6 Recurrences

**Algorithm 2** mergesort(list $L$)

1: $n \leftarrow \text{size}(L)$
2: if $n \leq 1$ return $L$
3: $L_1 \leftarrow L[1 \cdots \lceil n/2 \rceil]$
4: $L_2 \leftarrow L[\lfloor n/2 \rfloor + 1 \cdots n]$
5: mergesort($L_1$)
6: mergesort($L_2$)
7: $L \leftarrow \text{merge}(L_1, L_2)$
8: return $L$

This algorithm requires

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

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

---

**Recurrences**

How do we bring the expression for the number of comparisons (= 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.

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.
First we need to get rid of the $\Theta$-notation in our recurrence:

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

Assume that instead we had

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

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

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

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

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

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

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

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.

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 $T(n) \leq dn \log n$ and get

$$T(n) \leq 2T\left(\frac{n}{2}\right) + cn$$

$$\leq 2\left(\frac{n}{2}\right) \log \left(\frac{n}{2}\right) + cn$$

$$\leq 2(d(n/2 + 1) \log(n/2 + 1)) + cn$$

$$\leq dn \log \left(\frac{n}{2}\right) + 2d \log n + cn$$

$$\leq dn \log n + (\log 9 - 4)dn + 2d \log n + cn$$

$$\leq dn \log n + (\log 9 - 3.5)dn + cn$$

$$\leq dn \log n - 0.33dn + cn$$

$$\leq dn \log n$$

for a suitable choice of $d$.

6.2 Master Theorem

**Lemma 1**

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

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

**Case 1.**

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

**Case 2.**

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

**Case 3.**

If $f(n) = \Omega(n \log_b(a) + \epsilon)$ and for sufficiently large $n$

$$af\left(\frac{n}{b}\right) \leq cf(n)$$

for some constant $c < 1$ then $T(n) = \Theta(f(n))$.

The Recursion Tree

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

- $f(n)$
- $af\left(\frac{n}{b}\right)$
- $a^2f\left(\frac{n}{b^2}\right)$
- $\ldots$
- $a^{\log_b n}$
- $n \log_b a$
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 \( f(n) \leq cn^{\log_b a - \epsilon} \).

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left( \frac{n}{b^i} \right) \\
\leq c \sum_{i=0}^{\log_b n-1} a^i \left( \frac{n}{b^i} \right)^{\log_b a} \\
= c n^{\log_b a} \sum_{i=0}^{\log_b n-1} \left( \frac{b^i}{n} \right)^{\log_b a} \\
= c n^{\log_b a} \sum_{i=0}^{\log_b n-1} \left( \frac{b^i}{n} \right)^{\log_b n} \\
= c \left( \frac{b^{i+1} - 1}{b^i - 1} \right) \left( \frac{n^{\log_b a} - 1}{n^{\log_b a}} \right) \\
\Rightarrow T(n) = \Theta(n^{\log_b a}).
\]

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

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left( \frac{n}{b^i} \right) \\
\leq c \sum_{i=0}^{\log_b n-1} a^i \left( \frac{n}{b^i} \right)^{\log_b a} \\
= c n^{\log_b a} \sum_{i=0}^{\log_b n-1} 1 \\
= c n^{\log_b a} \log_b n \\
\Rightarrow T(n) = \Theta(n^{\log_b a} \log_b n),
\]

Hence,

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

\[
T(n) - n^{\log_b a} = \sum_{i=0}^{\log_b n-1} a^i f\left( \frac{n}{b^i} \right) \\
\geq c \sum_{i=0}^{\log_b n-1} a^i \left( \frac{n}{b^i} \right)^{\log_b a} \\
= c n^{\log_b a} \sum_{i=0}^{\log_b n-1} 1 \\
= c n^{\log_b a} \log_b n \\
\Rightarrow T(n) = \Omega(n^{\log_b a} \log_b n),
\]

Hence,

\[
T(n) = \Omega(n^{\log_b a} \log_b n) \\
\Rightarrow T(n) = \Omega(n^{\log_b a} \log n),
\]
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 \( B \):

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0
\end{array}
\]

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

Example: Multiplying Two Integers

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

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & & & & & \\
1 & 0 & 1 & & & & & & & \\
0 & 0 & 0 & & & & & & & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1
\end{array}
\]

