

Vorkurs **Datenstrukturen**

Sommersemester 2011

Vorkurs **Datenstrukturen**

Sommersemester 2011

Herzlich willkommen!

Abstrakte Datentypen und Datenstrukturen

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
- Eine **Datenstruktur** implementiert einen abstrakten Datentyps.

Abstrakte Datentypen und Datenstrukturen

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!

Abstrakte Datentypen und Datenstrukturen

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!
 - Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:

Abstrakte Datentypen und Datenstrukturen

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!
 - Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
 - ▶ Wenn der jüngste Schlüssel zu entfernen ist:
Wähle die Datenstruktur

Abstrakte Datentypen und Datenstrukturen

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!
 - Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
 - ▶ Wenn der jüngste Schlüssel zu entfernen ist: Wähle die Datenstruktur „**Stack**“.

Abstrakte Datentypen und Datenstrukturen

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!
 - Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
 - ▶ Wenn der jüngste Schlüssel zu entfernen ist:
Wähle die Datenstruktur „**Stack**“.
 - ▶ Wenn der älteste Schlüssel zu entfernen ist:
Wähle die Datenstruktur

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!
 - Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
 - ▶ Wenn der jüngste Schlüssel zu entfernen ist:
Wähle die Datenstruktur „**Stack**“.
 - ▶ Wenn der älteste Schlüssel zu entfernen ist:
Wähle die Datenstruktur „**Schlange**“.

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!
 - Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
 - ▶ Wenn der jüngste Schlüssel zu entfernen ist:
Wähle die Datenstruktur „**Stack**“.
 - ▶ Wenn der älteste Schlüssel zu entfernen ist:
Wähle die Datenstruktur „**Schlange**“.
 - ▶ Wenn Schlüssel mit zugeordneter Prioritäten eingefügt werden und der Schlüssel mit jeweils höchster Priorität zu entfernen ist:
Benutze die Datenstruktur

Abstrakte Datentypen und Datenstrukturen

- Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
 - Eine **Datenstruktur** implementiert einen abstrakten Datentyps.
-
- Warum abstrakte Datentypen?
 - ▶ In Anwendungen treten dieselben Operationen für **unterschiedlichste** Datenmengen auf.
 - ▶ Gib **eine** Implementierung an!
 - Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
 - ▶ Wenn der jüngste Schlüssel zu entfernen ist:
Wähle die Datenstruktur „**Stack**“.
 - ▶ Wenn der älteste Schlüssel zu entfernen ist:
Wähle die Datenstruktur „**Schlange**“.
 - ▶ Wenn Schlüssel mit zugeordneter Prioritäten eingefügt werden und der Schlüssel mit jeweils höchster Priorität zu entfernen ist:
Benutze die Datenstruktur „**Heap**“.

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

- Wie skaliert die Laufzeit einer Implementierung mit wachsender Eingabelänge?
 - ▶ Das Verhalten für große Eingabelängen ist kritisch.

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

- Wie skaliert die Laufzeit einer Implementierung mit wachsender Eingabelänge?
 - ▶ Das Verhalten für große Eingabelängen ist kritisch.
 - ▶ Für **Eingabelänge n** sei $T(n)$ die maximale Laufzeit für eine Eingabe der Länge n .

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

- Wie skaliert die Laufzeit einer Implementierung mit wachsender Eingabelänge?
 - ▶ Das Verhalten für große Eingabelängen ist kritisch.
 - ▶ Für **Eingabelänge n** sei $T(n)$ die maximale Laufzeit für eine Eingabe der Länge n .
- Führe eine asymptotische Analyse von Laufzeit und Speicherplatzverbrauch durch.

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

- Wie skaliert die Laufzeit einer Implementierung mit wachsender Eingabelänge?
 - ▶ Das Verhalten für große Eingabelängen ist kritisch.
 - ▶ Für **Eingabelänge n** sei $T(n)$ die maximale Laufzeit für eine Eingabe der Länge n .
- Führe eine asymptotische Analyse von Laufzeit und Speicherplatzverbrauch durch.
 - ▶ Die Laufzeit ist **linear**, falls

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

- Wie skaliert die Laufzeit einer Implementierung mit wachsender Eingabelänge?
 - ▶ Das Verhalten für große Eingabelängen ist kritisch.
 - ▶ Für **Eingabelänge n** sei $T(n)$ die maximale Laufzeit für eine Eingabe der Länge n .
- Führe eine asymptotische Analyse von Laufzeit und Speicherplatzverbrauch durch.
 - ▶ Die Laufzeit ist **linear**, falls $0 < \lim_{n \rightarrow \infty} \frac{T(n)}{n} < \infty$
 - ▶ und **quadratisch**, falls

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

- Wie skaliert die Laufzeit einer Implementierung mit wachsender Eingabelänge?
 - ▶ Das Verhalten für große Eingabelängen ist kritisch.
 - ▶ Für **Eingabelänge n** sei $T(n)$ die maximale Laufzeit für eine Eingabe der Länge n .
- Führe eine asymptotische Analyse von Laufzeit und Speicherplatzverbrauch durch.
 - ▶ Die Laufzeit ist **linear**, falls $0 < \lim_{n \rightarrow \infty} \frac{T(n)}{n} < \infty$
 - ▶ und **quadratisch**, falls $0 < \lim_{n \rightarrow \infty} \frac{T(n)}{n^2} < \infty$.

Wir müssen uns mit Grenzwerten beschäftigen.

Angenommen, wir möchten n Zahlen sortieren.

Angenommen, wir möchten n Zahlen sortieren.

- Wir definieren die Eingabelänge als die Anzahl n der Zahlen.
- Ein Sortierverfahren wie **Bubble Sort** benötigt (ungefähr) quadratisch viele Vergleiche im schlimmsten Fall.

Angenommen, wir möchten n Zahlen sortieren.

- Wir definieren die Eingabelänge als die Anzahl n der Zahlen.
- Ein Sortierverfahren wie **Bubble Sort** benötigt (ungefähr) quadratisch viele Vergleiche im schlimmsten Fall.
- **Merge Sort** hingegen kommt für jede Eingabe mit (ungefähr) $n \cdot \log_2 n$ Vergleichen aus.

Angenommen, wir möchten n Zahlen sortieren.

- Wir definieren die Eingabelänge als die Anzahl n der Zahlen.
- Ein Sortierverfahren wie **Bubble Sort** benötigt (ungefähr) quadratisch viele Vergleiche im schlimmsten Fall.
- **Merge Sort** hingegen kommt für jede Eingabe mit (ungefähr) $n \cdot \log_2 n$ Vergleichen aus.

