Recurrence Relations and their closed-form solutions

In "divide and conquer" algorithms one usually ends up with a recurrence relation that "defines" the "timing function", $T(n)$. For example, it could look like

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

In order to assess the "goodness" of the proposed algorithm by comparison to either our expectations or to another algorithm, we need to know $T(n)$ in "closed" form in terms of known functions such as n^r for $r > 0$, c^n for $c > 1$, $\log_b n$ for some integer $b > 1$.

Often a preliminary analysis need only worry about the "asymptotic behaviour" of the algorithm, i.e., the behaviour for *large* inputs (n is the input size). "Big-O" notation is an excellent tool in this case, therefore the solution of recurrences is often sought in such notation. On occasion one requires an "exact" solution (this is much harder to achieve in general).

There is a big variety of recurrence relations and an equally big variety of solution techniques. Some restricted cases are handled well by packages such as *Mathematica* or *Maple V*. For the mathematical reasons that make the solutions tick the best reference is perhaps Knuth *et al.* "*Concrete Mathematics*" (Addison-Wesley).

In this note we restrict attention to simple classes of recurrences taken from both the "additive" and "multiplicative" cases (the latter characterizations refer to the manner of handling the "index" or argument of the recurrence; see below).

1. The Additive Case

The general case here is of the form†

$$\begin{aligned} T_0 &= k \\ s_n T_n &= v_n T_{n-1} + f(n) \textbf{ if } n > 0 \end{aligned}$$

a recurrence defining the *sequence* T_n, or equivalently, the *function* $T(n)$ (both jargons and notations spell out the same thing), in terms of the *known* functions (sequences) $s_n, v_n, f(n)$.

For the general case see Knuth cited above. Here we will restrict attention to the case $s_n = 1$ for all n and $v_n = a$ (a constant) for all n.

Subcase 1. ($a = 1$) Solve

$$\begin{aligned} T_0 &= k \\ T_n &= T_{n-1} + f(n) \textbf{ if } n > 0 \end{aligned} \tag{1}$$

From (1), $T_n - T_{n-1} = f(n)$, thus

$$\sum_{i=1}^{n} (T_i - T_{i-1}) = \sum_{i=1}^{n} f(i)$$

therefore

$$\begin{aligned} T_n &= T_0 + \sum_{i=1}^{n} f(i) \\ &= k + \sum_{i=1}^{n} f(i) \end{aligned} \tag{2}$$

If we know how to get the sum in (2) in closed form, then we solved the problem.

1.1 Example. Solve

$$p_n = \begin{cases} 2 & \textbf{if } n = 1 \\ p_{n-1} + n & \textbf{otherwise} \end{cases} \tag{3}$$

Here

$$\sum_{i=2}^{n} (p_i - p_{i-1}) = \sum_{i=2}^{n} i$$

Note the lower bound of the summation: It is here 2, for the "basis" case in (3) is for $n = 1$ rather than $n = 0$.

Thus,

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

† Note the "additivity" in the relation between indices: n vs $n - 1$.

(Where did I get the $(n+2)(n-1)/2$ from?) The above answer is the same as (verify!)

$$p_n = 1 + \frac{(n+1)n}{2}$$

obtained by writing

$$2 + \sum_{i=2}^{n} i = 1 + \sum_{i=1}^{n} i$$

□

Subcase 2. ($a \neq 1$) Solve

$$\begin{aligned} T_0 &= k \\ T_n &= aT_{n-1} + f(n) \textbf{ if } n > 0 \end{aligned} \tag{4}$$

(4) is the same as

$$\frac{T_n}{a^n} = \frac{T_{n-1}}{a^{n-1}} + \frac{f(n)}{a^n}$$

To simplify notation, set

$$t_n \stackrel{\text{def}}{=} \frac{T_n}{a^n}$$

thus the recurrence (4) becomes

$$\begin{aligned} t_0 &= k \\ t_n &= t_{n-1} + \frac{f(n)}{a^n} \textbf{ if } n > 0 \end{aligned} \tag{5}$$

By subcase 1, this yields

$$t_n = k + \sum_{i=1}^{n} \frac{f(i)}{a^i}$$

from which

$$T_n = ka^n + a^n \sum_{i=1}^{n} \frac{f(i)}{a^i} \tag{6}$$

1.2 Example. As an illustration, solve the recurrence that captures the "Towers of Hanoi" solution timing.

$$T_n = \begin{cases} 1 & \textbf{if } n = 1 \\ 2T_{n-1} + 1 & \textbf{otherwise} \end{cases}$$

To avoid trouble, note that when the basis of (4) is at $n = 1$, the basis of (5) is $t_1 = k/a$ rather than $t_0 = k$. So, the right hand side of (6) will have ka^{n-1} instead of ka^n (Why?) and the indexing in the summation will start at $i = 2$ (Why?)