Time requirement:
- Computing intermediate results: \( O(nm) \).
- Adding \( m \) numbers of length \( \leq 2n \): \( O((m + n)m) = O(nm) \).
Example: Multiplying Two Integers

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

![Image of multiplication]

Then it holds that

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

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

**Master Theorem:** Recurrence: \( T[n] = aT\left(\frac{n}{2}\right) + f(n) \).

- Case 1: \( f(n) = \Theta(n^{\log_2 a - \epsilon}) \), \( T(n) = \Theta(n^{\log_2 a}) \)
- Case 2: \( f(n) = \Theta(n^{\log_2 a \log k n}) \), \( T(n) = \Theta(n^{\log_2 a \log k + 1}) \)
- Case 3: \( f(n) = \Omega(n^{\log_2 a + \epsilon}) \), \( T(n) = \Theta(f(n)) \)

In our case \( a = 4 \), \( b = 2 \), and \( f(n) = \Theta(n) \). Hence, we are in Case 1, since \( n = \Theta(n^{2 - \epsilon}) = \Theta(n^{\log_2 a - \epsilon}) \).

We get a running time of \( \Theta(n^2) \) for our algorithm.

\[ \Rightarrow \text{Not better than the "school method".} \]

Example: Multiplying Two Integers

**Algorithm 3** \( \text{mult}(A,B) \)

1. if \( |A| = |B| = 1 \) then \( \Theta(1) \)
2. \( \text{return} \ a_0 \cdot b_0 \)
3. split \( A \) into \( A_0 \) and \( A_1 \)
4. split \( B \) into \( B_0 \) and \( B_1 \)
5. \( Z_2 = \text{mult}(A_1,B_1) \)
6. \( Z_1 = \text{mult}(A_0,B_0) + \text{mult}(A_0,B_1) \)
7. \( Z_0 = \text{mult}(A_0,B_0) \)
8. \( \text{return} \ Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0 \)

We get the following recurrence:

\[ T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) \]

Example: Multiplying Two Integers

We can use the following identity to compute \( Z_1 \):

\[ Z_1 = A_1 B_0 + A_0 B_1 = Z_2 = Z_0 = (A_0 + A_1) \cdot (B_0 + B_1) - A_1 B_1 - A_0 B_0 \]

Hence,

**Algorithm 4** \( \text{mult}(A,B) \)

1. if \( |A| = |B| = 1 \) then \( \Theta(1) \)
2. \( \text{return} \ a_0 \cdot b_0 \)
3. split \( A \) into \( A_0 \) and \( A_1 \)
4. split \( B \) into \( B_0 \) and \( B_1 \)
5. \( Z_2 = \text{mult}(A_1,B_1) \)
6. \( Z_1 = \text{mult}(A_0,B_0) + \text{mult}(A_0,B_1) - Z_2 - Z_0 \)
7. \( Z_0 = \text{mult}(A_0,B_0) \)
8. \( \text{return} \ Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0 \)

\[ \Rightarrow \text{Not better then the "school method".} \]
Example: Multiplying Two Integers

We get the following recurrence:

\[ T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) . \]

**Master Theorem:**

**Recurrence:** \( T[n] = aT\left(\frac{n}{b}\right) + f(n) \).

- **Case 1:** \( f(n) = \Theta(n^{\log_b a - \epsilon}) \)  
  \( T(n) = \Theta(n^{\log_b a}) \)
- **Case 2:** \( f(n) = \Theta(n^{\log_b a \log_k n}) \)  
  \( T(n) = \Theta(n^{\log_b a \log_k n} + 1) \)
- **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

**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-consecutive 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

\[ S = \left\{ T = T[1], T[2], T[3], \ldots \mid T \text{ fulfills recurrence relation} \right\} \]

is a vector space. This means that if \( T_1, T_2 \in S \), then also \( \alpha T_1 + \beta T_2 \in S \), for arbitrary constants \( \alpha, \beta \).

**How do we find a non-trivial solution?**

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

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

for all \( n \geq k \).
Dividing by \( \lambda^{n-k} \) gives that all these constraints are identical to
\[
\begin{align*}
&c_0 \lambda^k + c_1 \lambda^{k-1} + c_2 \cdot \lambda^{k-2} + \cdots + c_k = 0 \\
&\text{characteristic polynomial } P[\lambda]
\end{align*}
\]
This means that if \( \lambda_i \) 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 \( \alpha_i \).

