Algorithmen, Pseudocode, Sortieren

Manfred Hauswirth | Open Distributed Systems | Einführung in die Programmierung, WS 24/25

Rückblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort
VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort
VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortierverfahren
VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue
VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen
VL 6 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C
VL 7 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort
VL 8 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise
VL 9 „Prioritätenschlangen/Halden/Heaps “: Heap Sort, Binärer Heap, Heap Operationen
VL 10 „Fortgeschrittene Sortierverfahren“: Quick Sort, Radix Sort
VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung
VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem
VL 13 „Q & A“: Offene Vorlesung/Wiederholung

Was ist Informatik?

Vorbemerkungen

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

Was ist 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.”

> **Marvin Lee Minsky** (* 9. August 1927 in New York; † 24. Januar 2016 in Boston, Massachusetts[1]) war ein amerikanischer Forscher auf dem Gebiet der künstlichen Intelligenz (KI). Gemeinsam mit John McCarthy, Nathaniel Rochester und Claude Shannon begründete er 1956 auf der Dartmouth Conference den Begriff der künstlichen Intelligenz. Später waren er und Seymour Papert auch Begründer des Labors für Künstliche Intelligenz am Massachusetts Institute of Technology (AI Lab).
>
> Quelle: Wikipedia

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

> **Juris Hartmanis** (* 5. Juli 1928 in Riga, Lettland; † 29. Juli 2022[1]) war ein lettisch-US-amerikanischer Informatiker, der gemeinsam mit Richard E. Stearns 1993 den Turing Award für seine Forschungsleistungen auf dem Gebiet der Komplexitätstheorie erhielt.

Was ist Informatik / Computer Science?

- Donald Knuth: “The ‘study’ of **algorithms**”

> **Donald Ervin „Don“ Knuth** [kəˈnuːθ][1] (* 10. Januar 1938 in Milwaukee, Wisconsin) ist ein US-amerikanischer Informatiker. Er ist emeritierter Professor an der Stanford University, Autor des Standardwerks *The Art of Computer Programming* und Urheber des Textsatzsystems TeX.
>
> Quelle: Wikipedia

  - 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 information
  - How to process information
  - And the machines that do this

Was ist Informatik / Computer Science?

- Fred Brooks: “CS = **engineering**”

Frederick Phillips Brooks, Jr. (* 19. April 1931 in Durham, North Carolina, USA; † 17. November 2022[1] in Chapel Hill, North Carolina) war ein US-amerikanischer Informatiker. Bekannt wurde Brooks zunächst als Verantwortlicher für die Entwicklung des OS/360 bei IBM und später für die sehr authentische Beschreibung des Entwicklungsprozesses in seinem Buch *The Mythical Man-Month* (deutsch *Vom Mythos des Mann-Monats: Essays über Software-Engineering*). Dieses Buch enthält auch eine viel zitierte Aussage, die als Brooks'sches Gesetz bekannt wurde:

“Adding manpower to a late software project makes it later.”

Quelle: Wikipedia

  - “CS ≠ science”
  - “CS = **engineering**”
  - Concerned with “making”: Physical computers, S/W systems

Was ist Informatik / Computer Science?

- Peter Denning: “Computing is a 4th great domain of science alongside the physical, life, and social sciences.”

Peter James Denning (* 6. Januar 1942 in New York City) ist ein US-amerikanischer Informatiker. Er war Hochschullehrer an mehreren Universitäten.
Er ist besonders für Beiträge zur Verwaltung des Arbeitsspeicherbedarfs von Programmen bekannt (Einführung des Working Set als Arbeitsspeicherbedarf pro Zeiteinheit in seiner Dissertation 1968) mit Anwendung auf Seitenflattern (Trashing) und einen Aufsatz von 1970 zur Klärung der Eigenschaften der (damals noch umstrittenen) Virtuellen Speicherverwaltung. Außerdem veröffentlichte er über Betriebssysteme (mit einem einflussreichen Lehrbuch mit Edward G. Coffman), Warteschlangentheorie (Operationsanalyse von Warteschlangen in Netzwerken) und allgemeine Prinzipien der Informatik und Innovation. 1981 war er einer der Gründer von CSNET.