Thus, by (6),

$$\begin{aligned} T_n &= 2^{n-1} + 2^n \sum_{i=2}^{n} \frac{1}{2^i} \\ &= 2^{n-1} + 2^n (\frac{(2^{-1})^{n+1} - 1}{2^{-1} - 1} - 1 - \frac{1}{2}) \\ &= 2^{n-1} + 2^n (2 - 2^{-n} - 1 - \frac{1}{2}) \\ &= 2^n - 1 \end{aligned}$$

□

2. The Multiplicative Case

Subcase 1.

$$T(n) = \begin{cases} k & \textbf{if } n = 1 \\ aT(n/b) + c & \textbf{if } n > 1 \end{cases} \tag{1}$$

were a, b are positive integer constants ($b > 1$) and k, c any constants. Recurrences like (1) above occur in divide and conquer solutions to problems. For example, *binary search* has timing governed by the above recurrence with $b = 2, a = c = k = 1$.

We seek a general solution in "Big-O" notation.

First convert to an "additive indices" problem: To this end, seek a solution in the *restricted* set $\{n \in \mathbb{N} : n = b^m \text{ for some } m \in \mathbb{N}\}$. Next, set

$$t(m) = T(b^m) \tag{2}$$

so that the recurrence becomes

$$t(m) = \begin{cases} k & \textbf{if } m = 0 \\ at(m-1) + c & \textbf{if } m > 0 \end{cases} \tag{3}$$

hence, from the work in the previous section,

$$\sum_{i=1}^{m} (\frac{t(i)}{a^i} - \frac{t(i-1)}{a^{i-1}}) = c \sum_{i=1}^{m} a^{-i}$$

therefore

$$t(m) = a^m k + c a^m \begin{cases} m & \textbf{if } a = 1 \\ a^{-1} \dfrac{(a^{-1})^m - 1}{a^{-1} - 1} & \textbf{if } a \neq 1 \end{cases}$$

or, more simply,

$$t(m) = \begin{cases} k + cm & \textbf{if } a = 1 \\ a^m k + c \dfrac{a^m - 1}{a - 1} & \textbf{if } a \neq 1 \end{cases}$$

Using O-notation, and going back to T we get:

$$T(b^m) = \begin{cases} O(m) & \textbf{if } a = 1 \\ O(a^m) & \textbf{if } a \neq 1 \end{cases} \tag{4}$$

or, *provided we remember that this solution relies on the assumption that* n has the form b^m:

$$T(n) = \begin{cases} O(\log n) & \textbf{if } a = 1 \\ O(a^{\log_b n}) & \textbf{if } a \neq 1 \end{cases} = \begin{cases} O(\log n) & \textbf{if } a = 1 \\ O(n^{\log_b a}) & \textbf{if } a \neq 1 \end{cases} \tag{5}$$

If $a > b$ then we get slower than linear "run time" $O(n^{\log_b a})$. If on the other hand $b > a > 1$ then we get a sublinear run time, since then $\log_b a < 1$.

Now an important observation. For functions $T(n)$ that are *increasing*,† i.e., $T(i) \leq T(j)$ if $i < j$ the restriction of n to have form b^m proves to be *irrelevant* in obtaining the solution. The solution is still given by (5) *for all* n. Here's why:

In the general case, n satisfies

$$b^{m-1} < n \leq b^m \text{ for some } m \geq 0 \tag{6}$$

Suppose now that $a = 1$ (upper case in (4)). We want to establish that $T(n) = O(\log n)$ for the general n (of (6)). By monotonicity of T and the second inequality of (6) we get

$$T(n) \overset{\text{by (6)}}{\leq} T(b^m) \overset{\text{by (4)}}{=} O(m) \overset{\text{by (6)}}{=} O(\log n)$$

The last invocation of (6) above used the first inequality therein.

