> *"...O Zarathustra, who you are and must become" behold you are the teacher of the eternal recurrence – that is your destiny! That you as the first must teach this doctrine – how could this great destiny not be your greatest danger and sickness too?*
>
> — Friedrich Nietzsche, *Also sprach Zarathustra* (1885)
> [translated by Walter Kaufmann]

Solving Recurrences

1 Introduction

A ***recurrence*** is a recursive description of a function, usually of the form $F : \mathbb{N} \to \mathbb{R}$, or a description of such a function in terms of itself. Like all recursive structures, a recurrence consists of one or more *base cases* and one or more *recursive cases*. Each of these cases is an equation or inequality, with some function value $f(n)$ on the left side. The base cases give explicit values for a (typically finite, typically small) subset of the possible values of n. The recursive cases relate the function value $f(n)$ to function value $f(k)$ for one or more integers $k < n$; typically, each recursive case applies to an infinite number of possible values of n.

For example, the following recurrence (written in two different but standard ways) describes the identity function $f(n) = n$:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ f(n-1)+1 & \text{otherwise} \end{cases} \qquad \begin{aligned} &f(0) = 0 \\ &f(n) = f(n-1)+1 \text{ for all } n > 0 \end{aligned}$$

In both presentations, the first line is the only base case, and the second line is the only recursive case. The same function can satisfy *many* different recurrences; for example, both of the following recurrences also describe the identity function:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) & \text{otherwise} \end{cases} \qquad f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 2 \cdot f(n/2) & \text{if } n \text{ is even and } n > 0 \\ f(n-1)+1 & \text{if } n \text{ is odd} \end{cases}$$

We say that a particular function ***satisfies*** a recurrence, or is the ***solution*** to a recurrence, if each of the statements in the recurrence is true. Most recurrences—at least, those that we will encounter in this class—have a solution; moreover, if every case of the recurrence is an equation, that solution is unique. Specifically, if we transform the recursive formula into a recursive *algorithm*, the solution to the recurrence is the function computed by that algorithm!

Recurrences arise naturally in the analysis of algorithms, especially recursive algorithms. In many cases, we can express the running time of an algorithm as a recurrence, where the recursive cases of the recurrence correspond exactly to the recursive cases of the algorithm. Recurrences are also useful tools for solving counting problems—How many objects of a particular kind exist?

By itself, a recurrence is not a satisfying description of the running time of an algorithm or a bound on the number of widgets. Instead, we need a ***closed-form*** solution to the recurrence; this is a *non-recursive* description of a function that satisfies the recurrence. For recurrence *equations*, we sometimes prefer an *exact* closed-form solution, but such a solution may not exist, or may be too complex to be useful. Thus, for most recurrences, especially those arising in algorithm analysis, we can be satisfied with an *asymptotic* solution of the form $\Theta(f(n))$, for some explicit (non-recursive) function $g(n)$.

For recursive *inequalities*, we prefer a ***tight*** solution; this is a function that would still satisfy the recurrence if all the inequalities were replaced with the corresponding equations. Again, exactly tight solutions may not exist, or may be too complex to be useful, so we may have to settle for a looser solution and/or an asymptotic solution of the form $O(g(n))$ or $\Omega(g(n))$.

2 The Ultimate Method: Guess and Confirm

Ultimately, there is only one fail-safe method to solve *any* recurrence:

Guess the answer, and then prove it correct by induction.

Later sections of these notes describe techniques to generate guesses that are guaranteed to be correct, provided you use them correctly. But if you're faced with a recurrence that doesn't seem to fit any of these methods, or if you've forgotten how those techniques work, don't despair! If you guess a closed-form solution and then try to verify your guess inductively, usually either the proof will succeed, in which case you're done, or the proof will fail, in which case *the failure will help you refine your guess*. Where you get your initial guess is utterly irrelevant[1]—from a classmate, from a textbook, on the web, from the answer to a different problem, scrawled on a bathroom wall in Siebel, included in a care package from your mom, dictated by the machine elves, whatever. If you can prove that the answer is correct, then it's correct!

2.1 Tower of Hanoi

The classical Tower of Hanoi problem gives us the recurrence $\boldsymbol{T(n) = 2T(n-1) + 1}$ with base case $\boldsymbol{T(0) = 0}$. Just looking at the recurrence we can guess that $T(n)$ is something like 2^n. If we write out the first few values of $T(n)$, we discover that they are each one less than a power of two.

$$T(0) = 0, \quad T(1) = 1, \quad T(2) = 3, \quad T(3) = 7, \quad T(4) = 15, \quad T(5) = 31, \quad T(6) = 63, \quad \ldots,$$

It looks like $\boldsymbol{T(n) = 2^n - 1}$ might be the right answer. Let's check.

$$\begin{aligned} T(0) &= 0 = 2^0 - 1 \quad \checkmark \\ T(n) &= 2T(n-1) + 1 \\ &= 2(2^{n-1} - 1) + 1 && \text{[induction hypothesis]} \\ &= 2^n - 1 \quad \checkmark && \text{[algebra]} \end{aligned}$$

We were right! Hooray, we're done!

Another way we can guess the solution is by ***unrolling*** the recurrence, by substituting it into itself:

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2\,(2T(n-2) + 1) + 1 \\ &= 4T(n-2) + 3 \\ &= 4\,(2T(n-3) + 1) + 3 \\ &= 8T(n-2) + 7 \\ &= \cdots \end{aligned}$$

[1] . . . except of course during exams, where you aren't supposed to use *any* outside sources

It looks like unrolling the initial Hanoi recurrence k times, for any non-negative integer k, will give us the new recurrence $T(n) = 2^k T(n-k) + (2^k - 1)$. Let's prove this by induction:

$$\begin{aligned}
T(n) &= 2T(n-1) + 1 \quad \checkmark && [k = 0, \text{ by definition}] \\
T(n) &= 2^{k-1} T(n-(k-1)) + (2^{k-1} - 1) && [\text{inductive hypothesis}] \\
&= 2^{k-1}\big(2T(n-k) + 1\big) + (2^{k-1} - 1) && [\text{initial recurrence for } T(n-(k-1))] \\
&= 2^k T(n-k) + (2^k - 1) \quad \checkmark && [\text{algebra}]
\end{aligned}$$

Our guess was correct! In particular, unrolling the recurrence n times give us the recurrence $T(n) = 2^n T(0) + (2^n - 1)$. Plugging in the base case $T(0) = 0$ give us the closed-form solution $T(n) = 2^n - 1$.

2.2 Fibonacci numbers

Let's try a less trivial example: the Fibonacci numbers $\boldsymbol{F_n = F_{n-1} + F_{n-2}}$ with base cases $\boldsymbol{F_0 = 0}$ and $\boldsymbol{F_1 = 1}$. There is no obvious pattern in the first several values (aside from the recurrence itself), but we can reasonably guess that F_n is exponential in n. Let's try to prove inductively that $F_n \le \alpha \cdot c^n$ for some constants $a > 0$ and $c > 1$ and see how far we get.

$$\begin{aligned}
F_n &= F_{n-1} + F_{n-2} \\
&\le \alpha \cdot c^{n-1} + \alpha \cdot c^{n-2} && [\text{“induction hypothesis”}] \\
&\le \alpha \cdot c^n \ \textbf{???}
\end{aligned}$$

The last inequality is satisfied if $c^n \ge c^{n-1} + c^{n-2}$, or more simply, if $c^2 - c - 1 \ge 0$. The smallest value of c that works is $\phi = (1 + \sqrt{5})/2 \approx 1.618034$; the other root of the quadratic equation has smaller absolute value, so we can ignore it.

