Einführung in die Programmierung

Algorithmen, Pseudocode, Sortieren

Heutige Themen

- Kommentare zum C-Kurs
- Hinweise zur Tutorienorganisation
- Motivation: Datenstrukturen und Algorithmen
- Algorithmen und Pseudo Code
- Unser erster Sortieralgorithmus

C-Kurs

C-Kurs: Wie wars?

Organisation

Organisation – Termine

Was	Wie	Kommentar	Termin
Mantelbogen	Prüfungsamt	Registrierung im Prüfungsamt	ASAP
Modulanmeldung	Prüfungsamt	Qispos Formular (Gelber Zettel)	Bis 04.11.2015
Hausaufgaben-anmeldung	OSIRIS	Tutorienplatzbestätigung	Bis 29.10.2015 21:59
Tutorientausch	OSIRIS	Timeslotmatch führt zu Tausch	Bis 02.11.2015 12:59
Zusätzliche Tutorienplätze	OSIRIS	Für Teilnehmer ohne Moses Tutorium	30.11.2015 – 02.11.2015 12:59
Bestätigung Tutoriumslot	OSIRIS	Nachschauen!	Ab 03.11.2015 18:00

Organisation – Termine

Was	Wie	Kommentar	Termin
Tutorienstart		Hingehen☺	Ab 04.11.2015
Gruppenbildung	OSIRIS	Selber mit TubIT-login des Gruppenpartners (per Einladung) Gruppenbildung im Tutorium	Ab 04.11.2015 Ca. bis 09.11.2015
Gruppenbildung	durch Tutor	Sichtbar in OSIRIS	Danach

Motivation
Was ist Informatik?

Vorwort

- Informatik ist ein Kunstwort aus
  - Information und Automatik
  - Information und Mathematik
- Informatik hat viele Facetten
- Einige definieren Informatik darüber was Informatiker machen
  - Viele, viele Bereiche
  - Von Büro, Maschinenhalle, Unterhaltungsbranche, …
- Informatik bedeutet für viele was anderes, …

Informatik / Computer Science?

- Marvin Minsky: "Computer science has such intimate relations with so many other subjects that it is hard to see it as a thing in itself."

- Juris Hartmanis: "Computer science differs from the known sciences so deeply that it has to be viewed as a new species among the sciences."

Informatik / Computer Science?

- Donald Knuth: "The "study" of algorithms"
  - Algorithms ≈ what you can teach a computer
  - Which functions can be efficiently computed?
  - Need a computer to find out!

- Juris Hartmanis: "Study of **information**"
  - How to represent info
  - How to process info
  - And the machines that do this

Informatik / Computer Science?

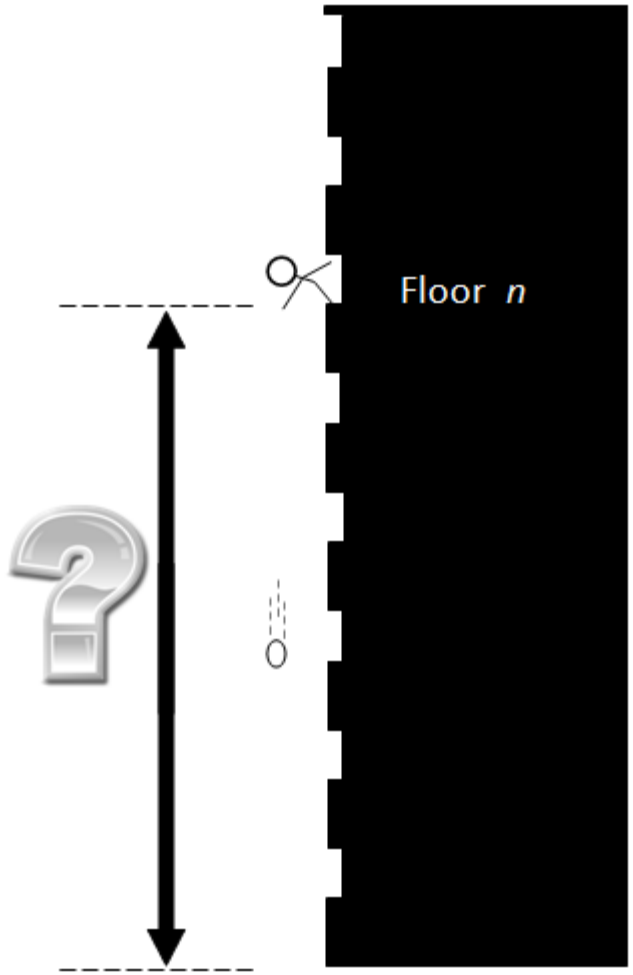
- Fred Brooks:
  "CS ≠ science"
  "CS = **engineering**"
  - Concerned with "making": Physical computers; S/W systems
- Peter Denning: "Computing is a 4th great domain of science alongside the physical, life, and social sciences."
- Informatik =
  - discovery (science)
  - & implementation (engineering)
  - of information processes

Algorithmik
Ein wesentlicher Teil der Informatik

Denkanstoß

- Eierfall Problem:
  Gegeben: Hochhaus mit 100 Stockwerken und zwei Eier
  Gesucht: Das höchste Stockwerk aus dem man ein Ei fallen lassen kann, so dass es nicht zerbricht

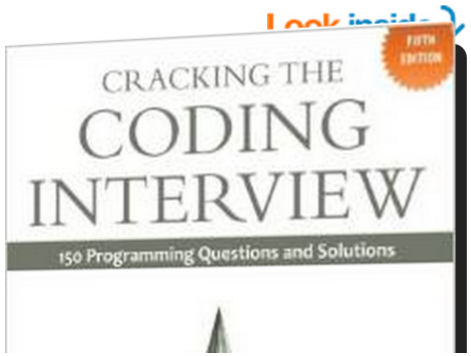
- Wie???



Hintergrund des Denkanstoß

- Frage aus „technischen Bewerbungsgesprächen“ für Softwareengineering Positionen

- Siehe auch:

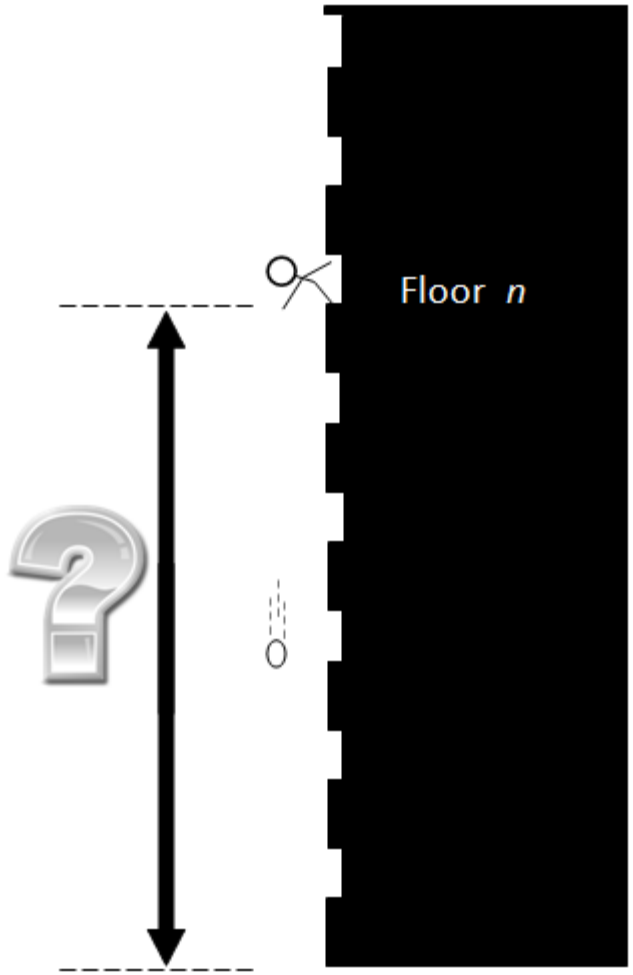


Cracking the Coding Interview: 150 Programming Questions and Solutions Paperback – August 22, 2011
by Gayle Laakmann McDowell (Author)
396 customer reviews
#1 Best Seller in Job Interviewing
ISBN-13: 978-0984782802 | ISBN-10: 098478280X | Edition: 5th Revised & enlarged