Quelle: Wikipedia

- Informatik =
  - discovery (science)
  - implementation (engineering)

  of information processes

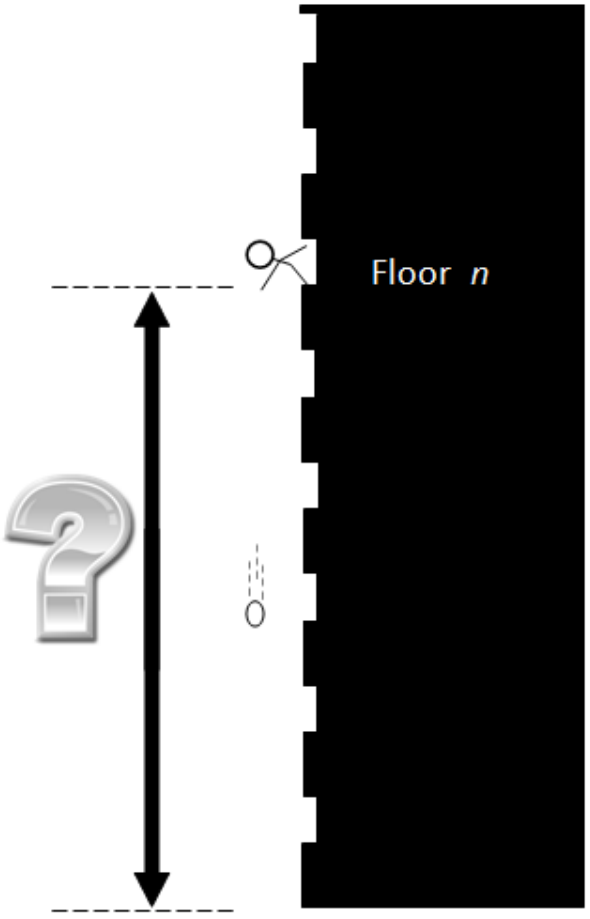
Algorithmik: Ein wesentlicher Teil der Informatik

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???

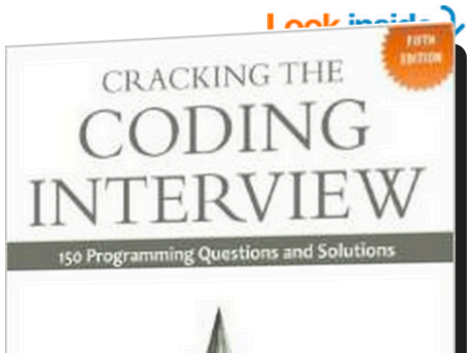
➢ http://datagenetics.com/blog/july22012/



Denkanstoß

- Frage aus „technischen Bewerbungsgesprächen“ für Software-Engineering-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

Warum solche Fragen?

- Algorithmisches Denken ist die Grundlage der Informatik
- Algorithmen und Datenstrukturen findet man überall in der Informatik wieder
- In dieser Vorlesung
  - Basis-Datenstrukturen
  - Basis-Algorithmen

Themengebiete in der Informatik

- Datenbanken
- Kommunikationsnetze
- Verteilte Systeme
- Graphische Datenverarbeitung
- Robotik
- Künstliche Intelligenz

… und viele weitere mehr !

Algorithmen

Programm vs. Algorithmus

- Algorithmen beschreiben in prinzipiellen Elementen, was ein Computer ausführen soll

- Programmiersprachen stellen eine Schnittstelle dar, um Algorithmen auf einem Computer definieren und 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 vom Computer ausgeführt werden müssen, um eine bestimmte Funktionalität zu erreichen
  - „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 Algorithmenanalyse

Grundlagen der Algorithmen-Analyse

Technische Universität Berlin

Inhalt

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

Wie beschreibt man einen Algorithmus?

Spiel: einen Algorithmus praktisch ausprobieren.

Wie beschreibt man einen Algorithmus?

Problemstellung

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

Gesucht: Exakte, kompakte, einfache „Notation“

Pseudocode

- Beschreibungssprache
- Losgelöst von spezifischer Programmiersprache/Umgebung
- Manchmal kann auch ein vollständiger Satz die beste Beschreibung sein
- Wir ignorieren dabei Details, wie
  - Variablen-Deklaration
  - Bibliotheken, …
- 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

- 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], …
- Indizierung 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: ein Array hat eine Länge
- Zugriff über Funktionen, z.B. die Funktion length(A) gibt die *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 nine thousand“

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. lokale 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

