6 Recurrences

Algorithm 2 mergesort(list L)

$n \leftarrow \text{size}(L)$
if $n \leq 1$ **return** L
$L_1 \leftarrow L[1 \cdots \lfloor \frac{n}{2} \rfloor]$
$L_2 \leftarrow L[\lfloor \frac{n}{2} \rfloor + 1 \cdots n]$
$\text{mergesort}(L_1)$
$\text{mergesort}(L_2)$
$L \leftarrow \text{merge}(L_1, L_2)$
return L

This algorithm requires

$$T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \mathcal{O}(n) \leq 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \mathcal{O}(n)$$

comparisons when $n > 1$ and 0 comparisons when $n \leq 1$.

Recurrences

How do we bring the expression for the number of comparisons ($\approx$ running time) into a closed form?

For this we need to solve the recurrence.

Methods for Solving Recurrences

1. **Guessing+Induction**
   Guess the right solution and prove that it is correct via induction. It needs experience to make the right guess.
2. **Master Theorem**
   For a lot of recurrences that appear in the analysis of algorithms this theorem can be used to obtain tight asymptotic bounds. It does not provide exact solutions.
3. **Characteristic Polynomial**
   Linear homogenous recurrences can be solved via this method.

Methods for Solving Recurrences

4. **Generating Functions**
   A more general technique that allows to solve certain types of linear inhomogenous relations and also sometimes non-linear recurrence relations.
5. **Transformation of the Recurrence**
   Sometimes one can transform the given recurrence relations so that it e.g. becomes linear and can therefore be solved with one of the other techniques.

6.1 Guessing+Induction

First we need to get rid of the $\mathcal{O}$-notation in our recurrence:

$$T(n) \leq \begin{cases} 2T(\lceil \frac{n}{2} \rceil) + cn & n \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

Assume that instead we had

$$T(n) \leq \begin{cases} 2T(\frac{n}{2}) + cn & n \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

One way of solving such a recurrence is to guess a solution, and check that it is correct by plugging it in.

6.1 Guessing+Induction

Suppose we guess $T(n) \leq dn \log n$ for a constant d. Then

$$\begin{aligned} T(n) &\leq 2T\left(\frac{n}{2}\right) + cn \\ &\leq 2\left(d\frac{n}{2}\log\frac{n}{2}\right) + cn \\ &= dn(\log n - 1) + cn \\ &= dn\log n + (c - d)n \\ &\leq dn\log n \end{aligned}$$

if we choose $d \geq c$.

Formally one would make an induction proof, where the above is the induction step. The base case is usually trivial.

6.1 Guessing+Induction

$$T(n) \leq \begin{cases} 2T(\frac{n}{2}) + cn & n \geq 16 \\ b & \text{otw.} \end{cases}$$

Guess: $T(n) \leq dn \log n$.
Proof. (by induction)

- **base case** $(2 \leq n < 16)$: true if we choose $d \geq b$.
- **induction step** $2 \ldots n-1 \to n$:

  Suppose statem. is true for $n' \in \{2, \ldots, n-1\}$, and $n \geq 16$.
  We prove it for n:

$$\begin{aligned} T(n) &\leq 2T\left(\frac{n}{2}\right) + cn \\ &\leq 2\left(d\frac{n}{2}\log\frac{n}{2}\right) + cn \\ &= dn(\log n - 1) + cn \\ &= dn\log n + (c - d)n \\ &\leq dn\log n \end{aligned}$$

- Note that this proves the statement for $n \in \mathbb{N}_{\geq 2}$, as the statement is wrong for $n = 1$.
- The base case is usually omitted, as it is the same for different recurrences.

Hence, statement is true if we choose $d \geq c$.

6.1 Guessing+Induction

Why did we change the recurrence by getting rid of the ceiling?

If we do not do this we instead consider the following recurrence:

$$T(n) \leq \begin{cases} 2T(\lceil \frac{n}{2} \rceil) + cn & n \geq 16 \\ b & \text{otherwise} \end{cases}$$

Note that we can do this as for constant-sized inputs the running time is always some constant (b in the above case).

6.1 Guessing+Induction

We also make a guess of $T(n) \le dn\log n$ and get

$$\begin{aligned}
T(n) &\le 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + cn \\
&\le 2\left(d\left\lceil \frac{n}{2} \right\rceil \log\left\lceil \frac{n}{2} \right\rceil\right) + cn \\
\boxed{\left\lceil \tfrac{n}{2} \right\rceil \le \tfrac{n}{2}+1} \quad &\le 2(d(n/2+1)\log(n/2+1)) + cn \\
\boxed{\tfrac{n}{2}+1 \le \tfrac{9}{16}n} \quad &\le dn\log\left(\frac{9}{16}n\right) + 2d\log n + cn \\
\boxed{\log \tfrac{9}{16}n = \log n + (\log 9 - 4)} \quad &= dn\log n + (\log 9 - 4)dn + 2d\log n + cn \\
\boxed{\log n \le \tfrac{n}{4}} \quad &\le dn\log n + (\log 9 - 3.5)dn + cn \\
&\le dn\log n - 0.33dn + cn \\
&\le dn\log n
\end{aligned}$$

for a suitable choice of d.

6.2 Master Theorem

Note that the cases do not cover all possibilities.

Lemma 1

Let $a \ge 1, b \ge 1$ and $\epsilon > 0$ denote constants. Consider the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)\ .$$

Case 1.
If $f(n) = \mathcal{O}(n^{\log_b(a)-\epsilon})$ then $T(n) = \Theta(n^{\log_b a})$.

Case 2.
If $f(n) = \Theta(n^{\log_b(a)}\log^k n)$ then $T(n) = \Theta(n^{\log_b a}\log^{k+1} n)$, $k \ge 0$.

