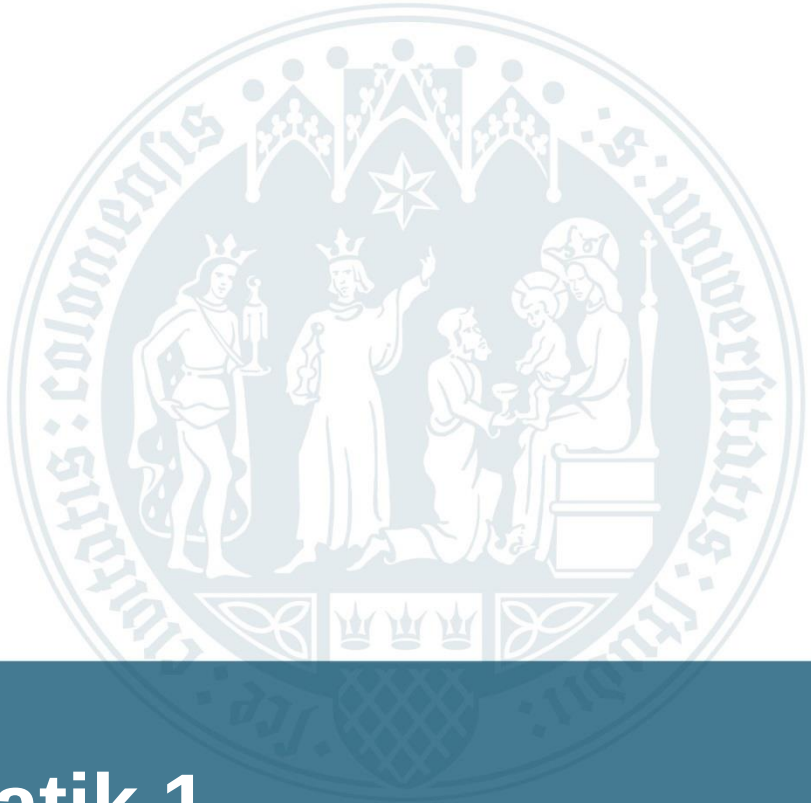
Universität zu Köln

Universität zu Köln



Grundzüge der Informatik 1

Vorlesung 4

Überblick

Überblick

- Wiederholung O-Notation
- Ω-Notation
- Θ-Notation
- o-Notation
- ω-Notation
- Korrektheitsbeweise (Anfang)
- Mathematische Induktion
- Schleifeninvarianten

Wiederholung

Grundideen (asymptotische Analyse)

- Ignoriere konstante Faktoren
- Betrachte das Verhältnis von Laufzeiten für $n \to \infty$
- Klassifizieren Laufzeiten durch Angabe von „einfachen Vergleichsfunktionen“

O-Notation

- Lässt Konstanten und Terme niederer Ordnung weg
- Beschreibt Menge von Funktionen, die höchstens genau so schnell wachsen wie die Vergleichsfunktion
- Kann genutzt werden, um obere Schranken für die Laufzeit anzugeben

Landau Notation

Definition 4.1 (O-Notation)

- $O(g(n)) = \{f(n) :$ Es gibt positive Konstanten c und n_0, so dass für alle $n \geq n_0$ gilt: $0 \leq f(n) \leq c \cdot g(n)\}$

Wiederholung

Ω-Notation

- Lässt Konstanten und Terme niederer Ordnung weg
- Beschreibt Menge von Funktionen, die mindestens genau so schnell wachsen wie die Vergleichsfunktion
- Kann genutzt werden, um untere Schranken für die Laufzeit anzugeben

Landau Notation

Definition 4.2 (Ω-Notation)

- $\Omega(g(n)) = \{f(n)$: Es gibt positive Konstanten c und n_0, so dass für alle $n \geq n_0$ gilt: $0 \leq c \cdot g(n) \leq f(n)\}$

Landau Notation

Satz 4.1

- $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

Beweis

- „$\Rightarrow$“ :
- Annahme: Es gilt $f(n) \in O(g(n))$
- Z.z.: $g(n) \in \Omega(f(n))$
- Da laut Annahme $f(n) \in O(g(n))$ gilt, gibt es pos. Konstanten c‘ und n_0, so dass für alle $n \geq n_0$ gilt: $0 \leq f(n) \leq c' \cdot g(n)$
- Damit gilt auch $0 \leq 1/c' \; f(n) \leq g(n)$ für alle $n \geq n_0$
- Es folgt ebenso für alle $n \geq n_0$: $0 \leq c \; f(n) \leq g(n)$, wobei $c=1/c'>0$ ist
- Somit gilt $g(n) \in \Omega(f(n))$