So we have *most* of an inductive proof that $\boldsymbol{F_n \le \alpha \cdot \phi^n}$ for *some* constant α. All that we're missing are the base cases, which (we can easily guess) must determine the value of the coefficient a. We quickly compute

$$\frac{F_0}{\phi^0} = \frac{0}{1} = 0 \quad \text{and} \quad \frac{F_1}{\phi^1} = \frac{1}{\phi} \approx 0.618034 > 0,$$

so the base cases of our induction proof are correct as long as $\alpha \ge 1/\phi$. It follows that $\boldsymbol{F_n \le \phi^{n-1}}$ for all $n \ge 0$.

What about a matching lower bound? Essentially the same inductive proof implies that $F_n \ge \beta \cdot \phi^n$ for some constant β, but the only value of β that works for *all* n is the trivial $\beta = 0$! We could try to find some lower-order term that makes the base case non-trivial, but an easier approach is to recall that asymptotic $\Omega(\,)$ bounds only have to work for *sufficiently large* n. So let's ignore the trivial base case $F_0 = 0$ and assume that $F_2 = 1$ is a base case instead. Some more easy calculation gives us

$$\frac{F_2}{\phi^2} = \frac{1}{\phi^2} \approx 0.381966 < \frac{1}{\phi}.$$

Thus, the new base cases of our induction proof are correct as long as $\beta \le 1/\phi^2$, which implies that $\boldsymbol{F_n \ge \phi^{n-2}}$ for all $n \ge 1$.

Putting the upper and lower bounds together, we obtain the tight asymptotic bound $\boldsymbol{F_n = \Theta(\phi^n)}$. It *is* possible to get a more exact solution by speculatively refining and conforming our current bounds, but it's not easy. Fortunately, if we really need it, we can get an exact solution using the *annihilator* method, which we'll see later in these notes.

2.3 Mergesort

Mergesort is a classical recursive divide-and-conquer algorithm for sorting an array. The algorithm splits the array in half, recursively sorts the two halves, and then merges the two sorted subarrays into the final sorted array.

```
MERGESORT(A[1..n]):
  if (n > 1)
      m ← ⌊n/2⌋
      MERGESORT(A[1..m])
      MERGESORT(A[m+1..n])
      MERGE(A[1..n], m)
```

```
MERGE(A[1..n], m):
  i ← 1;  j ← m+1
  for k ← 1 to n
      if j > n
          B[k] ← A[i];  i ← i+1
      else if i > m
          B[k] ← A[j];  j ← j+1
      else if A[i] < A[j]
          B[k] ← A[i];  i ← i+1
      else
          B[k] ← A[j];  j ← j+1
  for k ← 1 to n
      A[k] ← B[k]
```

Let $T(n)$ denote the worst-case running time of MERGESORT when the input array has size n. The MERGE subroutine clearly runs in $\Theta(n)$ time, so the function $T(n)$ satisfies the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T\left(\lceil n/2 \rceil\right) + T\left(\lfloor n/2 \rfloor\right) + \Theta(n) & \text{otherwise.} \end{cases}$$

For now, let's consider the special case where n is a power of 2; this assumption allows us to take the floors and ceilings out of the recurrence. (We'll see how to deal with the floors and ceilings later; the short version is that they don't matter.)

Because the recurrence itself is given only asymptotically—in terms of $\Theta(\,)$ expressions—we can't hope for anything but an asymptotic solution. So we can safely simplify the recurrence further by removing the Θ's; any asymptotic solution to the simplified recurrence will also satisfy the original recurrence. (This simplification is actually important for another reason; if we kept the asymptotic expressions, we might be tempted to simplify them inappropriately.)

Our simplified recurrence now looks like this:

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2T(n/2) + n & \text{otherwise.} \end{cases}$$

To guess at a solution, let's try unrolling the recurrence.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2\big(2T(n/4) + n/2\big) + n \\ &= 4T(n/4) + 2n \\ &= 8T(n/8) + 3n = \cdots \end{aligned}$$

It looks like $T(n)$ satisfies the recurrence $T(n) = 2^k T(n/2^k) + kn$ for *any* positive integer k. Let's verify this by induction.

$$T(n) = 2T(n/2) + n = 2^1 T(n/2^1) + 1 \cdot n \quad \checkmark \qquad [k = 1, \text{given recurrence}]$$

$$\begin{aligned} T(n) &= 2^{k-1} T(n/2^{k-1}) + (k-1)n && [\text{inductive hypothesis}] \\ &= 2^{k-1}\left(2T(n/2^k) + n/2^{k-1}\right) + (k-1)n && [\text{substitution}] \\ &= 2^k T(n/2^k) + kn \quad \checkmark && [\text{algebra}] \end{aligned}$$

Our guess was right! The recurrence becomes trivial when $n/2^k = 1$, or equivalently, when $k = \log_2 n$:

$$T(n) = nT(1) + n\log_2 n = n\log_2 n + n.$$

Finally, we have to put back the Θ's we stripped off; our final closed-form solution is $\boldsymbol{T(n) = \Theta(n \log n)}$.

2.4 An uglier divide-and-conquer example

Consider the divide-and-conquer recurrence $\boldsymbol{T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n}$. This doesn't fit into the form required by the Master Theorem (which we'll see below), but it still sort of resembles the Mergesort recurrence—the total size of the subproblems at the first level of recursion is n—so let's *guess* that $T(n) = O(n \log n)$, and then try to prove that our guess is correct. (We could also attack this recurrence by unrolling, but let's see how far just guessing will take us.)

Let's start by trying to prove an upper bound $T(n) \le a\, n \lg n$ for all sufficiently large n and some constant a to be determined later:

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\le \sqrt{n} \cdot a\sqrt{n} \lg \sqrt{n} + n && [\text{induction hypothesis}] \\ &= (a/2) n \lg n + n && [\text{algebra}] \\ &\le an \lg n \checkmark && [\text{algebra}] \end{aligned}$$

The last inequality assumes only that $1 \le (a/2)\log n$,or equivalently, that $n \ge 2^{2/a}$. In other words, the induction proof is correct if n is sufficiently large. So we were right!

But before you break out the champagne, what about the multiplicative constant a? The proof worked for *any* constant a, no matter how small. This strongly suggests that our upper bound $T(n) = O(n \log n)$ is not tight. Indeed, if we try to prove a matching lower bound $T(n) \ge b\, n \log n$ for sufficiently large n, we run into trouble.

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\ge \sqrt{n} \cdot b\sqrt{n} \log \sqrt{n} + n && [\text{induction hypothesis}] \\ &= (b/2) n \log n + n \\ &\not\ge bn \log n \end{aligned}$$

The last inequality would be correct only if $1 > (b/2)\log n$, but that inequality is false for large values of n, no matter which constant b we choose.

Okay, so $\Theta(n \log n)$ is too big. How about $\Theta(n)$? The lower bound is easy to prove directly:

$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n \ge n \checkmark$$

But an inductive proof of the upper bound fails.

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\le \sqrt{n} \cdot a\sqrt{n} + n && [\text{induction hypothesis}] \\ &= (a+1)n && [\text{algebra}] \\ &\not\le an \end{aligned}$$

Hmmm. So what's bigger than n and smaller than $n \lg n$? How about $n\sqrt{\lg n}$?

$$\begin{aligned} T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n &\le \sqrt{n} \cdot a\sqrt{n}\sqrt{\lg \sqrt{n}} + n && \text{[induction hypothesis]} \\ &= (a/\sqrt{2})\, n\sqrt{\lg n} + n && \text{[algebra]} \\ &\le a\, n\sqrt{\lg n} \quad \text{for large enough } n \checkmark \end{aligned}$$

Okay, the upper bound checks out; how about the lower bound?

$$\begin{aligned} T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n &\ge \sqrt{n} \cdot b\sqrt{n}\sqrt{\lg \sqrt{n}} + n && \text{[induction hypothesis]} \\ &= (b/\sqrt{2})\, n\sqrt{\lg n} + n && \text{[algebra]} \\ &\not\ge b\, n\sqrt{\lg n} \end{aligned}$$

No, the last step doesn't work. So $\Theta(n\sqrt{\lg n})$ doesn't work.