Die asymptotische Analyse zeigt:

$$\lim_{n \rightarrow \infty} \frac{n \cdot \log_2 n}{n^2} = 0$$

Merge Sort ist wesentlich schneller als Bubble Sort.

Wie schnell dominiert die Asymptotik?

Annahme: Ein einfacher Befehl benötigt 10^{-9} Sekunden.

Wie schnell dominiert die Asymptotik?

Annahme: Ein einfacher Befehl benötigt 10^{-9} Sekunden.

n	n^2	n^3	n^{10}	2^n	$n!$
16	256	4.096	$\geq 10^{12}$	65536	$\geq 10^{13}$
32	1.024	32.768	$\geq 10^{15}$	$\geq 4 \cdot 10^9$	$\geq 10^{31}$
64	4.096	262.144	$\geq 10^{18}$	$\geq 6 \cdot 10^{19}$	$\geq 10^{31}$
128	16.384	2.097.152	mehr als	mehr als	mehr als
256	65.536	16.777.216	10 Jahre	600 Jahre	10^{14} Jahre
512	262.144	134.217.728			
1024	1.048.576	$\geq 10^9$			
Million	$\geq 10^{12}$	$\geq 10^{18}$			
	mehr als	mehr als			
	15 Minuten	10 Jahre			

Angenommen, wir möchten feststellen, ob eine Zahl x in einem **sortierten** Array von n Zahlen vorkommt.

Angenommen, wir möchten feststellen, ob eine Zahl x in einem **sortierten** Array von n Zahlen vorkommt.

- Wir definieren die Eingabelänge wieder als die Anzahl n der Zahlen.
- **Lineare Suche**: Wir durchlaufen das Array von „links nach rechts“ und überprüfen jeweils ob wir x gefunden haben.
Mindestens n Vergleiche sind notwendig, wenn x nicht vorkommt.

Angenommen, wir möchten feststellen, ob eine Zahl x in einem **sortierten** Array von n Zahlen vorkommt.

- Wir definieren die Eingabelänge wieder als die Anzahl n der Zahlen.
- **Lineare Suche**: Wir durchlaufen das Array von „links nach rechts“ und überprüfen jeweils ob wir x gefunden haben.
Mindestens n Vergleiche sind notwendig, wenn x nicht vorkommt.
- **Binärsuche** vergleicht x mit $A[n/2]$.
 - ▶ Wenn $x = A[n/2]$, dann haben wir x gefunden!
 - ▶ Wenn $x < A[n/2]$, dann suche in der linken Hälfte $(A[1], \dots, A[n/2] - 1)$,
 - ▶ und sonst in rechten Hälfte $(A[n/2 + 1], \dots, A[n])$.

Angenommen, wir möchten feststellen, ob eine Zahl x in einem **sortierten** Array von n Zahlen vorkommt.

- Wir definieren die Eingabelänge wieder als die Anzahl n der Zahlen.
- **Lineare Suche**: Wir durchlaufen das Array von „links nach rechts“ und überprüfen jeweils ob wir x gefunden haben.
Mindestens n Vergleiche sind notwendig, wenn x nicht vorkommt.
- **Binärsuche** vergleicht x mit $A[n/2]$.
 - ▶ Wenn $x = A[n/2]$, dann haben wir x gefunden!
 - ▶ Wenn $x < A[n/2]$, dann suche in der linken Hälfte $(A[1], \dots, A[n/2] - 1)$,
 - ▶ und sonst in rechten Hälfte $(A[n/2 + 1], \dots, A[n])$.

Wer ist schneller: Lineare Suche oder Binärsuche?

```
void Suche( int unten, int oben){  
    if (oben < unten)  
        std::cout << x << " wurde nicht gefunden."  
        << std::endl;
```

Binärsuche

```
void Suche( int unten, int oben){  
    if (oben < unten)  
        std::cout << x << " wurde nicht gefunden."  
        << std::endl;  
    int mitte = (unten+oben)/2;
```

Binärsuche

```
void Suche( int unten, int oben){  
    if (oben < unten)  
        std::cout << x << " wurde nicht gefunden."  
        << std::endl;  
    int mitte = (unten+oben)/2;  
    if (A[mitte] == x)
```

```
void Suche( int unten, int oben){  
    if (oben < unten)  
        std::cout << x << " wurde nicht gefunden."  
        << std::endl;  
    int mitte = (unten+oben)/2;  
    if (A[mitte] == x)  
        std::cout << x << " wurde in Position "  
        << mitte << " gefunden." << std::endl;  
}
```

```
void Suche( int unten, int oben){
    if (oben < unten)
        std::cout << x << " wurde nicht gefunden."
            << std::endl;
    int mitte = (unten+oben)/2;
    if (A[mitte] == x)
        std::cout << x << " wurde in Position "
            << mitte << " gefunden." << std::endl;
    else {
```

```
void Suche( int unten, int oben){
    if (oben < unten)
        std::cout << x << " wurde nicht gefunden."
            << std::endl;
    int mitte = (unten+oben)/2;
    if (A[mitte] == x)
        std::cout << x << " wurde in Position "
            << mitte << " gefunden." << std::endl;
    else {
        if (x < A[mitte])
            Suche(unten, mitte-1);
        else
            Suche(mitte+1, oben);
    }
}
```

- Wie kann man ein rekursives Programm verifizieren?
Wie lassen sich Aussagen über die Laufzeit machen?
- Mit Hilfe der **vollständigen Induktion**.

Warum ist Suche (a, b) korrekt?

Wir beweisen Korrektheit durch Induktion nach der Anzahl $n = \textit{oben} - \textit{unten} + 1$ der Zahlen.

Warum ist Suche (a, b) korrekt?

Wir beweisen Korrektheit durch Induktion nach der Anzahl $n = \text{oben} - \text{unten} + 1$ der Zahlen.

- Der **Basisschritt** für $n \leq 0$:
 - ▶ Wenn $n \leq 0$, dann ist das Array leer und x kann nicht vorkommen.
 - ▶ Suche (unten, oben) ist für $n \leq 0$ korrekt.

Warum ist Suche (a, b) korrekt?

Wir beweisen Korrektheit durch Induktion nach der Anzahl $n = \text{oben} - \text{unten} + 1$ der Zahlen.

