technische universität
dortmund

LS 2 /
Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Algorithmenentwurf

Anforderungen

- Korrektheit
- Effizienz (Laufzeit, Speicherplatz)

Entwurf umfasst

1. Beschreibung des Berechnungsproblems
2. Beschreibung des Algorithmus/der Datenstruktur
3. Korrektheitsbeweis
4. Analyse von Laufzeit- und Speicherplatzbedarf

Was ist ein mathematischer Beweis?

Informale Definition

Ein Beweis ist eine Herleitung einer Aussage aus bereits bewiesenen Aussagen und/oder Grundannahmen (Axiomen).

Korrektheitsbeweise

Was muss ich eigentlich zeigen?

Häufiges Problem: Was muss man in einem Korrektheitsbeweis beweisen?

Was wissen wir?

Problembeschreibung definiert zulässige Eingaben und zugehörige (gewünschte) Ausgaben

Korrektheitsbeweise

Wann ist ein Algorithmus korrekt?

- Wir bezeichnen einen Algorithmus für eine vorgegebene Problembeschreibung als korrekt, wenn er für jede Eingabe die in der Problembeschreibung spezifizierte Ausgabe berechnet.
- Streng genommen kann man also nur von Korrektheit sprechen, wenn vorher das angenommene Verhalten des Algorithmus geeignet beschrieben wurde.

Beispiel: Sortieren

- Problem: Sortieren
- Eingabe: Folge von n Zahlen $(a_1, \dots, a_n)$
- Ausgabe: Permutation $(a'_1, \dots, a'_n)$ von $(a_1, \dots, a_n)$ so dass $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Korrektheitsbeweise

Was müssen wir zeigen?

Für jede gültige Eingabe sortiert unser Algorithmus korrekt

Aber wie? (auf welchen Annahmen können wir aufbauen?)

- Die Grundannahme in der Algorithmik ist, dass ein Pseudocodebefehl gemäß seiner Spezifikation ausgeführt wird
- Z.B.: Die Anweisung $x \leftarrow x + 1$ bewirkt, dass die Variable x um eins erhöht wird

Korrektheitsbeweise

Ein triviales Beispiel

$\mathrm{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:

Korrektheitsbeweise

Ein triviales Beispiel

$\mathrm{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.

Korrektheitsbeweise

Ein triviales Beispiel

$\text{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.

Der Befehl in Zeile 1 weist *X* den Wert 10 zu.

Korrektheitsbeweise

Ein triviales Beispiel

$\text{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.

Der Befehl in Zeile 1 weist X den Wert 10 zu.

Der Befehl in Zeile 2 weist Y den Wert n zu.

Korrektheitsbeweise

Ein triviales Beispiel

$\text{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.
Der Befehl in Zeile 1 weist *X* den Wert 10 zu.
Der Befehl in Zeile 2 weist *Y* den Wert n zu.
Der Befehl in Zeile 3 weist *X* den Wert *X* + *Y* zu. Da *X* vor der Zuweisung den Wert 10 enthielt und *Y* den Wert n, wird *X* auf $10 + n$ gesetzt.

Korrektheitsbeweise

Ein triviales Beispiel

$\text{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.
Der Befehl in Zeile 1 weist *X* den Wert 10 zu.
Der Befehl in Zeile 2 weist *Y* den Wert n zu.
Der Befehl in Zeile 3 weist *X* den Wert *X* + *Y* zu. Da *X* vor der Zuweisung den Wert 10 enthielt und *Y* den Wert n, wird *X* auf $10 + n$ gesetzt. Der Befehl in Zeile 4 gibt *X* zurück. Da *X* zu diesem Zeitpunkt den Wert $10 + n$ hat, folgt die Behauptung.

Korrektheitsbeweise

Ein triviales Beispiel

$\text{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Ein **Korrektheitsbeweis** vollzieht also das Programm Schritt für Schritt nach und leitet durch Verknüpfen der einzelnen Befehle eine Aussage über die Ausgabe her.

