

KORREKTHEITSBEWEIFE VON ALGORITHMEN

Grundlagen der Algorithmen Analyse

Inhalt

- Wie beschreibt man einen Algorithmus?
- **Rechenmodell**
- **Laufzeitanalyse (Zeitkomplexität)**
- **Speicherplatzanalyse (Raumkomplexität)**
- **Wie beweist man die Korrektheit eines Algorithmus?**

Was ist ein mathematischer Beweis?

Informale Definition

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

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

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 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

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

Erster Korrektheitsbeweis: Rückgabe ist $10+n$

Ein triviales Beispiel

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:

Erster Korrektheitsbeweis: Rückgabe ist $10+n$

Ein triviales Beispiel

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.

Erster Korrektheitsbeweis: Rückgabe ist $10+n$

Ein triviales Beispiel

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.***

Ein triviales Beispiel

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.***

Erster Korrektheitsbeweis: Rückgabe ist $10+n$

Ein triviales Beispiel

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.***

Erster Korrektheitsbeweis: Rückgabe ist $10+n$

Ein triviales Beispiel

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.***

Erster Korrektheitsbeweis: Rückgabe ist $10+n$

Ein triviales Beispiel

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.**

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 Korrektheit der Behauptung.

Korrektheitsbeweis: Maximumssuche

Ein erstes nicht triviales Beispiel

Algorithmus Max-Search(Array A)

1. $\text{max} \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
3. **if** $A[j] > A[\text{max}]$ **then** $\text{max} \leftarrow j$
4. **return** max // ist der Index eines maximalen Elements

Aufgabe:

Korrektheitsbeweis muss das Programm Schritt für Schritt nachvollziehen

Problem

Wir wissen nicht, wie viele Durchläufe die **for**-Schleife benötigt.

Das hängt von der Eingabelänge ab.

Korrektheitsbeweis: Maximumssuche

Ein erstes nicht triviales Beispiel

Algorithmus Max-Search(Array A)

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

Abhilfe

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

Zustand

Werte der Variablen des Programms

Korrektheitsbeweis: Schleifeninvariante

Ein erstes nicht triviales Beispiel

Algorithmus Max-Search(Array A)

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

Definition **Schleifeninvariante**

Eine Schleifeninvariante ist eine Aussage $A(i)$, die

- Zu Beginn des ersten Durchlaufs gilt (Initialisierung)
- Zu Beginn des i -ten Schleifendurchlaufs gilt (in Abhängigkeit von i)

Und eine Aussage über den Zustand nach Ablauf der Schleife erlaubt, den Austrittszustand.

Korrektheitsbeweis: Schleifeninvariante

Ein erstes nicht triviales Beispiel

Algorithmus Max-Search(Array A)

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

Schleifeninvariante – Austrittszustand

Eine Schleife wird beendet, wenn die Schleifenbedingung nicht mehr gilt.
Dieser Zustand wird als Austrittszustand bezeichnet.

D.h. im Austrittszustand ist der Wert von j mit dem die Schleife verlassen wird

D.h. $j = \text{length}(A) + 1$

Korrektheitsbeweis: Schleifeninvariante

Ein erstes nicht triviales Beispiel

Algorithmus Max-Search(Array A)

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

*Schleifeninvariante – Konventionen für **for**-Schleifen*

Bei einer for-Schleife nehmen wir an:

- Laufvariablen werden am Ende des Schleifendurchlaufs erhöht
- Bei der Initialisierung die Laufvariablen mit dem Startwert initialisiert sind.

Somit kann (und sollte) die Invariante von der Laufvariablen abhängen.

Korrektheitsbeweis: Maximumssuche

Ein erstes nicht triviales Beispiel

Abkürzung:

$A[1..j-1]$ entspricht $A[1, \dots, j-1]$

Algorithmus Max-Search(Array A)

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

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende **Schleifeninvariante**:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis:

Vor der Schleife setzt der Befehl in Zeile 1 max auf 1.

Wir zeigen in Abhängigkeit von der Laufvariable j , dass die Invariante erfüllt ist.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis:

Vor der Schleife setzt der Befehl in Zeile 1 max auf 1.

Wir zeigen in Abhängigkeit von der Laufvariable j , dass die Invariante erfüllt ist.