Okay... what else is between n and $n \lg n$? How about $n \lg\lg n$?

$$\begin{aligned} T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n &\le \sqrt{n} \cdot a\sqrt{n} \lg\lg \sqrt{n} + n && \text{[induction hypothesis]} \\ &= a\, n \lg\lg n - a\, n + n && \text{[algebra]} \\ &\le a\, n \lg\lg n \qquad \textbf{if } \boldsymbol{a \ge 1} \checkmark \end{aligned}$$

Hey look at that! For once, our upper bound proof requires a constraint on the hidden constant a. This is an good indication that we've found the right answer. Let's try the lower bound:

$$\begin{aligned} T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n &\ge \sqrt{n} \cdot b\sqrt{n} \lg\lg \sqrt{n} + n && \text{[induction hypothesis]} \\ &= b\, n \lg\lg n - b\, n + n && \text{[algebra]} \\ &\ge b\, n \lg\lg n \qquad \textbf{if } \boldsymbol{b \le 1} \checkmark \end{aligned}$$

Hey, it worked! We have most of an inductive proof that $T(n) \le an \lg\lg n$ for any $a \ge 1$ and most of an inductive proof that $T(n) \ge bn \lg\lg n$ for any $b \le 1$. Technically, we're still missing the base cases in both proofs, but we can be fairly confident at this point that $\boldsymbol{T(n) = \Theta(n \log\log n)}$.

3 Divide and Conquer Recurrences (Recursion Trees)

Many divide and conquer algorithms give us running-time recurrences of the form

$$\boxed{T(n) = a\, T(n/b) + f(n)} \tag{1}$$

where a and b are constants and $f(n)$ is some other function. There is a simple and general technique for solving many recurrences in this and similar forms, using a ***recursion tree***. The root of the recursion tree is a box containing the value $f(n)$; the root has a children, each of which is the root of a (recursively defined) recursion tree for the function $T(n/b)$.

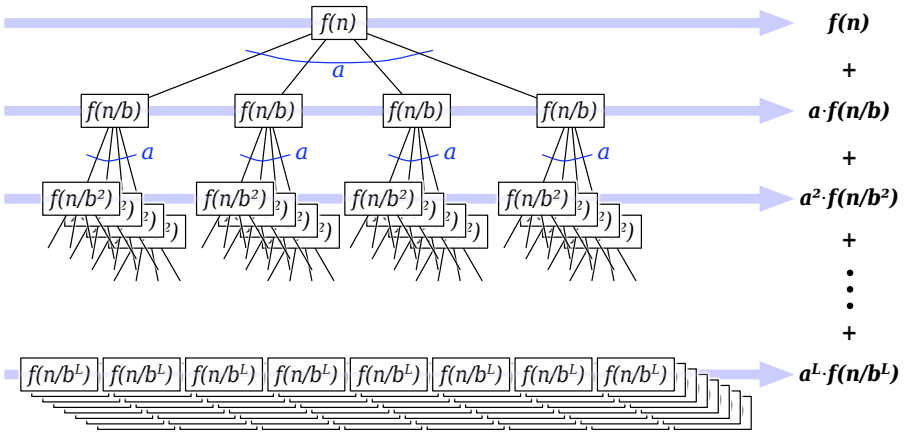
Equivalently, a recursion tree is a complete a-ary tree where each node at depth i contains the value $f(n/b^i)$. The recursion stops when we get to the base case(s) of the recurrence. Because we're only looking for asymptotic bounds, the exact base case doesn't matter; we can safely assume that $T(1) = \Theta(1)$, or even that $T(n) = \Theta(1)$ for all $n \le 10^{100}$. I'll also assume for simplicity that n is an integral power of b; we'll see how to avoid this assumption later (but to summarize: it doesn't matter).

Now $T(n)$ is just the sum of all values stored in the recursion tree. For each i, the ith level of the tree contains a^i nodes, each with value $f(n/b^i)$. Thus,

$$\boxed{T(n) = \sum_{i=0}^{L} a^i f(n/b^i)} \qquad (\Sigma)$$

where L is the depth of the recursion tree. We easily see that $\boldsymbol{L = \log_b n}$, because $n/b^L = 1$. The base case $f(1) = \Theta(1)$ implies that the last non-zero term in the summation is $\Theta(a^L) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$.

For *most* divide-and-conquer recurrences, the level-by-level sum (Σ) is a *geometric* series—each term is a constant factor larger or smaller than the previous term. In this case, only the largest term in the geometric series matters; all of the other terms are swallowed up by the $\Theta(\cdot)$ notation.



A recursion tree for the recurrence $T(n) = a\,T(n/b) + f(n)$

Here are several examples of the recursion-tree technique in action:

- **Mergesort (simplified): $\boldsymbol{T(n) = 2T(n/2) + n}$**

  There are 2^i nodes at level i, each with value $n/2^i$, so every term in the level-by-level sum (Σ) is the same:

  $$T(n) = \sum_{i=0}^{L} n.$$

  The recursion tree has $L = \log_2 n$ levels, so $\boldsymbol{T(n) = \Theta(n \log n)}$.

- **Randomized selection: $\boldsymbol{T(n) = T(3n/4) + n}$**

  The recursion tree is a single path. The node at depth i has value $(3/4)^i n$, so the level-by-level sum (Σ) is a decreasing geometric series:

  $$T(n) = \sum_{i=0}^{L} (3/4)^i n.$$

  This geometric series is dominated by its initial term n, so $\boldsymbol{T(n) = \Theta(n)}$. The recursion tree has $L = \log_{4/3} n$ levels, but so what?

- **Karatsuba's multiplication algorithm: $T(n) = 3T(n/2) + n$**

  There are 3^i nodes at depth i, each with value $n/2^i$, so the level-by-level sum (Σ) is an increasing geometric series:

  $$T(n) = \sum_{i=0}^{L} (3/2)^i n.$$

  This geometric series is dominated by its final term $(3/2)^L n$. Each leaf contributes 1 to this term; thus, the final term is equal to the number of leaves in the tree! The recursion tree has $L = \log_2 n$ levels, and therefore $3^{\log_2 n} = n^{\log_2 3}$ leaves, so $\boldsymbol{T(n) = \Theta(n^{\log_2 3})}$.

- **$T(n) = 2T(n/2) + n/\lg n$**

  The sum of all the nodes in the ith level is $n/(\lg n - i)$. This implies that the depth of the tree is at most $\lg n - 1$. The level sums are neither constant nor a geometric series, so we just have to evaluate the overall sum directly.

  Recall (or if you're seeing this for the first time: Behold!) that the nth *harmonic number* H_n is the sum of the reciprocals of the first n positive integers:

  $$H_n := \sum_{i=1}^{n} \frac{1}{i}$$

  It's nat hard to show that $H_n = \Theta(\log n)$; in fact, we have the stronger inequalities $\ln(n+1) \le H_n \le \ln n + 1$.

  $$T(n) = \sum_{i=0}^{\lg n - 1} \frac{n}{\lg n - i} = \sum_{j=1}^{\lg n} \frac{n}{j} = nH_{\lg n} = \boldsymbol{\Theta(n \lg \lg n)}$$

- **$T(n) = 4T(n/2) + n \lg n$**

  There are 4^i nodes at each level i, each with value $(n/2^i)\lg(n/2^i) = (n/2^i)(\lg n - i)$; again, the depth of the tree is at most $\lg n - 1$. We have the following summation:

  $$T(n) = \sum_{i=0}^{\lg n - 1} n2^i(\lg n - i)$$

  We can simplify this sum by substituting $j = \lg n - i$:

  $$T(n) = \sum_{j=i}^{\lg n} n2^{\lg n - j} j = \sum_{j=i}^{\lg n} n^2 j/2^j = \boldsymbol{\Theta(j^2)}$$

  The last step uses the fact that $\sum_{i=1}^{\infty} j/2^j = 2$. Although this is not quite a geometric series, it is still dominated by its largest term.