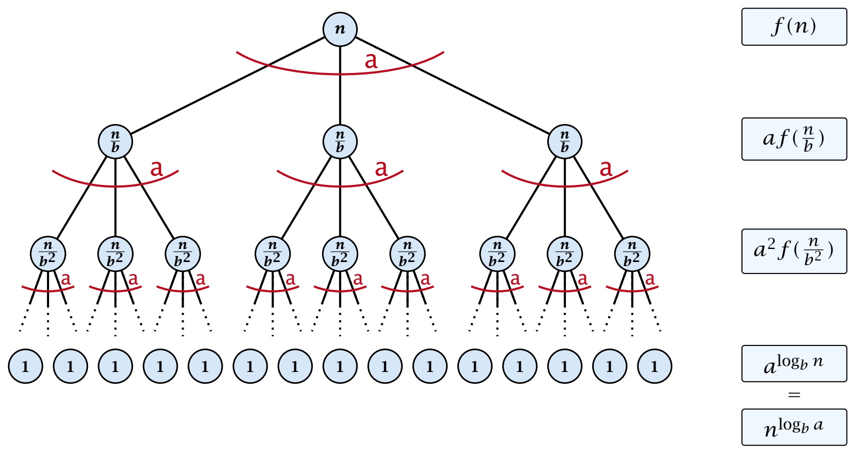
Case 3.
If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ and for sufficiently large n $af(\frac{n}{b}) \le cf(n)$ for some constant $c < 1$ then $T(n) = \Theta(f(n))$.

6.2 Master Theorem

We prove the Master Theorem for the case that n is of the form b^ℓ, and we assume that the non-recursive case occurs for problem size 1 and incurs cost 1.

The Recursion Tree

The running time of a recursive algorithm can be visualized by a recursion tree:



6.2 Master Theorem

This gives

$$T(n) = n^{\log_b a} + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) .$$

Case 1. Now suppose that $f(n) \le cn^{\log_b a - \epsilon}$.

$$\begin{aligned} T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\ &\le c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon} \\ &= cn^{\log_b a - \epsilon} \sum_{i=0}^{\log_b n - 1} \left(b^\epsilon\right)^i \\ &= cn^{\log_b a - \epsilon}(b^{\epsilon \log_b n} - 1)/(b^\epsilon - 1) \\ &= cn^{\log_b a - \epsilon}(n^\epsilon - 1)/(b^\epsilon - 1) \\ &= \frac{c}{b^\epsilon - 1} n^{\log_b a}(n^\epsilon - 1)/(n^\epsilon) \end{aligned}$$

$b^{-i(\log_b a - \epsilon)} = b^{\epsilon i}(b^{\log_b a})^{-i} = b^{\epsilon i} a^{-i}$

$\sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1}$

Hence,

$$T(n) \le \left(\frac{c}{b^\epsilon - 1} + 1\right) n^{\log_b (a)}$$

$\Rightarrow T(n) = \mathcal{O}(n^{\log_b a})$.

Case 2. Now suppose that $f(n) \le cn^{\log_b a}$.

$$\begin{aligned} T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\ &\le c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\ &= cn^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\ &= cn^{\log_b a} \log_b n \end{aligned}$$

Hence,

$$T(n) = \mathcal{O}(n^{\log_b a} \log_b n)$$

$\Rightarrow T(n) = \mathcal{O}(n^{\log_b a} \log n)$.

Case 2. Now suppose that $f(n) \ge cn^{\log_b a}$.

$$\begin{aligned} T(n) - n^{\log_b a} &= \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) \\ &\ge c \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\ &= cn^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \\ &= cn^{\log_b a} \log_b n \end{aligned}$$

Hence,

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

$\Rightarrow T(n) = \Omega(n^{\log_b a} \log n)$.

Case 2. Now suppose that $f(n) \leq cn^{\log_b a}(\log_b(n))^k$.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq c \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \cdot \left(\log_b\left(\frac{n}{b^i}\right)\right)^k$$

$n = b^\ell \Rightarrow \ell = \log_b n$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} \left(\log_b\left(\frac{b^\ell}{b^i}\right)\right)^k$$

$$= cn^{\log_b a} \sum_{i=0}^{\ell-1} (\ell - i)^k$$

$$= cn^{\log_b a} \sum_{i=1}^{\ell} i^k \approx \frac{1}{k}\ell^{k+1}$$

$$\approx \frac{c}{k} n^{\log_b a} \ell^{k+1}$$

$\Rightarrow T(n) = \mathcal{O}(n^{\log_b a} \log^{k+1} n)$.

Case 3. Now suppose that $f(n) \geq dn^{\log_b a + \epsilon}$, and that for sufficiently large n: $af(n/b) \leq cf(n)$, for $c < 1$.

From this we get $a^i f(n/b^i) \leq c^i f(n)$, where we assume that $n/b^{i-1} \geq n_0$ is still sufficiently large.

$$T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

$$\leq \sum_{i=0}^{\log_b n-1} c^i f(n) + \mathcal{O}(n^{\log_b a})$$

$q < 1 : \sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q} \leq \frac{1}{1-q}$

$$\leq \frac{1}{1-c} f(n) + \mathcal{O}(n^{\log_b a})$$

Hence,

$$T(n) \leq \mathcal{O}(f(n))$$

$\Rightarrow T(n) = \Theta(f(n))$.

Where did we use $f(n) \geq \Omega(n^{\log_b a + \epsilon})$?

Example: Multiplying Two Integers

Suppose we want to multiply two n-bit Integers, but our registers can only perform operations on integers of constant size.

For this we first need to be able to add two integers A and B:

	1	1	0	1	1	0	1	0	1	A
	1	0	0	0	1	0	0	1	1	B
1	0	0	1	1	0	1	1	1		(carries)
1	0	1	1	0	0	1	0	0	0	

This gives that two n-bit integers can be added in time $\mathcal{O}(n)$.

