} i^k = \Theta(n^{k+1}), \qquad \sum_{i=1}^{n} i \log i = \Theta(n^2 \log n). \tag{29}$$

2. *If S_n is an exponential type summation, replace the* entire sum *by its largest term.* Since the largest term is $f(n)$ if the terms are growing exponentially large, and $f(1)$ if the terms are growing exponentially small, we get $S_n = \Theta(f(n))$ or $S_n = \Theta(f(1))$, respectively. Example: For constants $k > 0$ and $x \neq 1$,

$$\sum_{i=1}^{n} i^k x^i = \begin{cases} \Theta(1) & \text{if } x < 1, \\ \Theta(n^k x^n) & \text{if } x > 1. \end{cases} \tag{30}$$

Proof. For a polynomial type summation,

$$\frac{n}{2} f(n/2) \leq S_n \leq \sum_{i=1}^{n} \mathcal{O}_1(f(n/2)) = \mathcal{O}_1(nf(n/2)).$$

The result follows since we have $f(n/2) = \Theta(f(n))$. For an exponentially large type summation, there is some $C > 1$ such that

$$f(n) \leq S_n \leq f(n) + f(n-1) + f(n-2) + \cdots \leq f(n)\left[1 + \frac{1}{C} + \frac{1}{C^2} + \cdots\right] < \frac{C}{C-1} f(n).$$

Similarly for an exponentially small summation, there is an appropriate $c < 1$ such that

$$f(1) \leq S_n \leq f(1) + f(2) + f(3) + \cdots \leq f(1)\left[1 + c + c^2 + \cdots\right] < f(1)\frac{1}{c-1}.$$

Q.E.D.

Breaking Up a Sum into Small and Large Parts. A general technique is to break up a sum into two parts, one containing the "small terms" and the other containing the "big terms". Let us illustrate this by showing that

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = o(n).$$

It is sufficient to show that
$$S_n := H_n/n = \sum_{i=1}^{n} \frac{1}{i \cdot n}$$
goes to 0 as $n \to \infty$. Write $S_n = A_n + B_b$ where
$$A_n = \sum_{i=1}^{\lfloor \sqrt{n} \rfloor} \frac{1}{i \cdot n}.$$
Then we see that
$$A_n \le \sum_{i=1}^{\lfloor \sqrt{n} \rfloor} \frac{1}{n} \le \frac{1}{\sqrt{n}}.$$
Also,
$$B_n = \sum_{i=\lfloor \sqrt{n} \rfloor + 1}^{n} \frac{1}{i \cdot n} \le \sum_{i=1}^{n} \frac{1}{\sqrt{n} \cdot n} = \frac{1}{\sqrt{n}}.$$
Thus $S_n \le \frac{2}{\sqrt{n}} \to 0$ as $n \to \infty$.

Exercises

Exercise 6.1: (a) Verify that each of the examples in (27) and (28) are polynomial type or exponential type, as claimed. For each, state the bound according to our summation rules.
(b) Is the summation $\sum_{i=1}^{n} i^{\lg i}$ an exponential type or polynomial type? Give bounds for the summation. ♢

Exercise 6.2: (a) Use a direct estimate to show that $H_n = o(n^\alpha)$ for any $\alpha > 0$. Generalize the argument in the text (do not use calculus, properties of $\log x$ such as $x/\log x \to \infty$, etc.)
(b) Likewise, show by direct argument that $H_n \to \infty$ as $n \to \infty$. ♢

Exercise 6.3: Extend our summation rules to the case where $f(i)$ is "decreasing polynomially". ♢

§7. Domain transformation

So our goal for a general recurrence is to transform it into the standard form. You may think of change of domain as a "change of scale". Transforming the domain of a recurrence equation may sometimes bring it into standard form. Consider
$$T(N) = T(N/2) + N. \tag{31}$$
We define
$$t(n) := T(2^n), \quad N = 2^n.$$
This transforms the original N-domain into the n-domain. The new recurrence is now in standard form,
$$t(n) = t(n-1) + 2^n.$$
Choosing the boundary condition $t(0) = 1$, we get $t(n) = \sum_{i=0}^{n} 2^i$. This is a geometric series which we know how to sum, $t(n) = 2^{n+1} - 1$; hence, $T(N) = 2N - 1$.