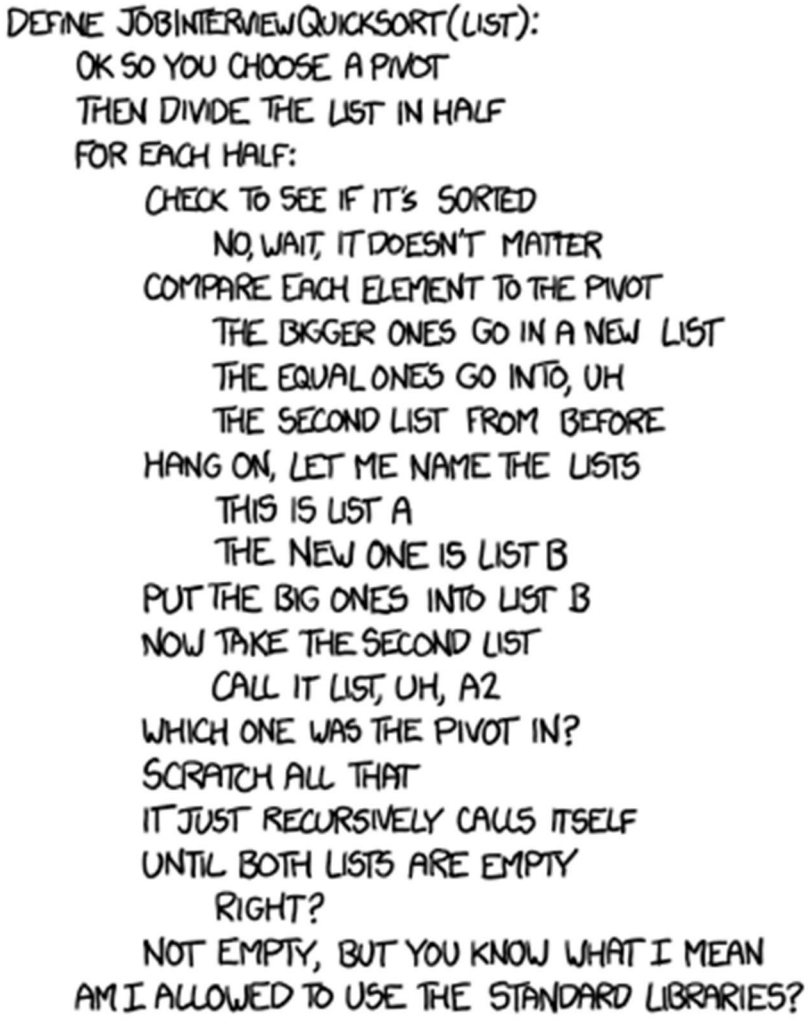
```
AlgorithmFoo(Array A)
1. for j ← 2 to length(A) do
2.         key ← A[j]
3.         i ← j-1
4.         while i>0 and A[i]>key do
5.             A[i+1] ← A[i]
6.             i ← i-1
7.         A[i+1] ← key
```

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

Pseudocode

- Kommentare durch ➢, oder //

Pseudocode in Jobinterviews



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

Beispiel: Sortieren

Sortieren

Problem: Sortieren

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

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
```

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
```