Example: Multiplying Two Integers

Suppose that we want to multiply an n-bit integer A and an m-bit integer B ($m \leq n$).

```
1 0 0 0 1 × 1 0 1 1
-------------------
        1 0 0 0 1
      1 0 0 0 1 0
    0 0 0 0 0 0 0
  1 0 0 0 1 0 0 0
-------------------
  1 0 1 1 1 0 1 1
```

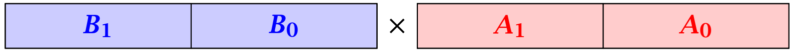
- This is also nown as the "school method" for multiplying integers.
- Note that the intermediate numbers that are generated can have at most $m + n \leq 2n$ bits.

Time requirement:

- Computing intermediate results: $\mathcal{O}(nm)$.
- Adding m numbers of length $\leq 2n$: $\mathcal{O}((m+n)m) = \mathcal{O}(nm)$.

Example: Multiplying Two Integers

A recursive approach:
Suppose that integers A and B are of length $n = 2^k$, for some k.



Then it holds that

$$A = A_1 \cdot 2^{\frac{n}{2}} + A_0 \text{ and } B = B_1 \cdot 2^{\frac{n}{2}} + B_0$$

Hence,

$$A \cdot B = A_1 B_1 \cdot 2^n + (A_1 B_0 + A_0 B_1) \cdot 2^{\frac{n}{2}} + A_0 B_0$$

Example: Multiplying Two Integers

Algorithm 3 mult(A, B)

if $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
return $a_0 \cdot b_0$	$\mathcal{O}(1)$
split A into A_0 and A_1	$\mathcal{O}(n)$
split B into B_0 and B_1	$\mathcal{O}(n)$
$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
$Z_1 \leftarrow \text{mult}(A_1, B_0) + \text{mult}(A_0, B_1)$	$2T(\frac{n}{2}) + \mathcal{O}(n)$
$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
return $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

We get the following recurrence:

$$T(n) = 4T\left(\frac{n}{2}\right) + \mathcal{O}(n) \ .$$

Example: Multiplying Two Integers

Master Theorem: Recurrence: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $\quad T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $\quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\quad T(n) = \Theta(f(n))$

In our case $a = 4$, $b = 2$, and $f(n) = \Theta(n)$. Hence, we are in Case 1, since $n = \mathcal{O}(n^{2-\epsilon}) = \mathcal{O}(n^{\log_b a - \epsilon})$.

We get a running time of $\mathcal{O}(n^2)$ for our algorithm.

$\Longrightarrow$ Not better then the "school method".

Example: Multiplying Two Integers

We can use the following identity to compute Z_1:

$$\begin{aligned} Z_1 &= A_1 B_0 + A_0 B_1 \\ &= (A_0 + A_1) \cdot (B_0 + B_1) - \underbrace{A_1 B_1}_{= Z_2} - \underbrace{A_0 B_0}_{= Z_0} \end{aligned}$$

Hence,

Algorithm 4 mult(A, B)

if $\lvert A\rvert = \lvert B\rvert = 1$ **then**	$\mathcal{O}(1)$
return $a_0 \cdot b_0$	$\mathcal{O}(1)$
split A into A_0 and A_1	$\mathcal{O}(n)$
split B into B_0 and B_1	$\mathcal{O}(n)$
$Z_2 \leftarrow \text{mult}(A_1, B_1)$	$T(\frac{n}{2})$
$Z_0 \leftarrow \text{mult}(A_0, B_0)$	$T(\frac{n}{2})$
$Z_1 \leftarrow \text{mult}(A_0 + A_1, B_0 + B_1) - Z_2 - Z_0$	$T(\frac{n}{2}) + \mathcal{O}(n)$
return $Z_2 \cdot 2^n + Z_1 \cdot 2^{\frac{n}{2}} + Z_0$	$\mathcal{O}(n)$

A more precise (correct) analysis would say that computing Z_1 needs time $T(\frac{n}{2} + 1) + \mathcal{O}(n)$.

Example: Multiplying Two Integers

We get the following recurrence:

$$T(n) = 3T\Big(\frac{n}{2}\Big) + \mathcal{O}(n) \ .$$

Master Theorem: Recurrence: $T[n] = aT(\frac{n}{b}) + f(n)$.

- Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ $T(n) = \Theta(n^{\log_b a})$
- Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$ $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ $T(n) = \Theta(f(n))$

Again we are in Case 1. We get a running time of $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$.

A huge improvement over the "school method".

6.3 The Characteristic Polynomial

Consider the recurrence relation:

$$c_0T(n) + c_1T(n-1) + c_2T(n-2) + \cdots + c_kT(n-k) = f(n)$$

This is the general form of a linear recurrence relation of order k with constant coefficients ($c_0, c_k \neq 0$).

- $T(n)$ only depends on the k preceding values. This means the recurrence relation is of order k.
- The recurrence is linear as there are no products of $T[n]$'s.
- If $f(n) = 0$ then the recurrence relation becomes a linear, homogenous recurrence relation of order k.

Note that we ignore boundary conditions for the moment.

6.3 The Characteristic Polynomial

Observations:

- The solution $T[1], T[2], T[3], \ldots$ is completely determined by a set of boundary conditions that specify values for $T[1], \ldots, T[k]$.
- In fact, any k consecutive values completely determine the solution.
- k non-concecutive values might not be an appropriate set of boundary conditions (depends on the problem).

Approach:

- First determine all solutions that satisfy recurrence relation.
- Then pick the right one by analyzing boundary conditions.
- First consider the homogenous case.

The Homogenous Case

The solution space

$$S = \Big\{\mathcal{T} = T[1], T[2], T[3], \ldots \ \Big|\ \mathcal{T} \text{ fulfills recurrence relation}\Big\}$$

is a vector space. This means that if $\mathcal{T}_1, \mathcal{T}_2 \in S$, then also $\alpha\mathcal{T}_1 + \beta\mathcal{T}_2 \in S$, for arbitrary constants α, β.