Denkanstoß

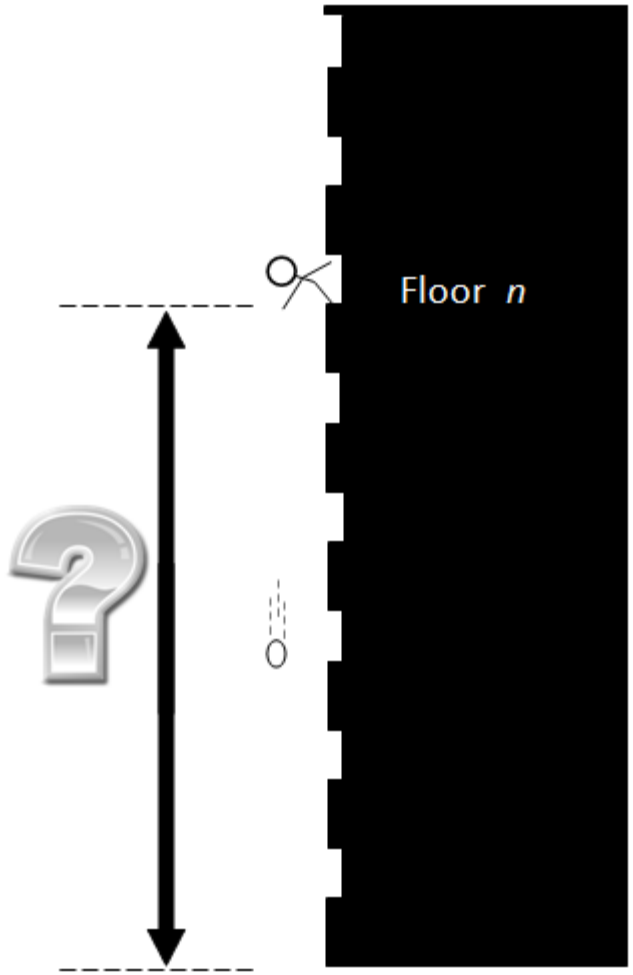
- Eierfall Problem:
  Gegeben: Hochhaus mit 100 Stockwerken und zwei Eier
  Gesucht: Das höchste Stockwerk aus dem man ein Ei fallen lassen kann, so dass es nicht zerbricht

- Wie???



Denkanstoß – Zwischenschritt

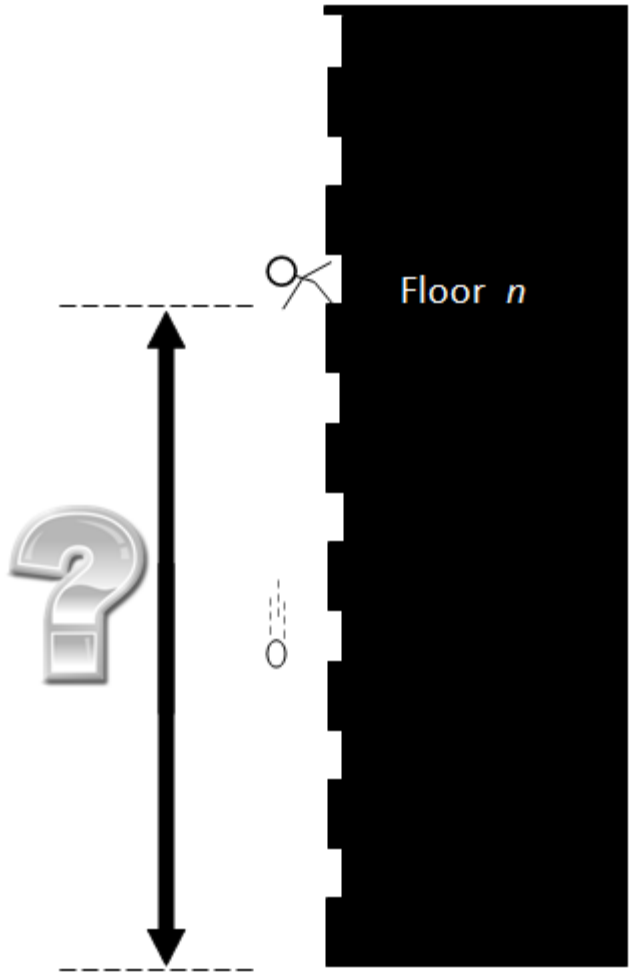
- Eierfall Problem:
  Gegeben: Hochhaus mit 100 Stockwerken und ein Ei
  Gesucht: Das höchste Stockwerk aus dem man ein Ei fallen lassen kann, so dass es nicht zerbricht



- Wie???

Denkanstoß – Zwischenschritt

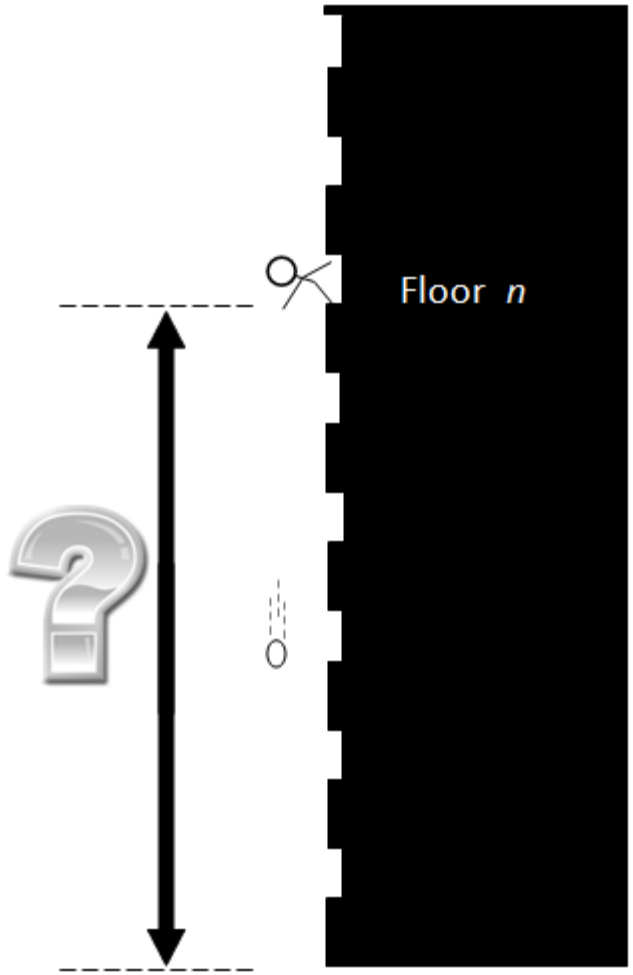
- Eierfall Problem:
  Gegeben: Hochhaus mit 100 Stockwerken und ein Ei
  Gesucht: Das höchste Stockwerk aus dem man ein Ei fallen lassen kann, so dass es nicht zerbricht



- Hinweis: Man kann das Ei mehrfach fallenlassen....

Denkanstoß

- Eierfall Problem:
  Gegeben: Hochhaus mit 100 Stockwerken und zwei Eier
  Gesucht: Das höchste Stockwerk aus dem man ein Ei fallen lassen kann, so dass es nicht zerbricht



- Wie – gegeben die Möglichkeiten mit einem Ei?

Warum solche Fragen?

- Algorithmisches Denken ist die Grundlage der Informatik
- D.h. Algorithmen und Datenstrukturen findet man überall in der Informatik wieder
- In dieser Vorlesung
  - Basis Datenstrukturen
  - Basis Algorithmen
- Siehe z.B: http://datagenetics.com/blog/july22012/index.html

Themengebiete in der Informatik

- Datenbank
- Kommunikation
- Graphik
- Robotik
- Künstliche Intelligenz
- …

Algorithmen

Programm vs. Algorithmus

- Algorithmen beschreiben was die Computer ausführen sollen in prinzipiellen Elementen
- Programmiersprachen stellen eine Schnittstelle da, um die Algorithmen auf dem Computer ausführen zu können

Programm vs. Algorithmus (2.)

- Algorithmen fokussieren auf Korrektheit, Vollständigkeit, und Komplexität
- Programmiersprachen müssen zusätzlich alle Details des Computers berücksichtigen

Beispiel: Zweier Potenzen

- Berechne die zweier Potenzen bis n:

```
m ← 0;
p ← 1;
while (p < n)
    Ausgabe von: „2^m ist p“;
    m ← m + 1;
    p ← p * 2;
```

- Der Algorithmus ist in Pseudocode beschrieben!

Algorithmus

- Ein Algorithmus ist eine Liste von Anweisungen, die Essenz eines Programms

- Wichtige Aspekte:
  - Korrektheit: Erfüllt der Algorithmus seine Anforderungen?
  - Effizienz: Wie viel Zeit und wie viel Speicherplatz braucht er?
  - Terminierung: Hält der Algorithmus immer an?