- Der **Basisschritt** für $n \leq 0$:
 - ▶ Wenn $n \leq 0$, dann ist das Array leer und x kann nicht vorkommen.
 - ▶ Suche (unten, oben) ist für $n \leq 0$ korrekt.
- Der **Induktionsschritt**:
 - ▶ Wir können die **Induktionsvoraussetzung** annehmen:
Suche (unten', oben') ist korrekt, wenn $\text{oben}' - \text{unten}' \leq n$.
 - ▶ Wir müssen zeigen, dass Suche (unten, oben) für $\text{oben} - \text{unten} = n + 1$ korrekt ist.

- Wenn an der Position `mitte` die Zahl x steht, wird sie ordnungsgemäß gefunden.
- Andernfalls ruft sich der Algorithmus rekursiv auf einem Teilproblem auf.

- Wenn an der Position `mitte` die Zahl x steht, wird sie ordnungsgemäß gefunden.
- Andernfalls ruft sich der Algorithmus rekursiv auf einem Teilproblem auf.
 - ▶ Das Teilproblem hat **weniger** als $\text{oben} - \text{unten} - 1 = n$ Zahlen.

- Wenn an der Position `mitte` die Zahl x steht, wird sie ordnungsgemäß gefunden.
- Andernfalls ruft sich der Algorithmus rekursiv auf einem Teilproblem auf.
 - ▶ Das Teilproblem hat **weniger** als $\text{oben} - \text{unten} - 1 = n$ Zahlen.
 - ▶ Auf einem Problem dieser Größe arbeitet der Algorithmus jedoch **nach Induktionsannahme** korrekt!

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \textit{oben} - \textit{unten} + 1$ der Elemente des Arrays ab.

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \textit{oben} - \textit{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) =$

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) = 1$.

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) = 1$.
 - ▶ Suche (unten, oben) arbeitet entweder in der linken Hälfte oder in der rechten Hälfte rekursiv weiter:
 - ★ Beide Hälften bestehen aus genau $2^{k-1} - 1$ Zahlen.

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) = 1$.
 - ▶ Suche (unten, oben) arbeitet entweder in der linken Hälfte oder in der rechten Hälfte rekursiv weiter:
 - ★ Beide Hälften bestehen aus genau $2^{k-1} - 1$ Zahlen.
 - ▶ Es ist $T(2^k - 1) = T(2^{k-1} - 1) + 1$, falls $k \geq 1$.

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) = 1$.
 - ▶ Suche (unten, oben) arbeitet entweder in der linken Hälfte oder in der rechten Hälfte rekursiv weiter:
 - ★ Beide Hälften bestehen aus genau $2^{k-1} - 1$ Zahlen.
 - ▶ Es ist $T(2^k - 1) = T(2^{k-1} - 1) + 1$, falls $k \geq 1$.
 - ▶ Es ist $T(2^k - 1) =$

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) = 1$.
 - ▶ Suche (unten, oben) arbeitet entweder in der linken Hälfte oder in der rechten Hälfte rekursiv weiter:
 - ★ Beide Hälften bestehen aus genau $2^{k-1} - 1$ Zahlen.
 - ▶ Es ist $T(2^k - 1) = T(2^{k-1} - 1) + 1$, falls $k \geq 1$.
 - ▶ Es ist $T(2^k - 1) = k$. Warum?

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) = 1$.
 - ▶ Suche (unten, oben) arbeitet entweder in der linken Hälfte oder in der rechten Hälfte rekursiv weiter:
 - ★ Beide Hälften bestehen aus genau $2^{k-1} - 1$ Zahlen.
 - ▶ Es ist $T(2^k - 1) = T(2^{k-1} - 1) + 1$, falls $k \geq 1$.
 - ▶ Es ist $T(2^k - 1) = k$. Warum?
- Und wenn n eine beliebige natürliche Zahl ist? Dann ist

$$T(n) =$$

Wieviele Vergleiche maximal führt suche (unten, oben) aus?

Die Anzahl der Vergleiche hängt nur von der Zahl
 $n = \text{oben} - \text{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
 - ▶ Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n .
 - ▶ Dann ist $T(1) = 1$.
 - ▶ Suche (unten, oben) arbeitet entweder in der linken Hälfte oder in der rechten Hälfte rekursiv weiter:
 - ★ Beide Hälften bestehen aus genau $2^{k-1} - 1$ Zahlen.
 - ▶ Es ist $T(2^k - 1) = T(2^{k-1} - 1) + 1$, falls $k \geq 1$.
 - ▶ Es ist $T(2^k - 1) = k$. Warum?
- Und wenn n eine beliebige natürliche Zahl ist? Dann ist

$$T(n) = \lceil \log_2(n + 1) \rceil.$$

Binärsuche ist viel schneller als lineare Suche, denn $\lim_{n \rightarrow \infty} \frac{\log_2 n}{n} = 0$.

Wir machen mit

rekursiver Programmierung / vollständiger Induktion

weiter und kehren dann zur

asymptotischen Analyse

zurück.

- Die **Anfangskonfiguration**:
 - ▶ Wir haben drei Stäbe 1, 2 und 3.
 - ▶ Ursprünglich besitzt Stab 1 N Ringe, wobei die Ringe in absteigender Größe auf dem Stab aufgereiht sind:
 - ★ Der größte Ring von Stab 1 ist also der unterste Ring.
 - ▶ Die Stäbe 2 und 3 sind zu Anfang leer.

- Die **Anfangskonfiguration**:

- ▶ Wir haben drei Stäbe 1, 2 und 3.
- ▶ Ursprünglich besitzt Stab 1 N Ringe, wobei die Ringe in absteigender Größe auf dem Stab aufgereiht sind:
 - ★ Der größte Ring von Stab 1 ist also der unterste Ring.
- ▶ Die Stäbe 2 und 3 sind zu Anfang leer.

- Die **Züge**

- ▶ Bewege einen zuoberst liegenden Ring von einem Stab zu einem anderen.
- ▶ Der Zug ist nur dann erlaubt, wenn der Ring auf einen größeren Ring gelegt wird oder wenn der Stab leer ist.

- Die **Anfangskonfiguration**:

- ▶ Wir haben drei Stäbe 1, 2 und 3.
- ▶ Ursprünglich besitzt Stab 1 N Ringe, wobei die Ringe in absteigender Größe auf dem Stab aufgereiht sind:
 - ★ Der größte Ring von Stab 1 ist also der unterste Ring.
- ▶ Die Stäbe 2 und 3 sind zu Anfang leer.

- Die **Züge**

- ▶ Bewege einen zuoberst liegenden Ring von einem Stab zu einem anderen.
- ▶ Der Zug ist nur dann erlaubt, wenn der Ring auf einen größeren Ring gelegt wird oder wenn der Stab leer ist.