How do we find a non-trivial solution?

We guess that the solution is of the form λ^n, $\lambda \neq 0$, and see what happens. In order for this guess to fulfill the recurrence we need

$$c_0\lambda^n + c_1\lambda^{n-1} + c_2 \cdot \lambda^{n-2} + \cdots + c_k \cdot \lambda^{n-k} = 0$$

for all $n \geq k$.

The Homogenous Case

Dividing by λ^{n-k} gives that all these constraints are identical to

$$\underbrace{c_0\lambda^k + c_1\lambda^{k-1} + c_2 \cdot \lambda^{k-2} + \cdots + c_k}_{\text{characteristic polynomial } P[\lambda]} = 0$$

This means that if λ_i is a root (Nullstelle) of $P[\lambda]$ then $T[n] = \lambda_i^n$ is a solution to the recurrence relation.

Let $\lambda_1, \ldots, \lambda_k$ be the k (complex) roots of $P[\lambda]$. Then, because of the vector space property

$$\alpha_1\lambda_1^n + \alpha_2\lambda_2^n + \cdots + \alpha_k\lambda_k^n$$

is a solution for arbitrary values α_i.

The Homogenous Case

Lemma 2

Assume that the characteristic polynomial has k distinct roots $\lambda_1, \ldots, \lambda_k$. Then all solutions to the recurrence relation are of the form

$$\alpha_1\lambda_1^n + \alpha_2\lambda_2^n + \cdots + \alpha_k\lambda_k^n .$$

Proof.

There is one solution for every possible choice of boundary conditions for $T[1], \ldots, T[k]$.

We show that the above set of solutions contains one solution for every choice of boundary conditions.

The Homogenous Case

Proof (cont.).

Suppose I am given boundary conditions $T[i]$ and I want to see whether I can choose the $\alpha_i's$ such that these conditions are met:

$$\begin{array}{ccccccccc}
\alpha_1 \cdot \lambda_1 & + & \alpha_2 \cdot \lambda_2 & + & \cdots & + & \alpha_k \cdot \lambda_k & = & T[1] \\
\alpha_1 \cdot \lambda_1^2 & + & \alpha_2 \cdot \lambda_2^2 & + & \cdots & + & \alpha_k \cdot \lambda_k^2 & = & T[2] \\
 & & & & \vdots & & & & \\
\alpha_1 \cdot \lambda_1^k & + & \alpha_2 \cdot \lambda_2^k & + & \cdots & + & \alpha_k \cdot \lambda_k^k & = & T[k]
\end{array}$$

The Homogenous Case

Proof (cont.).

Suppose I am given boundary conditions $T[i]$ and I want to see whether I can choose the $\alpha_i's$ such that these conditions are met:

$$\begin{pmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_k \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_k^2 \\ & & \vdots & \\ \lambda_1^k & \lambda_2^k & \cdots & \lambda_k^k \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \end{pmatrix} = \begin{pmatrix} T[1] \\ T[2] \\ \vdots \\ T[k] \end{pmatrix}$$

We show that the column vectors are linearly independent. Then the above equation has a solution.

Computing the Determinant

$$\begin{vmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_{k-1} & \lambda_k \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_{k-1}^2 & \lambda_k^2 \\ \vdots & \vdots & & \vdots & \vdots \\ \lambda_1^k & \lambda_2^k & \cdots & \lambda_{k-1}^k & \lambda_k^k \end{vmatrix} = \prod_{i=1}^{k} \lambda_i \cdot \begin{vmatrix} 1 & 1 & \cdots & 1 & 1 \\ \lambda_1 & \lambda_2 & \cdots & \lambda_{k-1} & \lambda_k \\ \vdots & \vdots & & \vdots & \vdots \\ \lambda_1^{k-1} & \lambda_2^{k-1} & \cdots & \lambda_{k-1}^{k-1} & \lambda_k^{k-1} \end{vmatrix}$$

$$= \prod_{i=1}^{k} \lambda_i \cdot \begin{vmatrix} 1 & \lambda_1 & \cdots & \lambda_1^{k-2} & \lambda_1^{k-1} \\ 1 & \lambda_2 & \cdots & \lambda_2^{k-2} & \lambda_2^{k-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \lambda_k & \cdots & \lambda_k^{k-2} & \lambda_k^{k-1} \end{vmatrix}$$

Computing the Determinant

$$\begin{vmatrix} 1 & \lambda_1 & \cdots & \lambda_1^{k-2} & \lambda_1^{k-1} \\ 1 & \lambda_2 & \cdots & \lambda_2^{k-2} & \lambda_2^{k-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \lambda_k & \cdots & \lambda_k^{k-2} & \lambda_k^{k-1} \end{vmatrix} =$$

$$\begin{vmatrix} 1 & \lambda_1 - \lambda_1 \cdot 1 & \cdots & \lambda_1^{k-2} - \lambda_1 \cdot \lambda_1^{k-3} & \lambda_1^{k-1} - \lambda_1 \cdot \lambda_1^{k-2} \\ 1 & \lambda_2 - \lambda_1 \cdot 1 & \cdots & \lambda_2^{k-2} - \lambda_1 \cdot \lambda_2^{k-3} & \lambda_2^{k-1} - \lambda_1 \cdot \lambda_2^{k-2} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \lambda_k - \lambda_1 \cdot 1 & \cdots & \lambda_k^{k-2} - \lambda_1 \cdot \lambda_k^{k-3} & \lambda_k^{k-1} - \lambda_1 \cdot \lambda_k^{k-2} \end{vmatrix}$$