The case where $a > 1$ is handled similarly. Here we found an answer $O(n^r)$ (where $r = \log_b a > 0$) *provided* $n = b^m$ (some m). Relax this proviso, and assume (6).

Now

$$T(n) \overset{\text{by (6)}}{\leq} T(b^m) \overset{\text{by (4)}}{=} O(a^m) = O((b^m)^r) \overset{\text{Why?}}{=} O((b^{m-1})^r) \overset{\text{by (6)}}{=} O(n^r)$$

where again the last invocation of (6) above used the first inequality therein.

Subcase 2.

$$T(n) = \begin{cases} k & \textbf{if } n = 1 \\ aT(n/b) + cn & \textbf{if } n > 1 \end{cases} \tag{1'}$$

were a, b are positive integer constants ($b > 1$) and k, c any constants. Recurrences like (1′) above occur in divide and conquer solutions to problems. For example, *two-way merge sort* has timing governed by the above recurrence with $a = b = 2$ and $c = 1/2$. Quicksort has *average* run time governed, essentially, by the above with $a = b = 2$ and $c = 1$. Both lead to $O(n \log n)$ solutions. Also, *Karatsuba integer multiplication* has run time recurrence as above with $a = 3, b = 2$.

† Such are the "complexity" or "timing" functions of algorithms.

Setting at first (the famous *initial* restriction on n) $n = b^m$ for some $m \in \mathbb{N}$ and using (2) above we end up with a variation on (3):

$$t(m) = \begin{cases} k & \textbf{if } m = 0 \\ at(m-1) + cb^m & \textbf{if } m > 0 \end{cases} \tag{3'}$$

thus we need do

$$\sum_{i=1}^{m} \left(\frac{t(i)}{a^i} - \frac{t(i-1)}{a^{i-1}}\right) = c \sum_{i=1}^{m} (b/a)^i$$

therefore

$$t(m) = a^m k + ca^m \begin{cases} m & \textbf{if } a = b \\ (b/a)\dfrac{(b/a)^m - 1}{b/a - 1} & \textbf{if } a \neq b \end{cases}$$

Using O-notation, and using cases according as to $a < b$ or $a > b$ we get:

$$t(m) = \begin{cases} O(b^m m) & \textbf{if } a = b \\ a^m O(1) = O(a^m) & \textbf{if } b < a \\ O(b^m - a^m) = O(b^m) & \textbf{if } b > a \end{cases} \qquad /^*\ (b/a)^m \to 0 \text{ as } m \to \infty\ ^*/$$

or, in terms of T and n, which is *restricted* to form b^m (using same calculational "tricks" as before):

$$T(n) = \begin{cases} O(n \log n) & \textbf{if } a = b \\ O(n^{\log_b a}) & \textbf{if } b < a \\ O(n) & \textbf{if } b > a \end{cases} \tag{4'}$$

The above solution is valid for *any* n without restriction, *provided* T is increasing. The proof is as before, so we will not redo it (you may wish to check the "new case" $O(n \log n)$ as an exercise).

In terms of complexity of algorithms, the above solution says that in a divide and conquer algorithm (governed by (1')) we have the following cases:

- The total size of all subproblems we solve (recursively) is *equal* to the original problem's size. Then we have a $O(n \log n)$ algorithm (e.g., merge sort).
- The total size of all subproblems we solve is *more* than the original problem's size. Then we go worse than linear ($\log_b a > 1$ in this case). An example is Karatsuba multiplication that runs in $O(n^{\log_2 3})$ time.
- The total size of all subproblems we solve is *less* than the original problem's size. Then we go in linear time (e.g., the problem of finding the k-th smallest in a *set* of n elements).

3. Generating Functions

We saw some simple cases of recurrence relations with additive and multiplicative index structure (we reduced the latter to the former). Now we turn to a wider class of additive index structure problems where our previous technique of utilizing a "telescoping sum"

$$\sum_{i=1}^{n}(t(i)-t(i-1))$$

does not apply because the right hand side still refers to $t(i)$ for some $i < n$. Such is the case of the well known Fibonacci sequence F_n given by

$$F_n = \begin{cases} 0 & \textbf{if } n = 0 \\ 1 & \textbf{if } n = 1 \\ F_{n-1} + F_{n-1} & \textbf{if } n > 1 \end{cases}$$

The method of *generating functions* that solves this harder problem also solves the previous problems we saw.