**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.

**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:
\[
\begin{align*}
\alpha_1 \cdot \lambda_1 + \alpha_2 \cdot \lambda_2 + \cdots + \alpha_k \cdot \lambda_k &= T[1] \\
\alpha_1 \cdot \lambda_1^2 + \alpha_2 \cdot \lambda_2^2 + \cdots + \alpha_k \cdot \lambda_k^2 &= T[2] \\
&\vdots \\
\alpha_1 \cdot \lambda_1^k + \alpha_2 \cdot \lambda_2^k + \cdots + \alpha_k \cdot \lambda_k^k &= T[k]
\end{align*}
\]
We show that the column vectors are linearly independent. Then the above equation has a solution.
### Computing the Determinant

\[
\begin{vmatrix}
\lambda_1 & \lambda_2 & \ldots & \lambda_{k-1} & \lambda_k \\
\lambda_1^2 & \lambda_2^2 & \ldots & \lambda_{k-1}^2 & \lambda_k^2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\lambda_1^k & \lambda_2^k & \ldots & \lambda_{k-1}^k & \lambda_k^k
\end{vmatrix}
= \prod_{i=1}^{k} \lambda_i 
\]

\[
\begin{vmatrix}
1 & 1 & \ldots & 1 & 1 \\
\lambda_1 & \lambda_2 & \ldots & \lambda_{k-1} & \lambda_k \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\lambda_1^{k-1} & \lambda_2^{k-1} & \ldots & \lambda_{k-1}^{k-1} & \lambda_k^{k-1}
\end{vmatrix}
= \prod_{i=1}^{k} \lambda_i 
\]

### Computing the Characteristic Polynomial

\[
\prod_{i=1}^{k} (\lambda - \lambda_i)
\]

\[
= \prod_{i=1}^{k} \lambda_i 
\]

\[
= \prod_{i=1}^{k} (\lambda - \lambda_i) 
\]

\[
= \prod_{i=1}^{k} (\lambda - \lambda_i) 
\]

\[
= \prod_{i=1}^{k} (\lambda - \lambda_i) 
\]
Computing the Determinant

Repeating the above steps gives:

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

Hence, if all \(\lambda_i\)'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 \(\lambda_i\) with multiplicity (Vielfachheit) at least 2. Then not only is \(\lambda_i^n\) 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 \(\lambda_i\) 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 \(\lambda_i\).

This means

\[
c_0n\lambda_i^{n-1} + c_1(n-1)\lambda_i^{n-2} + \cdots + c_k(n-k)\lambda_i^{n-k-1} = 0
\]

Hence,

\[
c_0n\lambda_i^n + c_1(n-1)\lambda_i^{n-1} + \cdots + c_k(n-k)\lambda_i^{n-k} = 0
\]

\[
\frac{T(n)}{T(n-1)} + \frac{T(n)}{T(n-1)} + \cdots + \frac{T(n)}{T(n-k)} = 0
\]

The Homogeneous Case

Suppose \(\lambda_i\) has multiplicity \(j\). We know that

\[
c_0n\lambda_i^n + c_1(n-1)\lambda_i^{n-1} + \cdots + c_k(n-k)\lambda_i^{n-k} = 0
\]

(after taking the derivative; multiplying with \(\lambda\); plugging in \(\lambda_i\))

Doing this again gives

\[
c_0n^2\lambda_i^n + c_1(n-1)^2\lambda_i^{n-1} + \cdots + c_k(n-k)^2\lambda_i^{n-k} = 0
\]

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

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

The full proof is omitted. We have only shown that any choice of $\alpha_{ij}$’s is a solution to the recurrence.

---

**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

\[ \alpha \left( \frac{1 + \sqrt{5}}{2} \right) + \beta \left( \frac{1 - \sqrt{5}}{2} \right) = 1 \Rightarrow \alpha - \beta = \frac{2}{\sqrt{5}} \]

---

**Example: Fibonacci Sequence**

The characteristic polynomial is

\[ \lambda^2 - \lambda - 1 \]

Finding the roots, gives

\[ \lambda_{1/2} = \frac{1}{2} \pm \sqrt{\frac{1}{4} + 1} = \frac{1}{2} (1 \pm \sqrt{5}) \]

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.