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.
Der Befehl in Zeile 1 weist *X* den Wert 10 zu.
Der Befehl in Zeile 2 weist *Y* den Wert n zu.
Der Befehl in Zeile 3 weist *X* den Wert *X* + *Y* zu. Da *X* vor der Zuweisung den Wert 10 enthielt und *Y* den Wert n, wird *X* auf $10 + n$ gesetzt. Der Befehl in Zeile 4 gibt *X* zurück. Da *X* zu diesem Zeitpunkt den Wert $10 + n$ hat, folgt die Behauptung.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

Problem

Wir wissen nicht, wie viele Durchläufe die **for**-Schleife benötigt. Dies hängt sogar von der Eingabelänge ab.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

Abhilfe

Wir benötigen eine Aussage, die den Zustand am Ende der Schleife nach einer beliebigen Anzahl Schleifendurchläufe angibt.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

Definition (Schleifeninvariante)

Eine **Schleifeninvariante** ist eine i.A. von der Anzahl i der Schleifendurchläufe abhängige Aussage $A(i)$, die zu Beginn des i-ten Schleifendurchlauf gilt. Mit $A(1)$ beziehen wir uns also auf den Zustand zu Beginn des ersten Durchlaufs. Dieser wird auch als Initialisierung bezeichnet.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

*Schleifeninvariante (Konventionen für **for**-Schleifen)*

Bei einer **for**-Schleife nehmen wir dabei an, dass bereits am Ende eines Schleifendurchlaufs die Laufvariable erhöht wird. Außerdem nehmen wir an, dass zur Initialisierung die Laufvariable bereits auf ihren Startwert initialisiert wurde.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

*Schleifeninvariante (Konventionen für **for**-Schleifen, Teil 2)*

Da bei **for**-Schleifen die Anzahl der Durchläufe direkt von der Laufvariable abhängt, können wir eine Schleifeninvariante auch in Abhängigkeit der Laufvariablen formulieren.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

Schleifeninvariante (Zustand nach Austritt aus der Schleife)

Eine Schleife wird beendet, wenn beim Überprüfen der Schleifenbedingung eine Verletzung derselben festgestellt wird. Der danach angenommene Zustand des Algorithmus wird als Austrittszustand bezeichnet und sollte i.A. nicht direkt von der Anzahl der Schleifendurchläufe abhängen.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

Lemma 1

Die **for**-Schleife in Algorithmus MaxSearch erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis:

Der Befehl in Zeile 1 des Algorithmus setzt $\max$ auf 1. Wir zeigen per Induktion über die Laufvariable j, dass (Inv.) erfüllt ist.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis:

Der Befehl in Zeile 1 des Algorithmus setzt $\max$ auf 1. Wir zeigen per Induktion über die Laufvariable j, dass (Inv.) erfüllt ist.

(I.A.) Zur Initialisierung der Schleife ist $\max = 1$ und $j = 2$. Außerdem enthält $A[1..1]$ nur ein Element, nämlich $A[1]$. Da $A[\max] = A[1]$ ist, ist $A[\max]$ ein größtes Element aus $A[1..1]$. Daher gilt die Invariante zur Initialisierung.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis:

Der Befehl in Zeile 1 des Algorithmus setzt $\max$ auf 1. Wir zeigen per Induktion über die Laufvariable j, dass (Inv.) erfüllt ist.

(I.A.) Zur Initialisierung der Schleife ist $\max = 1$ und $j = 2$. Außerdem enthält $A[1..1]$ nur ein Element, nämlich $A[1]$. Da $A[\max] = A[1]$ ist, ist $A[\max]$ ein größtes Element aus $A[1..1]$. Daher gilt die Invariante zur Initialisierung.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis (fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j = j_0 < \mathrm{length}[A] + 1$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis (fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j = j_0 < \mathrm{length}[A] + 1$.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für $j + 1$. $(j \to j + 1)$

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis (fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j = j_0 < \mathrm{length}[A] + 1$.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für $j + 1$. $(j \to j + 1)$

Wir betrachten den Durchlauf der Schleife mit Laufvariable $j = j_0$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis (fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j = j_0 < \mathrm{length}[A] + 1$.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für $j + 1$. $(j \to j + 1)$

Wir betrachten den Durchlauf der Schleife mit Laufvariable $j = j_0$.

Falls $A[j] \leq A[\max]$ ist, so wird die **then**-Anweisung nicht ausgeführt. Dann ist $A[\max]$ auch größtes Element aus $A[1..j]$. Am Ende der Schleife wird j um 1 erhöht. Somit gilt die Invariante auch für $j + 1$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis (fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j = j_0 < \mathrm{length}[A] + 1$.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für $j + 1$. $(j \to j + 1)$

Wir betrachten den Durchlauf der Schleife mit Laufvariable $j = j_0$.

Falls $A[j] \leq A[\max]$ ist, so wird die **then**-Anweisung nicht ausgeführt. Dann ist $A[\max]$ auch größtes Element aus $A[1..j]$. Am Ende der Schleife wird j um 1 erhöht. Somit gilt die Invariante auch für $j + 1$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis (fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j = j_0 < \mathrm{length}[A] + 1$.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für $j + 1$. $(j \to j + 1)$

Wir betrachten den Durchlauf der Schleife mit Laufvariable $j = j_0$.