Here's the method in outline. We will then embark on a number of fully worked out solutions.

Given a recurrence relation

$$t_n = \ldots t_{n-1} \ldots t_{n-2} \ldots t_{n-3} \ldots \tag{1}$$

with the appropriate "starting" (initial) conditions. We want t_n in "closed form" in terms of known functions. Here are the steps:

- Define a *generating function* of the *sequence* $t_0, t_1, \ldots, t_n, \ldots$

$$\begin{aligned} G(z) &= \sum_{i=0}^{\infty} t_i z^i \\ &= t_0 + t_1 z + t_2 z^2 + \cdots + t_n z^n + \cdots \end{aligned} \tag{2}$$

  (2) is a *formal power series*, where *formal* means that we only are interested in the *form* of the "infinite sum" and *not* in any issues of convergence† (therefore "meaning") of the sum. It is stressed that our disinterest in convergence matters is *not* a simplifying convenience but it is due to the fact that convergence issues are *irrelevant* to the problem at hand.


In particular, we will *never* have to consider values of z or make substitutions in z.


- Using the recurrence (1), find a *closed form* of $G(z)$ as a function of z (this *can* be done *prior* to knowing the t_n in closed form!)

† In Calculus one learns that power series converge in an interval like $|z| < r$ for some real $r \geq 0$. The $r = 0$ case means the series diverges for *all* z.

- Expand the closed form $G(z)$ back into a power series

$$\begin{aligned} G(z) &= \sum_{i=0}^{\infty} a_i z^i \\ &= a_0 + a_1 z + a_2 z^2 + \cdots + a_n z^n + \cdots \end{aligned} \tag{3}$$

  But now we *do have* the a_n's in terms of known functions, because we know $G(z)$ in closed form! We only need to compare (2) and (3) and proclaim

$$t_n = a_n \quad \text{for } n = 0, 1, \ldots$$

  The problem has been solved.

Steps bullet 2 and 3 embody all the real work. We will illustrate by examples how this is done in practice, but first we need some "tools":

The Binomial Expansion. For our purposes we will be content with just one tool, the "binomial expansion theorem" of calculus:

$$\begin{aligned} \text{For any } \textit{real } m, \qquad (1+z)^m &= \sum_{r=0}^{\infty} \binom{m}{r} z^r \\ &= \cdots + \binom{m}{r} z^r + \cdots \end{aligned} \tag{4}$$

where for any $r \in \mathbb{N}$ and $m \in \mathbb{R}$

$$\binom{m}{r} \stackrel{\text{def}}{=} \begin{cases} 1 & \textbf{if } r = 0 \\ \dfrac{m(m-1)\cdots(m-[r-1])}{r!} & \textbf{otherwise} \end{cases} \tag{5}$$

The expansion (4) terminates with last term

$$\binom{m}{m} z^m \stackrel{\text{by (5)}}{=} z^m$$

as the "binomial theorem of *Algebra* says, iff m is a *positive integer*. In *all* other cases (4) is non-terminating (infinitely many terms). As we remarked before, we will not be concerned with when (4) converges.

Note that (5) gives the familiar

$$\begin{aligned} \binom{m}{r} &= \frac{m(m-1)\cdots(m-[r-1])}{r!} \\ &= \frac{m(m-1)\cdots(m-[r-1])(m-r)\cdots 2\cdot 1}{r!(m-r)!} \\ &= \frac{m!}{r!(m-r)!} \end{aligned}$$

when $m \in \mathbb{N}$. *In all other cases we use* (5) for if $m \notin \mathbb{N}$, then "$m!$" is meaningless.