(Erster Schritt) Zur Initialisierung der Schleife ist $\text{max}=1$ und $j=2$.

$A[1..1]$ enthält nur ein Element, nämlich $A[1]$.

Da $A[\text{max}] = A[1]$ ist, ist $A[\text{max}]$ ein größtes Element aus $A[1..1]$.

=> Die Invariante gilt zur Initialisierung.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis:

Vor der Schleife setzt der Befehl in Zeile 1 max auf 1.

Wir zeigen in Abhängigkeit von der Laufvariable j , dass Invariante erfüllt ist.

(Erster Schritt) Zur Initialisierung der Schleife ist $\text{max}=1$ und $j=2$.

$A[1..1]$ enthält nur ein Element, nämlich $A[1]$.

Da $A[\text{max}] = A[1]$ ist, ist $A[\text{max}]$ ein größtes Element aus $A[1..1]$.

=> Die Invariante gilt zur Initialisierung.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

(Voraussetzung) Sei die Invariante erfüllt für $j=j_0 < \text{length}(A)+1$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

(Voraussetzung) Sei die Invariante erfüllt für $j=j_0 < \text{length}(A)+1$.

Zu zeigen: Die Invariante ist erfüllt für $j + 1$. („Induktionsschritt“: $j \rightarrow j+1$)

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

(Voraussetzung) Sei die Invariante erfüllt für $j=j_0 < \text{length}(A)+1$.

Zu zeigen: Die Invariante ist erfüllt für $j+1$. („Induktionsschritt“: $j \rightarrow j+1$)

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

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

(Voraussetzung) Sei die Invariante erfüllt für $j=j_0 < \text{length}(A)+1$.

Zu zeigen: Die Invariante ist erfüllt für $j+1$. („Induktionsschritt“: $j \rightarrow j+1$)

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

Falls $A[j] \leq A[\text{max}]$ ist, so wird die **then**-Anweisung nicht ausgeführt.

Dann ist $A[\text{max}]$ auch größtes Element aus $A[1..j]$.

Am Ende der Schleife wird j um 1 erhöht.

Somit gilt die Aussage auch für $j+1$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

(Voraussetzung) Sei die Invariante erfüllt für $j=j_0 < \text{length}(A)+1$.

Zu zeigen: Die Invariante ist erfüllt für $j+1$. („Induktionsschritt“: $j \rightarrow j+1$)

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

Falls $A[j] \leq A[\text{max}]$ ist, so wird die **then**-Anweisung nicht ausgeführt.

Dann ist $A[\text{max}]$ auch größtes Element aus $A[1..j]$.

Am Ende der Schleife wird j um 1 erhöht.

Somit gilt die Aussage auch für $j+1$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

Falls $A[j] > A[\text{max}]$ ist, so ist nach I.V.

$A[j]$ größer als das größte Element aus $A[1..j-1]$
somit ist $A[j]$ das größte Element aus $A[1..j]$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

Falls $A[j] > A[\text{max}]$ ist, so ist nach I.V.

$A[j]$ größer als das größte Element aus $A[1..j-1]$
somit ist $A[j]$ das größte Element aus $A[1..j]$.

In der **then**-Anweisung wird $\text{max}=j$ gesetzt.

Damit ist $A[\text{max}]$ das größte Element aus $A[1..j]$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

Falls $A[j] > A[\text{max}]$ ist, so ist nach I.V.

$A[j]$ größer als das größte Element aus $A[1..j-1]$

somit ist $A[j]$ das größte Element aus $A[1..j]$.

In der **then**-Anweisung wird $\text{max}=j$ gesetzt.

Damit ist $A[\text{max}]$ das größte Element aus $A[1..j]$.

Am Ende der Schleife wird j um 1 erhöht.

Damit gilt die Aussage auch für $j + 1$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

Falls $A[j] > A[\text{max}]$ ist, so ist nach I.V.

$A[j]$ größer als das größte Element aus $A[1..j-1]$

somit ist $A[j]$ das größte Element aus $A[1..j]$.

In der **then**-Anweisung wird $\text{max}=j$ gesetzt.

Damit ist $A[\text{max}]$ das größte Element aus $A[1..j]$.

Am Ende der Schleife wird j um 1 erhöht.