Logarithmic transform. More generally, consider the recurrence

$$T(N) = T\left(\frac{N}{c} - d\right) + F(N), \quad c > 1, \tag{32}$$

and d is an arbitrary constant. It is instructive to begin with the case $d = 0$. Then it is easy to see that the "logarithmic transformation" of the argument N to the new argument $n := \log_c(N)$ converts this to the new recurrence

$$t(n) = t(n-1) + F(c^n)$$

where we define

$$t(n) := T(c^n) = T(N).$$

There is some possible confusion in such manipulations, so let us state the connection between t and T more formally. Let τ denote the **domain transformation function**,

$$\tau(N) = \log_c(N)$$

(so "n" is only a short-hand for "$\tau(N)$"). Then $t(\tau(N))$ is defined to be $T(N)$, valid for large enough N. In order for this to be well defined, we need τ to have an inverse for large enough N. Then we can write

$$t(n) := T(\tau^{-1}(N)).$$

We now return to the general case where d is an arbitrary constant. Note that if $d < 0$ then we must assume that N is sufficiently large (how large?) so that the recurrence (32) is meaningful (*i.e.*, $(N/c) - d < N$). The following transformation

$$n := \tau(N) = \log_c(N + \frac{cd}{c-1})$$

will reduce the recurrence to standard from. To see this, note that the "inverse transformation" is

$$\begin{aligned} N &:= c^n - \frac{cd}{c-1} \\ &= \tau^{-1}(n) \\ (N/c) - d &= c^{n-1} - \frac{cd}{c-1} \\ &= \tau^{-1}(n-1). \end{aligned}$$

Writing $t(n)$ for $T(\tau^{-1}(n))$ and $f(n)$ for $F(\tau^{-1}(n))$, we convert equation (32) to

$$\begin{aligned} t(n) &= t(n-1) + F\left(c^n - \frac{cd}{c-1}\right) \\ &= t(n-1) + f(n) \\ &= \sum_{i=1}^{n} f(i). \end{aligned}$$

To finally "solve" for $t(n)$ we need to know more about the function $F(N)$. For example, if $F(N)$ is a polynomially bounded function, then $f(n) = F(c^n + \frac{cd}{c-1})$ would be $\Theta(F(c^n))$. This is the usual justification for ignoring the additive term "d" in the equation (32).

Multiplicative transform. Notice that the logarithmic transform case does not quite capture the following closely related recurrence

$$T(N) = T(N-d) + F(N), d > 0. \tag{33}$$

It is easy to concoct the necessary domain transformation: replace N by $n = N/d$ and substituting

$$t(n) = T(dn)$$

will transform it to the standard form,

$$t(n) = t(n-1) + F(dn).$$

Again, to be formal, we can explicitly introduce the transform function $\tau(N) = N/d$, etc. This may be called the "multiplicative transform".

More generally, we consider $T(N) = T(r(N)) + f(N)$ where $r(N) < N$. We want a domain transform $\tau(N)$ so that $\tau(r(N)) = \tau(N) - 1$. For instance, if $r(N) = \sqrt{(N)}$, then $\tau(N) = \lg\lg(N)$ implies $\tau(\sqrt{(N)}) = \lg\lg N - 1 = \tau(N) - 1$.

Remarks. You may need to perform several of the above transformations to get the standard form. For instance, with

$$T(n) = T(\sqrt{n}) + N$$

you need to apply the logarithmic transformation twice to obtain the transformation $N = \lg\lg n$. If you then define $t(N) = T(n)$, you get $t(N) = t(N-1) + 2^{2^N}$. We note that the application of domain transformations is often confusing for students who have difficulty keeping the similar-looking symbols, 'n' versus 'N' and 't' versus 'T', straight. Of course, these symbols are mnemonically chosen. You can choose anything you want, but your reader may get even more confused.