- Die **Zielkonfiguration**:

- ▶ Alle Ringe müssen nach Stab 2 bewegt werden.

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
```

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
```

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab2;
else {
```

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab2;
else {
    Hanoi(N-1,stab1,stab3,stab2);
```

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab2;
else {
    Hanoi(N-1,stab1,stab3,stab2);
    bewege einen Ring von Stab stab1 nach Stab stab2;
```

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab2;
else {
    Hanoi(N-1,stab1,stab3,stab2);
    bewege einen Ring von Stab stab1 nach Stab stab2;
    Hanoi(N-1,stab3,stab2,stab1); }}}
```

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab2;
else {
    Hanoi(N-1,stab1,stab3,stab2);
    bewege einen Ring von Stab stab1 nach Stab stab2;
    Hanoi(N-1,stab3,stab2,stab1); }}}
```

- Wie zeigt man, dass `Hanoi` korrekt ist?

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab2;
else {
    Hanoi(N-1,stab1,stab3,stab2);
    bewege einen Ring von Stab stab1 nach Stab stab2;
    Hanoi(N-1,stab3,stab2,stab1); }}}
```

- Wie zeigt man, dass `Hanoi` korrekt ist?
- Sei $T(N)$ die Anzahl der Ringbewegungen nach Aufruf des Programms `Hanoi(N,*,*,*)`: Wie bestimmt man $T(N)$?

Die Summe der ersten n Zahlen

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

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$:

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$.

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :

▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i =$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) =$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n + 1) = \frac{n \cdot (n+1)}{2} + (n + 1) =$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$.

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!

- ▶ **Induktionsschritt** von n auf $n+1$:

- ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.

- ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n + 1) = \frac{n \cdot (n+1)}{2} + (n + 1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten:

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!
- Ein direktes Argument:
 - ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n + 1) = \frac{n \cdot (n+1)}{2} + (n + 1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
 - ★ Die Hauptdiagonale besitzt n Gitterpunkte.

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
 - ★ Die Hauptdiagonale besitzt n Gitterpunkte.
 - ★ Unterhalb der Hauptdiagonale befindet sich die Hälfte der verbleibenden $n^2 - n$ Gitterpunkte.

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
 - ★ Die Hauptdiagonale besitzt n Gitterpunkte.
 - ★ Unterhalb der Hauptdiagonale befindet sich die Hälfte der verbleibenden $n^2 - n$ Gitterpunkte.
- ▶ Also folgt $\sum_{i=1}^n i =$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n + 1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n + 1) = \frac{n \cdot (n+1)}{2} + (n + 1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
 - ★ Die Hauptdiagonale besitzt n Gitterpunkte.
 - ★ Unterhalb der Hauptdiagonale befindet sich die Hälfte der verbleibenden $n^2 - n$ Gitterpunkte.
- ▶ Also folgt $\sum_{i=1}^n i = n +$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
 - ★ Die Hauptdiagonale besitzt n Gitterpunkte.
 - ★ Unterhalb der Hauptdiagonale befindet sich die Hälfte der verbleibenden $n^2 - n$ Gitterpunkte.
- ▶ Also folgt $\sum_{i=1}^n i = n + \frac{n^2 - n}{2}$

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
 - ★ Die Hauptdiagonale besitzt n Gitterpunkte.
 - ★ Unterhalb der Hauptdiagonale befindet sich die Hälfte der verbleibenden $n^2 - n$ Gitterpunkte.
- ▶ Also folgt $\sum_{i=1}^n i = n + \frac{n^2 - n}{2} = \frac{n \cdot (n+1)}{2}$.

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$:

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=0}^0 a^i = 1$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=0}^0 a^i = 1$ und $\frac{a^{0+1} - 1}{a - 1} = 1$.

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=0}^0 a^i = 1$ und $\frac{a^{0+1} - 1}{a - 1} = 1$. Stimmt!

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=0}^0 a^i = 1$ und $\frac{a^{0+1} - 1}{a - 1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=0}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=0}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt.

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=1}^{n+1} a^i$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=1}^{n+1} a^i = \sum_{i=1}^n a^i + a^{n+1}$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :

- ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
- ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=1}^{n+1} a^i = \sum_{i=1}^n a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1}$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=1}^{n+1} a^i = \sum_{i=1}^n a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1} = \frac{a^{n+2}-1}{a-1}$. Stimmt!

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=1}^{n+1} a^i = \sum_{i=1}^n a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1} = \frac{a^{n+2}-1}{a-1}$. Stimmt!
- Ein direktes Argument:

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=1}^{n+1} a^i = \sum_{i=1}^n a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1} = \frac{a^{n+2}-1}{a-1}$. Stimmt!
- Ein direktes Argument:

$$(a - 1) \cdot \sum_{i=0}^n a^i = a \cdot \sum_{i=0}^n a^i - \sum_{i=0}^n a^i$$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=1}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=1}^{n+1} a^i = \sum_{i=1}^n a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1} = \frac{a^{n+2}-1}{a-1}$. Stimmt!
- Ein direktes Argument:

$$\begin{aligned}(a-1) \cdot \sum_{i=0}^n a^i &= a \cdot \sum_{i=0}^n a^i - \sum_{i=0}^n a^i \\ &= \sum_{i=1}^{n+1} a^i - \sum_{i=0}^n a^i =\end{aligned}$$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=0}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=0}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=0}^{n+1} a^i = \sum_{i=0}^n a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1} = \frac{a^{n+2}-1}{a-1}$. Stimmt!
- Ein direktes Argument:

$$\begin{aligned}(a-1) \cdot \sum_{i=0}^n a^i &= a \cdot \sum_{i=0}^n a^i - \sum_{i=0}^n a^i \\ &= \sum_{i=1}^{n+1} a^i - \sum_{i=0}^n a^i = a^{n+1} - a^0\end{aligned}$$

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n :
 - ▶ **Induktionsbasis** für $n = 0$: $\sum_{i=0}^0 a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
 - ▶ **Induktionsschritt** von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=0}^n a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
 - ★ $\sum_{i=0}^{n+1} a^i = \sum_{i=0}^n a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1} = \frac{a^{n+2}-1}{a-1}$. Stimmt!
- Ein direktes Argument:

$$\begin{aligned}(a-1) \cdot \sum_{i=0}^n a^i &= a \cdot \sum_{i=0}^n a^i - \sum_{i=0}^n a^i \\ &= \sum_{i=1}^{n+1} a^i - \sum_{i=0}^n a^i = a^{n+1} - a^0 = a^{n+1} - 1\end{aligned}$$

Die Menge U bestehe aus n Objekten.