Grundlagen der Algorithmen Analyse

Inhalt

- **Wie beschreibt man einen Algorithmus?**
- Rechenmodell
- Laufzeitanalyse
- Wie beweist man die Korrektheit eines Algorithmus?

Wie beschreibt man einen Algorithmus?

Problemstellung

- Menschen wollen über Algorithmen reden, sie beschreiben
- Vergleichen von Algorithmen
- Programmiersprachen benötigen oft viel Code für „nichts“
- Kern des Algorithmus dann oft nicht mehr erkennbar
- Exakter Prosatext ist ebenfalls zu lang

Gesucht: Exakte, kompakte, einfache „Notation“

Pseudocode

- Beschreibungssprache ähnlich wie Java, C, Pascal, etc…
- Losgelöst von spezifischer Programmiersprache/Umgebung
- Manchmal kann auch ein vollständiger Satz die beste Beschreibung sein
- Wir ignorieren dabei Details, wie
  - Variablen Deklaration
  - Include files, …
- Wir ignorieren dabei Software Engineering Aspekte wie
  - Modularität
  - Fehlerbehandlung, …

Pseudocode: Beispiel

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Schleifen (**for**, **while**, **repeat**)

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Schleifen (**for**, **while**, **repeat**)

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Zuweisungen durch ←

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Variablen (z.B. i, j, key) sind lokal definiert

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Keine Typdeklaration, wenn Typ aus dem Kontext klar

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Auch komplexere Datenstrukturen möglich, z.B: Array

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Zugriff auf Feldelemente eines Arrays mit [ ]: z.B: A[1], A[i], A[i + 1], …
- Indexierung beginnt mit 1!

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Datenstrukturen/Objekte können weitere Eigenschaften haben, z.B: Arrays haben Längen
- Zugriff über Funktionen, z.B. Funktion length(A) gibt *Länge* des Arrays A zurück

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Blockstruktur durch Einrücken, d.h. Klammern nicht unbedingt benötigt!

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Bedingte Verzweigungen (**if then else**)

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Bedingte Verzweigungen (**if then else**)
- If summe > 9000 then
  print “over ninethousand“

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Funktionen „call-by-value“: jede aufgerufene Funktion erhält neue Kopie der übergebenen Variable, d.h. lokalen Änderungen sind nicht global sichtbar
- Bei Objekten wird der Zeiger kopiert, lokale Änderungen am Objekt global sichtbar

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Rückgabe von Werten durch **return**

Pseudocode

```
AlgorithmFoo(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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Pseudocode

- Kommentare durch ➢, oder //

Pseudocode in Jobinterviews

Quelle: http://xkcd.com/1185/

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

Beispiel: Sortieren

Sortieren

Problem: Sortieren

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

Beispiel:

- Eingabe: 15, 7, 3, 18, 8, 4
- Ausgabe: 3, 4, 7, 8, 15, 18

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
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

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

Idee InsertionSort

- *Die ersten j-1 Elemente sind sortiert (zu Beginn j=2)*
- *Innerhalb eines Schleifendurchlaufs wird das j-te Element in die sortierte Folge eingefügt*
- *Am Ende ist die gesamte Folge sortiert*

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

Beispiel

8	15	3	14	7	6	18	19

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
```

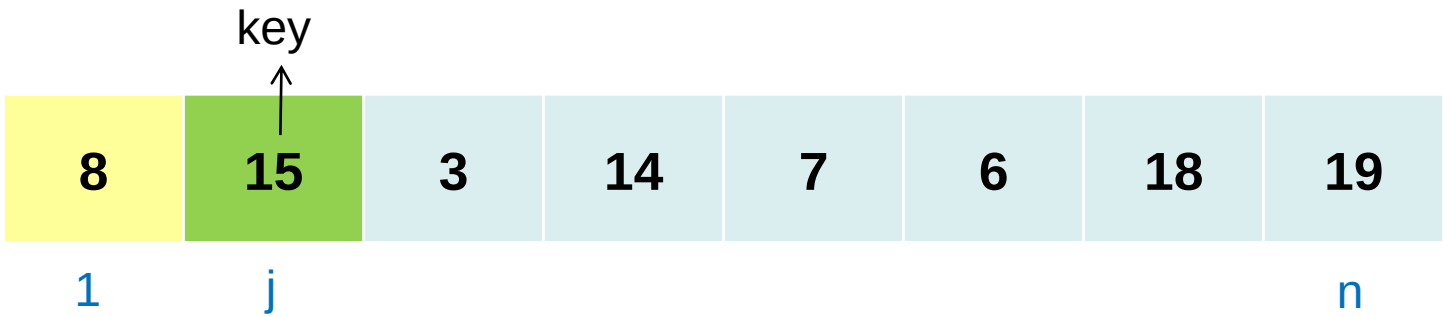
- Eingabegröße n
- length(A) = n

8	15	3	14	7	6	18	19
1	j						n

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
```

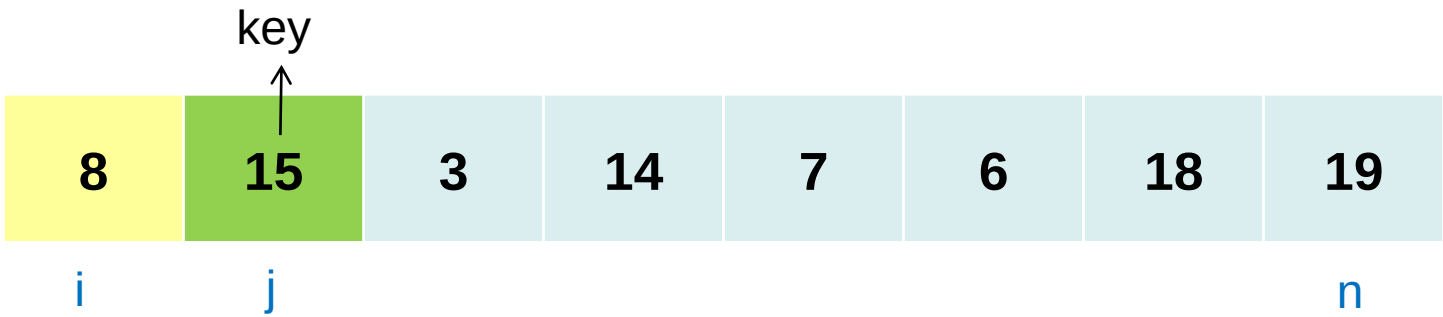
- Eingabegröße n
- length(A) = n



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
```