Landau Notation

Satz 4.1

- $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

Beweis

- „$\Leftarrow$“ :
- Annahme: Es gilt $g(n) \in \Omega(f(n))$
- Z.z.: $f(n) \in O(g(n))$
- Da laut Annahme $g(n) \in \Omega(f(n))$ gilt, gibt es pos. Konstanten c‘ und n_0, so dass für alle $n \geq n_0$ gilt: $0 \leq c' \cdot f(n) \leq g(n)$
- Damit gilt auch $0 \leq f(n) \leq 1/c' \; g(n)$ für alle $n \geq n_0$
- Es folgt ebenso für alle $n \geq n_0$: $0 \leq f(n) \leq c \; g(n)$, wobei $c=1/c'>0$ ist
- Somit gilt auch $f(n) \in O(g(n))$

Landau Notation

Beispiele Ω-Notation

- 0.1n ist in $\Omega(n)$
- $1000n \notin \Omega(n^2)$
- $\Omega(1000n)=\Omega(n)$

Beweisskizze

- Ähnlich wie bei der O-Notation
- Man kann die ersten beiden Aussagen wegen unseres Satzes auch in O-Notation umwandeln:
- 0.1n ist in $\Omega(n)$ ist äquivalent zu n ist in $O(0.1n)$
- $1000n \notin \Omega(n^2)$ ist äquivalent zu $n^2 \notin O(1000n)$

Landau Notation

Satz 4.2

- Die Worst-Case Laufzeit von Algorithmus InsertionSort ist in $\Omega(n^2)$.

Beweis

- Wir wissen, dass die Worst Case Laufzeit von InsertionSort $T(n)= (3n^2+7n-8)/2$ ist
- Z.z.: Es gibt positive Konstanten c und n_0, so dass für alle $n \geq n_0$ gilt: $0 \leq c \cdot n^2 \leq (3n^2+7n-8)/2$
- Mit $c=1$ und $n_0=1$ gilt:
- $(3n^2+7n-8)/2 = (2n^2 + n^2+7n -8)/2 \geq 2n^2/2 = n^2$
- Damit ist $(3n^2+7n-8)/2$ in $\Omega(n^2)$

Landau Notation

Satz 4.2

- Die Worst-Case Laufzeit von Algorithmus InsertionSort ist in $\Omega(n^2)$.

Bedeutung

- Für jedes hinreichend große n ist die Worst-Case Laufzeit ungefähr n^2
- D.h. es gibt für jedes hinreichend große n eine Eingabe der Größe n mit ungefähr dieser Laufzeit

Landau Notation

Definition 4.3 (Θ-Notation)

- $f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in O(g(n))$ und $f(n) \in \Omega(g(n))$

Interpretation

- $f(n) \in \Theta(g(n))$ bedeutet, dass f(n) genauso stark wächst wie g(n) für $n \to \infty$
- Dabei ignorieren wir beim Wachstum konstante Faktoren
- Die Θ-Notation liefert eine obere und untere Schranke
- Die Funktionen f(n) und g(n) müssen asymptotisch nicht-negativ sein (für n groß genug, sind die Funktionen nicht-negativ)

Landau Notation

Korollar 4.1

- Die Worst-Case Laufzeit von Algorithmus InsertionSort ist in $\Theta(n^2)$.

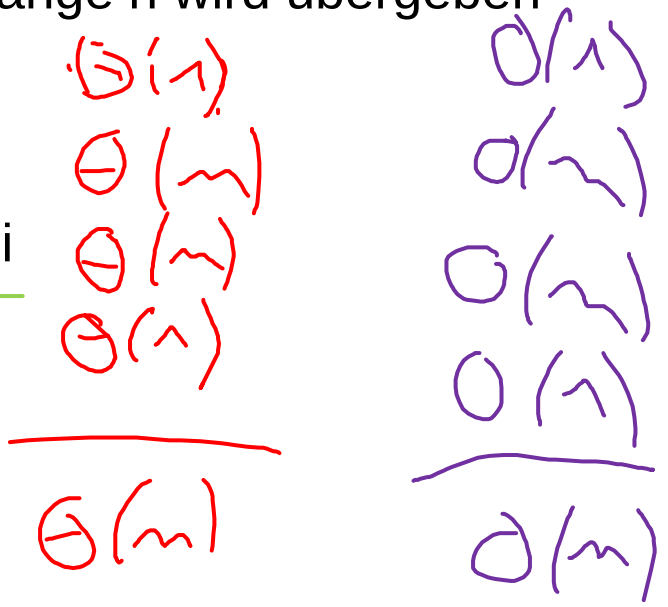
Interpretation des Resultats

- Für jedes hinreichend große n ist die Laufzeit jeder Eingabe höchstens n^2
- Für jedes hinreichend große n ist die Laufzeit mindestens einer Eingabe der Größe n mindestens n^2
- In den Aussagen ignoriere ich multiplikative Konstanten