Die Menge U bestehe aus n Objekten.

- (a) Zeige, dass U genau 2^n Teilmengen besitzt, wenn wir die leere Menge als eine Teilmenge von U zulassen.

Die Menge U bestehe aus n Objekten.

- (a) Zeige, dass U genau 2^n Teilmengen besitzt, wenn wir die leere Menge als eine Teilmenge von U zulassen.
- (b) Zeige, dass U genau $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ Teilmengen mit k Elementen besitzt.

Die Menge U bestehe aus n Objekten.

- (a) Zeige, dass U genau 2^n Teilmengen besitzt, wenn wir die leere Menge als eine Teilmenge von U zulassen.
- (b) Zeige, dass U genau $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ Teilmengen mit k Elementen besitzt.
- (c) Zeige: Es gibt genau $n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$ verschiedene Permutationen von n Objekten.

Binomischer Lehrsatz

Zeige: $(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k}$. (Zeige $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$ zuerst.)

- Teile ein Rechteck durch Geraden in Teilflächen.
- Kann man die Teilflächen immer so mit den Farben Schwarz und Weiß färben, dass Teilflächen, die an einer Kante zusammenstoßen, verschiedene Farben besitzen?

Wo ist der Fehler?

Wir zeigen, dass je zwei natürliche Zahlen a und b gleich sind.

Wo ist der Fehler?

Wir zeigen, dass je zwei natürliche Zahlen a und b gleich sind.

- Wir setzen $k = \max\{a, b\}$ und führen eine Induktion nach k .

Wir zeigen, dass je zwei natürliche Zahlen a und b gleich sind.

- Wir setzen $k = \max\{a, b\}$ und führen eine Induktion nach k .
 - ▶ Im Basisschritt haben wir $k = 0$ und deshalb ist $a = 0 = b$ und das war zu zeigen.

Wir zeigen, dass je zwei natürliche Zahlen a und b gleich sind.

- Wir setzen $k = \max\{a, b\}$ und führen eine Induktion nach k .
 - ▶ Im Basisschritt haben wir $k = 0$ und deshalb ist $a = 0 = b$ und das war zu zeigen.
 - ▶ Im Induktionsschritt ist $\max\{a, b\} = k + 1$.
 - ★ Wir können die Induktionsbehauptung auf $a - 1$ und $b - 1$ anwenden, denn $\max\{a - 1, b - 1\} = k$.

Wir zeigen, dass je zwei natürliche Zahlen a und b gleich sind.

- Wir setzen $k = \max\{a, b\}$ und führen eine Induktion nach k .
 - ▶ Im Basisschritt haben wir $k = 0$ und deshalb ist $a = 0 = b$ und das war zu zeigen.
 - ▶ Im Induktionsschritt ist $\max\{a, b\} = k + 1$.
 - ★ Wir können die Induktionsbehauptung auf $a - 1$ und $b - 1$ anwenden, denn $\max\{a - 1, b - 1\} = k$.
 - ★ Also ist $a - 1 = b - 1$ und die Behauptung $a = b$ folgt.

Wo ist der Fehler?

Wir zeigen, dass je zwei natürliche Zahlen a und b gleich sind.

- Wir setzen $k = \max\{a, b\}$ und führen eine Induktion nach k .
 - ▶ Im Basisschritt haben wir $k = 0$ und deshalb ist $a = 0 = b$ und das war zu zeigen.
 - ▶ Im Induktionsschritt ist $\max\{a, b\} = k + 1$.
 - ★ Wir können die Induktionsbehauptung auf $a - 1$ und $b - 1$ anwenden, denn $\max\{a - 1, b - 1\} = k$.
 - ★ Also ist $a - 1 = b - 1$ und die Behauptung $a = b$ folgt.
- Die Behauptung ist richtig für $k = 0$, aber falsch schon für $k = 1$.

- Wir haben behauptet, dass Mergesort viel schneller als Bubblesort ist, weil

$$\lim_{n \rightarrow \infty} \frac{n \cdot \log_2 n}{n^2} = 0$$

- ▶ Warum ist der Grenzwert 0?

- Wir haben behauptet, dass Mergesort viel schneller als Bubblesort ist, weil

$$\lim_{n \rightarrow \infty} \frac{n \cdot \log_2 n}{n^2} = 0$$

- ▶ Warum ist der Grenzwert 0?

- Wir haben behauptet, dass Binärsuche viel schneller als lineare Suche ist, weil

$$\lim_{n \rightarrow \infty} \frac{\log_2 n}{n} = 0$$

- ▶ Warum ist auch dieser Grenzwert 0?

- Wir haben behauptet, dass Mergesort viel schneller als Bubblesort ist, weil

$$\lim_{n \rightarrow \infty} \frac{n \cdot \log_2 n}{n^2} = 0$$

- ▶ Warum ist der Grenzwert 0?

- Wir haben behauptet, dass Binärsuche viel schneller als lineare Suche ist, weil

$$\lim_{n \rightarrow \infty} \frac{\log_2 n}{n} = 0$$

- ▶ Warum ist auch dieser Grenzwert 0?

- Wir untersuchen unendliche Folgen $(a_n) = a_1, a_2, \dots, a_k \dots$
- Wann konvergiert (a_n) gegen einen Grenzwert a ?

Man sagt, dass die Folge $(a_n) = a_0, a_1, \dots$ gegen $a \in \mathbb{R}$ konvergiert, und nennt a dann den Grenzwert oder Limes der Folge (a_n) , formal

$$\lim_{n \rightarrow \infty} a_n = a,$$

Man sagt, dass die Folge $(a_n) = a_0, a_1, \dots$ gegen $a \in \mathbb{R}$ konvergiert, und nennt a dann den Grenzwert oder Limes der Folge (a_n) , formal

$$\lim_{n \rightarrow \infty} a_n = a,$$

wenn es für jedes $\varepsilon > 0$ ein $N \in \mathbb{N}$ gibt, so dass $|a_n - a| < \varepsilon$ für alle $n > N$ gilt.

Man sagt, dass die Folge $(a_n) = a_0, a_1, \dots$ gegen $a \in \mathbb{R}$ konvergiert, und nennt a dann den Grenzwert oder Limes der Folge (a_n) , formal

$$\lim_{n \rightarrow \infty} a_n = a,$$

wenn es für jedes $\varepsilon > 0$ ein $N \in \mathbb{N}$ gibt, so dass $|a_n - a| < \varepsilon$ für alle $n > N$ gilt.

- **Zeige:** Jede Folge (a_n) besitzt höchstens einen Grenzwert.