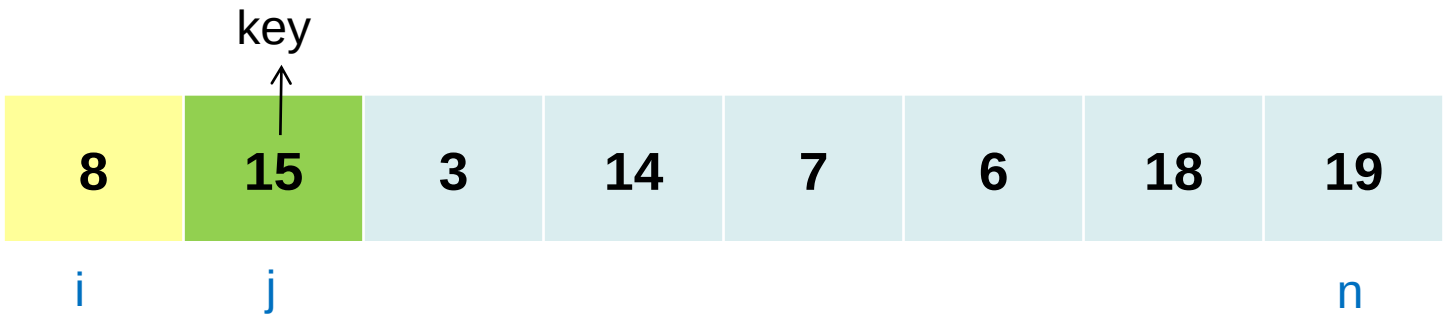
- Eingabegröße n
- length(A) = n



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
```

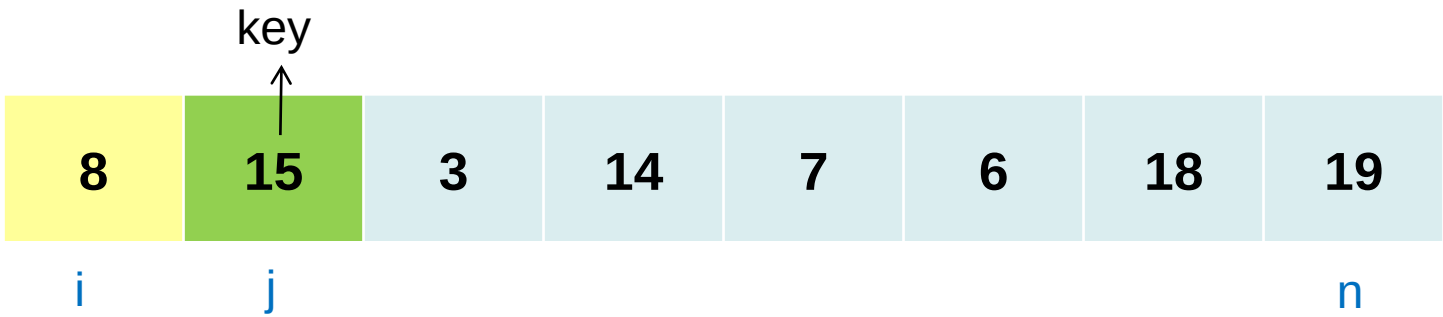
- Eingabegröße n
- length(A) = n



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
- length(A) = n

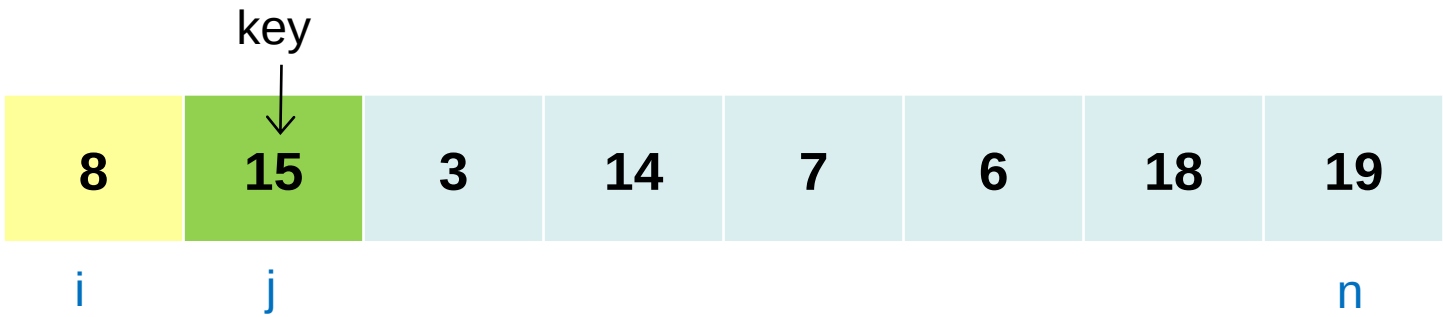


Insertion Sort

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

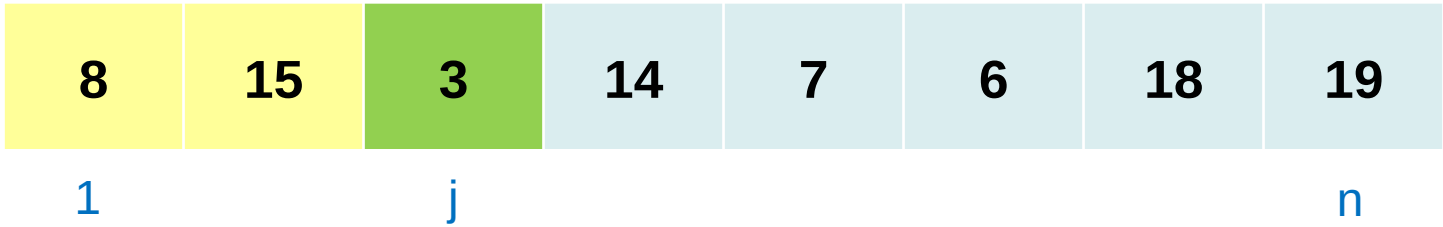
- Eingabegröße n
- length(A) = n



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
- length(A) = n

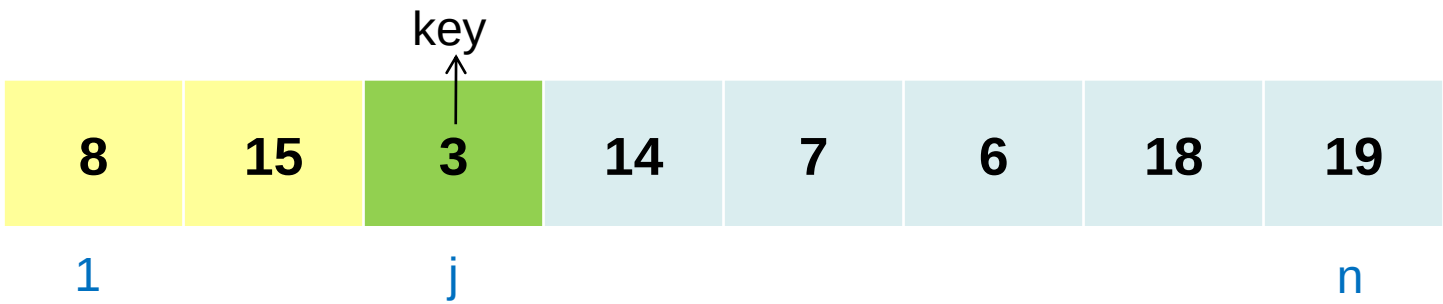


Insertion Sort

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
- length(A) = n



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
- length(A) = n

key=3

8	15	3	14	7	6	18	19
1	i	j					n

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
- length(A) = n

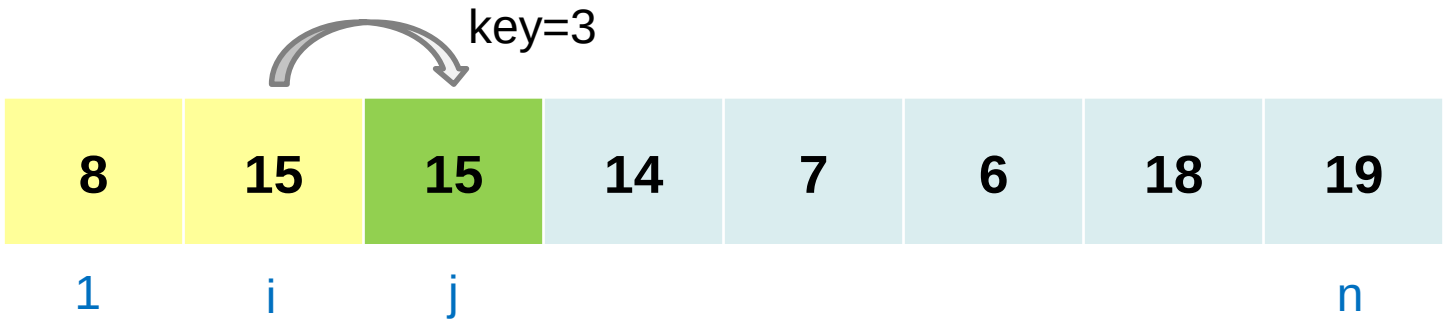
key=3

8	15	3	14	7	6	18	19
1	i	j					n

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
- length(A) = n

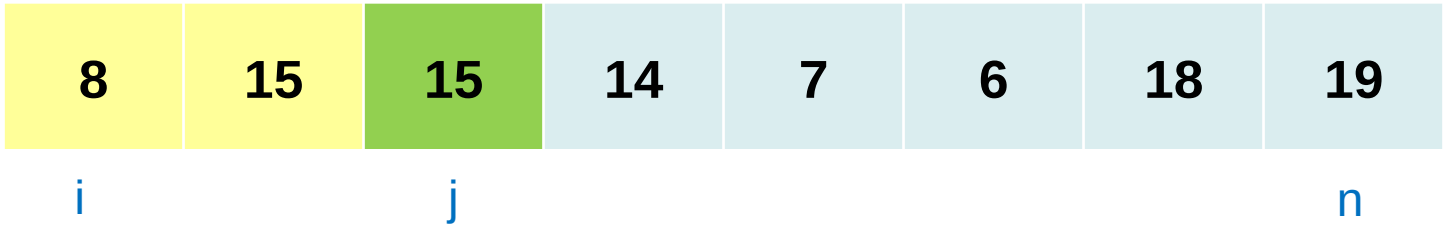


Insertion Sort