Worst-Case Laufzeitanalyse mit Θ-Notation

MaxSuche(A, n) * Array A der Länge n wird übergeben

1. max = 1
2. **for** i=2 **to** n **do**
3. **if** A[i] > A[max] **then** max = i
4. **return** max



Landau Notation

Definition 4.4 (o-Notation)

- $o(g(n)) = \{f(n)$: Für jede Konstante $c>0$ gibt es eine Konstante $n_0>0$, so dass für alle $n \geq n_0$ gilt: $0 \leq f(n) \leq c \cdot g(n)\}$

Interpretation

- $f(n) \in o(g(n))$ bedeutet, dass $f(n)$ weniger stark wächst als $g(n)$ für $n \to \infty$
- Dabei ignorieren wir beim Wachstum konstante Faktoren
- Die Funktionen $f(n)$ und $g(n)$ müssen asymptotisch nicht-negativ sein (für n groß genug, sind die Funktionen nicht-negativ)

Landau Notation

Beispiel

- $n \in o(n^2)$

Beweis

- Z.z.: Für jedes $c>0$ gibt es $n_0>0$, so dass für alle $n \geq n_0$ gilt: $0 \leq n \leq c \cdot n^2$
- Sei also $c>0$ beliebig
- Wir wählen $n_0 = 1/c$
- Dann gilt $n \geq 1/c$ und somit $cn \geq 1$
- Damit folgt: $0 \leq n \leq cn\ n = c\ n^2$
- Da wir somit für jedes beliebige $c>0$ ein $n_0 >0$ gefunden haben, so dass für alle $n \geq n_0$ gilt: $0 \leq n \leq c \cdot n^2$, folgt, dass $n \in o(n^2)$

Landau Notation

Definition 4.5 (ω-Notation)

- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

Interpretation

- $f(n) \in \omega(g(n))$ bedeutet, dass f(n) echt stärker wächst als g(n) für $n \to \infty$
- Dabei ignorieren wir beim Wachstum konstante Faktoren
- Die Funktionen f(n) und g(n) müssen asymptotisch nicht-negativ sein (für n groß genug, sind die Funktionen nicht-negativ)

Landau Notation

Schreibweise

- Es ist üblich $f(n)=O(n)$ zu schreiben anstelle von $f(n) \in O(n)$
- Dies gilt auch für Ω-, Θ-, o- und ω-Notation.

Korrektheitsbeweise

Korrektheitsbeweis

- Formale Argumentation, dass ein Algorithmus korrekt arbeitet

Problembeschreibung

- Definiert für eine Menge von zulässigen Eingaben die zugehörigen gewünschten Ausgaben

Korrektheit

- Wir bezeichnen einen Algorithmus für eine vorgegebene Problembeschreibung als korrekt, wenn er für jede zulässige Eingabe die in der Problembeschreibung spezifizierte Ausgabe berechnet
- Streng genommen kann man also nur von Korrektheit sprechen, wenn vorher festgelegt wurde, was der Algorithmus eigentlich tun soll

Korrektheitsbeweise

Beispiel

- Eingabe: Folge von n Zahlen: $(x_1,\ldots, x_n)$
- Ausgabe: Permutation $(x_1',\ldots,x_n')$ von $(x_1,\ldots, x_n)$ mit $x_1' \leq x_2' \leq \ldots \leq x_n'$

Korrektheitsbeweise

Beispiel

- Eingabe: Folge von n Zahlen: $(x_1,\ldots, x_n)$
- Ausgabe: Permutation $(x_1',\ldots,x_n')$ von $(x_1,\ldots, x_n)$ mit $x_1' \leq x_2' \leq \ldots \leq x_n'$

Was muss man zeigen?

- Um zu zeigen, dass ein Sortieralgorithmus korrekt ist, muss man zeigen, dass er jede Folge von Eingabezahlen richtig sortiert

Korrektheitsbeweise

Motivation

- Elemente eines Korrektheitsbeweise (z.B. Schleifeninvarianten) können zur Überprüfung der Funktionsweise während der Laufzeit verwendet werden
- Außerdem lassen sich aus Korrektheitsbeweisen häufig gute Kommentare herleiten
- Letztendlich hält ein Korrektheitsbeweis nur die Überlegungen fest, die ein Entwickler sowieso machen muss
- Korrektheitsbeweise helfen dabei, sich dieser Überlegungen bewusst zu werden, und Fehler zu vermeiden

Korrektheitsbeweise – ein triviales Beispiel

EinfacherAlgorithmus(n)

1. X=10
2. Y=n
3. X=X+Y
4. **return** X

Behauptung

- EinfacherAlgorithmus(n) gibt n+10 zurück.