Example:
\[ T[n] = T[n-1] + 1 \quad T[0] = 1 \]

Then,
\[ T[n-1] = T[n-2] + 1 \quad (n \geq 2) \]

Subtracting the first from the second equation gives,
\[ T[n] - T[n-1] = T[n-1] - T[n-2] \quad (n \geq 2) \]
or
\[ T[n] = 2T[n-1] - T[n-2] \quad (n \geq 2) \]

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

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.

Example: Characteristic polynomial:
\[ \frac{\lambda^2 - 2\lambda + 1}{(\lambda-1)^2} = 0 \]

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 \( 1 + \beta = 2 \Rightarrow \beta = 1 \).
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:

\[
\]

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

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

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

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.

---

### 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.

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.,

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

This is well-defined.

---

Suppose we are given the generating function

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

We can compute the derivative:

\[
\sum_{n \geq 1} n z^{n-1} = \frac{1}{(1-z)^2}
\]

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

We can repeat this
\[ \sum_{n \geq 0} (n + 1)z^n = \frac{1}{(1 - z)^2}. \]

Derivative:
\[ \sum_{n \geq 1} n(n + 1)z^{n-1} = \frac{2}{(1 - z)^3} \]
\[ \sum_{n \geq 0} (n+1)(n+2)z^n \]

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

Computing the \( k \)-th derivative of \( \sum z^n \).
\[ \sum_{n \geq k} n(n - 1) \cdots (n - k + 1)z^{n-k} = \sum_{n \geq 0} (n + k) \cdots (n + 1)z^n \]
\[ = \frac{k!}{(1 - z)^{k+1}}. \]

Hence:
\[ \sum_{n \geq 0} \binom{n+k}{k}z^n = \frac{1}{(1 - z)^{k+1}}. \]

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

We know
\[ \sum_{n \geq 0} y^n = \frac{1}{1 - y}. \]

Hence,
\[ \sum_{n \geq 0} a^n z^n = \frac{1}{1 - az} \]
\[ \sum_{n \geq 0} a^n z^n = \frac{1}{1 - az}. \]

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 \( a_0 = 1 \).

\[
A(z) = \sum_{n=0}^{\infty} a_n z^n \\
= a_0 + \sum_{n=1}^{\infty} (a_{n-1} + 1) z^n \\
= 1 + z \sum_{n=1}^{\infty} a_{n-1} z^{n-1} + z^n \\
= z \sum_{n=0}^{\infty} a_n z^n + \sum_{n=0}^{\infty} z^n \\
= zA(z) + \sum_{n=0}^{\infty} z^n \\
= zA(z) + \frac{1}{1-z}
\]

Solving for \( A(z) \) gives
\[
\sum_{n=0}^{\infty} a_n z^n = A(z) = \frac{1}{(1-z)^2} = \sum_{n=0}^{\infty} (n+1) z^n
\]
Hence, \( a_n = n+1 \).

### Some Generating Functions

<table>
<thead>
<tr>
<th>( n )-th sequence element</th>
<th>generating function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \frac{1}{1-z} )</td>
</tr>
<tr>
<td>( n + 1 )</td>
<td>( \frac{1}{(1-z)^2} )</td>
</tr>
<tr>
<td>( \binom{n+k}{k} )</td>
<td>( \frac{1}{(1-z)^{k+1}} )</td>
</tr>
<tr>
<td>( n )</td>
<td>( \frac{z}{(1-z)^2} )</td>
</tr>
<tr>
<td>( a^n )</td>
<td>( \frac{1}{1-az} )</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>( \frac{z(1+z)}{(1-z)^3} )</td>
</tr>
<tr>
<td>( \frac{1}{n!} )</td>
<td>( e^z )</td>
</tr>
</tbody>
</table>

### Some Generating Functions

<table>
<thead>
<tr>
<th>( n )-th sequence element</th>
<th>generating function</th>
</tr>
</thead>
<tbody>
<tr>
<td>( cf_n )</td>
<td>( cF )</td>
</tr>
<tr>
<td>( f_n + gn )</td>
<td>( F + G )</td>
</tr>
<tr>
<td>( \sum_{i=0}^{n} f_i \theta_{n-i} )</td>
<td>( F \cdot G )</td>
</tr>
<tr>
<td>( f_{n-k} ) ( (n \geq k) ); 0 otw.</td>
<td>( z^k F )</td>
</tr>
<tr>
<td>( \sum_{i=0}^{n} f_i )</td>
<td>( \frac{F(z)}{1-z} )</td>
</tr>
<tr>
<td>( nf_n )</td>
<td>( z \frac{dF(z)}{dz} )</td>
</tr>
<tr>
<td>( c^n f_n )</td>
<td>( F(cz) )</td>
</tr>
</tbody>
</table>
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: \( a_n = 2a_{n-1}, a_0 = 1 \)