Computing the Determinant

$$\begin{vmatrix} 1 & \lambda_1 - \lambda_1 \cdot 1 & \cdots & \lambda_1^{k-2} - \lambda_1 \cdot \lambda_1^{k-3} & \lambda_1^{k-1} - \lambda_1 \cdot \lambda_1^{k-2} \\ 1 & \lambda_2 - \lambda_1 \cdot 1 & \cdots & \lambda_2^{k-2} - \lambda_1 \cdot \lambda_2^{k-3} & \lambda_2^{k-1} - \lambda_1 \cdot \lambda_2^{k-2} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \lambda_k - \lambda_1 \cdot 1 & \cdots & \lambda_k^{k-2} - \lambda_1 \cdot \lambda_k^{k-3} & \lambda_k^{k-1} - \lambda_1 \cdot \lambda_k^{k-2} \end{vmatrix} =$$

$$\begin{vmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & (\lambda_2 - \lambda_1) \cdot 1 & \cdots & (\lambda_2 - \lambda_1) \cdot \lambda_2^{k-3} & (\lambda_2 - \lambda_1) \cdot \lambda_2^{k-2} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & (\lambda_k - \lambda_1) \cdot 1 & \cdots & (\lambda_k - \lambda_1) \cdot \lambda_k^{k-3} & (\lambda_k - \lambda_1) \cdot \lambda_k^{k-2} \end{vmatrix}$$

Computing the Determinant

$$\begin{vmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & (\lambda_2 - \lambda_1) \cdot 1 & \cdots & (\lambda_2 - \lambda_1) \cdot \lambda_2^{k-3} & (\lambda_2 - \lambda_1) \cdot \lambda_2^{k-2} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & (\lambda_k - \lambda_1) \cdot 1 & \cdots & (\lambda_k - \lambda_1) \cdot \lambda_k^{k-3} & (\lambda_k - \lambda_1) \cdot \lambda_k^{k-2} \end{vmatrix} =$$

$$\prod_{i=2}^{k} (\lambda_i - \lambda_1) \cdot \begin{vmatrix} 1 & \lambda_2 & \cdots & \lambda_2^{k-3} & \lambda_2^{k-2} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \lambda_k & \cdots & \lambda_k^{k-3} & \lambda_k^{k-2} \end{vmatrix}$$

Computing the Determinant

Repeating the above steps gives:

$$\begin{vmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_{k-1} & \lambda_k \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_{k-1}^2 & \lambda_k^2 \\ \vdots & \vdots & & \vdots & \vdots \\ \lambda_1^k & \lambda_2^k & \cdots & \lambda_{k-1}^k & \lambda_k^k \end{vmatrix} = \prod_{i=1}^{k} \lambda_i \cdot \prod_{i>\ell} (\lambda_i - \lambda_\ell)$$

Hence, if all λ_i's are different, then the determinant is non-zero.

The Homogeneous Case

What happens if the roots are not all distinct?

Suppose we have a root λ_i with multiplicity (Vielfachheit) at least 2. Then not only is λ_i^n a solution to the recurrence but also $n\lambda_i^n$.

To see this consider the polynomial

$$P[\lambda] \cdot \lambda^{n-k} = c_0\lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \cdots + c_k\lambda^{n-k}$$

Since λ_i is a root we can write this as $Q[\lambda] \cdot (\lambda - \lambda_i)^2$. Calculating the derivative gives a polynomial that still has root λ_i.

This means

$$c_0 n\lambda_i^{n-1} + c_1(n-1)\lambda_i^{n-2} + \cdots + c_k(n-k)\lambda_i^{n-k-1} = 0$$

Hence,

$$\underbrace{c_0 n\lambda_i^n}_{T[n]} + \underbrace{c_1(n-1)\lambda_i^{n-1}}_{T[n-1]} + \cdots + \underbrace{c_k(n-k)\lambda_i^{n-k}}_{T[n-k]} = 0$$

The Homogeneous Case

Suppose λ_i has multiplicity j. We know that

$$c_0 n\lambda_i^n + c_1(n-1)\lambda_i^{n-1} + \cdots + c_k(n-k)\lambda_i^{n-k} = 0$$

(after taking the derivative; multiplying with λ; plugging in λ_i)

Doing this again gives

$$c_0 n^2\lambda_i^n + c_1(n-1)^2\lambda_i^{n-1} + \cdots + c_k(n-k)^2\lambda_i^{n-k} = 0$$

We can continue $j-1$ times.

Hence, $n^\ell\lambda_i^n$ is a solution for $\ell \in 0, \ldots, j-1$.

The Homogeneous Case

Lemma 3
Let $P[\lambda]$ denote the characteristic polynomial to the recurrence

$$c_0 T[n] + c_1 T[n-1] + \cdots + c_k T[n-k] = 0$$

Let λ_i, $i = 1, \ldots, m$ be the (complex) roots of $P[\lambda]$ with multiplicities ℓ_i. Then the general solution to the recurrence is given by

$$T[n] = \sum_{i=1}^{m} \sum_{j=0}^{\ell_i - 1} \alpha_{ij} \cdot (n^j \lambda_i^n) \ .$$

The full proof is omitted. We have only shown that any choice of α_{ij}'s is a solution to the recurrence.

Example: Fibonacci Sequence

$$T[0] = 0$$
$$T[1] = 1$$
$$T[n] = T[n-1] + T[n-2] \text{ for } n \geq 2$$

The characteristic polynomial is

$$\lambda^2 - \lambda - 1$$

Finding the roots, gives

$$\lambda_{1/2} = \frac{1}{2} \pm \sqrt{\frac{1}{4} + 1} = \frac{1}{2}\left(1 \pm \sqrt{5}\right)$$

Example: Fibonacci Sequence

Hence, the solution is of the form

$$\alpha \left(\frac{1+\sqrt{5}}{2}\right)^n + \beta \left(\frac{1-\sqrt{5}}{2}\right)^n$$

$T[0] = 0$ gives $\alpha + \beta = 0$.