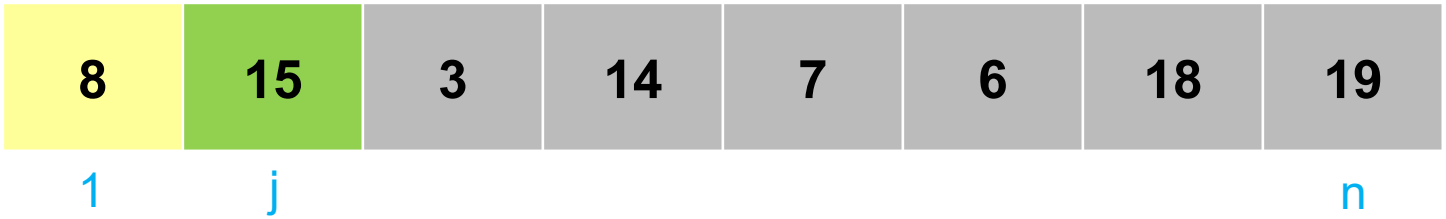
Beispiel

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
```

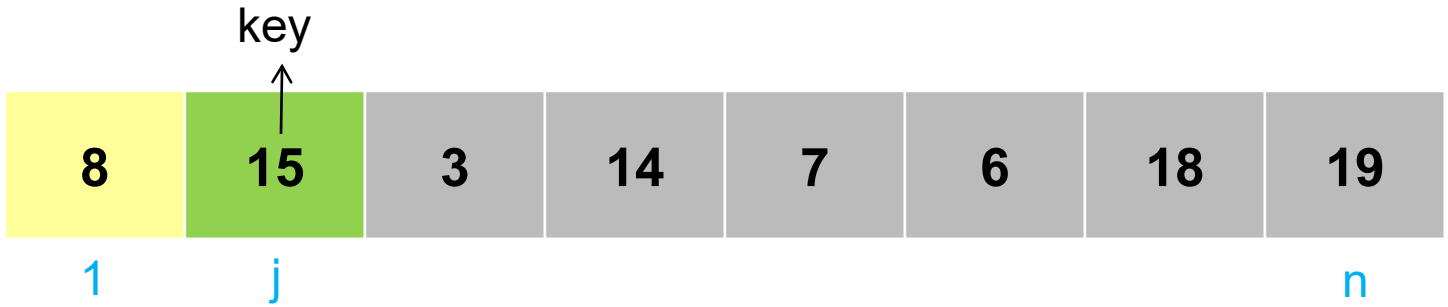
- Eingabegröße n
- length(A) = n



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
```

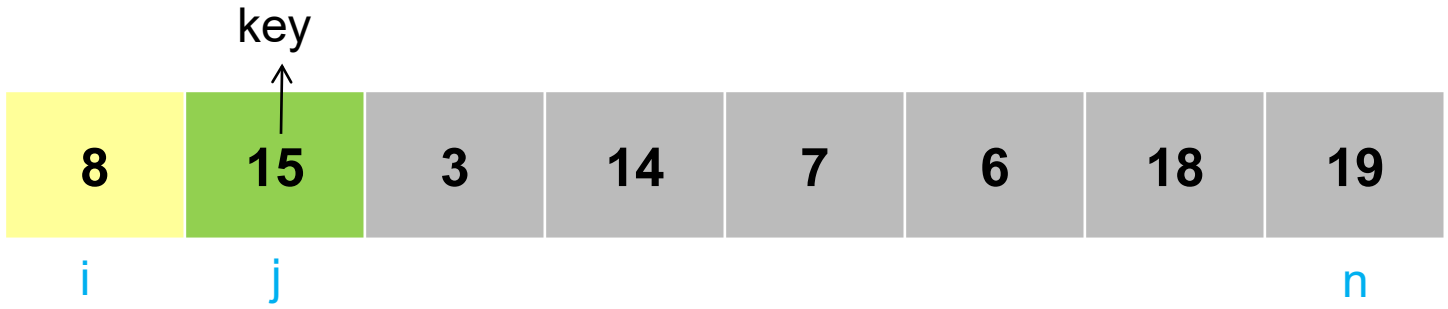
- Eingabegröße n
- length(A) = n



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
```

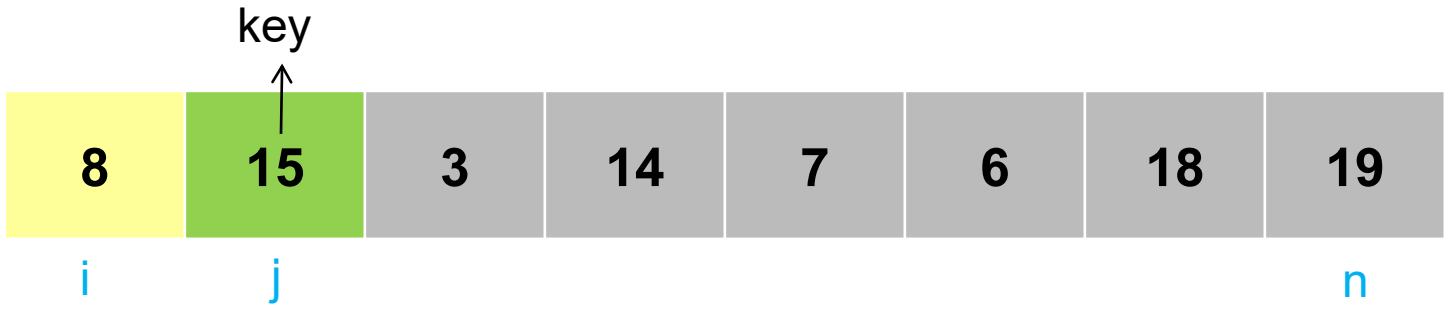
- Eingabegröße n
- length(A) = n



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
```