- **Ugly divide and conquer: $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$**

  We solved this recurrence earlier by guessing the right answer and verifying, but we can use recursion trees to get the correct answer directly. The *degree* of the nodes in the recursion tree is

no longer constant, so we have to be a bit more careful, but the same basic technique still applies. It's not hard to see that the nodes in any level sum to n. The depth L satisfies the identity $n^{2^{-L}} = 2$ (we can't get all the way down to 1 by taking square roots), so $L = \lg\lg n$ and $\boldsymbol{T(n) = \Theta(n \lg\lg n)}$.

- **Randomized quicksort: $\boldsymbol{T(n) = T(3n/4) + T(n/4) + n}$**

  This recurrence isn't in the standard form described earlier, but we can still solve it using recursion trees. Now modes in the same level of the recursion tree have different values, and different leaves are at different levels. However, the nodes in any *complete* level (that is, above any of the leaves) sum to n. Moreover, every leaf in the recursion tree has depth between $\log_4 n$ and $\log_{4/3} n$. To derive an upper bound, we overestimate $T(n)$ by ignoring the base cases and extending the tree downward to the level of the *deepest* leaf. Similarly, to derive a lower bound, we overestimate $T(n)$ by counting only nodes in the tree up to the level of the *shallowest* leaf. These observations give us the upper and lower bounds $n\log_4 n \le T(n) \le n\log_{4/3} n$. Since these bounds differ by only a constant factor, we have $\boldsymbol{T(n) = \Theta(n \log n)}$.

- **Deterministic selection: $\boldsymbol{T(n) = T(n/5) + T(7n/10) + n}$**

  Again, we have a lopsided recursion tree. If we look only at complete levels of the tree, we find that the level sums form a descending geometric series $T(n) = n + 9n/10 + 81n/100 + \cdots$. We can get an upper bound by ignoring the base cases entirely and growing the tree out to infinity, and we can get a lower bound by only counting nodes in complete levels. Either way, the geometric series is dominated by its largest term, so $\boldsymbol{T(n) = \Theta(n)}$.

- **Randomized search trees: $\boldsymbol{T(n) = \frac{1}{4}T(n/4) + \frac{3}{4}T(3n/4) + 1}$**

  This looks like a divide-and-conquer recurrence, but what does it mean to have a quarter of a child? The right approach is to imagine that each node in the recursion tree has a *weight* in addition to its value. Alternately, we get a standard recursion tree again if we add a second real parameter to the recurrence, defining $T(n) = T(n, 1)$, where

  $$T(n, \alpha) = T(n/4, \alpha/4) + T(3n/4, 3\alpha/4) + \alpha.$$

  In each complete level of the tree, the (weighted) node values sum to exactly 1. The leaves of the recursion tree are at different levels, but all between $\log_4 n$ and $\log_{4/3} n$. So we have upper and lower bounds $\log_4 n \le T(n) \le \log_{4/3} n$, which differ by only a constant factor, so $\boldsymbol{T(n) = \Theta(\log n)}$.

- **Ham-sandwich trees: $\boldsymbol{T(n) = T(n/2) + T(n/4) + 1}$**

  Again, we have a lopsided recursion tree. If we only look at complete levels, we find that the level sums form an *ascending* geometric series $T(n) = 1 + 2 + 4 + \cdots$, so the solution is dominated by the number of leaves. The recursion tree has $\log_4 n$ complete levels, so there are more than $2^{\log_4 n} = n^{\log_4 2} = \sqrt{n}$; on the other hand, every leaf has depth at most $\log_2 n$, so the total number of leaves is at most $2^{\log_2 n} = n$. Unfortunately, the crude bounds $\boldsymbol{\sqrt{n} \ll T(n) \ll n}$ are the best we can derive using the techniques we know so far!

The following theorem completely describes the solution for any divide-and-conquer recurrence in the 'standard form' $T(n) = aT(n/b) + f(n)$, where a and b are constants and $f(n)$ is a polynomial. This theorem allows us to bypass recursion trees for 'standard' recurrences, but many people (including Jeff) find it harder to remember than the more general recursion-tree technique. Your mileage may vary.

The Master Theorem. *The recurrence $T(n) = aT(n/b) + f(n)$ can be solved as follows.*

- *If $a\,f(n/b) = \kappa\, f(n)$ for some constant $\kappa < 1$, then $T(n) = \Theta(f(n))$.*
- *If $a\,f(n/b) = K\, f(n)$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$.*
- *If $a\,f(n/b) = f(n)$, then $T(n) = \Theta(f(n)\log_b n)$.*
- *If none of these three cases apply, you're on your own.*

Proof: If $f(n)$ is a *constant factor larger* than $a\,f(b/n)$, then by induction, the sum is a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$.

If $f(n)$ is a *constant factor smaller* than $a\,f(b/n)$, then by induction, the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $\Theta(n^{\log_b a})$.

Finally, if $a\,f(b/n) = f(n)$, then by induction, each of the $L+1$ terms in the sum is equal to $f(n)$. □

4 Linear Recurrences (Annihilators)

Another common class of recurrences, called ***linear*** recurrences, arises in the context of recursive backtracking algorithms and counting problems. These recurrences express each function value $f(n)$ as a *linear* combination of a small number of nearby values $f(n-1), f(n-2), f(n-3), \ldots$. The Fibonacci recurrence is a typical example:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{otherwise} \end{cases}$$

It turns out that the solution to *any* linear recurrence is a simple combination of polynomial and exponential functions in n. For example, we can verify by induction that the linear recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \text{ or } n = 2 \\ 3T(n-1) - 8T(n-2) + 4T(n-3) & \text{otherwise} \end{cases}$$

has the closed-form solution $\boldsymbol{T(n) = (n-3)2^n + 4}$. First we check the base cases:

$$\begin{aligned} T(0) &= (0-3)2^0 + 4 = 1 \quad \checkmark \\ T(1) &= (1-3)2^1 + 4 = 0 \quad \checkmark \\ T(2) &= (2-3)2^2 + 4 = 0 \quad \checkmark \end{aligned}$$

And now the recursive case:

$$\begin{aligned} T(n) &= 3T(n-1) - 8T(n-2) + 4T(n-3) \\ &= 3((n-4)2^{n-1} + 4) - 8((n-5)2^{n-2} + 4) + 4((n-6)2^{n-3} + 4) \\ &= \left(\frac{3}{2} - \frac{8}{4} + \frac{4}{8}\right) n \cdot 2^n - \left(\frac{12}{2} - \frac{40}{4} + \frac{24}{8}\right) 2^n + (2 - 8 + 4) \cdot 4 \\ &= (n-3) \cdot 2^n + 4 \quad \checkmark \end{aligned}$$

But how could we have possibly come up with that solution? In this section, I'll describe a general method for solving linear recurrences that's arguably easier than the induction proof!

4.1 Operators

Our technique for solving linear recurrences relies on the theory of ***operators***. Operators are higher-order functions, which take one or more functions as input and produce different functions as output. For example, your first two semesters of calculus focus almost exclusively on the *differential* and *integral* operators $\frac{d}{dx}$ and $\int dx$. All the operators we will need are combinations of three elementary building blocks:

- ***Sum:*** $(f + g)(n) := f(n) + g(n)$
- ***Scale:*** $(\alpha \cdot f)(n) := \alpha \cdot (f(n))$
- ***Shift:*** $(\boldsymbol{E}f)(n) := f(n+1)$

The shift and scale operators are ***linear***, which means they can be distributed over sums; for example, for any functions f, g, and h, we have $\boldsymbol{E}(f - 3(g - h)) = \boldsymbol{E}f + (-3)\boldsymbol{E}g + 3\boldsymbol{E}h$.