Damit gilt die Aussage auch für $j + 1$.

Korrektheitsbeweis: Maximumssuche

Lemma

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Invariante) $A[\text{max}]$ ist ein größtes Element aus $A[1..j-1]$.

Beweis(fortgesetzt):

Das entspricht dem Prinzip der vollständigen Induktion: Ist die Invariante vor jedem Schleifendurchlauf und vor dem Schleifenaustritt erfüllt, gilt sie für jeden Schleifendurchlauf und insbesondere am Ende der Schleife!

Korrektheitsbeweis: Maximumssuche

Korrektheit von Max-Search

Für den Algorithmus Max-Search gilt folgende Korrektheitsaussage:

$A[\text{Max-Search}(A)]$ ist ein größtes Element aus $A[1..\text{length}(A)]$.

Beweis:

Durch Ausnutzen der Schleifeninvariante

Am Ende der Schleife gilt:

$A[\text{max}]$ ist ein größtes Element aus $A[1..(\text{length}(A) + 1) - 1]$.

d.h. $A[\text{max}]$ ist ein größtes Element aus $A[1..\text{length}(A)]$.

Korrektheitsbeweis: Maximumssuche

Ein erstes nicht triviales Beispiel

Algorithmus Max-Search(Array A)

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

Invarianten bei der Programmierung

Invarianten sollten zur Kommentierung von Schleifen benutzt werden.

Diese kann man u.a. mit Hilfe von „Assertions“ zur Laufzeit überprüfen.

AUSFLUG: VOLLSTÄNDIGE INDUKTION

Vollständige Induktion

- **Zu zeigen:** $p: N_0 \rightarrow \text{Boolean}$ (Prädikat)
 $N_0 = \{0, 1, \dots\}$
- **Induktionsanfang:** Zu beweisen:
 $p(0)$ ist WAHR
- **Induktionsvoraussetzung:** Für alle $n \in N_0$, mit $n \leq n_0$ gilt:
 $p(n)$ ist WAHR
- **Induktionsschritt:** Zu beweisen ist:
 $p(n)$ ist WAHR für $n \leq n_0 \Rightarrow p(n+1)$ ist WAHR
- **Induktionsschluss:** Für alle $n \in N_0$ gilt
 $p(n)$ ist WAHR.

Beispiel für vollständigen Induktion

Zu zeigen: Für alle n in N_0 gilt

$$p(n): 1 + 2 + 3 + \dots + (n - 1) + n = \frac{n(n+1)}{2}$$

Beispiel für vollständigen Induktion

$$p(n): 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$\text{Induktionsanfang: } 1 = \frac{1(1+1)}{2} = \frac{2}{2}$$

Induktionsschritt:

$$\begin{aligned} 1 + 2 + 3 + \dots + n + (n+1) &= \frac{n(n+1)}{2} + (n+1) \\ &= (n+1)\left(\frac{n}{2} + 1\right) \\ &= \frac{(n+1)(n+2)}{2} \end{aligned}$$

Beispiel für vollständigen Induktion

Induktionsschluss: Für alle n in N_0

$$1 + 2 + 3 + \dots + (n - 1) + n = \frac{n(n+1)}{2}$$

KORREKTHEIT INSERTIONSORT

Insertion Sort

InsertionSort(Array A)