Let us record the very useful special case when m is a *negative integer*, $-n$ $(n > 0)$.

$$\begin{aligned}(1+z)^{-n} &= \cdots + \frac{-n(-n-1)\cdots(-n-[r-1])}{r!}z^r + \cdots \\ &= \cdots + (-1)^r \frac{n(n+1)\cdots(n+[r-1])}{r!}z^r + \cdots \\ &= \cdots + (-1)^r \frac{(n+[r-1])\cdots(n+1)n}{r!}z^r + \cdots \\ &= \cdots + (-1)^r \binom{n+r-1}{r} z^r + \cdots \end{aligned} \tag{6}$$

Finally, let us record important special cases of (6)

$$(1-z)^{-n} = \cdots + \binom{n+r-1}{r} z^r + \cdots \tag{7}$$

$$\begin{aligned}(1-z)^{-1} = \frac{1}{1-z} &= \cdots + \binom{r}{r} z^r + \cdots \\ &= \cdots + z^r + \cdots \end{aligned} \tag{8}$$

the familiar "converging geometric progression" (converging for $|z| < 1$, that is, but this is the last time I'll raise irrelevant convergence issues). Two more special cases of (6) will be helpful:

$$\begin{aligned}(1-z)^{-2} = \frac{1}{(1-z)^2} &= \cdots + \binom{r+1}{r} z^r + \cdots \\ &= \cdots + (r+1)z^r + \cdots \end{aligned} \tag{9}$$

and

$$\begin{aligned}(1-z)^{-3} = \frac{1}{(1-z)^3} &= \cdots + \binom{r+2}{r} z^r + \cdots \\ &= \cdots + \frac{(r+2)(r+1)}{2} z^r + \cdots \end{aligned} \tag{10}$$

3.1 Example. Solve the recurrence

$$\begin{aligned} a_0 &= 1 \\ a_n &= 2a_{n-1} + 1 \qquad \textbf{if } n > 0 \end{aligned} \tag{i}$$

Write (i) as

$$a_n - 2a_{n-1} = 1 \tag{ii}$$

Next, form the generating function for a_n, and a "shifted" copy of it (multiplied by 2) underneath it (this was "inspired" by (ii)):

$$\begin{aligned} G(z) &= a_0 + a_1 z + a_2 z^2 + \cdots + a_n z^n + \cdots \\ 2zG(z) &= \quad 2a_0 z + 2a_1 z^2 + \cdots + 2a_{n-1} z^n + \cdots \end{aligned}$$

Subtract the above term-by-term to get

$$\begin{aligned} G(z)(1-2z) &= 1 + z + z^2 + z^3 + \cdots \\ &= \frac{1}{1-z} \end{aligned}$$

Hence

$$G(z) = \frac{1}{(1-2z)(1-z)} \qquad (iii)$$

(iii) is $G(z)$ in closed form. To expand it back to a (known) power series we first we use the "partial fractions" method (familiar to students of calculus) to write $G(z)$ as the sum of two fractions with *linear* denominators. I.e., find constants A and B such that (iv) below is true for all z:

$$\frac{1}{(1-2z)(1-z)} = \frac{A}{(1-2z)} + \frac{B}{(1-z)}$$

or

$$1 = A(1-z) + B(1-2z)$$

Setting in turn $z \leftarrow 1$ and $z \leftarrow 1/2$ we find $B = -1$ and $A = 2$, hence

$$\begin{aligned} G(z) &= \frac{2}{1-2z} - \frac{1}{1-z} \\ &= 2\big(\cdots (2z)^n \cdots\big) - \big(\cdots z^n \cdots\big) \\ &= \cdots (2^{n+1} - 1) z^n \cdots \end{aligned}$$

Comparing this known expansion with the original power series above, we conclude that

$$a_n = 2^{n+1} - 1$$

Of course, we solved this problem much easier in section 1. However due to its simplicity it was worked out here again to illustrate this new method. Normally, you apply the method of generating functions when there is *no other simpler way to do it.* □

3.2 Example. Solve

$$\begin{aligned} p_1 &= 2 \\ p_n &= p_{n-1} + n \qquad \textbf{if } n > 1 \end{aligned} \qquad (i)$$

Write (i) as

$$p_n - p_{n-1} = n \qquad (ii)$$

Next, form the generating function for p_n, and a "shifted" copy of it underneath it (this was "inspired" by (ii)).
Note how this sequence starts with p_1 (rather than p_0). Correspondingly, the constant term of the generating function is p_1.

$$\begin{aligned} G(z) &= p_1 + p_2 z + p_3 z^2 + \cdots + p_{n+1} z^n + \cdots \\ zG(z) &= \quad p_1 z + p_2 z^2 + \cdots + p_n z^n + \cdots \end{aligned}$$