```
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
- length(A) = n

key=3



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
- length(A) = n

key=3

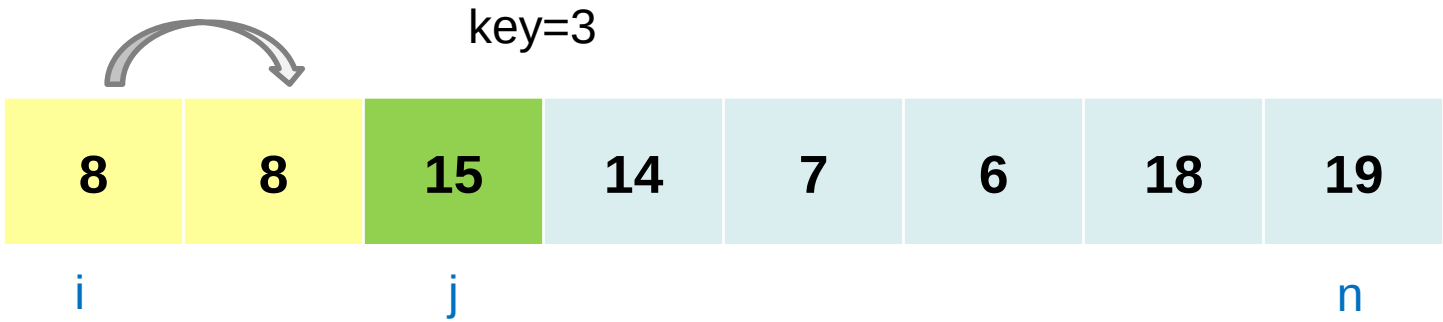
8	15	15	14	7	6	18	19
i		j					n

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
- length(A) = n



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
- length(A) = n

key=3

8	8	15	14	7	6	18	19

i 1 j n

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
- length(A) = n

key=3

8	8	15	14	7	6	18	19

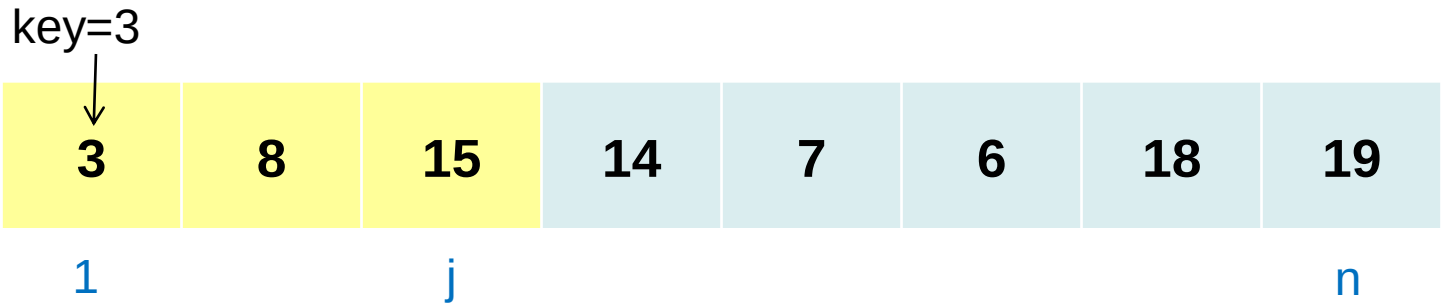
i 1 j n

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

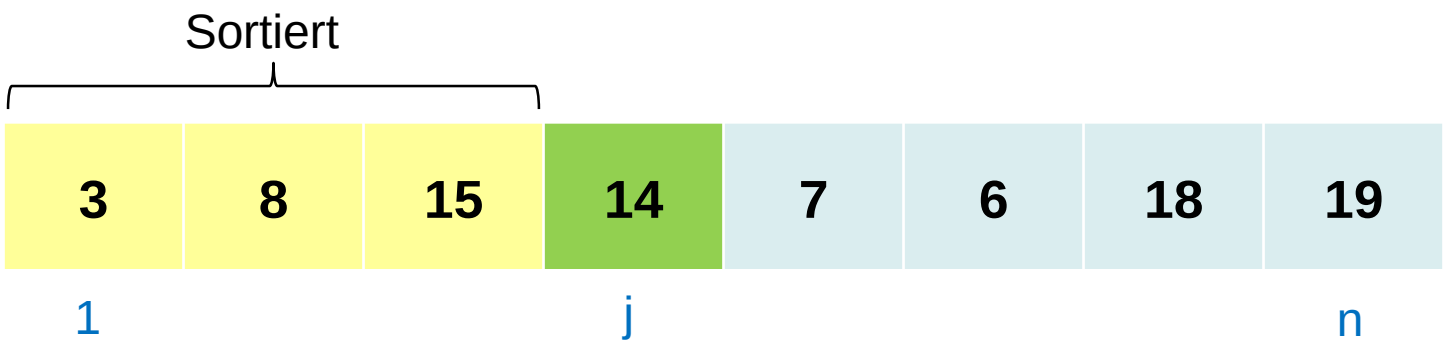
- Eingabegröße n
- length(A) = n



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
```

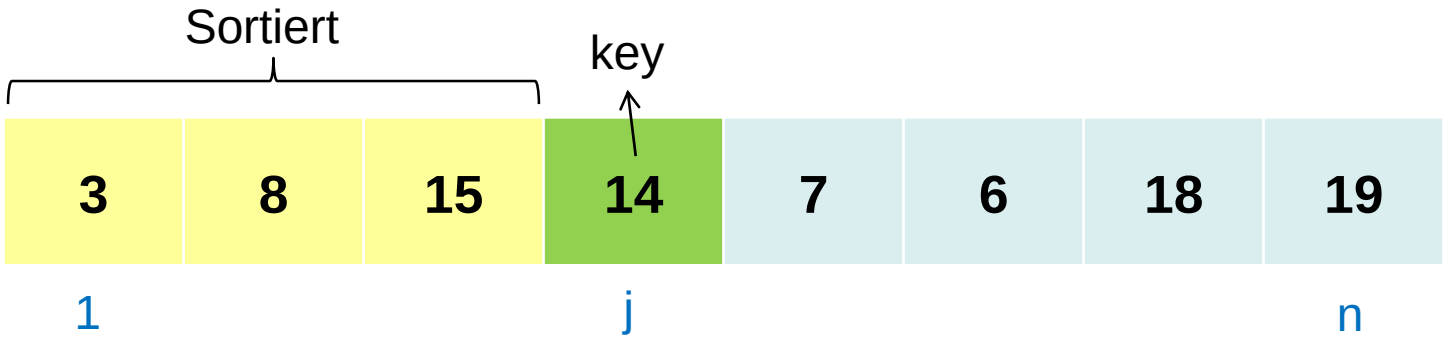
- Eingabegröße n
- length(A) = n



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
```