Exercises

Exercise 7.1: Justify the simplification step (iv) in §1 (where we replace $\lceil n/2 \rceil$ by $n/2$). $\diamondsuit$

Exercise 7.2: Solve recurrence (32) in these cases:
(a) $F(N) = N^k$.
(b) $F(N) = \log N$. $\diamondsuit$

Exercise 7.3: Construct examples where you need to compose two or more of the above domain transformations. $\diamondsuit$

End Exercises

§8. Range transformation

A transformation of the range is sometimes called for. For instance, consider

$$T(n) = 2T(n-1) + n.$$

To put this into standard form, we could define

$$t(n) := \frac{T(n)}{2^n}$$

and get the standard form recurrence

$$t(n) = t(n-1) + \frac{n}{2^n}.$$

Telescoping gives us a series of the type in equation (20), which we know how to sum.

We have transformed the range of $T(n)$ by introducing a multiplicative factor 2^n: this factor is called the **summation factor**. The reader familiar with linear differential equations will see an analogy with "integrating factor". (In the same spirit, the previous trick of domain transformation is simply a "change of variable".)

In general, a range transformation converts a recurrence of the form

$$T(n) = c_n T(n-1) + F(n) \tag{34}$$

into standard form. Here c_n is a constant depending on n. Let us discover which summation factor will work. If $C(n)$ is the summation factor, we get

$$t(n) := \frac{T(n)}{C(n)},$$

and hence

$$\begin{aligned} t(n) &= \frac{T(n)}{C(n)} \\ &= \frac{c_n}{C(n)} T(n-1) + \frac{F(n)}{C(n)} \\ &= \frac{T(n-1)}{C(n-1)} + \frac{F(n)}{C(n)}, \qquad \text{(provided } C(n) = c_n C(n-1)\text{)} \\ &= t(n-1) + \frac{F(n)}{C(n)}. \end{aligned}$$

Thus we need $C(n) = c_n C(n-1)$ which expands into

$$C(n) = c_n c_{n-1} \cdots c_1.$$

Exercises

Exercise 8.1: Solve the recurrence (34) in the case where $c_n = 1/n$ and $F(n) = 1$. ♢

Exercise 8.2: (a) Reduce the following recurrence

$$T(n) = 4T(n/2) + \frac{n^2}{\lg n}$$

to standard form. Then solve it exactly when n is a power of 2. For general n, use our generalized Harmonic numbers H_x for real $x \geq 2$ (see §2). You may choose any suitable initial conditions, but please state it explicitly.
(b) Solve the variations

$$T(n) = 4T(n/2) + \frac{n^2}{\lg^2 n}$$

and

$$T(n) = 4T(n/2) + \frac{n^2}{\sqrt{\lg n}}.$$

♢

§9. Examples

There is a wide variety of recurrences. This section looks at some recurrences, some of which falling outside our transformation techniques.

§9.1. Recurrences with Max

A class of recurrences that arises frequently in computer science involves the max operation. Fredman has investigated the solution of a class of recurrences involving max.

Consider the following variant of QuickSort: each time after we partition the problem into two subproblems, we will solve the subproblem that has the smaller size first (if their sizes are equal, it does not matter which order is used). We want to analyze the depth of the recursion stack. If a problem of size n is split into two subproblems of sizes n_1, n_2 then $n_1 + n_2 = n - 1$. Without loss of generality, let $n_1 \le n_2$. So $0 \le n_1 \le \lfloor (n-1)/2 \rfloor$. If the stack contains problems of sizes $(n_1 \ge n_2 \ge \cdots \ge n_k \ge 1)$ where n_k is the problem size at the top of the stack, then we have

$$n_{i-1} \ge n_i + n_{i+1}.$$

Since $n_1 \le n$, this easily implies $n_{2i+1} \le n/2^i$ or $k \le 2 \lg n$. A tighter bound is $k \le \log_\phi n$ where $\phi = 1.618\ldots$ is the golden ratio. This is not tight either.