Man sagt, dass die Folge $(a_n) = a_0, a_1, \dots$ gegen $a \in \mathbb{R}$ konvergiert, und nennt a dann den Grenzwert oder Limes der Folge (a_n) , formal

$$\lim_{n \rightarrow \infty} a_n = a,$$

wenn es für jedes $\varepsilon > 0$ ein $N \in \mathbb{N}$ gibt, so dass $|a_n - a| < \varepsilon$ für alle $n > N$ gilt.

- **Zeige:** Jede Folge (a_n) besitzt höchstens einen Grenzwert.
- Warum gilt $\lim_{n \rightarrow \infty} \frac{1}{n} = 0$?
 - ▶ Für $\varepsilon > 0$ wähle $N =$

Man sagt, dass die Folge $(a_n) = a_0, a_1, \dots$ gegen $a \in \mathbb{R}$ konvergiert, und nennt a dann den Grenzwert oder Limes der Folge (a_n) , formal

$$\lim_{n \rightarrow \infty} a_n = a,$$

wenn es für jedes $\varepsilon > 0$ ein $N \in \mathbb{N}$ gibt, so dass $|a_n - a| < \varepsilon$ für alle $n > N$ gilt.

- **Zeige:** Jede Folge (a_n) besitzt höchstens einen Grenzwert.
- Warum gilt $\lim_{n \rightarrow \infty} \frac{1}{n} = 0$?
 - ▶ Für $\varepsilon > 0$ wähle $N = \lceil \frac{1}{\varepsilon} \rceil$.
 - ▶ Dann gilt $|\frac{1}{n} - 0| = \frac{1}{n} < \varepsilon$ für $n > N$.

Man sagt, dass die Folge $(a_n) = a_0, a_1, \dots$ gegen $a \in \mathbb{R}$ konvergiert, und nennt a dann den Grenzwert oder Limes der Folge (a_n) , formal

$$\lim_{n \rightarrow \infty} a_n = a,$$

wenn es für jedes $\varepsilon > 0$ ein $N \in \mathbb{N}$ gibt, so dass $|a_n - a| < \varepsilon$ für alle $n > N$ gilt.

- **Zeige:** Jede Folge (a_n) besitzt höchstens einen Grenzwert.
- Warum gilt $\lim_{n \rightarrow \infty} \frac{1}{n} = 0$?
 - ▶ Für $\varepsilon > 0$ wähle $N = \lceil \frac{1}{\varepsilon} \rceil$.
 - ▶ Dann gilt $|\frac{1}{n} - 0| = \frac{1}{n} < \varepsilon$ für $n > N$.
- Warum gilt $\lim_{n \rightarrow \infty} \frac{1}{n^{1/k}} = 0$? für jede natürliche Zahl k ?

- Hat eine Folge keinen Grenzwert, so heißt sie **divergent**.
 - ▶ Für jede Zahl $a \in \mathbb{R}$ gibt es ein $\varepsilon > 0$, so dass unendlich viele Folgenglieder *außerhalb* der ε -Umgebung von a liegen.

- Hat eine Folge keinen Grenzwert, so heißt sie **divergent**.
 - ▶ Für jede Zahl $a \in \mathbb{R}$ gibt es ein $\varepsilon > 0$, so dass unendlich viele Folgenglieder *außerhalb* der ε -Umgebung von a liegen.
 - ▶ Die alternierende Folge $a_n = (-1)^n$ divergiert, da sie unendlich oft zwischen 1 und -1 hin und her springt.

- Hat eine Folge keinen Grenzwert, so heißt sie **divergent**.
 - ▶ Für jede Zahl $a \in \mathbb{R}$ gibt es ein $\varepsilon > 0$, so dass unendlich viele Folgenglieder *außerhalb* der ε -Umgebung von a liegen.
 - ▶ Die alternierende Folge $a_n = (-1)^n$ divergiert, da sie unendlich oft zwischen 1 und -1 hin und her springt.
- Wir sagen, dass eine Folge gegen „ ∞ **konvergiert**“, falls für jede Zahl $a \in \mathbb{R}$ fast alle Folgenglieder größer als a sind.

- Hat eine Folge keinen Grenzwert, so heißt sie **divergent**.
 - ▶ Für jede Zahl $a \in \mathbb{R}$ gibt es ein $\varepsilon > 0$, so dass unendlich viele Folgenglieder *außerhalb* der ε -Umgebung von a liegen.
 - ▶ Die alternierende Folge $a_n = (-1)^n$ divergiert, da sie unendlich oft zwischen 1 und -1 hin und her springt.
- Wir sagen, dass eine Folge gegen „ ∞ **konvergiert**“, falls für jede Zahl $a \in \mathbb{R}$ fast alle Folgenglieder größer als a sind.
 - ▶ Konvergiert die Folge $a_n = \sum_{k=1}^n \frac{1}{k^2}$ gegen ∞ ?

- Hat eine Folge keinen Grenzwert, so heißt sie **divergent**.
 - ▶ Für jede Zahl $a \in \mathbb{R}$ gibt es ein $\varepsilon > 0$, so dass unendlich viele Folgenglieder *außerhalb* der ε -Umgebung von a liegen.
 - ▶ Die alternierende Folge $a_n = (-1)^n$ divergiert, da sie unendlich oft zwischen 1 und -1 hin und her springt.
- Wir sagen, dass eine Folge gegen „ ∞ **konvergiert**“, falls für jede Zahl $a \in \mathbb{R}$ fast alle Folgenglieder größer als a sind.
 - ▶ Konvergiert die Folge $a_n = \sum_{k=1}^n \frac{1}{k^2}$ gegen ∞ ?
 - ▶ Konvergiert die Folge $a_n = \sum_{k=1}^n \frac{1}{k}$ gegen ∞ ?

- Hat eine Folge keinen Grenzwert, so heißt sie **divergent**.
 - ▶ Für jede Zahl $a \in \mathbb{R}$ gibt es ein $\varepsilon > 0$, so dass unendlich viele Folgenglieder *außerhalb* der ε -Umgebung von a liegen.
 - ▶ Die alternierende Folge $a_n = (-1)^n$ divergiert, da sie unendlich oft zwischen 1 und -1 hin und her springt.
- Wir sagen, dass eine Folge gegen „ ∞ **konvergiert**“, falls für jede Zahl $a \in \mathbb{R}$ fast alle Folgenglieder größer als a sind.
 - ▶ Konvergiert die Folge $a_n = \sum_{k=1}^n \frac{1}{k^2}$ gegen ∞ ?
 - ▶ Konvergiert die Folge $a_n = \sum_{k=1}^n \frac{1}{k}$ gegen ∞ ?
- Konvergenz gegen $-\infty$ wird analog definiert.