We can combine these building blocks to obtain more complex *compound* operators. For example, the compound operator $\boldsymbol{E} - 2$ is defined by setting $(\boldsymbol{E} - 2)f := \boldsymbol{E}f + (-2)f$ for any function f. We can also apply the shift operator twice: $(\boldsymbol{E}(\boldsymbol{E}f))(n) = f(n+2)$; we write usually $\boldsymbol{E}^2 f$ as a synonym for $\boldsymbol{E}(\boldsymbol{E}f)$. More generally, for any positive integer k, the operator $\boldsymbol{E}^k$ shifts its argument k times: $\boldsymbol{E}^k f(n) = f(n+k)$. Similarly, $(\boldsymbol{E} - 2)^2$ is shorthand for the operator $(\boldsymbol{E} - 2)(\boldsymbol{E} - 2)$, which applies $(\boldsymbol{E} - 2)$ twice.

For example, here are the results of applying different operators to the function $f(n) = 2^n$:

$$\begin{aligned}
2f(n) &= 2 \cdot 2^n = 2^{n+1} \\
3f(n) &= 3 \cdot 2^n \\
\boldsymbol{E}f(n) &= 2^{n+1} \\
\boldsymbol{E}^2 f(n) &= 2^{n+2} \\
(\boldsymbol{E} - 2)f(n) &= \boldsymbol{E}f(n) - 2f(n) = 2^{n+1} - 2^{n+1} = 0 \\
(\boldsymbol{E}^2 - 1)f(n) &= \boldsymbol{E}^2 f(n) - f(n) = 2^{n+2} - 2^n = 3 \cdot 2^n
\end{aligned}$$

These compound operators can be manipulated exactly as though they were polynomials over the 'variable' $\boldsymbol{E}$. In particular, we can 'factor' compound operators into 'products' of simpler operators, and the order of the factors is unimportant. For example, the compound operators $\boldsymbol{E}^2 - 3\boldsymbol{E} + 2$ and $(\boldsymbol{E} - 1)(\boldsymbol{E} - 2)$ are equivalent:

$$\text{Let} \quad g(n) := (\boldsymbol{E} - 2)f(n) = f(n+1) - 2f(n).$$

$$\begin{aligned}
\text{Then} \quad (\boldsymbol{E} - 1)(\boldsymbol{E} - 2)f(n) &= (\boldsymbol{E} - 1)g(n) \\
&= g(n+1) - g(n) \\
&= (f(n+2) - 2f(n-1)) - (f(n+1) - 2f(n)) \\
&= f(n+2) - 3f(n+1) + 2f(n) \\
&= (\boldsymbol{E}^2 - 3\boldsymbol{E} + 2)f(n). \quad \checkmark
\end{aligned}$$

It is an easy exercise to confirm that $\boldsymbol{E}^2 - 3\boldsymbol{E} + 2$ is also equivalent to the operator $(\boldsymbol{E} - 2)(\boldsymbol{E} - 1)$.

The following table summarizes everything we need to remember about operators.

Operator	Definition
addition	$(f+g)(n) := f(n)+g(n)$
subtraction	$(f-g)(n) := f(n)-g(n)$
multiplication	$(\alpha \cdot f)(n) := \alpha \cdot (f(n))$
shift	$\boldsymbol{E}f(n) := f(n+1)$
k-fold shift	$\boldsymbol{E}^k f(n) := f(n+k)$
composition	$(\boldsymbol{X}+\boldsymbol{Y})f := \boldsymbol{X}f + \boldsymbol{Y}f$
	$(\boldsymbol{X}-\boldsymbol{Y})f := \boldsymbol{X}f - \boldsymbol{Y}f$
	$\boldsymbol{X}\boldsymbol{Y}f := \boldsymbol{X}(\boldsymbol{Y}f) = \boldsymbol{Y}(\boldsymbol{X}f)$
distribution	$\boldsymbol{X}(f+g) = \boldsymbol{X}f + \boldsymbol{X}g$

4.2 Annihilators

An ***annihilator*** of a function f is any nontrivial operator that transforms f into the zero function. (We can trivially annihilate any function by multiplying it by zero, so as a technical matter, we do not consider the zero operator to be an annihilator.) Every compound operator we consider annihilates a specific class of functions; conversely, every function composed of polynomial and exponential functions has a unique (minimal) annihilator.

We have already seen that the operator $(\boldsymbol{E}-2)$ annihilates the function 2^n. It's not hard to see that the operator $(\boldsymbol{E}-c)$ annihilates the function $\alpha \cdot c^n$, for any constants c and α. More generally, the operator $(\boldsymbol{E}-c)$ annihilates the function a^n if and only if $c = a$:

$$(\boldsymbol{E}-c)a^n = \boldsymbol{E}a^n - c \cdot a^n = a^{n+1} - c \cdot a^n = (a-c)a^n.$$

Thus, $(\boldsymbol{E}-2)$ is essentially the *only* annihilator of the function 2^n.

What about the function $2^n + 3^n$? The operator $(\boldsymbol{E}-2)$ annihilates the function 2^n, but leaves the function 3^n unchanged. Similarly, $(\boldsymbol{E}-3)$ annihilates 3^n while *negating* the function 2^n. But if we apply *both* operators, we annihilate both terms:

$$\begin{aligned}(\boldsymbol{E}-2)(2^n+3^n) &= \boldsymbol{E}(2^n+3^n) - 2(2^n+3^n) \\ &= (2^{n+1}+3^{n+1}) - (2^{n+1}+2\cdot 3^n) = 3^n \\ \Longrightarrow (\boldsymbol{E}-3)(\boldsymbol{E}-2)(2^n+3^n) &= (\boldsymbol{E}-3)3^n = 0\end{aligned}$$

In general, for any integers $a \neq b$, the operator $(\boldsymbol{E}-a)(\boldsymbol{E}-b) = (\boldsymbol{E}-b)(\boldsymbol{E}-a) = (\boldsymbol{E}^2 - (a+b)\boldsymbol{E} + ab)$ annihilates any function of the form $\alpha a^n + \beta b^n$, but nothing else.

What about the operator $(\boldsymbol{E}-a)(\boldsymbol{E}-a) = (\boldsymbol{E}-a)^2$? It turns out that this operator annihilates all functions of the form $(\alpha n + \beta)a^n$:

$$\begin{aligned}(\boldsymbol{E}-a)((\alpha n+\beta)a^n) &= (\alpha(n+1)+\beta)a^{n+1} - a(\alpha n+\beta)a^n \\ &= \alpha a^{n+1} \\ \Longrightarrow (\boldsymbol{E}-a)^2((\alpha n+\beta)a^n) &= (\boldsymbol{E}-a)(\alpha a^{n+1}) = 0\end{aligned}$$

More generally, the operator $(\boldsymbol{E}-a)^d$ annihilates all functions of the form $p(n)\cdot a^n$, where $p(n)$ is a polynomial of degree at most $d-1$. For example, $(\boldsymbol{E}-1)^3$ annihilates any polynomial of degree at most 2.

The following table summarizes everything we need to remember about annihilators.

Operator	Functions annihilated	
$\boldsymbol{E}-1$	α	
$\boldsymbol{E}-a$	αa^n	
$(\boldsymbol{E}-a)(\boldsymbol{E}-b)$	$\alpha a^n + \beta b^n$	[if $a \neq b$]
$(\boldsymbol{E}-a_0)(\boldsymbol{E}-a_1)\cdots(\boldsymbol{E}-a_k)$	$\sum_{i=0}^{k} \alpha_i a_i^n$	[if a_i distinct]
$(\boldsymbol{E}-1)^2$	$\alpha n + \beta$	
$(\boldsymbol{E}-a)^2$	$(\alpha n + \beta)a^n$	
$(\boldsymbol{E}-a)^2(\boldsymbol{E}-b)$	$(\alpha n + \beta)a^b + \gamma b^n$	[if $a \neq b$]
$(\boldsymbol{E}-a)^d$	$\left(\sum_{i=0}^{d-1} \alpha_i n^i\right)a^n$	

If $\boldsymbol{X}$ annihilates f, then $\boldsymbol{X}$ also annihilates $\boldsymbol{E}f$.
If $\boldsymbol{X}$ annihilates both f and g, then $\boldsymbol{X}$ also annihilates $f \pm g$.
If $\boldsymbol{X}$ annihilates f, then $\boldsymbol{X}$ also annihilates αf, for any constant α.