$T[1] = 1$ gives

$$\alpha \left(\frac{1+\sqrt{5}}{2}\right) + \beta \left(\frac{1-\sqrt{5}}{2}\right) = 1 \Longrightarrow \alpha - \beta = \frac{2}{\sqrt{5}}$$

Example: Fibonacci Sequence

Hence, the solution is

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

The Inhomogeneous Case

Consider the recurrence relation:

$$c_0 T(n) + c_1 T(n-1) + c_2 T(n-2) + \cdots + c_k T(n-k) = f(n)$$

with $f(n) \neq 0$.

While we have a fairly general technique for solving homogeneous, linear recurrence relations the inhomogeneous case is different.

The Inhomogeneous Case

The general solution of the recurrence relation is

$$T(n) = T_h(n) + T_p(n) \ ,$$

where T_h is any solution to the homogeneous equation, and T_p is one particular solution to the inhomogeneous equation.

There is no general method to find a particular solution.

The Inhomogeneous Case

Example:

$$T[n] = T[n-1] + 1 \qquad T[0] = 1$$

Then,

$$T[n-1] = T[n-2] + 1 \qquad (n \geq 2)$$

Subtracting the first from the second equation gives,

$$T[n] - T[n-1] = T[n-1] - T[n-2] \qquad (n \geq 2)$$

or

$$T[n] = 2T[n-1] - T[n-2] \qquad (n \geq 2)$$

I get a completely determined recurrence if I add $T[0] = 1$ and $T[1] = 2$.

The Inhomogeneous Case

Example: Characteristic polynomial:

$$\underbrace{\lambda^2 - 2\lambda + 1}_{(\lambda-1)^2} = 0$$

Then the solution is of the form

$$T[n] = \alpha 1^n + \beta n 1^n = \alpha + \beta n$$

$T[0] = 1$ gives $\alpha = 1$.

$T[1] = 2$ gives $1 + \beta = 2 \Longrightarrow \beta = 1$.

The Inhomogeneous Case

If $f(n)$ is a polynomial of degree r this method can be applied $r+1$ times to obtain a homogeneous equation:

$$T[n] = T[n-1] + n^2$$

Shift:

$$T[n-1] = T[n-2] + (n-1)^2 = T[n-2] + n^2 - 2n + 1$$

Difference:

$$T[n] - T[n-1] = T[n-1] - T[n-2] + 2n - 1$$

$$T[n] = 2T[n-1] - T[n-2] + 2n - 1$$

$$T[n] = 2T[n-1] - T[n-2] + 2n - 1$$

Shift:

$$\begin{aligned} T[n-1] &= 2T[n-2] - T[n-3] + 2(n-1) - 1 \\ &= 2T[n-2] - T[n-3] + 2n - 3 \end{aligned}$$

Difference:

$$\begin{aligned} T[n] - T[n-1] =& 2T[n-1] - T[n-2] + 2n - 1 \\ & - 2T[n-2] + T[n-3] - 2n + 3 \end{aligned}$$

$$T[n] = 3T[n-1] - 3T[n-2] + T[n-3] + 2$$

and so on...

6.4 Generating Functions

Definition 4 (Generating Function)

Let $(a_n)_{n\geq 0}$ be a sequence. The corresponding

- generating function (Erzeugendenfunktion) is

$$F(z) := \sum_{n\geq 0} a_n z^n;$$

- exponential generating function (exponentielle Erzeugendenfunktion) is

$$F(z) = \sum_{n\geq 0} \frac{a_n}{n!} z^n.$$

6.4 Generating Functions

Example 5

1. The generating function of the sequence $(1, 0, 0, \ldots)$ is

$$F(z) = 1.$$

2. The generating function of the sequence $(1, 1, 1, \ldots)$ is

$$F(z) = \frac{1}{1-z}.$$

6.4 Generating Functions

There are two different views:

A generating function is a formal power series (formale Potenzreihe).

Then the generating function is an algebraic object.

Let $f = \sum_{n\geq 0} a_n z^n$ and $g = \sum_{n\geq 0} b_n z^n$.

- **Equality:** f and g are equal if $a_n = b_n$ for all n.
- **Addition:** $f + g := \sum_{n\geq 0}(a_n + b_n)z^n$.
- **Multiplication:** $f \cdot g := \sum_{n\geq 0} c_n z^n$ with $c_n = \sum_{p=0}^{n} a_p b_{n-p}$.

There are no convergence issues here.

6.4 Generating Functions

The arithmetic view:

We view a power series as a function $f : \mathbb{C} \to \mathbb{C}$.

Then, it is important to think about convergence/convergence radius etc.

6.4 Generating Functions

What does $\sum_{n\geq 0} z^n = \frac{1}{1-z}$ mean in the algebraic view?

It means that the power series $1 - z$ and the power series $\sum_{n\geq 0} z^n$ are invers, i.e.,

$$\left(1 - z\right) \cdot \left(\sum_{n\geq 0}^{\infty} z^n\right) = 1 \ .$$

This is well-defined.

6.4 Generating Functions

Suppose we are given the generating function

$$\sum_{n\geq 0} z^n = \frac{1}{1-z} \ .$$

We can compute the derivative:

$$\underbrace{\sum_{n\geq 1} nz^{n-1}}_{\sum_{n\geq 0}(n+1)z^n} = \frac{1}{(1-z)^2}$$

Hence, the generating function of the sequence $a_n = n + 1$ is $1/(1-z)^2$.

Formally the derivative of a formal power series $\sum_{n\geq 0} a_n z^n$ is defined as $\sum_{n\geq 0} n a_n z^{n-1}$.

The known rules for differentiation work for this definition. In particular, e.g. the derivative of $\frac{1}{1-z}$ is $\frac{1}{(1-z)^2}$.

Note that this requires a proof if we consider power series as algebraic objects. However, we did not prove this in the lecture.

6.4 Generating Functions

We can repeat this

$$\sum_{n\geq 0}(n+1)z^n = \frac{1}{(1-z)^2} \enspace .$$

Derivative:

$$\underbrace{\sum_{n\geq 1} n(n+1)z^{n-1}}_{\sum_{n\geq 0}(n+1)(n+2)z^n} = \frac{2}{(1-z)^3}$$

Hence, the generating function of the sequence $a_n = (n+1)(n+2)$ is $\frac{2}{(1-z)^3}$.