3. Transform right hand side so that infinite sums can be replaced by \( A(z) \) or by simple function.
   \[
   A(z) = 1 + \sum_{n \geq 1} (2a_{n-1}) z^n
   = 1 + 2z \sum_{n \geq 1} a_{n-1} z^{n-1}
   = 1 + 2z \sum_{n \geq 0} a_n z^n
   = 1 + 2z \cdot A(z)
   \]
4. Solve for \( A(z) \).
   \[
   A(z) = \frac{1}{1 - 2z}
   \]
Example: $a_n = 3a_{n-1} + n$, $a_0 = 1$

1. Set up generating function:

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

2. Transform right hand side:

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

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

$$= 1 + \sum_{n \geq 1} (3a_{n-1} + n)z^n$$

$$= 1 + 3z \sum_{n \geq 1} a_{n-1} z^{n-1} + \sum_{n \geq 1} nz^n$$

$$= 1 + 3z \sum_{n \geq 0} a_n z^n + \sum_{n \geq 0} nz^n$$

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

4. Solve for $A(z)$:

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

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

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

We use partial fraction decomposition:

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

This gives

$$z^2 - z + 1 = A(1-z)^2 + B(1-3z)(1-z) + C(1-3z)$$

$$= A(1-2z + z^2) + B(1-4z + 3z^2) + C(1-3z)$$

$$= (A+3B)z^2 + (-2A-4B-3C)z + (A+B+C)$$
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:

\[
\begin{align*}
A + B + C &= 1 \\
2A + 4B + 3C &= 1 \\
A + 3B &= 1
\end{align*}
\]

which gives

\[
A = \frac{7}{4}, \quad B = -\frac{1}{4}, \quad C = -\frac{1}{2}
\]

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

\[
\begin{align*}
f_0 &= 1 \\
f_1 &= 2 \\
f_n &= f_{n-1} \cdot f_{n-2} \text{ for } n \geq 2.
\end{align*}
\]

Define

\[
g_n := \log f_n.
\]

Then

\[
\begin{align*}
g_n &= g_{n-1} + g_{n-2} \text{ for } n \geq 2 \\
g_1 &= \log 2 = 1 \text{ (for } \log = \log_2), \quad g_0 = 0 \\
g_n &= F_n \text{ (} n \text{-th Fibonacci number)} \\
f_n &= 2^{F_n}
\end{align*}
\]

Example 7

\[
\begin{align*}
f_1 &= 1 \\
n &= 3f_{\frac{n}{2}} + n; \text{ for } n = 2^k, \ k \geq 1
\end{align*}
\]

Define

\[
g_k := f_{2^k}.
\]

Then:

\[
\begin{align*}
g_0 &= 1 \\
g_k &= 3g_{k-1} + 2^k, \ k \geq 1
\end{align*}
\]
We get

\[ g_k = 3 [g_{k-1}] + 2^k \]

\[ = 3 \left[ 3g_{k-2} + 2^{k-1} \right] + 2^k \]

\[ = 3^2 \left[ g_{k-2} \right] + 32^{k-1} + 2^k \]

\[ = 3^2 \left[ 3g_{k-3} + 2^{k-2} \right] + 32^{k-1} + 2^k \]

\[ = 3^3 g_{k-3} + 3^2 2^{k-2} + 32^{k-1} + 2^k \]

\[ = 2^k \cdot \sum_{i=0}^{k} \left( \frac{3}{2} \right)^i \]

\[ = 2^k \cdot \left( \frac{1}{2} \right)^{k+1} - \frac{1}{1/2} = 3^{k+1} - 2^{k+1} \]

Let \( n = 2^k \):

\[ g_k = 3^{k+1} - 2^{k+1} \]

\[ f_n = 3 \cdot 3^k - 2 \cdot 2^k \]

\[ = 3 \left( 2^{\log_2 3} \right)^k - 2 \cdot 2^k \]

\[ = 3 \left( 2^k \right)^{\log_2 3} - 2 \cdot 2^k \]

\[ = 3 n^{\log_3 3} - 2 n \]

---

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.