If $\boldsymbol{X}$ annihilates f and $\boldsymbol{Y}$ annihilates g, then $\boldsymbol{XY}$ annihilates $f \pm g$.

4.3 Annihilating Recurrences

Given a linear recurrence for a function, it's easy to extract an annihilator for that function. For many recurrences, we only need to rewrite the recurrence in operator notation. Once we have an annihilator, we can factor it into operators of the form $(\boldsymbol{E}-c)$; the table on the previous page then gives us a generic solution with some unknown coefficients. If we are given explicit base cases, we can determine the coefficients by examining a few small cases; in general, this involves solving a small system of linear equations. If the base cases are not specified, the generic solution almost always gives us an asymptotic solution. Here is the technique step by step:

1. Write the recurrence in operator form
2. Extract an annihilator for the recurrence
3. Factor the annihilator (if necessary)
4. Extract the *generic solution* from the annihilator
5. Solve for coefficients using base cases (if known)

Here are several examples of the technique in action:

- **$r(n) = 5r(n-1)$, *where* $r(0) = 3$.**

  1. We can write the recurrence in operator form as follows:

  $$r(n) = 5r(n-1) \implies r(n+1) - 5r(n) = 0 \implies (\boldsymbol{E}-5)r(n) = 0.$$

  2. We immediately see that $(\boldsymbol{E}-5)$ annihilates the function $r(n)$.
  3. The annihilator $(\boldsymbol{E}-5)$ is already factored.
  4. Consulting the annihilator table on the previous page, we find the generic solution $r(n) = \alpha 5^n$ for some constant α.
  5. The base case $r(0) = 3$ implies that $\alpha = 3$.

We conclude that $\boldsymbol{r(n) = 3 \cdot 5^n}$. We can easily verify this closed-form solution by induction:

$$\begin{aligned} r(0) &= 3 \cdot 5^0 = 3 \quad \checkmark && \text{[definition]} \\ r(n) &= 5r(n-1) && \text{[definition]} \\ &= 5 \cdot (3 \cdot 5^{n-1}) && \text{[induction hypothesis]} \\ &= 5^n \cdot 3 \quad \checkmark && \text{[algebra]} \end{aligned}$$

- ***Fibonacci numbers:*** $\boldsymbol{F(n) = F(n-1) + F(n-2)}$***, where*** $\boldsymbol{F(0) = 0}$ ***and*** $\boldsymbol{F(1) = 1}$**.**

  1. We can rewrite the recurrence as $(\boldsymbol{E}^2 - \boldsymbol{E} - 1)F(n) = 0$.
  2. The operator $\boldsymbol{E}^2 - \boldsymbol{E} - 1$ clearly annihilates $F(n)$.
  3. The quadratic formula implies that the annihilator $\boldsymbol{E}^2 - \boldsymbol{E} - 1$ factors into $(\boldsymbol{E} - \phi)(\boldsymbol{E} - \hat{\phi})$, where $\phi = (1 + \sqrt{5})/2 \approx 1.618034$ is the golden ratio and $\hat{\phi} = (1 - \sqrt{5})/2 = 1 - \phi = -1/\phi$.
  4. The annihilator implies that $\boldsymbol{F(n) = \alpha\phi^n + \hat{\alpha}\hat{\phi}^n}$ for some unknown constants α and $\hat{\alpha}$.
  5. The base cases give us two equations in two unknowns:

     $$\begin{aligned} F(0) &= 0 = \alpha + \hat{\alpha} \\ F(1) &= 1 = \alpha\phi + \hat{\alpha}\hat{\phi} \end{aligned}$$

     Solving this system of equations gives us $\alpha = 1/(2\phi - 1) = 1/\sqrt{5}$ and $\hat{\alpha} = -1/\sqrt{5}$.

  We conclude with the following exact closed form for the nth Fibonacci number:

  $$\boldsymbol{F(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n}$$

  With all the square roots in this formula, it's quite amazing that Fibonacci numbers are integers. However, if we do all the math correctly, all the square roots cancel out when i is an integer. (In fact, this is pretty easy to prove using the binomial theorem.)

- ***Towers of Hanoi:*** $\boldsymbol{T(n) = 2T(n-1) + 1}$***, where*** $\boldsymbol{T(0) = 0}$**.** This is our first example of a *non-homogeneous* recurrence, which means the recurrence has one or more non-recursive terms.

  1. We can rewrite the recurrence as $(\boldsymbol{E} - 2)T(n) = 1$.
  2. The operator $(\boldsymbol{E} - 2)$ doesn't quite annihilate the function; it leaves a *residue* of 1. But we can annihilate the residue by applying the operator $(\boldsymbol{E} - 1)$. Thus, the compound operator $(\boldsymbol{E} - 1)(\boldsymbol{E} - 2)$ annihilates the function.
  3. The annihilator is already factored.
  4. The annihilator table gives us the generic solution $T(n) = \alpha 2^n + \beta$ for some unknown constants α and β.
  5. The base cases give us $T(0) = 0 = \alpha 2^0 + \beta$ and $T(1) = 1 = \alpha 2^1 + \beta$. Solving this system of equations, we find that $\alpha = 1$ and $\beta = -1$.

  We conclude that $\boldsymbol{T(n) = 2^n - 1}$.

For the remaining examples, I won't explicitly enumerate the steps in the solution.

- ***Height-balanced trees:*** $\boldsymbol{H(n) = H(n-1)+H(n-2)+1}$***, where*** $\boldsymbol{H(-1) = 0}$ ***and*** $\boldsymbol{H(0) = 1}$**.** (Yes, we're starting at -1 instead of 0. So what?)

  We can rewrite the recurrence as $(\boldsymbol{E}^2 - \boldsymbol{E} - 1)H = 1$. The residue 1 is annihilated by $(\boldsymbol{E} - 1)$, so the compound operator $(\boldsymbol{E} - 1)(\boldsymbol{E}^2 - \boldsymbol{E} - 1)$ annihilates the recurrence. This operator factors into $(\boldsymbol{E}-1)(\boldsymbol{E}-\phi)(\boldsymbol{E}-\hat{\phi})$, where $\phi = (1+\sqrt{5})/2$ and $\hat{\phi} = (1-\sqrt{5})/2$. Thus, we get the generic solution $H(n) = \alpha \cdot \phi^n + \beta + \gamma \cdot \hat{\phi}^n$, for some unknown constants α, β, γ that satisfy the following system of equations:

$$\begin{aligned} H(-1) = 0 = \alpha\phi^{-1} + \beta + \gamma\hat{\phi}^{-1} &= \alpha/\phi + \beta - \gamma/\hat{\phi} \\ H(0) = 1 = \alpha\phi^{0} + \beta + \gamma\hat{\phi}^{0} &= \alpha + \beta + \gamma \\ H(1) = 2 = \alpha\phi^{1} + \beta + \gamma\hat{\phi}^{1} &= \alpha\phi + \beta + \gamma\hat{\phi} \end{aligned}$$

  Solving this system (using Cramer's rule or Gaussian elimination), we find that $\alpha = (\sqrt{5}+2)/\sqrt{5}$, $\beta = -1$, and $\gamma = (\sqrt{5}-2)/\sqrt{5}$. We conclude that

$$\boldsymbol{H(n) = \frac{\sqrt{5}+2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - 1 + \frac{\sqrt{5}-2}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.}$$

- $\boldsymbol{T(n) = 3T(n-1) - 8T(n-2) + 4T(n-3)}$***, where*** $\boldsymbol{T(0) = 1}$**,** $\boldsymbol{T(1) = 0}$***, and*** $\boldsymbol{T(2) = 0}$**.** This was our original example of a linear recurrence.