The depth of recursion satisfies

$$D(n) = \max_{n_1=0}^{\lfloor (n-1)/2 \rfloor} [\max\{1 + D(n_1), D(n_2)\}]$$

This recurrence involving max is actually easy to solve. Assuming $D(n) \le D(m)$ for all $n \le m$, and for any real x, $D(x) = D(\lfloor x \rfloor)$, it is easy to see that $D(n) = 1 + D(n/2)$. Using the fact that $D(1) = 0$, we obtain $D(n) \le \lg n$. [Note: $D(1) = 0$ means that all problems on the stack has size ≥ 2.

§9.2. The Master Theorem

We first look at a recurrence that does fall under our transformation techniques. If $a \ge 1, b > 1$ are constants, we consider the **master recurrence**

$$T(n) = aT(n/b) + f(n) \tag{35}$$

where $f(n)$ is some function. Evidently, this is the recurrence to solve if we manage to solve a problem of size n by breaking it up into a subproblems each of size n/b, and merging these a subsolutions in time $f(n)$. The recurrence was systematically studied by Bentley, Haken and Saxe [1]. Solving it requires a combination of domain and range transformation.

First apply a domain transformation by defining

$$t(k) := T(b^k) \quad \text{(for all } k\text{)}.$$

Hence

$$t(k) = a\, t(k-1) + f(b^k).$$

Next, transform the range by using the summation factor $1/a^k$. This defines the function $s(k)$:

$$s(k) := t(k)/a^k.$$

Now $s(k)$ satisfies a recurrence in standard form:

$$\begin{aligned} s(k) &= \frac{t(k)}{a^k} \\ &= \frac{t(k-1)}{a^{k-1}} + \frac{f(b^k)}{a^k} \\ &= s(k-1) + \frac{f(b^k)}{a^k} \end{aligned}$$

Telescoping, we get

$$s(k) = s(k) - s(0) = \sum_{i=1}^{k} \frac{f(b^i)}{a^i},$$

where we have chosen the boundary condition $s(0) = 0$. *Now, we cannot proceed any further without knowing the nature of the function f.*

The master theorem considers three possibilities for f. The easiest possibility is what we call is what we call the "watershed" case (CASE (0)). This is when $f(n) = \Theta(n^{\log_b a})$ (we may call $n^{\log_b a}$ the "watershed function"). The other two possibilities are where f grows "polynomially slower" (CASE (-1)) or "polynomially faster" (CASE $(+1)$) than the watershed case.

CASE (0) This is when $f(n)$ satisfies

$$f(n) = \Theta(n^{\log_b a}). \tag{36}$$

Then $f(b^i) = \Theta(a^i)$ and $s(k) = \sum_{i=1}^{k} f(b^i)/a^i = \Theta(k)$.

CASE (-1) This is when $f(n)$ **grows polynomially slower** than the watershed function:

$$f(n) = \mathcal{O}(n^{-\epsilon + \log_b a}), \tag{37}$$

for some $\epsilon > 0$. Then $f(b^i) = \mathcal{O}(b^{i(\log_b a - \epsilon)})$. Let $f(b^i) = \mathcal{O}_1(a^i b^{-i\epsilon})$ (using the subscripting notation for $\mathcal{O}$). So $s(k) = \sum_{i=1}^{k} f(b^i)/a^i = \sum \mathcal{O}_1(b^{-i\epsilon}) = \mathcal{O}_2(1)$, since $b > 1$ implies $b^{-\epsilon} < 1$. Hence $s(k) = \Theta(1)$.