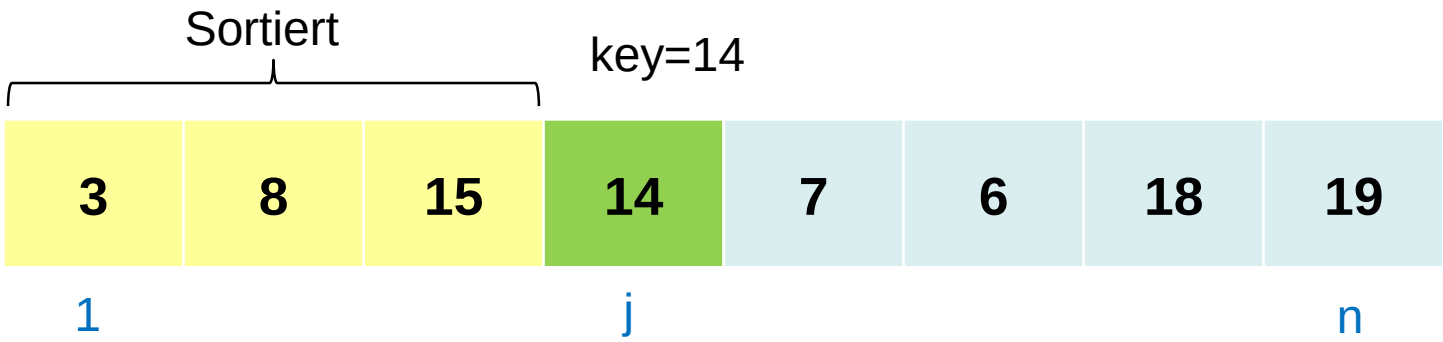
- Eingabegröße n
- length(A) = n



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
- length(A) = n

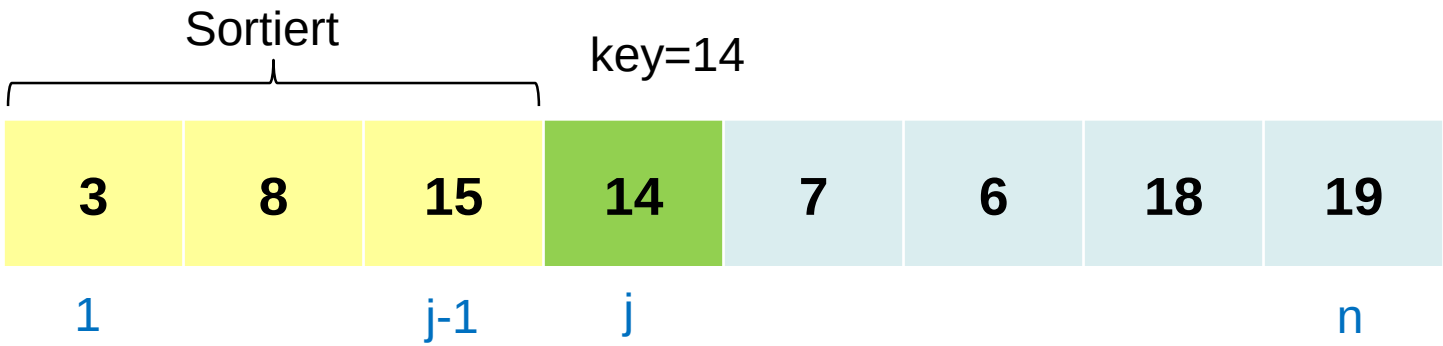


Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

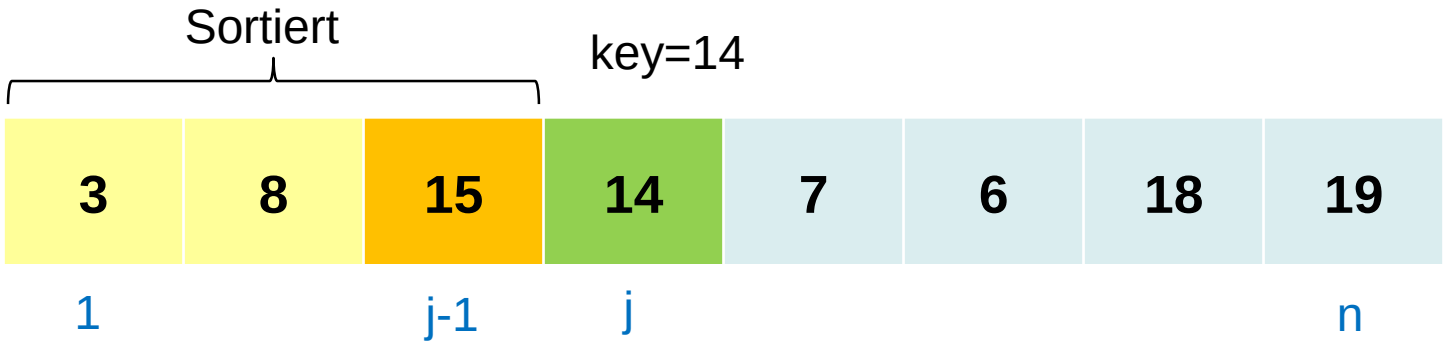


Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts



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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

key=14

3	8	15	15	7	6	18	19
1	i	j-1	j				n

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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

key=14

3	8	15	15	7	6	18	19
1	i	i+1	j				n

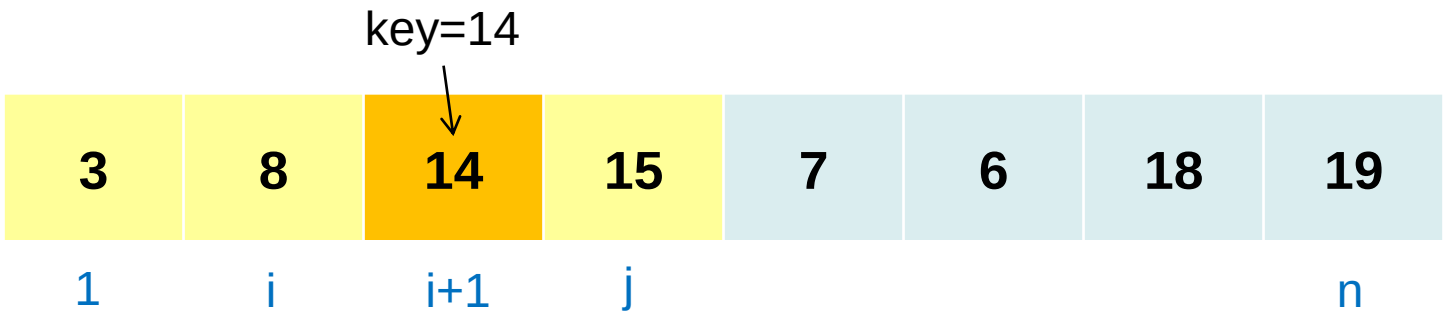
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



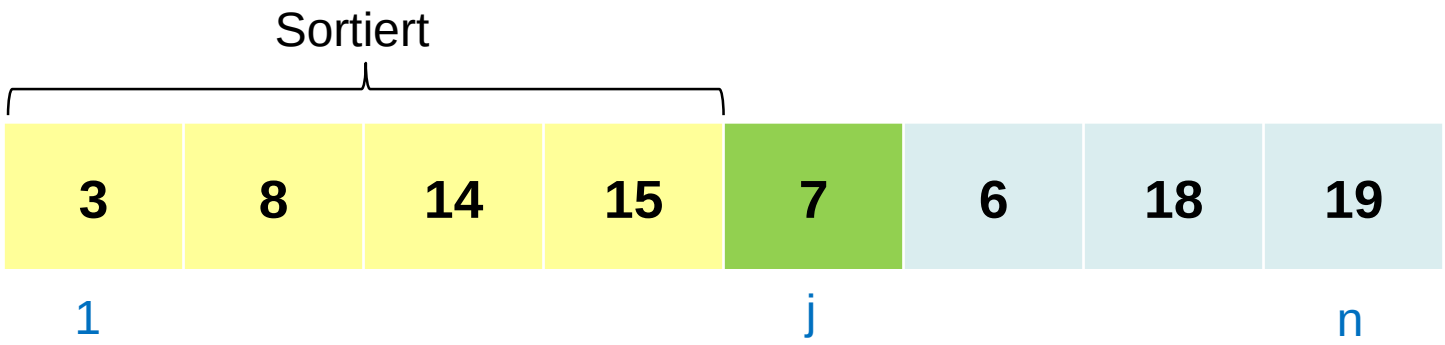
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



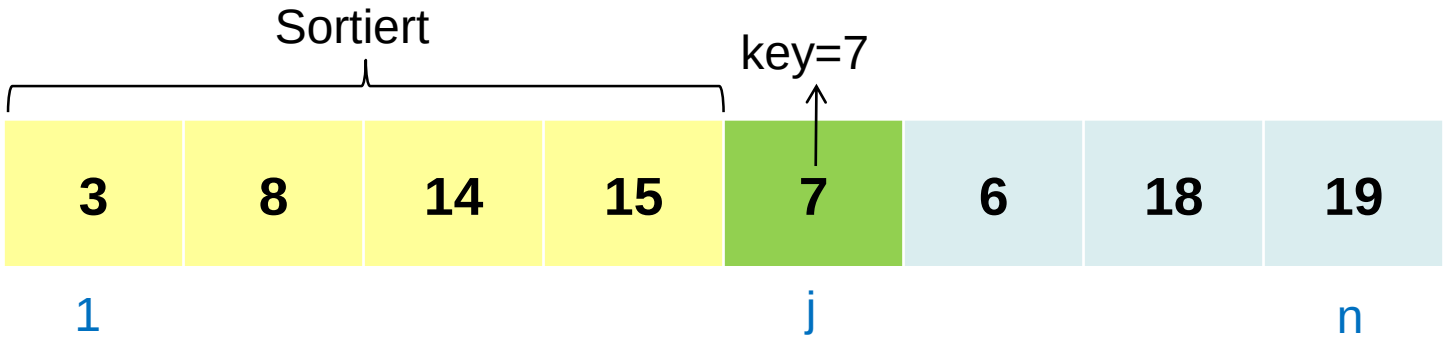
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