- Folgen (a_n) mit $\lim a_n = 0$ heißen **Nullfolgen**.

Nullfolgen spielen eine Sonderrolle, denn $\lim a_n = a$ gilt genau dann, wenn $(a_n - a)$ eine Nullfolge ist.

- Folgen (a_n) mit $\lim a_n = 0$ heißen **Nullfolgen**.
Nullfolgen spielen eine Sonderrolle, denn $\lim a_n = a$ gilt genau dann, wenn $(a_n - a)$ eine Nullfolge ist.
- Gilt $|a_n| \leq c \cdot |b_n|$ für eine Konstante c und fast alle n , dann ist die Folge (b_n) eine **Majorante** der Folge (a_n) .

- Folgen (a_n) mit $\lim a_n = 0$ heißen **Nullfolgen**.
Nullfolgen spielen eine Sonderrolle, denn $\lim a_n = a$ gilt genau dann, wenn $(a_n - a)$ eine Nullfolge ist.
- Gilt $|a_n| \leq c \cdot |b_n|$ für eine Konstante c und fast alle n , dann ist die Folge (b_n) eine **Majorante** der Folge (a_n) .
- (a_n) heißt **beschränkt**, wenn die Folge $b_n = 1$ eine Majorante ist.

Das Einschließungs- und Majorantenkriterium

- Folgen (a_n) mit $\lim a_n = 0$ heißen **Nullfolgen**.
Nullfolgen spielen eine Sonderrolle, denn $\lim a_n = a$ gilt genau dann, wenn $(a_n - a)$ eine Nullfolge ist.
 - Gilt $|a_n| \leq c \cdot |b_n|$ für eine Konstante c und fast alle n , dann ist die Folge (b_n) eine **Majorante** der Folge (a_n) .
 - (a_n) heißt **beschränkt**, wenn die Folge $b_n = 1$ eine Majorante ist.
- Das **Einschließungskriterium**: Gilt $\lim a_n = \lim b_n = a$ und $a_n \leq x_n \leq b_n$ für fast alle n , so gilt auch $\lim x_n = a$.

Das Einschließungs- und Majorantenkriterium

- Folgen (a_n) mit $\lim a_n = 0$ heißen **Nullfolgen**.
Nullfolgen spielen eine Sonderrolle, denn $\lim a_n = a$ gilt genau dann, wenn $(a_n - a)$ eine Nullfolge ist.
 - Gilt $|a_n| \leq c \cdot |b_n|$ für eine Konstante c und fast alle n , dann ist die Folge (b_n) eine **Majorante** der Folge (a_n) .
 - (a_n) heißt **beschränkt**, wenn die Folge $b_n = 1$ eine Majorante ist.
- Das **Einschließungskriterium**: Gilt $\lim a_n = \lim b_n = a$ und $a_n \leq x_n \leq b_n$ für fast alle n , so gilt auch $\lim x_n = a$.
Zeige, dass $\lim \frac{1}{n^k} = 0$ für jedes $k \geq 1$ gilt.

Das Einschließungs- und Majorantenkriterium

- Folgen (a_n) mit $\lim a_n = 0$ heißen **Nullfolgen**.
Nullfolgen spielen eine Sonderrolle, denn $\lim a_n = a$ gilt genau dann, wenn $(a_n - a)$ eine Nullfolge ist.
 - Gilt $|a_n| \leq c \cdot |b_n|$ für eine Konstante c und fast alle n , dann ist die Folge (b_n) eine **Majorante** der Folge (a_n) .
 - (a_n) heißt **beschränkt**, wenn die Folge $b_n = 1$ eine Majorante ist.
-
- Das **Einschließungskriterium**: Gilt $\lim a_n = \lim b_n = a$ und $a_n \leq x_n \leq b_n$ für fast alle n , so gilt auch $\lim x_n = a$.
Zeige, dass $\lim \frac{1}{n^k} = 0$ für jedes $k \geq 1$ gilt.
 - Das **Majorantenkriterium für Nullfolgen**: Hat die Folge (a_n) eine Nullfolge als ihre Majorante, so ist auch (a_n) eine Nullfolge.

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

① $\lim (a_n \pm b_n) = a \pm b;$

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

① $\lim (a_n \pm b_n) = a \pm b;$

② $\lim (a_n \cdot b_n) = a \cdot b;$

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

- 1 $\lim (a_n \pm b_n) = a \pm b;$
- 2 $\lim (a_n \cdot b_n) = a \cdot b;$
- 3 $\lim \frac{a_n}{b_n} = \frac{a}{b},$ falls $b \neq 0$ und $b_n \neq 0$ für alle n .

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

- 1 $\lim (a_n \pm b_n) = a \pm b$;
- 2 $\lim (a_n \cdot b_n) = a \cdot b$;
- 3 $\lim \frac{a_n}{b_n} = \frac{a}{b}$, falls $b \neq 0$ und $b_n \neq 0$ für alle n .

1. ist einfach. Warum?

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

- 1 $\lim (a_n \pm b_n) = a \pm b$;
- 2 $\lim (a_n \cdot b_n) = a \cdot b$;
- 3 $\lim \frac{a_n}{b_n} = \frac{a}{b}$, falls $b \neq 0$ und $b_n \neq 0$ für alle n .

1. ist einfach. Warum? Ein Beweis für 2.

- Die konvergenten Folgen (a_n) und (b_n) sind beschränkt: (Warum?) Es gibt $c \in \mathbb{R}$ mit $|a_n|, |b_n| \leq c$ für alle n .

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

- 1 $\lim (a_n \pm b_n) = a \pm b$;
- 2 $\lim (a_n \cdot b_n) = a \cdot b$;
- 3 $\lim \frac{a_n}{b_n} = \frac{a}{b}$, falls $b \neq 0$ und $b_n \neq 0$ für alle n .

1. ist einfach. Warum? Ein Beweis für 2.

- Die konvergenten Folgen (a_n) und (b_n) sind beschränkt: (Warum?) Es gibt $c \in \mathbb{R}$ mit $|a_n|, |b_n| \leq c$ für alle n .
- Wie groß ist der Abstand zwischen $a_n b_n$ und ab ?

$$|a_n b_n - ab| = |a_n b_n - a_n b + a_n b - ab|$$

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

- 1 $\lim (a_n \pm b_n) = a \pm b$;
- 2 $\lim (a_n \cdot b_n) = a \cdot b$;
- 3 $\lim \frac{a_n}{b_n} = \frac{a}{b}$, falls $b \neq 0$ und $b_n \neq 0$ für alle n .