6.4 Generating Functions

Computing the k-th derivative of $\sum z^n$.

$$\begin{aligned}\sum_{n\geq k} n(n-1)\cdot\ldots\cdot(n-k+1)z^{n-k} &= \sum_{n\geq 0}(n+k)\cdot\ldots\cdot(n+1)z^n \\ &= \frac{k!}{(1-z)^{k+1}} \enspace .\end{aligned}$$

Hence:

$$\sum_{n\geq 0}\binom{n+k}{k}z^n = \frac{1}{(1-z)^{k+1}} \enspace .$$

The generating function of the sequence $a_n = \binom{n+k}{k}$ is $\frac{1}{(1-z)^{k+1}}$.

6.4 Generating Functions

$$\begin{aligned}\sum_{n\geq 0} nz^n &= \sum_{n\geq 0}(n+1)z^n - \sum_{n\geq 0} z^n \\ &= \frac{1}{(1-z)^2} - \frac{1}{1-z} \\ &= \frac{z}{(1-z)^2}\end{aligned}$$

The generating function of the sequence $a_n = n$ is $\frac{z}{(1-z)^2}$.

6.4 Generating Functions

We know

$$\sum_{n\geq 0} y^n = \frac{1}{1-y}$$

Hence,

$$\sum_{n\geq 0} a^n z^n = \frac{1}{1-az}$$

The generating function of the sequence $f_n = a^n$ is $\frac{1}{1-az}$.

Example: $a_n = a_{n-1} + 1$, $a_0 = 1$

Suppose we have the recurrence $a_n = a_{n-1} + 1$ for $n \geq 1$ and $a_0 = 1$.

$$\begin{aligned} A(z) &= \sum_{n\geq 0} a_n z^n \\ &= a_0 + \sum_{n\geq 1} (a_{n-1} + 1) z^n \\ &= 1 + z \sum_{n\geq 1} a_{n-1} z^{n-1} + \sum_{n\geq 1} z^n \\ &= z \sum_{n\geq 0} a_n z^n + \sum_{n\geq 0} z^n \\ &= zA(z) + \sum_{n\geq 0} z^n \\ &= zA(z) + \frac{1}{1-z} \end{aligned}$$

Example: $a_n = a_{n-1} + 1$, $a_0 = 1$

Solving for $A(z)$ gives

$$\sum_{n\geq 0} a_n z^n = A(z) = \frac{1}{(1-z)^2} = \sum_{n\geq 0} (n+1) z^n$$

Hence, $a_n = n + 1$.

Some Generating Functions

n-th sequence element	*generating function*
1	$\frac{1}{1-z}$
$n+1$	$\frac{1}{(1-z)^2}$
$\binom{n+k}{k}$	$\frac{1}{(1-z)^{k+1}}$
n	$\frac{z}{(1-z)^2}$
a^n	$\frac{1}{1-az}$
n^2	$\frac{z(1+z)}{(1-z)^3}$
$\frac{1}{n!}$	e^z

Some Generating Functions

n-th sequence element	*generating function*
cf_n	cF
$f_n + g_n$	$F + G$
$\sum_{i=0}^{n} f_i g_{n-i}$	$F \cdot G$
f_{n-k} $(n \geq k)$; 0 otw.	$z^k F$
$\sum_{i=0}^{n} f_i$	$\frac{F(z)}{1-z}$
nf_n	$z\frac{\mathrm{d}F(z)}{\mathrm{d}z}$
$c^n f_n$	$F(cz)$

Solving Recursions with Generating Functions

1. Set $A(z) = \sum_{n\geq 0} a_n z^n$.
2. Transform the right hand side so that boundary condition and recurrence relation can be plugged in.
3. Do further transformations so that the infinite sums on the right hand side can be replaced by $A(z)$.
4. Solving for $A(z)$ gives an equation of the form $A(z) = f(z)$, where hopefully $f(z)$ is a simple function.
5. Write $f(z)$ as a formal power series.
   Techniques:
   - partial fraction decomposition (Partialbruchzerlegung)
   - lookup in tables
6. The coefficients of the resulting power series are the a_n.

Example: $a_n = 2a_{n-1}$, $a_0 = 1$

1. Set up generating function:

$$A(z) = \sum_{n\geq 0} a_n z^n$$

2. Transform right hand side so that recurrence can be plugged in:

$$A(z) = a_0 + \sum_{n\geq 1} a_n z^n$$

2. Plug in:

$$A(z) = 1 + \sum_{n\geq 1} (2a_{n-1}) z^n$$

Example: $a_n = 2a_{n-1}$, $a_0 = 1$

3. Transform right hand side so that infinite sums can be replaced by $A(z)$ or by simple function.

$$\begin{aligned} A(z) &= 1 + \sum_{n\geq 1} (2a_{n-1}) z^n \\ &= 1 + 2z \sum_{n\geq 1} a_{n-1} z^{n-1} \\ &= 1 + 2z \sum_{n\geq 0} a_n z^n \\ &= 1 + 2z \cdot A(z) \end{aligned}$$

4. Solve for $A(z)$.

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

Example: $a_n = 2a_{n-1}$, $a_0 = 1$

5. Rewrite $f(z)$ as a power series:

$$\sum_{n\geq 0} a_n z^n = A(z) = \frac{1}{1-2z} = \sum_{n\geq 0} 2^n z^n$$

Example: $a_n = 3a_{n-1} + n$, $a_0 = 1$

1. Set up generating function:

$$A(z) = \sum_{n\geq 0} a_n z^n$$

Example: $a_n = 3a_{n-1} + n$, $a_0 = 1$

2./3. Transform right hand side:

$$\begin{aligned} A(z) &= \sum_{n\geq 0} a_n z^n \\ &= a_0 + \sum_{n\geq 1} a_n z^n \\ &= 1 + \sum_{n\geq 1} (3a_{n-1} + n) z^n \\ &= 1 + 3z \sum_{n\geq 1} a_{n-1} z^{n-1} + \sum_{n\geq 1} n z^n \\ &= 1 + 3z \sum_{n\geq 0} a_n z^n + \sum_{n\geq 0} n z^n \\ &= 1 + 3zA(z) + \frac{z}{(1-z)^2} \end{aligned}$$