  We can rewrite the recurrence as $(E^3 - 3E^2 + 8E - 4)T = 0$, so we immediately have an annihilator $E^3 - 3E^2 + 8E - 4$. Using high-school algebra, we can factor the annihilator into $(\boldsymbol{E} - 2)^2(\boldsymbol{E} - 1)$, which implies the generic solution $T(n) = \alpha n 2^n + \beta 2^n + \gamma$. The constants α, β, and γ are determined by the base cases:

$$\begin{aligned} T(0) &= 1 = \alpha \cdot 0 \cdot 2^0 + \beta 2^0 + \gamma = \beta + \gamma \\ T(1) &= 0 = \alpha \cdot 1 \cdot 2^1 + \beta 2^1 + \gamma = 2\alpha + 2\beta + \gamma \\ T(2) &= 0 = \alpha \cdot 2 \cdot 2^2 + \beta 2^2 + \gamma = 8\alpha + 4\beta + \gamma \end{aligned}$$

  Solving this system of equations, we find that $\alpha = 1$, $\beta = -3$, and $\gamma = 4$, so $\boldsymbol{T(n) = (n-3)2^n + 4}$.

- $\boldsymbol{T(n) = T(n-1) + 2T(n-2) + 2^n - n^2}$

  We can rewrite the recurrence as $(\boldsymbol{E}^2 - \boldsymbol{E} - 2)T(n) = \boldsymbol{E}^2(2^n - n^2)$. Notice that we had to shift up the non-recursive parts of the recurrence when we expressed it in this form. The operator $(\boldsymbol{E} - 2)(\boldsymbol{E} - 1)^3$ annihilates the residue $2^n - n^2$, and therefore also annihilates the shifted residue $\boldsymbol{E}^2(2^n + n^2)$. Thus, the operator $(\boldsymbol{E}-2)(\boldsymbol{E}-1)^3(\boldsymbol{E}^2-\boldsymbol{E}-2)$ annihilates the entire recurrence. We can factor the quadratic factor into $(\boldsymbol{E}-2)(E+1)$, so the annihilator factors into $(\boldsymbol{E}-2)^2(\boldsymbol{E}-1)^3(\boldsymbol{E}+1)$. So the generic solution is $\boldsymbol{T(n) = \alpha n 2^n + \beta 2^n + \gamma n^2 + \delta n + \varepsilon + \eta(-1)^n}$. The coefficients α, β, γ, δ, ε, η satisfy a system of six equations determined by the first six function values $T(0)$ through $T(5)$. For almost[2] every set of base cases, we have $\alpha \neq 0$, which implies that $\boldsymbol{T(n) = \Theta(n2^n)}$.

For a more detailed explanation of the annihilator method, see George Lueker, Some techniques for solving recurrences, *ACM Computing Surveys* 12(4):419-436, 1980.

[2] In fact, the only possible solutions with $\alpha = 0$ have the form $-2^{n-1} - n^2/2 - 5n/2 + \eta(-1)^n$ for some constant η.

5 Transformations

Sometimes we encounter recurrences that don't fit the structures required for recursion trees or annihilators. In many of those cases, we can transform the recurrence into a more familiar form, by defining a new function in terms of the one we want to solve. There are many different kinds of transformations, but these three are probably the most useful:

- Domain transformation: Define a new function $S(n) = T(f(n))$ with a simpler recurrence, for some simple function f.
- Range transformation: Define a new function $S(n) = f(T(n))$ with a simpler recurrence, for some simple function f.
- Difference transformation: Simplify the recurrence for $T(n)$ by considering the difference $T(n) - T(n-1)$.

Here are some examples of these transformations in action.

- **Unsimplified Mergesort: $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$**

  When n is a power of 2, we can simplify the mergesort recurrence to $T(n) = 2T(n/2) + \Theta(n)$, which has the solution $T(n) = \Theta(n \log n)$. Unfortunately, for other values values of n, this simplified recurrence is incorrect. When n is odd, then the recurrence calls for us to sort a fractional number of elements! Worse yet, if n is not a power of 2, we will *never* reach the base case $T(1) = 1$.

  So we really need to solve the original recurrence. We have no hope of getting an *exact* solution, even if we ignore the $\Theta(\,)$ in the recurrence; the floors and ceilings will eventually kill us. But we can derive a tight asymptotic solution using a domain transformation—we can rewrite the function $T(n)$ as a nested function $S(f(n))$, where $f(n)$ is a simple function and the function $S(\,)$ has an simpler recurrence.

  First let's overestimate the time bound, once by pretending that the two subproblem sizes are equal, and again to eliminate the ceiling:

  $$T(n) \le 2T\left(\lceil n/2 \rceil\right) + n \le 2T(n/2 + 1) + n.$$

  Now we define a new function $S(n) = T(n + \alpha)$, where α is a unknown constant, chosen so that $S(n)$ satisfies the Master-Theorem-ready recurrence $S(n) \le 2S(n/2) + O(n)$. To figure out the correct value of α, we compare two versions of the recurrence for the function $T(n + \alpha)$:

  $$\begin{aligned} S(n) \le 2S(n/2) + O(n) \quad &\Longrightarrow \quad T(n+\alpha) \le 2T(n/2 + \alpha) + O(n) \\ T(n) \le 2T(n/2 + 1) + n \quad &\Longrightarrow \quad T(n+\alpha) \le 2T((n+\alpha)/2 + 1) + n + \alpha \end{aligned}$$

  For these two recurrences to be equal, we need $n/2 + \alpha = (n + \alpha)/2 + 1$, which implies that $\alpha = 2$. The Master Theorem now tells us that $S(n) = O(n \log n)$, so

  $$T(n) = S(n-2) = O((n-2)\log(n-2)) = O(n \log n).$$

  A similar argument implies the matching lower bound $T(n) = \Omega(n \log n)$. So $\boldsymbol{T(n) = \Theta(n \log n)}$ after all, just as though we had ignored the floors and ceilings from the beginning!

  Domain transformations are useful for removing floors, ceilings, and lower order terms from the arguments of any recurrence that otherwise looks like it ought to fit either the Master Theorem or the recursion tree method. But now that we know this, we don't need to bother grinding through the actual gory details!

- **Ham-Sandwich Trees: $T(n) = T(n/2) + T(n/4) + 1$**

  As we saw earlier, the recursion tree method only gives us the uselessly loose bounds $\sqrt{n} \ll T(n) \ll n$ for this recurrence, and the recurrence is in the wrong form for annihilators. The authors who discovered ham-sandwich trees (yes, this is a real data structure) solved this recurrence by guessing the solution and giving a complicated induction proof.

  But a simple transformation allows us to solve the recurrence in just a few lines. We define a new function $t(k) = T(2^k)$, which satisfies the simpler linear recurrence $t(k) = t(k-1) + t(k-2) + 1$. This recurrence should immediately remind you of Fibonacci numbers. Sure enough, the annihilator method implies the solution $t(k) = \Theta(\phi^k)$, where $\phi = (1+\sqrt{5})/2$ is the golden ratio. We conclude that

  $$T(n) = t(\lg n) = \Theta(\phi^{\lg n}) = \boldsymbol{\Theta(n^{\lg \phi})} \approx \Theta(n^{0.69424}).$$

  Many other divide-and-conquer recurrences can be similarly transformed into linear recurrences and then solved with annihilators. Consider once more the simplified mergesort recurrence $T(n) = 2T(n/2) + n$. The function $t(k) = T(2^k)$ satisfies the recurrence $t(k) = 2t(k-1) + 2^k$. The annihilator method gives us the generic solution $t(k) = \Theta(k \cdot 2^k)$, which implies that $T(n) = t(\lg n) = \Theta(n \log n)$, just as we expected.