CASE $(+1)$ This is when $f(n)$ satisfies the **regularity condition**

$$af(n/b) \leq cf(n) \tag{38}$$

for some $c < 1$. Expanding this,

$$\begin{aligned} f(n) &\geq \frac{a}{c} f\left(\frac{n}{b}\right) \\ &\geq \left(\frac{a}{c}\right)^{\log_b n} f(1) \\ &= \Omega(n^{\epsilon + \log_b a}), \end{aligned}$$

where $\epsilon = -\log_b c > 0$. Thus the regularity condition implies that $f(n)$ **grows polynomially faster** than the watershed function,

$$f(n) = \Omega(n^{\epsilon + \log_b a}). \tag{39}$$

It follows from (38) that $f(b^{k-i}) \le (c/a)^i f(b^k)$. So

$$\begin{aligned} s(k) &= \sum_{i=1}^{k} f(b^i)/a^i \\ &= \sum_{i=0}^{k-1} f(b^{k-i})/a^{k-i} \\ &\le \sum_{i=0}^{k-1} (c/a)^i f(b^k)/a^{k-i} \\ &= f(b^k)/a^k \left(\sum_{i=0}^{k-1} c^{k-i} \right) \\ &= \mathcal{O}\left(\frac{f(b^k)}{a^k} \right), \end{aligned}$$

since $c < 1$. But clearly, $s(k) \ge f(b^k)/a^k$. Hence we have $s(k) = \Theta(f(b^k)/a^k)$.

Summarizing,

$$s(k) = \begin{cases} \Theta(1), & \text{CASE } (-1) \\ \Theta(k), & \text{CASE } (0) \\ \Theta(f(b^k)/a^k), & \text{CASE } (+1). \end{cases}$$

Back substituting,

$$t(k) = a^k s(k) = \begin{cases} \Theta(a^k), & \text{CASE } (-1) \\ \Theta(a^k k), & \text{CASE } (0) \\ \Theta(f(b^k)), & \text{CASE } (+1). \end{cases}$$

Since $T(n) = t(\log_b n)$, we conclude:

THEOREM 3 (MASTER THEOREM) *The master recurrence (35) has solution:*

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & \text{if } f(n) = \mathcal{O}(n^{-\epsilon+\log_b a}), \text{ for some } \epsilon > 0, \\ \Theta(n^{\log_b a} \log n), & \text{if } f(n) = \Theta(n^{\log_b a}), \\ \Theta(f(n)), & \text{if } af(n/b) \le cf(n) \text{ for some } c < 1. \end{cases}$$

In applications of the Master Theorem for case $(+)$, we often first to verify equation (39) mentally, before checking the stronger regularity condition (38). The Master Theorem is powerful but unfortunately, there are gaps between its 3 cases. For instance, $f(n) = n^{\log_b a} \log n$ grows faster than the watershed function, but not polynomially faster. Thus the Master Theorem is inapplicable for this $f(n)$. Yet it is just as easy to solve this case using the transformation techniques (see Exercise).

Note that the values a, b in the theorem are constants. Thus, attempting to apply this theorem to the recurrence

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

(with $a = 2^n$ and $b = 2$) leads to the false conclusion that $T(n) = \Theta(n^n \log n)$. See exercise. For a more general solution to the master recurrence, see [10].

Graphic Interpretation of the Master Recurrence. We imagine a "recursion tree" with branching factor of a at each node, and every leaf of the tree is at level $\log_b a$. Of course, this "tree" is not realizable unless a and $\log_b a$ are integers! We further associate a "size" of n/b^i and "cost" of $f(n/b^i)$ to each node at level i (root is at level $i = 0$). Then $T(n)$ is just the sum of the costs at all the nodes. The Master Theorem says this: In case (0), the total cost associated with nodes at any level is $\Theta(n^{\log_b a})$ and there are $\log_b n$ levels giving an overall cost of $\Theta(n^{\log_b a} \log n)$. In case (+1), the cost associated with the root is $\Theta(T(n))$. In case (−1), the total cost associated with the leaves is $\Theta(T(n))$.