Subtract the above term-by-term to get

$$\begin{aligned} G(z)(1-z) &= 2 + 2z + 3z^2 + 4z^3 + \cdots + (n+1)z^n + \cdots \\ &= 1 + \frac{1}{(1-z)^2} \qquad \text{by (9)} \end{aligned}$$

Hence

$$\begin{aligned} G(z) &= \frac{1}{1-z} + \frac{1}{(1-z)^3} \\ &= \left(\cdots z^n \cdots\right) + \left(\cdots \frac{(n+2)(n+1)}{2} z^n \cdots\right) \qquad \text{by (10)} \\ &= \cdots \left(1 + \frac{(n+2)(n+1)}{2}\right) z^n \cdots \end{aligned}$$

Comparing this known expansion with the original power series above, we conclude that

$$p_{n+1} = 1 + \frac{(n+2)(n+1)}{2}$$

or

$$p_n = 1 + \frac{(n+1)n}{2}$$

□

3.3 Example. Here is one that cannot be handled by the techniques of section 1.

$$\begin{aligned} s_0 &= 1 \\ s_1 &= 1 \\ s_n &= 4s_{n-1} - 4s_{n-2} \quad \textbf{if } n > 1 \end{aligned} \tag{i}$$

Write (i) as

$$s_n - 4s_{n-1} + 4s_{n-2} = 0 \tag{ii}$$

to "inspire"

$$\begin{aligned} G(z) &= s_0 + s_1 z + s_2 z^2 + \cdots + s_n z^n + \cdots \\ 4zG(z) &= \quad 4s_0 z + 4s_1 z^2 + \cdots + 4s_{n-1} z^n + \cdots \\ 4z^2 G(z) &= \quad\quad 4s_0 z^2 + \cdots + 4s_{n-2} z^n + \cdots \end{aligned}$$

By (ii),

$$\begin{aligned} G(z)(1-4z+4z^2) &= 1+(1-4)z \\ &= 1-3z \end{aligned}$$

Since $1-4z+4z^2=(1-2z)^2$ we get

$$\begin{aligned} G(z) &= \frac{1}{(1-2z)^2} - 3z\frac{1}{(1-2z)^2} \\ &= \big(\cdots(n+1)(2z)^n\cdots\big) - 3z\big(\cdots(n+1)(2z)^n\cdots\big) \\ &= \big(\cdots\big[(n+1)2^n - 3n2^{n-1}\big]z^n\cdots\big) \end{aligned}$$

Thus,

$$\begin{aligned} s_n &= (n+1)2^n - 3n2^{n-1} \\ &= 2^{n-1}(2n+2-3n) \\ &= 2^n(1-n/2) \end{aligned}$$

□

3.4 Example. Here is another one that cannot be handled by the techniques of section 1.

$$\begin{aligned} s_0 &= 0 \\ s_1 &= 8 \\ s_n &= 2s_{n-1}+3s_{n-2} \quad \textbf{if } n>1 \end{aligned} \tag{i}$$

Write (i) as

$$s_n - 2s_{n-1} - 3s_{n-2} = 0 \tag{ii}$$

Next,

$$\begin{array}{lll} G(z) & = s_0 + s_1z + s_2z^2 & +\cdots + s_nz^n \quad +\cdots \\ 2zG(z) & = \quad 2s_0z + 2s_1z^2 & +\cdots + 2s_{n-1}z^n +\cdots \\ 3z^2G(z) & = \quad\quad 3s_0z^2 & +\cdots + 3s_{n-2}z^n +\cdots \end{array}$$

By (ii),

$$G(z)(1-2z-3z^2) = 8z$$

The roots of $1-2z-3z^2=0$ are

$$z = \frac{-2\pm\sqrt{4+12}}{6} = \frac{-2\pm 4}{6} = \begin{cases} -1 \\ 1/3 \end{cases}$$

hence $1-2z-3z^2 = -3(z+1)(z-1/3) = (1-3z)(1+z)$, therefore

$$G(z) = \frac{8z}{(1-3z)(1+z)} = \frac{A}{1-3z} + \frac{B}{1+z} \text{ spliting into partial fractions}$$