Example: $a_n = 3a_{n-1} + n$, $a_0 = 1$

4. Solve for $A(z)$:

$$A(z) = 1 + 3zA(z) + \frac{z}{(1-z)^2}$$

gives

$$A(z) = \frac{(1-z)^2 + z}{(1-3z)(1-z)^2} = \frac{z^2 - z + 1}{(1-3z)(1-z)^2}$$

Example: $a_n = 3a_{n-1} + n$, $a_0 = 1$

5. Write $f(z)$ as a formal power series:

We use partial fraction decomposition:

$$\frac{z^2 - z + 1}{(1-3z)(1-z)^2} \stackrel{!}{=} \frac{A}{1-3z} + \frac{B}{1-z} + \frac{C}{(1-z)^2}$$

This gives

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

Example: $a_n = 3a_{n-1} + n$, $a_0 = 1$

5. Write $f(z)$ as a formal power series:

   This leads to the following conditions:

$$A + B + C = 1$$
$$2A + 4B + 3C = 1$$
$$A + 3B = 1$$

   which gives

$$A = \frac{7}{4} \quad B = -\frac{1}{4} \quad C = -\frac{1}{2}$$

Example: $a_n = 3a_{n-1} + n$, $a_0 = 1$

5. Write $f(z)$ as a formal power series:

$$\begin{aligned} A(z) &= \frac{7}{4}\cdot\frac{1}{1-3z} - \frac{1}{4}\cdot\frac{1}{1-z} - \frac{1}{2}\cdot\frac{1}{(1-z)^2} \\ &= \frac{7}{4}\cdot\sum_{n\ge 0} 3^n z^n - \frac{1}{4}\cdot\sum_{n\ge 0} z^n - \frac{1}{2}\cdot\sum_{n\ge 0}(n+1)z^n \\ &= \sum_{n\ge 0}\Big(\frac{7}{4}\cdot 3^n - \frac{1}{4} - \frac{1}{2}(n+1)\Big)z^n \\ &= \sum_{n\ge 0}\Big(\frac{7}{4}\cdot 3^n - \frac{1}{2}n - \frac{3}{4}\Big)z^n \end{aligned}$$

6. This means $a_n = \frac{7}{4}3^n - \frac{1}{2}n - \frac{3}{4}$.

6.5 Transformation of the Recurrence

Example 6

$$f_0 = 1$$
$$f_1 = 2$$
$$f_n = f_{n-1}\cdot f_{n-2} \text{ for } n \ge 2 .$$

Define

$$g_n := \log f_n .$$

Then

$$g_n = g_{n-1} + g_{n-2} \text{ for } n \ge 2$$
$$g_1 = \log 2 = 1 (\text{for } \log = \log_2),\ g_0 = 0$$
$$g_n = F_n \ (n\text{-th Fibonacci number})$$
$$f_n = 2^{F_n}$$

6.5 Transformation of the Recurrence

Example 7

$$f_1 = 1$$
$$f_n = 3f_{\frac{n}{2}} + n;\ \text{for } n = 2^k,\ k \ge 1\ ;$$

Define

$$g_k := f_{2^k} .$$

Then:

$$g_0 = 1$$
$$g_k = 3g_{k-1} + 2^k,\ k \ge 1$$

6 Recurrences

We get

$$\begin{aligned} g_k &= 3\left[g_{k-1}\right] + 2^k \\ &= 3\left[3g_{k-2} + 2^{k-1}\right] + 2^k \\ &= 3^2\left[g_{k-2}\right] + 32^{k-1} + 2^k \\ &= 3^2\left[3g_{k-3} + 2^{k-2}\right] + 32^{k-1} + 2^k \\ &= 3^3 g_{k-3} + 3^2 2^{k-2} + 32^{k-1} + 2^k \\ &= 2^k \cdot \sum_{i=0}^{k} \left(\frac{3}{2}\right)^i \\ &= 2^k \cdot \frac{(\frac{3}{2})^{k+1} - 1}{1/2} = 3^{k+1} - 2^{k+1} \end{aligned}$$

6 Recurrences

Let $n = 2^k$:

$$\begin{aligned} g_k &= 3^{k+1} - 2^{k+1}, \text{ hence} \\ f_n &= 3 \cdot 3^k - 2 \cdot 2^k \\ &= 3(2^{\log 3})^k - 2 \cdot 2^k \\ &= 3(2^k)^{\log 3} - 2 \cdot 2^k \\ &= 3n^{\log 3} - 2n\ . \end{aligned}$$

6 Recurrences

Bibliography

[MS08] Kurt Mehlhorn, Peter Sanders:
Algorithms and Data Structures — The Basic Toolbox,
Springer, 2008

[CLRS90] Thomas H. Cormen, Charles E. Leiserson, Ron L. Rivest, Clifford Stein:
Introduction to algorithms (3rd ed.),
MIT Press and McGraw-Hill, 2009

[Liu85] Chung Laung Liu:
Elements of Discrete Mathematics
McGraw-Hill, 1985

The Karatsuba method can be found in [MS08] Chapter 1. Chapter 4.3 of [CLRS90] covers the "Substitution method" which roughly corresponds to "Guessing+induction". Chapters 4.4, 4.5, 4.6 of this book cover the master theorem. Methods using the characteristic polynomial and generating functions can be found in [Liu85] Chapter 10.