EXERCISES

Exercise 9.1: State the solution, up to Θ-order of the following recurrences:

$$\begin{aligned} T(n) &= 10T(n/10) + \log^{10} n. \\ T(n) &= 100T(n/10) + n^{10}. \\ T(n) &= 10T(n/100) + (\log n)^{\log\log n}. \\ T(n) &= 16T(n/4) + 4^{\lg n}. \end{aligned}$$

$\diamondsuit$

Exercise 9.2: Solve the following using the Master's theorem whenever possible. If the Master's theorem is inapplicable, say so (or, you can solve it by other means).

$$\begin{aligned} T(n) &= 3T(n/25) + \log^3 n. \\ T(n) &= 25T(n/3) + (n/\log n)^3. \\ T(n) &= T(\sqrt{n}) + n. \end{aligned}$$

HINT: in the third problem, the Master theorem is applicable after a simple transformation. $\diamondsuit$

Exercise 9.3: Solve the master recurrence when $f(n) = n^{\log_b a} \log^k n$, for any $k \geq 1$. NOTE: the Master Theorem is not applicable here, but the method of its proof is applicable. $\diamondsuit$

Exercise 9.4: Re-prove the master theorem, but now apply the range transformation to the master recurrence before applying the domain transformation. $\diamondsuit$

Exercise 9.5: Show that the master theorem applies to the following variation of the master recurrence:

$$T(n) = a \cdot T(\frac{n+c}{b}) + f(n)$$

where $a > 0, b > 1$ and c is arbitrary. $\diamondsuit$

Exercise 9.6:
(a) Solve $T(n) = 2^n T(n/2) + n^n$ by direct expansion.
(b) Try to generalize the Master theorem to handle some cases of $T(n) = a_n T(n/b_n) + f(n)$ where a_n, b_n are both functions of n. $\diamondsuit$

END EXERCISES

§9.3. Generalized Master Theorem

Let us introduce what might be called the **multiterm master recurrence**:

$$T(n) = f(n) + \sum_{i=1}^{k} a_i T\left(\frac{n}{b_i}\right) \tag{40}$$

where $k \geq 1$, $a_i > 0$ (for all $i = 1, \ldots, k$) and $b_1 > b_2 > \cdots > b_k > 1$. The critical constant here is α such that

$$\sum_{i=1}^{k} \frac{a_i}{b_i^\alpha} = 1. \tag{41}$$

It is clear that α exists since the above sum tends to 0 (resp., ∞) as $\alpha \to \infty$ (resp., $\alpha \to -\infty$).

Theorem 4 (Multiterm Master Theorem)

$$T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \ f(n) = \mathcal{O}(n^{\alpha-\varepsilon}), \text{for some } \varepsilon > 0, \\ \Theta(n^\alpha \log n) & \text{if } \ f(n) = \Theta(n^\alpha) \\ \Omega(f(n) & \text{if } \ \sum_{i=1}^{k} a_i f(n/b_i) \leq cf(n), \text{for some } c < 1. \end{cases}$$

This can be shown by real induction.

Example: in linear time algorithms for medians (see Lecture XXX), we encounter recurrence of the form

$$T(n) \leq T(7n/10) + T(n/5) + \mathcal{O}(n).$$

By our multiterm master theorem, this has solution $T(n) = \Theta(n)$.

Exercises

Exercise 9.7: The following recurrence arises in the analysis of the running time of the "conjugation tree" in computational geometry:

$$T(n) = T(n/2) + T(n/4) + \lg^7 n.$$

Solve for $T(n)$. ♢

End Exercises

§10. Orders of Growth

The reader should first review the basic properties of the exponential and logarithm functions in the appendix.

Learning to judge the growth rates of complexity functions is a fundamental skill in algorithmics. This section is a practical one, designed to help students develop this skill.