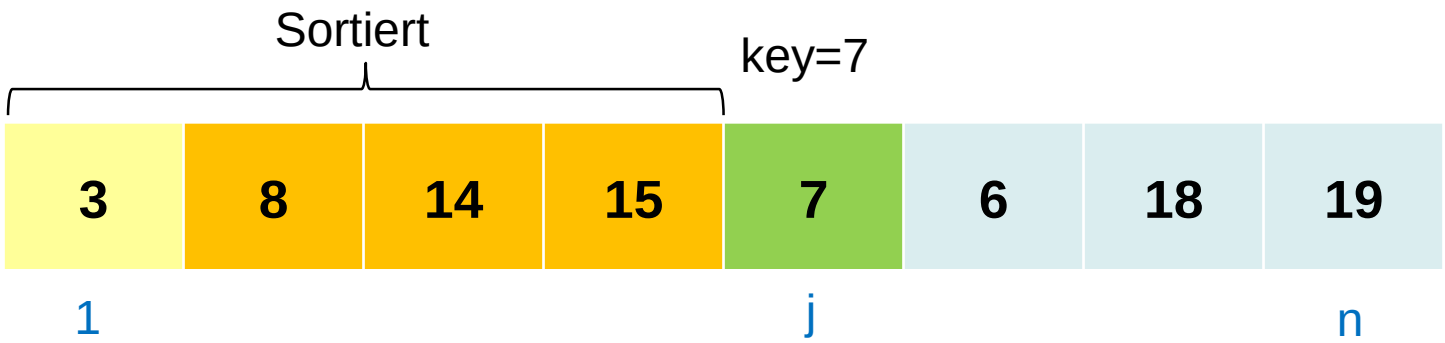
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke

key=7

3	8	8	14	15	6	18	19
1				j			n

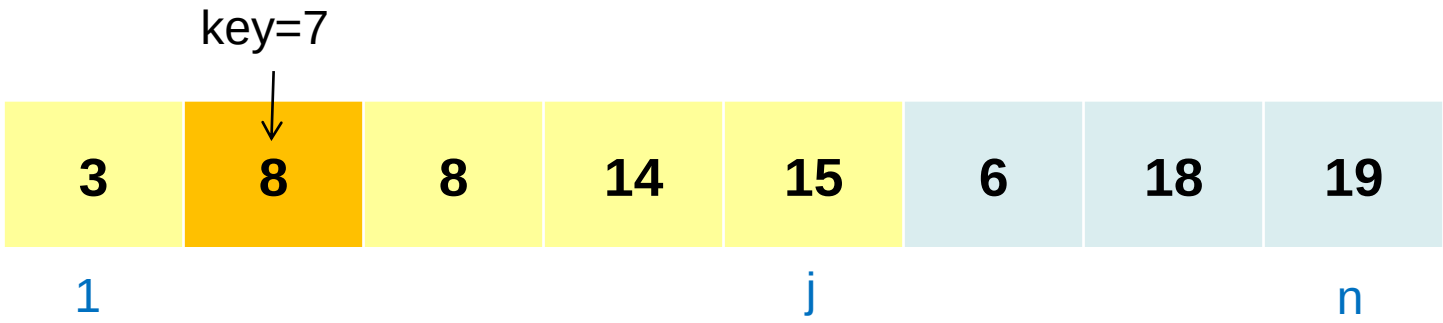
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



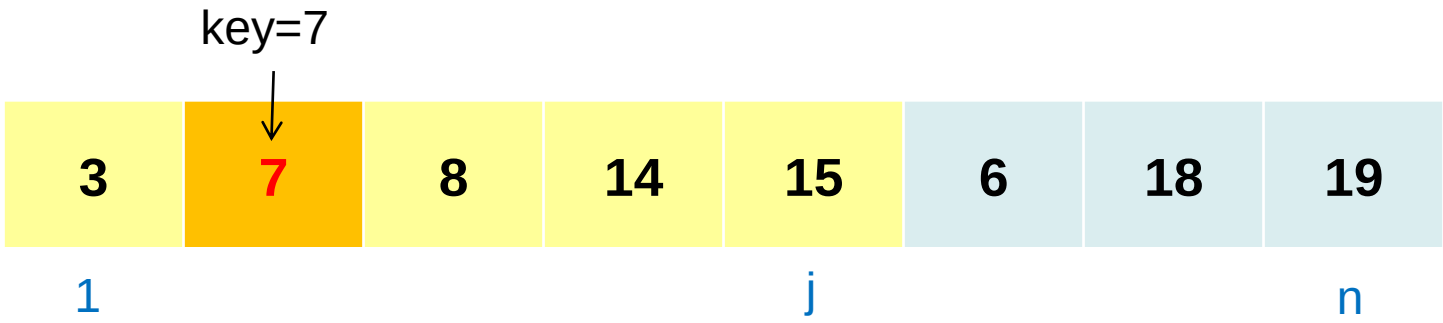
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



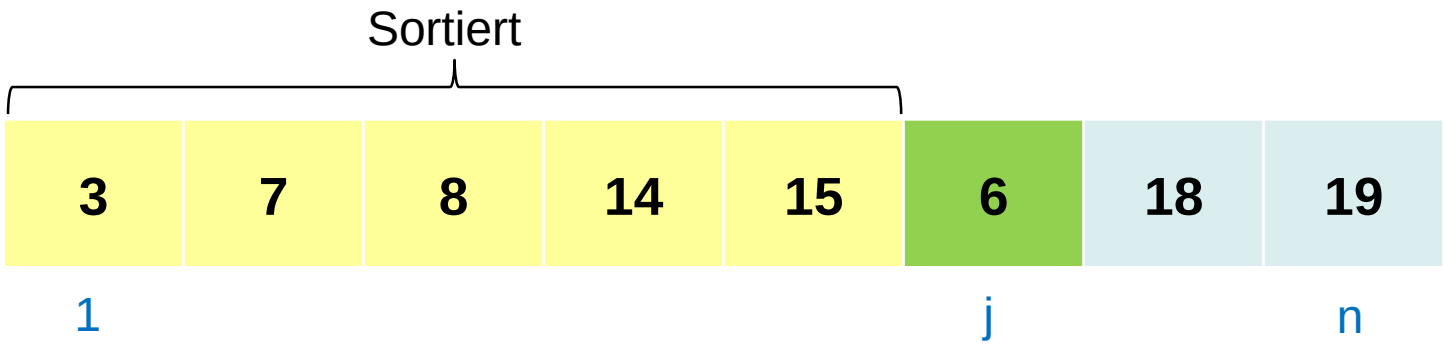
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



Insertion Sort

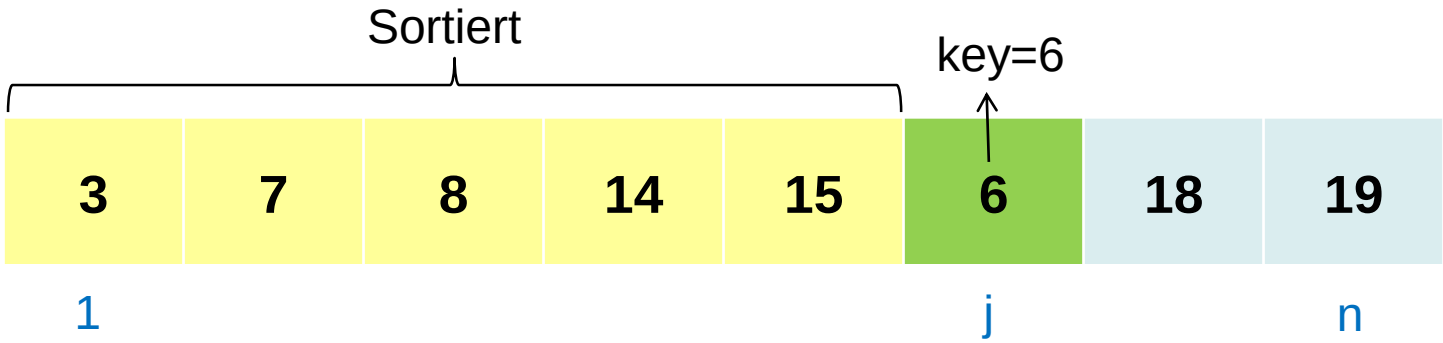
InsertionSort(Array A)

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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