By a calculation as in the previous example, $A = 2$ and $B = -2$, so

$$\begin{aligned} G(z) &= \frac{2}{1-3z} - \frac{2}{1+z} \\ &= 2\big(\cdots (3z)^n \cdots\big) - 2\big(\cdots (-z)^n \cdots\big) \\ &= (\cdots [2\cdot 3^n - 2(-1)^n] z^n \cdots) \end{aligned}$$

hence $s_n = 2\cdot 3^n - 2(-1)^n$ $\square$

3.5 Example. The Fibonacci recurrence.

$$\begin{aligned} &F_0 = 0 \\ &F_1 = 1 \\ &F_n = F_{n-1} + F_{n-2} \quad \textbf{if } n > 1 \end{aligned} \tag{i}$$

Write (i) as

$$F_n - F_{n-1} - F_{n-2} = 0 \tag{ii}$$

Next,

$$\begin{aligned} G(z) &= F_0 + F_1 z + F_2 z^2 + \cdots + F_n z^n + \cdots \\ zG(z) &= F_0 z + F_1 z^2 + \cdots + F_{n-1} z^n + \cdots \\ z^2 G(z) &= F_0 z^2 + \cdots + F_{n-2} z^n + \cdots \end{aligned}$$

By (ii),

$$G(z)(1 - z - z^2) = z$$

The roots of $1 - z - z^2 = 0$ are

$$z = \frac{-1 \pm \sqrt{1+4}}{2} = \begin{cases} \dfrac{-1+\sqrt{5}}{2} \\ \dfrac{-1-\sqrt{5}}{2} \end{cases}$$

For convenience of notation, set

$$\phi_1 = \frac{-1+\sqrt{5}}{2}, \qquad \phi_2 = \frac{-1-\sqrt{5}}{2} \tag{iii}$$

Hence

$$\begin{aligned} 1 - z - z^2 &= -(z-\phi_1)(z-\phi_2) \\ &= -(\phi_1 - z)(\phi_2 - z) \end{aligned} \tag{iv}$$

therefore

$$G(z) = \frac{z}{1-z-z^2} = \frac{A}{\phi_1 - z} + \frac{B}{\phi_2 - z} \text{ spliting into partial fractions}$$

from which,

$$A = \frac{\phi_1}{\phi_1 - \phi_2}, \qquad B = \frac{\phi_2}{\phi_2 - \phi_1}$$

so

$$\begin{aligned} G(z) &= \frac{1}{\phi_1 - \phi_2}\Big[\frac{\phi_1}{\phi_1 - z} - \frac{\phi_2}{\phi_2 - z}\Big] \\ &= \frac{1}{\phi_1 - \phi_2}\Big[\frac{1}{1 - z/\phi_1} - \frac{1}{1 - z/\phi_2}\Big] \\ &= \frac{1}{\phi_1 - \phi_2}\left(\Big(\cdots [\frac{z}{\phi_1}]^n \cdots\Big) - \Big(\cdots [\frac{z}{\phi_2}]^n \cdots\Big)\right) \end{aligned}$$

therefore

$$F_n = \frac{1}{\phi_1 - \phi_2}\left(\frac{1}{\phi_1^n} - \frac{1}{\phi_2^n}\right) \qquad (v)$$

Let's simplify (v):

First, by brute force calculation, or by using the "known" relations between the roots of a 2nd degree equation, we find

$$\phi_1\phi_2 = -1, \qquad \phi_1 - \phi_2 = \sqrt{5}$$

so that (v) gives

$$\begin{aligned} F_n &= \frac{1}{\sqrt{5}}\left(\frac{\phi_2^n}{(\phi_1\phi_2)^n} - \frac{\phi_1^n}{(\phi_1\phi_2)^n}\right) \\ &= \frac{1}{\sqrt{5}}\left((-1)^n\frac{\big((1+\sqrt{5})/2\big)^n}{(-1)^n} - (-1)^n\frac{\big((1-\sqrt{5})/2\big)^n}{(-1)^n}\right) \\ &= \frac{1}{\sqrt{5}}\left(\left[\frac{1+\sqrt{5}}{2}\right]^n - \left[\frac{1-\sqrt{5}}{2}\right]^n\right) \end{aligned}$$

In particular, we find that

$$F_n = O\left(\left[\frac{1+\sqrt{5}}{2}\right]^n\right)$$

since

$$\left[\frac{1-\sqrt{5}}{2}\right]^n \to 0 \text{ as } n \to \infty$$

i.e., F_n grows exponentially with n, since $|\phi_2| > 1$. $\square$