Beweis:

- Zu Beginn des Algorithmus sind alle Variablen bis auf n undefiniert
- In Zeile 1 wird X der Wert 10 zugewiesen.
- In Zeile 2 wird Y der Wert n zugewiesen.
- In Zeile 3 wird X der Wert X+Y zugewiesen. Da X aktuell den Wert 10 hat und Y den Wert n, erhält X den Wert n+10.
- In Zeile 4 wird X zurückgegeben. Da X den Wert n+10 hat, folgt die Behauptung.

Korrektheitsbeweise

Beweisprinzip der mathematischen Induktion

- Sei A(n) eine Aussage über eine natürliche Zahl $n \in \mathbb{N}=\{1, 2, 3,\ldots\}$
- Wir wollen zeigen, dass die Aussage für alle natürlichen Zahlen gilt

Mathematische Induktion besteht aus 2 Hauptkomponenten

- Induktionsanfang: Aussage A(1) stimmt
- Induktionsschritt: Wenn A(n) gilt, dann gilt auch A(n+1)

Beispiel

- A(1) 
- A(2)
- A(3)
- …

Korrektheitsbeweise

Beispiel

- A(n) ist die Aussage: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Induktionsanfang:

- A(1): $\sum_{i=1}^{1} i = \frac{1(1+1)}{2}$ ist korrekt (beide Seiten sind 1)

Induktionsschritt:

- Annahme: Es gilt $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- Z.z.: Es gilt auch $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$
- $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + n + 1 = \frac{(n+2)(n+1)}{2}$

Korrektheitsbeweise - Schleifeninvarianten

Schleifeninvariante

- A(n) ist eine Aussage über den Zustand des Algorithmus vor dem n-ten Durchlauf einer Schleife
- Eine Schleifeninvariante ist korrekt, wenn Sie zu Beginn jedes Schleifendurchlaufs erfüllt ist.
- A(1) wird auch als Initialisierung bezeichnet.

Korrektheitsbeweis für Invarianten

- Induktionsanfang: Die Aussage A(1) gilt
- Induktionsschluss: Gilt A(n) und ist die Eintrittsbedingung der Schleife erfüllt so gilt auch A(n+1)

Korrektheitsbeweise - Schleifeninvarianten

Schleifeninvariante bei for-Schleifen - Annahmen

- Die Laufvariable wird am Ende der for-Schleife erhöht
- Zur Initialisierung wurde die Laufvariable bereits auf ihren Startwert gesetzt

Abhängigkeit von der Laufvariable

- Da bei for-Schleifen die Laufvariable i.a. eindeutig mit der Anzahl der Schleifendurchläufe zusammenhängt, können wir die Invariante auch in Abhängigkeit der Laufvariablen formulieren

Korrektheitsbeweise - Schleifeninvarianten

MaxSuche(A, n) * Array A der Länge n wird übergeben

1. max = 1
2. **for** i=2 **to** n **do**
3. **if** A[i] > A[max] **then** max = i
4. **return** max

Schleifeninvariante

- A(i): A[max] ist ein größtes Elements aus dem Teilarray A[1..i-1]

Korrektheitsbeweise - Schleifeninvarianten

```
MaxSuche(A, n)     \* Array A der Länge n wird übergeben
1.    max = 1
2.    for i=2 to n do
3.        if A[i] > A[max] then max = i
4.    return max
```

Schleifeninvariante

- A(i): A[max] ist ein größtes Elements aus dem Teilarray A[1..i-1]

Lemma 4.1

- A(i) ist eine korrekte Schleifeninvariante.

Korrektheitsbeweise - Schleifeninvarianten

MaxSuche(A, n) * Array A der Länge n wird übergeben

1. max = 1
2. **for** i=2 **to** n **do**
3. **if** A[i] > A[max] **then** max = i
4. **return** max

Beweis von Lemma 4.1:

- Induktionsanfang: Zu Beginn der for-Schleife (i=2) ist max=1 und damit Index eines größten Elements aus dem Teilarray A[1…i-1]
- Induktionsschluss:
- Annahme: max ist Index eines größten Elements aus A[1…i-1] und i ≤ n

Korrektheitsbeweise - Schleifeninvarianten

```
MaxSuche(A, n)     \* Array A der Länge n wird übergeben
1.    max = 1
2.    for i=2 to n do
3.        if A[i] > A[max] then max = i
4.    return max
```

Beweis von Lemma 4.1:

- Annahme: max ist Index eines größten Elements aus A[1…i-1] und i ≤ n
- Ist A[i]≤A[max], so ändert sich max nicht und max ist der Index eines größten Elements aus A[1…i]
- Ist A[i]>A[max], so wird max auf i gesetzt und damit ist max ebenfalls der Index eines größten Elements aus A[1…i]

Zusammenfassung

Zusammenfassung

- Asymptotische Notation
- Korrektheitsbeweise (Anfang)
- Mathematische Induktion
- Schleifeninvarianten