Falls $A[j] > A[\max]$ ist, so ist nach I.V. $A[j]$ größer als das größte Element aus $A[1..j-1]$ und somit das größte Element aus $A[1..j]$. In der **then**-Anweisung wird $\max = j$ gesetzt. Damit ist $A[\max]$ das größte Element aus $A[1..j]$. Am Ende der Schleife wird j um 1 erhöht. Damit gilt die Invariante auch für $j + 1$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus $\mathrm{MaxSearch}$ erfüllt folgende Schleifeninvariante:

(Inv.) $A[\max]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis (fortgesetzt):

Nach dem Prinzip der vollständigen Induktion ist somit die Invariante vor jedem Schleifendurchlauf und vor dem Schleifenaustritt erfüllt. Somit gilt die Invariante.

Korrektheitsbeweise

Satz 2

Algorithmus $\mathrm{MaxSearch}$ berechnet den Index eines größten Elements aus einem Feld A.

Beweis

Der Schleifenaustritt aus der **for**-Schleife (Zeile 2) erfolgt für $j = \mathrm{length}[A] + 1$. Nach Lemma 1 gilt die Invariante insbesondere beim Schleifenaustritt und somit, dass $A[\mathrm{max}]$ ein größtes Element aus $A[1..\mathrm{length}[A]]$ ist.

Der **return**-Befehl gibt mit max daher den Index eines größten Elements aus A zurück.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

Invarianten im Praktikum

Wir werden im Praktikum Invarianten zur Kommentierung von Schleifen nutzen. Diese kann man auch mit Hilfe von „Assertions“ zur Laufzeit überprüfen (dazu mehr im Praktikum).

Korrektheitsbeweise

Notation: Invarianten

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** $\text{length}[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** $\max$

- Kommentare (Invariante):
- Initialisierung: $\max = 1$, $j = 2$, $A[\max]$ ist Maximum von $A[1..1]$.
- Invariante: $A[\max]$ ist Maximum von $A[1..j-1]$
- Austritt: $A[\max]$ ist Maximum von $A[1..\text{length}[A]]$