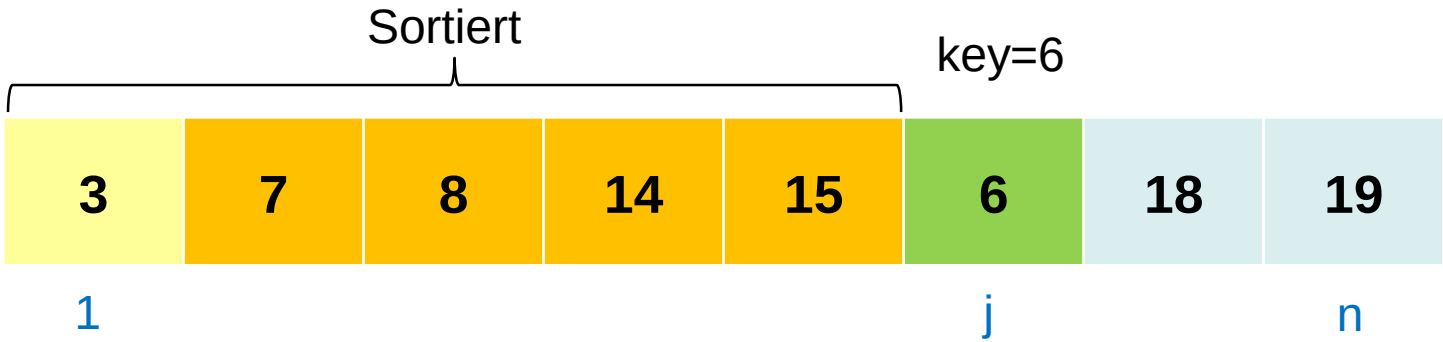
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke

key=6

3	7	7	8	14	15	18	19
1					j		n

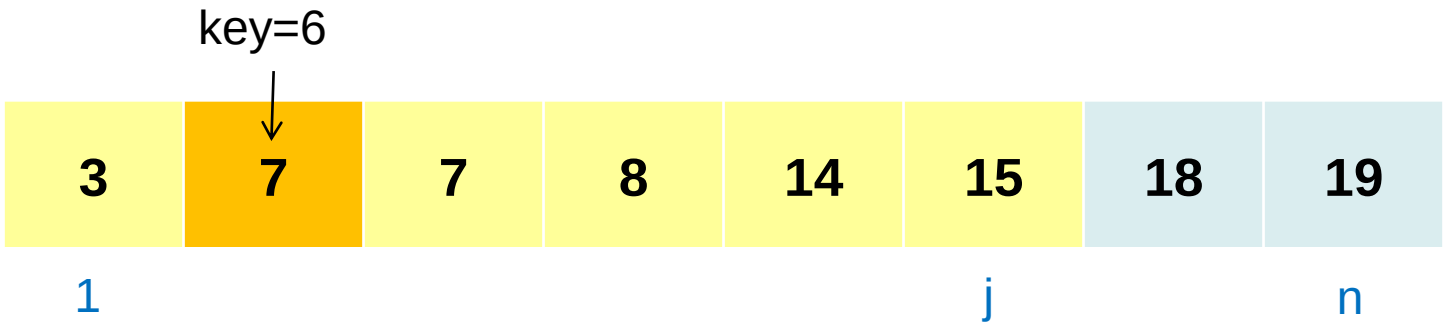
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

- Speichere key in Lücke



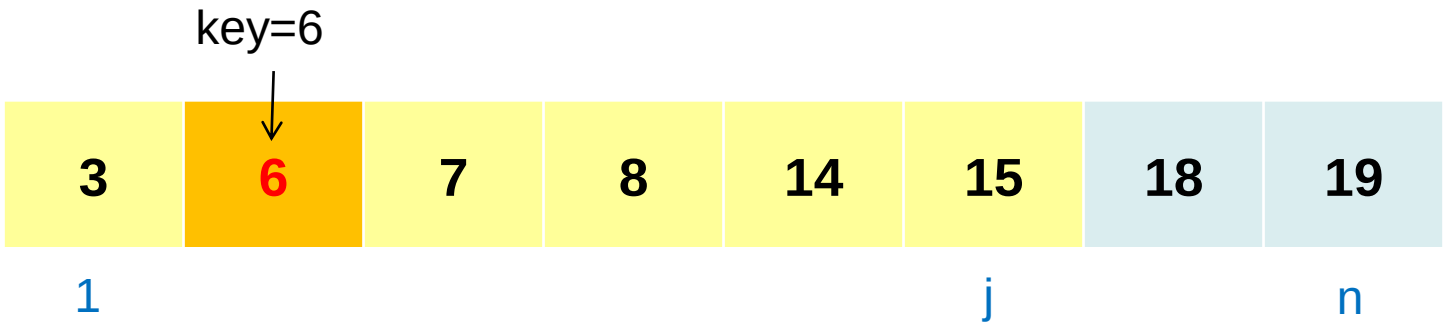
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als *key*
- sind eine Stelle nach rechts

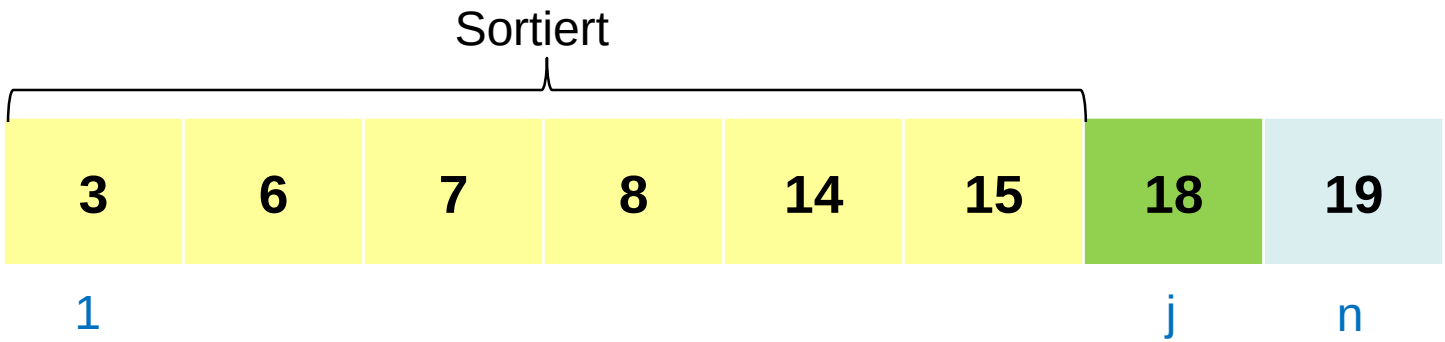
- Speichere key in Lücke



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
- length(A) = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts

- Speichere key in Lücke

3	6	7	8	14	15	18	19
1						j	n

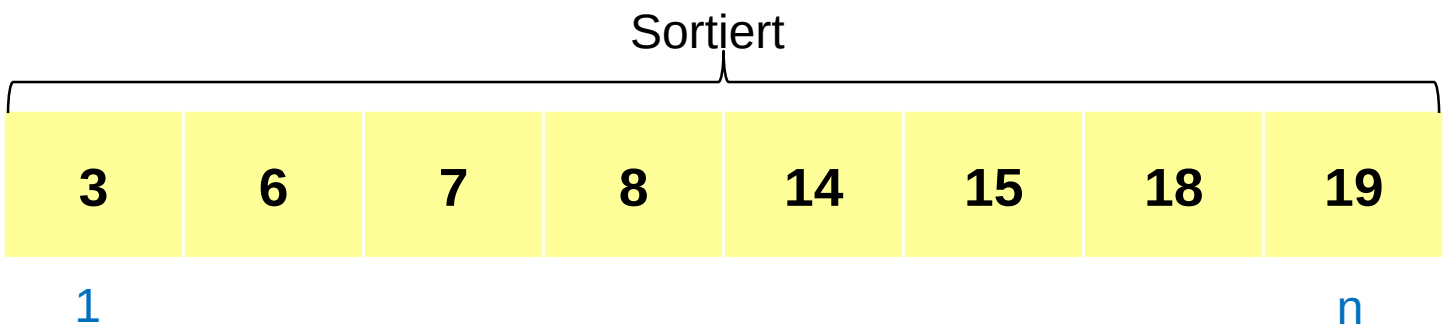
Insertion Sort

```
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
- length(A) = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts

- Speichere key in Lücke



Laufzeitanalyse

```
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
- length(A) = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke

Kernfragen

Wie kann man die Laufzeit eines Algorithmus bestimmen?
Sortiert der Algorithmus alle möglichen Eingaben auch wirklich korrekt?

Zum Knobeln: Was fehlt?

```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

Quelle: http://xkcd.com/1185/