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

➤ Eingabegröße n

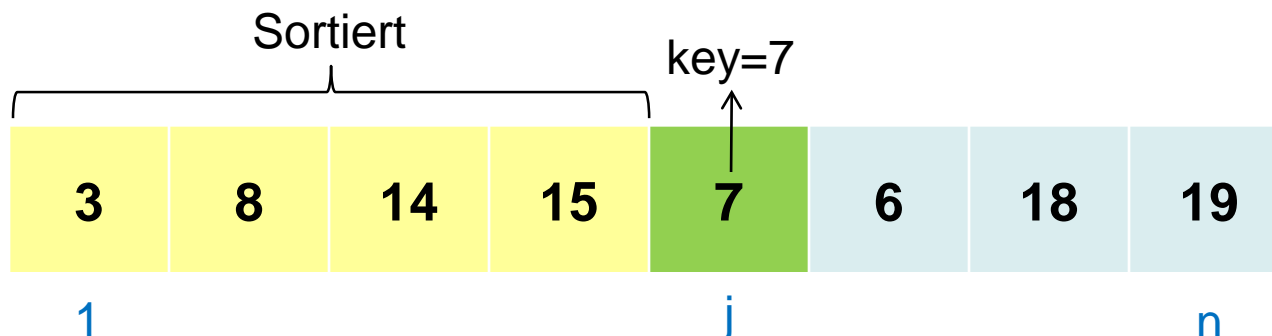
➤ $\text{length}(A) = n$

➤ verschiebe alle Elemente aus

➤ $A[1 \dots j-1]$, die größer als key

➤ sind eine Stelle nach rechts

➤ Speichere key in Lücke



Korrektheitsbeweis: Insertion Sort

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

Lemma

Die **for**-Schleife im Algorithmus Insertion Sort erfüllt folgende **Schleifeninvariante**:

(Invariante) $A[1..j-1]$ ist die sortierte Permutation von $A[1..j-1]$.
 $A[1..j-1]$ ist „sortiert“

Invarianten 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
```

➤ Initialisierung: $j=2$, $A[1..1]$ ist sortiert

➤ **Invariante:** $A[1..j-1]$ ist „sortiert“

➤ Austritt: $A[1..length(A)]$ ist sortiert

Invarianten 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
```

➤ Initialisierung: $j=2$, $A[1..1]$ ist sortiert

➤ **Invariante:** $A[1..j-1]$ ist „sortiert“

➤ Austritt: $A[1..length(A)]$ ist sortiert

Lemma

Die **while**-Schleife in Insertion Sort erfüllt folgende **Schleifeninvariante**:

(Invariante) $A[1..j-1] \setminus A[i+1]$ ist „sortiert“ und $A[i+1..j-1] > key$.

Invarianten 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
```

➤ Initialisierung: $j=2$, $A[1..1]$ ist sortiert

➤ **Invariante:** $A[1..j-1]$ ist „sortiert“

```
5.      A[i+1] ← A[i]
```

➤ Inv: $A[1..j-1] \setminus A[i+1]$ ist „sortiert“

➤ $A[i..j-1] > \text{key}$

```
6.      i ← i-1
```

```
7.      A[i+1] ← key
```

➤ Austritt: $A[1..\text{length}(A)]$ ist sortiert

Invarianten 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
```

➤ Initialisierung: $j=2$, $A[1..1]$ ist sortiert

➤ **Invariante:** $A[1..j-1]$ ist „sortiert“

➤ Initialisierung: $i=j-1$, $A[1..j-1]$ ist
➤ sortiert und $A[j] > \text{key}$

➤ **Inv:** $A[1..j-1] \setminus A[i+1]$ ist „sortiert“

➤ $A[i..j-1] > \text{key}$

```
7.  A[i+1] ← key
```

➤ Austritt: $A[1..\text{length}(A)]$ ist sortiert

Invarianten InsertionSort

InsertionSort(Array A)

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

- Initialisierung: $j=2$, $A[1..1]$ ist sortiert
- **Invariante:** $A[1..j-1]$ ist „sortiert“
- Initialisierung: $i=j-1$, $A[1..j] \setminus A[j]$ ist sortiert und $A[j] > \text{key}$
- **Inv:** $A[1..j-1] \setminus A[i+1]$ ist „sortiert“
- $A[i..j-1] > \text{key}$
- Austritt: $A[1..j] \setminus A[i+1]$ ist „sortiert“ und
- $A[i] \leq \text{key} < A[i+2]$ (wenn $i+1=j$, dann gilt die letzte Ungl. nicht unbedingt)
- oder $i=0$ und $\text{key} < A[2]$
- Austritt: $A[1..\text{length}(A)]$ ist „sortiert“

Korrektheitsbeweis: Insertion Sort

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

➤ $A[1 \dots j-1]$, die größer als key

➤ sind eine Stelle nach rechts

➤ Speichere key in Lücke

Korrektheit von InsertionSort

Für den Algorithmus Insertion Sort gilt folgende **Korrektheitsaussage**:

Nach dem Aufruf von **InsertionSort**($A[1..\text{length}(A)]$) ist A „sortiert“

Beweis: Durch Anwendung der äußeren Schleifeninvariante.