InsertionSort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j − 1
4.          while i > 0 and A[i] > key do
5.              A[i + 1] ← A[i]
6.              i ← i − 1
7.          A[i + 1] ← key
```

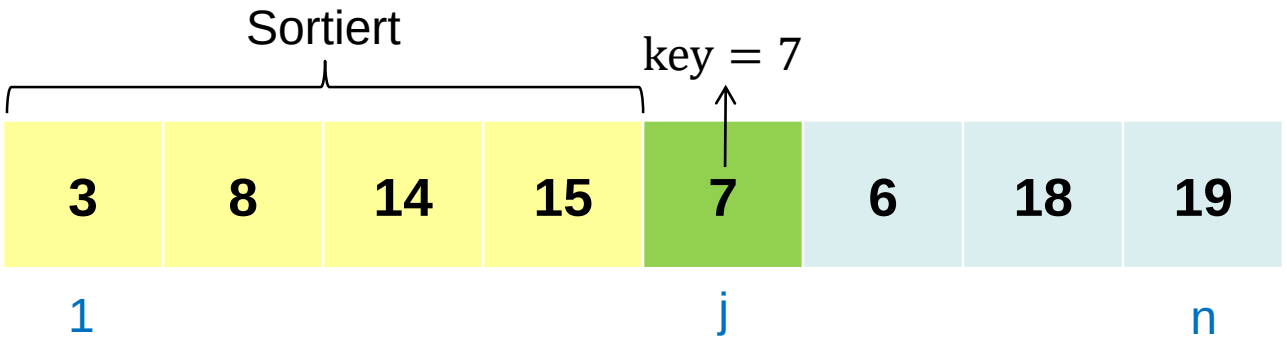
- Eingabegröße n
- $\text{length}[A] = n$
- verschiebe alle Elemente aus
- $A[1..j-1]$, die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke

Invarianten InsertionSort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
2. key $\leftarrow A[j]$
3. $i \leftarrow j - 1$
4. **while** $i > 0$ and $A[i] >$ key **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i - 1$
7. $A[i+1] \leftarrow$ key

- Initialisierung: $j = 2, A[1..1]$ ist sortiert
- Invariante: $A[1..j-1]$ ist sortiert
- Austritt: $A[1..\text{length}[A]]$ ist sortiert



Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Problem

Wir können nicht genau sagen, wie häufig eine Rekursion ausgeführt wird.

Abhilfe

Rekursion ist das Gegenstück zu Induktion. Man kann daher die Korrektheit leicht per Induktion zeigen.

Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Induktion über n):

(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Induktion über n):
(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.
(I.V.) Für $n - 1 > 0$ berechnet Sum(A,$n - 1$) die Summe der ersten $n - 1$ Einträge von A.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Korrektheitsbeweise - Rekursionen

Hier: Induktionsschritt von $n-1$ nach n. Dies ist leichter, weil es den Beweis der Rekursion im Algorithmus anpasst.

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Indukt

(I.A.) Ist $n = 1$, so g Algorithmus in Zeile 1 den Wert $A[1]$ zurüc Dies ist korrekt.

(I.V.) Für $n - 1 > 0$ berechnet Sum(A,$n-1$) die Summe der ersten $n-1$ Einträge von A.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Induktion über n):
(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.
(I.V.) Für $n - 1 > 0$ berechnet Sum(A,$n - 1$) die Summe der ersten $n - 1$ Einträge von A.
(I.S.) Wir betrachten den Aufruf von Sum(A, n).

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Induktion über n):
(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.
(I.V.) Für $n - 1 > 0$ berechnet Sum(A,$n - 1$) die Summe der ersten $n - 1$ Einträge von A.
(I.S.) Wir betrachten den Aufruf von Sum(A, n). Da $n > 1$ ist, wird der **else**-Fall der ersten **if**-Anweisung aufgerufen. Dort wird W auf Sum($A, n - 1$) gesetzt.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Induktion über n):
(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.
(I.V.) Für $n - 1 > 0$ berechnet Sum(A,$n - 1$) die Summe der ersten $n - 1$ Einträge von A.
(I.S.) Wir betrachten den Aufruf von Sum(A, n). Da $n > 1$ ist, wird der **else**-Fall der ersten **if**-Anweisung aufgerufen. Dort wird W auf Sum($A, n - 1$) gesetzt. Nach I.V. ist dies die Summe der ersten $n - 1$ Einträge von A.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Induktion über n):
(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.
(I.V.) Für $n - 1 > 0$ berechnet Sum(A,$n-1$) die Summe der ersten $n-1$ Einträge von A.
(I.S.) Wir betrachten den Aufruf von Sum(A, n). Da $n > 1$ ist, wird der **else**-Fall der ersten **if**-Anweisung aufgerufen. Dort wird W auf Sum($A, n-1$) gesetzt. Nach I.V. ist dies die Summe der ersten $n-1$ Einträge von A. Nun wird in Zeile 4 $A[n] + W$, also die Summe der ersten n Einträge von A zurückgegeben.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Korrektheitsbeweise - Rekursionen

$\mathrm{Sum}(A, n)$

1. **If** $n = 1$ **then return** $A[1]$
2. **else**
3. $\quad W \leftarrow \mathrm{Sum}(A, n-1)$
4. $\quad$ **return** $A[n] + W$

Beweis (Induktion über n):
(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.
(I.V.) Für $n - 1 > 0$ berechnet Sum(A,$n-1$) die Summe der ersten $n-1$ Einträge von A.
(I.S.) Wir betrachten den Aufruf von Sum(A, n). Da $n > 1$ ist, wird der **else**-Fall der ersten **if**-Anweisung aufgerufen. Dort wird W auf Sum$(A, n-1)$ gesetzt. Nach I.V. ist dies die Summe der ersten $n-1$ Einträge von A. Nun wird in Zeile 4 $A[n] + W$, also die Summe der ersten n Einträge von A zurückgegeben.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge eines Feldes A.

Zusammenfassung - Korrektheitsbeweise

- Grundannahme der Korrektheitsbeweise ist die korrekte Ausführung der Pseudocode Befehle
- Keine Schleifen: „Schrittweises Nachvollziehen des Programms“
- Schleifen: Korrektheit mittels Invarianten und Induktion
- Rekursion: Korrektheit mittels Induktion

Weiteres Beispiel: Zähle Auftreten eines Schlüssel

Count(Array A, Key k)

1. $count \leftarrow 0$
2. **for** $j \leftarrow 1$ **to** length$[A]$ **do**
3. **if** $A[j] = key$ **then** $count \leftarrow count + 1$
4. **return** $count$

Invariante: $count$ zählt die Auftreten von k in $A[1..j-1]$.

Weiteres Beispiel: Zähle Auftreten eines Schlüssel

CountRec(Array A, Key k, Int n)

1. **If** $n = 1$ **then** $count \leftarrow 0$
2. **else** $count \leftarrow \text{CountRec}(A, k, n-1)$
3. **If** $A[n] = key$ **then** $count \leftarrow count + 1$
4. **return** $count$

Induktion: $\text{CountRec}(A, k, \text{n})$ zählt Auftreten von *k* in *A[1…n]*.