Most complexity functions in practice are the so-called **logarithmico-exponential functions** (for short, L-functions): such functions $f(x)$ are real and defined for all $x \ge x_0$ for some x_0 depending of f. An L-function is either the identity function x or a constant $c \in \mathbb{R}$, or else obtained as a finite composition with the functions

$$A(x), \qquad \ln(x), \qquad e^x$$

where $A(x)$ denotes a real branch of an algebraical function. For instance, $A(x) = \sqrt{x}$ is the function that picks the real square-root of x. The reader may have noticed that all the common complexity functions are totally ordered in the sense that for any f, g, either $f \preceq g$ or $g \preceq f$. A theorem[4] of Hardy [4] confirms this: *if f and g are L-functions then $f \le g$ (ev.) or $g \le f$ (ev.).* In particular, each L-function f is eventually non-negative, $0 \le f$ (ev.), or or non-positive, $f \le 0$ (ev.).

The following are the common categories of functions you will encounter:

CATEGORY	SYMBOL	EXAMPLES
vanishing term	$o(1)$	$\frac{1}{n}$, 2^{-n}
constants	$\Theta(1)$	1, $2 - \frac{1}{n}$
polylogs	$\log^k n$ (for any $k > 0$)	H_n, $\log^2 n$
polynomials	n^k (for any $k > 0$)	n^3, $\sqrt{n}$
non-polynomials	$n^{\Omega(1)}$	$n!$, 2^n, $n^{\log\log n}$

Note that $n!$ and H_n are not L-functions, but they can be closely approximated by L-functions. The last category forms a grab-bag of anything growing faster than a polynomial. These 6 categories form a hierarchy of increasingly larger Θ-order.

Rules for comparing functions. We are interested in comparing functions up to their Θ-order. We list some simple rules. Most comparisons of interest to us can be reduced to repeated applications of these rules:

Sum: In a direct comparison involving a sum $f(n) + g(n)$, ignore the smaller term in this sum.
E.g., given $n^2 + n\log n + 5$, you should ignore the "$n \log n + 5$" term. However, beware that if the sum appears in an exponent, the neglected part may turn out be decisive when the dominant terms are identical.

Product: If $0 \preceq f \preceq f'$ and $0 \preceq g \preceq g'$ then $fg \preceq f'g'$. (If, in addition, $f \prec f'$ then we have $fg \prec f'g'$.)
E.g., this rule implies $n^b \prec n^c$ when $b < c$ (since $1 \prec n^{c-b}$, by the logarithm rule next).

Logarithm: $1 \prec \log^{(k+1)} n \prec (\log^{(k)} n)^c$ for any integer $k \ge 0$ and real $c > 0$. Here $\log^{(k)} n$ refers to the k-fold application of the logarithm function and $\log^{(0)} n = n$.

Exponentiation: If $1 \le f \le g$ (ev.) then $d^f \preceq d^g$ for any constant $d > 1$. If $1 \le f \le cg$ (ev.) for some $c < 1$ then $d^f \prec d^g$.

Example. Suppose we want to compare $n^{\log n}$ versus $(\log n)^n$. By the rule of exponentiation, $n^{\log n} \prec (\log n)^n$ follows if we take logs and show that $\log^2 n \le 0.5 n \log\log n$ (ev.). In fact, we show the stronger $\log^2 n \prec n \log\log n$. Taking logs again, and by the rule of sum, it is sufficient to show $2\log\log n \prec \log n$.

[4] In the literature on L-functions, the notation "$f \preceq g$" actually means $f \le g$ (ev.). There is a deep theory involving such functions, with connection to Nevanlinna theory.

Taking logs again, and by the rule of sum again, it is suffices to show $\log^{(3)} n \prec \log^{(2)} n$. But the latter follows from the rule of logarithms.

Exercises

Exercise 10.1: (i) Simplify the following expressions: (a) $n^{1/\lg n}$, (b) $2^{2^{\lg\lg n - 1}}$, (c) $\sum_{i=0}^{k-1} 2^i$, (d) $2^{(\lg n)^2}$, (e) $4^{\lg n}$, (f) $(\sqrt{2})^{\lg n}$.
(ii) Re-do the above, replacing each occurrence of "2" (explicit or otherwise) in the previous expressions by some constant $c > 2$. ♢