  On the other hand, for some recurrences like $T(n) = T(n/3) + T(2n/3) + n$, the recursion tree method gives an easy solution, but there's no way to transform the recurrence into a form where we can apply the annihilator method directly.[3]

- **Random Binary Search Trees: $T(n) = \frac{1}{4}T(n/4) + \frac{3}{4}T(3n/4) + 1$**

  This looks like a divide-and-conquer recurrence, so we might be tempted to apply recursion trees, but what does it mean to have a quarter of a child? If we're not comfortable with weighted recursion trees, we can instead consider a new function $U(n) = n \cdot T(n)$, which satisfies the recurrence $U(n) = U(n/4) + U(3n/4) + n$. As we've already seen, recursion trees imply that $U(n) = \Theta(n \log n)$, which immediately implies that $\boldsymbol{T(n) = \Theta(\log n)}$.

- **Randomized Quicksort: $T(n) = \frac{2}{n}\sum_{k=0}^{n-1} T(k) + n$**

  This is our first example of a ***full history*** recurrence; each function value $T(n)$ is defined in terms of *all* previous function values $T(k)$ with $k < n$. Before we can apply any of our existing techniques, we need to convert this recurrence into an equivalent *limited history* form by shifting and subtracting away common terms. To make this step slightly easier, we first multiply both sides of the recurrence by n to get rid of the fractions.

[3] However, we can still get a solution via functional transformations as follows. The function $t(k) = T((3/2)^k)$ satisfies the recurrence $t(n) = t(n-1) + t(n-\lambda) + (3/2)^k$, where $\lambda = \log_{3/2} 3 = 2.709511\ldots$. The ***characteristic function*** for this recurrence is $(r^\lambda - r^{\lambda-1} - 1)(r - 3/2)$, which has a double root at $r = 3/2$ and nowhere else. Thus, $t(k) = \Theta(k(3/2)^k)$, which implies that $T(n) = t(\log_{3/2} n) = \Theta(n \log n)$.

$$n \cdot T(n) = 2\sum_{k=0}^{n-1} T(j) + n^2 \qquad \text{[multiply both sides by } n\text{]}$$

$$(n-1) \cdot T(n-1) = 2\sum_{k=0}^{n-2} T(j) + (n-1)^2 \qquad \text{[shift]}$$

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2n - 1 \qquad \text{[subtract]}$$

$$T(n) = \frac{n+1}{n} T(n-1) + 2 - \frac{1}{n} \qquad \text{[simplify]}$$

We can solve this limited-history recurrence using another functional transformation. We define a new function $t(n) = T(n)/(n+1)$, which satisfies the simpler recurrence

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

which we can easily unroll into a summation. If we only want an asymptotic solution, we can simplify the final recurrence to $t(n) = t(n-1) + \Theta(1/n)$, which unrolls into a very familiar summation:

$$t(n) = \sum_{i=1}^{n} \Theta(1/i) = \Theta(H_n) = \Theta(\log n).$$

Finally, substituting $T(n) = (n+1)t(n)$ gives us a solution to the original recurrence: $\boldsymbol{T(n) = \Theta(n \log n)}$.

Exercises

1. For each of the following recurrences, first ***guess*** an exact closed-form solution, and then prove your guess is correct. You are free to use any method you want to make your guess—unrolling the recurrence, writing out the first several values, induction proof template, recursion trees, annihilators, transformations, 'It looks like that other one', whatever—but please describe your method. All functions are from the non-negative integers to the reals. If it simplifies your solutions, express them in terms of Fibonacci numbers F_n, harmonic numbers H_n, binomial coefficients $\binom{n}{k}$, factorials $n!$, and/or the floor and ceiling functions $\lfloor x \rfloor$ and $\lceil x \rceil$.

   (a) $A(n) = A(n-1) + 1$, where $A(0) = 0$.

   (b) $B(n) = \begin{cases} 0 & \text{if } n < 5 \\ B(n-5) + 2 & \text{otherwise} \end{cases}$

   (c) $C(n) = C(n-1) + 2n - 1$, where $C(0) = 0$.

   (d) $D(n) = D(n-1) + \binom{n}{2}$, where $D(0) = 0$.

   (e) $E(n) = E(n-1) + 2^n$, where $E(0) = 0$.

   (f) $F(n) = 3 \cdot F(n-1)$, where $F(0) = 1$.

   (g) $G(n) = \frac{G(n-1)}{G(n-2)}$, where $G(0) = 1$ and $G(1) = 2$. *[Hint: This is easier than it looks.]*

   (h) $H(n) = H(n-1) + 1/n$, where $H(0) = 0$.

   (i) $I(n) = I(n-2) + 3/n$, where $I(0) = I(1) = 0$. *[Hint: Consider even and odd n separately.]*

   (j) $J(n) = J(n-1)^2$, where $J(0) = 2$.

   (k) $K(n) = K(\lfloor n/2 \rfloor) + 1$, where $K(0) = 0$.

   (l) $L(n) = L(n-1) + L(n-2)$, where $L(0) = 2$ and $L(1) = 1$.
   [Hint: Write the solution in terms of Fibonacci numbers.]

   (m) $M(n) = M(n-1) \cdot M(n-2)$, where $M(0) = 2$ and $M(1) = 1$.
   [Hint: Write the solution in terms of Fibonacci numbers.]

   (n) $N(n) = 1 + \sum_{k=1}^{n} (N(k-1) + N(n-k))$, where $N(0) = 1$.

   (p) $P(n) = \sum_{k=0}^{n-1} (k \cdot P(k-1))$, where $P(0) = 1$.

   (q) $Q(n) = \frac{1}{2 - Q(n-1)}$, where $Q(0) = 0$.

   (r) $R(n) = \max_{1 \le k \le n} \{R(k-1) + R(n-k) + n\}$

   (s) $S(n) = \max_{1 \le k \le n} \{S(k-1) + S(n-k) + 1\}$

   (t) $T(n) = \min_{1 \le k \le n} \{T(k-1) + T(n-k) + n\}$

   (u) $U(n) = \min_{1 \le k \le n} \{U(k-1) + U(n-k) + 1\}$

   (v) $V(n) = \max_{n/3 \le k \le 2n/3} \{V(k-1) + V(n-k) + n\}$

2. Use recursion trees to solve each of the following recurrences.

   (a) $A(n) = 2A(n/4) + \sqrt{n}$

   (b) $B(n) = 2B(n/4) + n$

   (c) $C(n) = 2C(n/4) + n^2$

   (d) $D(n) = 3D(n/3) + \sqrt{n}$

   (e) $E(n) = 3E(n/3) + n$

   (f) $F(n) = 3F(n/3) + n^2$

   (g) $G(n) = 4G(n/2) + \sqrt{n}$

   (h) $H(n) = 4H(n/2) + n$

   (i) $I(n) = 4I(n/2) + n^2$

   (j) $J(n) = J(n/2) + J(n/3) + J(n/6) + n$

   (k) $K(n) = K(n/2) + K(n/3) + K(n/6) + n^2$

   (l) $L(n) = L(n/15) + L(n/10) + 2L(n/6) + \sqrt{n}$

   (m) $M(n) = \sqrt{2n}\,M(\sqrt{2n}) + \sqrt{n}$

   (n) $N(n) = \sqrt{2n}\,N(\sqrt{2n}) + n$

   (p) $P(n) = \sqrt{2n}\,P(\sqrt{2n}) + n^2$

   (q) $Q(n) = Q(n-3) + 8^n$ — Don't use annihilators!

   (r) $R(n) = 2R(n-2) + 4^n$ — Don't use annihilators!

   (s) $S(n) = 4S(n-1) + 2^n$ — Don't use annihilators!

3. Make up a bunch of linear recurrences and then solve them using annihilators.

4. Solve the following recurrences, using any tricks at your disposal.

   (a) $T(n) = \sum_{i=1}^{\lg n} T(n/2^i) + n$ *[Hint: Assume n is a power of 2.]*

   (b) More to come...