1. ist einfach. Warum? Ein Beweis für 2.

- Die konvergenten Folgen (a_n) und (b_n) sind beschränkt: (Warum?) Es gibt $c \in \mathbb{R}$ mit $|a_n|, |b_n| \leq c$ für alle n .
- Wie groß ist der Abstand zwischen $a_n b_n$ und ab ?

$$\begin{aligned} |a_n b_n - ab| &= |a_n b_n - a_n b + a_n b - ab| \\ &\leq |a_n| \cdot |b_n - b| + |b| \cdot |a_n - a| \end{aligned}$$

Arithmetische Operationen auf Folgen

(a_n) und (b_n) seien konvergente Folgen mit den jeweiligen Grenzwerten a und b . Dann gilt:

- 1 $\lim (a_n \pm b_n) = a \pm b$;
- 2 $\lim (a_n \cdot b_n) = a \cdot b$;
- 3 $\lim \frac{a_n}{b_n} = \frac{a}{b}$, falls $b \neq 0$ und $b_n \neq 0$ für alle n .

1. ist einfach. Warum? Ein Beweis für 2.

- Die konvergenten Folgen (a_n) und (b_n) sind beschränkt: (Warum?) Es gibt $c \in \mathbb{R}$ mit $|a_n|, |b_n| \leq c$ für alle n .
- Wie groß ist der Abstand zwischen $a_n b_n$ und ab ?

$$\begin{aligned} |a_n b_n - ab| &= |a_n b_n - a_n b + a_n b - ab| \\ &\leq |a_n| \cdot |b_n - b| + |b| \cdot |a_n - a| \\ &\leq c \cdot (|b_n - b| + |a_n - a|). \end{aligned}$$

- Eine Funktion $f : [a, b] \rightarrow \mathbb{R}$ ist genau dann **stetig** im Punkt $c \in [a, b]$, wenn

$$\lim f(c_n) = f(c)$$

für jede Folge (c_n) mit $a \leq c_n \leq b$ und $\lim c_n = c$ gilt.

- Eine Funktion $f : [a, b] \rightarrow \mathbb{R}$ ist genau dann **stetig** im Punkt $c \in [a, b]$, wenn

$$\lim f(c_n) = f(c)$$

für jede Folge (c_n) mit $a \leq c_n \leq b$ und $\lim c_n = c$ gilt.

- Die Funktion f ist stetig, wenn f für alle Punkte $c \in [a, b]$ stetig ist.

- Eine Funktion $f : [a, b] \rightarrow \mathbb{R}$ ist genau dann **stetig** im Punkt $c \in [a, b]$, wenn

$$\lim f(c_n) = f(c)$$

für jede Folge (c_n) mit $a \leq c_n \leq b$ und $\lim c_n = c$ gilt.

- Die Funktion f ist stetig, wenn f für alle Punkte $c \in [a, b]$ stetig ist.
-
- Der Graph einer stetigen Funktion macht „keine Sprünge“ und kann deshalb in einem Strich gezeichnet werden.

- Eine Funktion $f : [a, b] \rightarrow \mathbb{R}$ ist genau dann **stetig** im Punkt $c \in [a, b]$, wenn

$$\lim f(c_n) = f(c)$$

für jede Folge (c_n) mit $a \leq c_n \leq b$ und $\lim c_n = c$ gilt.

- Die Funktion f ist stetig, wenn f für alle Punkte $c \in [a, b]$ stetig ist.

- Der Graph einer stetigen Funktion macht „keine Sprünge“ und kann deshalb in einem Strich gezeichnet werden.
- Für eine stetige Funktion dürfen wir Funktion und Grenzwert vertauschen:

Die Funktion $f : [a, b] \rightarrow \mathbb{R}$ sei stetig. Wenn alle Folgenglieder der Folge (c_n) zum Intervall $[a, b]$ gehören, dann folgt

$$\lim f(c_n) = f(\lim c_n).$$

Beispiele stetiger Funktionen

Wenn $f, g : [a, b] \rightarrow \mathbb{R}$ stetige Funktionen sind, dann sind auch

① $f + g$ und $f \cdot g$ stetig.

Beispiele stetiger Funktionen

Wenn $f, g : [a, b] \rightarrow \mathbb{R}$ stetige Funktionen sind, dann sind auch

- 1 $f + g$ und $f \cdot g$ stetig.
- 2 Besitzt die Funktion g keine Nullstelle im Intervall $[a, b]$, dann ist auch f/g eine stetige Funktion.

Wenn $f, g : [a, b] \rightarrow \mathbb{R}$ stetige Funktionen sind, dann sind auch

- 1 $f + g$ und $f \cdot g$ stetig.
 - 2 Besitzt die Funktion g keine Nullstelle im Intervall $[a, b]$, dann ist auch f/g eine stetige Funktion.
-
- Warum stimmen die Behauptungen? Arithmetische Operationen auf konvergenten Folgen führen zu konvergenten Folgen.

Beispiele stetiger Funktionen

Wenn $f, g : [a, b] \rightarrow \mathbb{R}$ stetige Funktionen sind, dann sind auch

- 1 $f + g$ und $f \cdot g$ stetig.
 - 2 Besitzt die Funktion g keine Nullstelle im Intervall $[a, b]$, dann ist auch f/g eine stetige Funktion.
- Warum stimmen die Behauptungen? Arithmetische Operationen auf konvergenten Folgen führen zu konvergenten Folgen.
 - Aus 1. folgt, dass alle Polynome $p(x) = \sum_{k=0}^n a_k x^k$ stetig sind.

Beispiele stetiger Funktionen

Wenn $f, g : [a, b] \rightarrow \mathbb{R}$ stetige Funktionen sind, dann sind auch

- 1 $f + g$ und $f \cdot g$ stetig.
- 2 Besitzt die Funktion g keine Nullstelle im Intervall $[a, b]$, dann ist auch f/g eine stetige Funktion.

- Warum stimmen die Behauptungen? Arithmetische Operationen auf konvergenten Folgen führen zu konvergenten Folgen.
- Aus 1. folgt, dass alle Polynome $p(x) = \sum_{k=0}^n a_k x^k$ stetig sind.
- Als Konsequenz von 2. sind alle rationale Funktionen, also die Quotienten $p_1(x)/p_2(x)$ von Polynomen, stetig in allen Punkten, die keine Nullstelle des Nenners sind.