Exercise 10.2: Order the following functions (be sure to parse these nested exponentiations correctly): (a) $n^{(\lg n)^{\lg n}}$, (b) $(\lg n)^{n^{\lg n}}$, (c) $(\lg n)^{(\lg n)^n}$, (d) $(n/\lg n)^{n^{n/(\lg n)}}$. (e) $n^{n^{(\lg n)/n}}$. ♢

Exercise 10.3: Order the following functions in non-increasing order of growth. Between consecutive pairs of functions, insert the appropriate ordering relationship: $\preceq$, $\asymp$, $\le$ (ev.), $=$.

	a	b	c	d	e	f
1.	$\lg\lg n$	$(\lg n)^{\lg n}$	2^n	$2^{\lg n}$	$2^{\lg^* n}$	$2^{2^{n+1}}$
2.	$(1/3)^n$	$n2^n$	$n^{\lg\lg n}$	e^n	$n^{1/\lg n}$	$(\lg n)!$
3.	$2^{\sqrt{2\lg n}}$	$(3/2)^n$	2	$\lg(n!)$	n	$\sqrt{\lg n}$
4.	$2^{(\lg n)^2}$	2^{2^n}	n^2	$n\lg n$	$(n+1)!$	$4^{\lg n}$
5.	$\lg(\lg^* n)$	$\lg^2 n$	$(1+\frac{1}{n})^n$	$n^{\lg n}$	$n!$	$2^{(\lg n)/n}$
6.	$(\sqrt{2})^{\lg n}$	$\lg^* n$	$(n/\lg n)^2$	$\sqrt{n})$	$\lg^*(\lg n)$	$1/n$

NOTE: to help in the organization of this large list of functions, we ask that you first order each row. Then the rows are merged in pairs. Finally, perform a 3-way merge of the 3 lists. Show the intermediate lists of your computation (it allows us to visually verify your work). ♢

Exercise 10.4: (Purdom-Brown)
(a) Show that $\sum_{i=1}^n i! = n![1 + \mathcal{O}(1/n)]$. NOTE: The summation rule gives only a Θ-order so this is more precise.
(b) $\sum_{i=1}^n 2^i \ln i = 2^{n+1}[\ln n - (1/n) + \mathcal{O}(n^{-2})]$. HINT: use $\ln i = \ln n - (i/n) + \mathcal{O}(i^2/n^2)$ for $i = 1, \ldots, n$. ♢

Exercise 10.5: (Knuth) What is the asymptotic behaviour of $n^{1/n}$? of $n(n^{1/n} - 1)$?
HINT: take logs. Alternatively, expand $\prod_{i=1}^n e^{1/(in)}$. ♢

Exercise 10.6: Estimate the growth behavior of the solution to this recurrence: $T(n) = T(n/2)^2 + 1$. ♢

References

[1] J. L. Bentley, D. Haken, and J. B. Saxe. A general method for solving divide-and-conquer recurrences. *ACM SIGACT News*, 12(3):36–44, 1980.

[2] W. Feller. *An introduction to Probability Theory and its Applications*. Wiley, New York, 2nd edition edition, 1957. (Volumes 1 and 2).

[3] D. H. Greene and D. E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhäuser, 2nd edition, 1982.

[4] G. H. Hardy. *Orders of Infinity*. Cambridge Tracts in Mathematics and Mathematical Physics, No. 12. Reprinted by Hafner Pub. Co., New York. Cambridge University Press, 1910.

[5] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, Boston, 2nd edition edition, 1975.

[6] G. S. Lueker. Some techniques for solving recurrences. *Computing Surveys*, 12(4), 1980.

[7] B. Mishra and A. Siegel. (Class Lecture Notes) Analysis of Algorithms, January 28, 1991.

[8] D. S. Mitrinović. *Analytic Inequalities*. Springer-Verlag, New York, 1970.

[9] J. Paul Walton Purdom and C. A. Brown. *The Analysis of Algorithms*. Holt, Rinehart and Winston, New York, 1985.

[10] X. Wang and Q. Fu. A frame for general divide-and-conquer recurrences. *Info. Processing Letters*, 59:45–51, 1996.