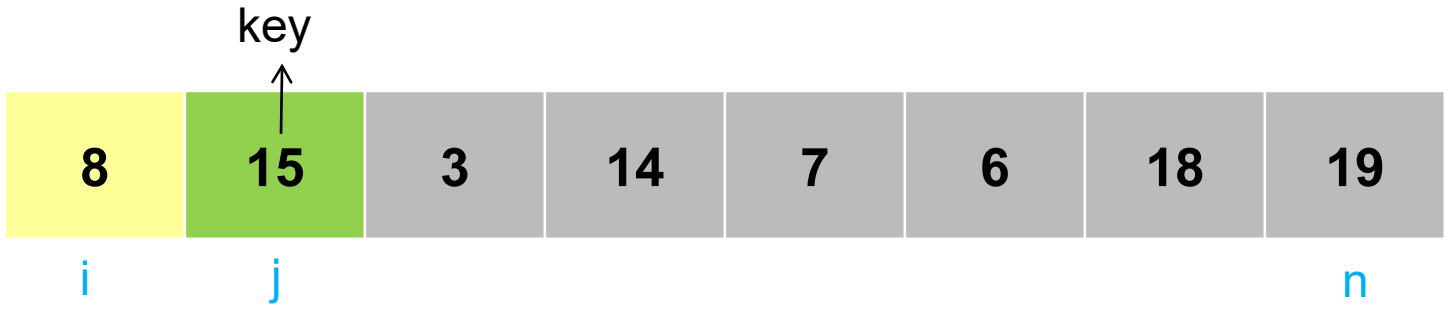
- Eingabegröße n
- length(A) = n



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
```

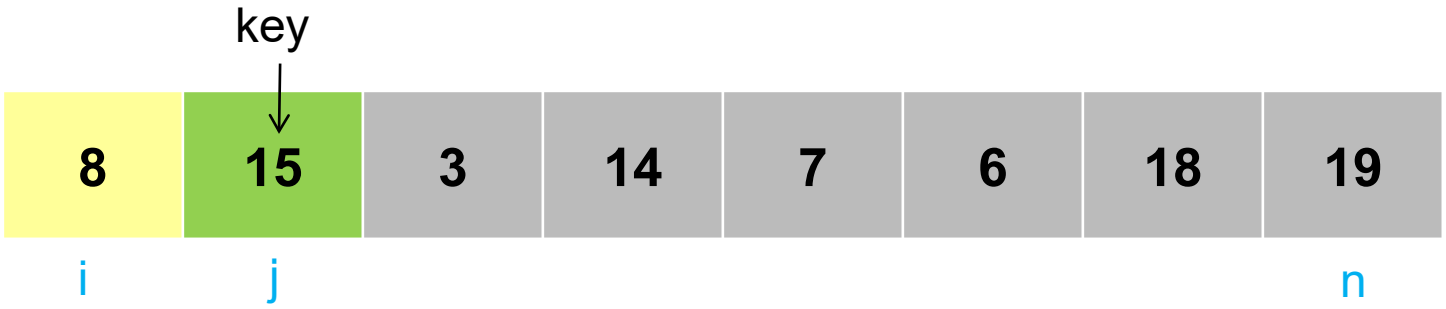
- Eingabegröße n
- length(A) = n



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
```

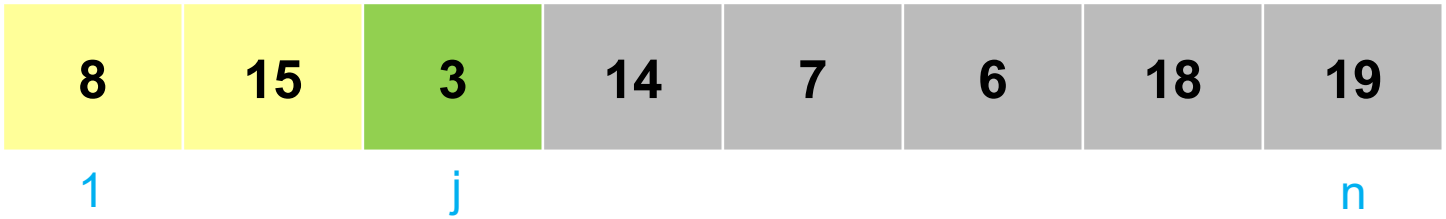
- Eingabegröße n
- length(A) = n



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
```

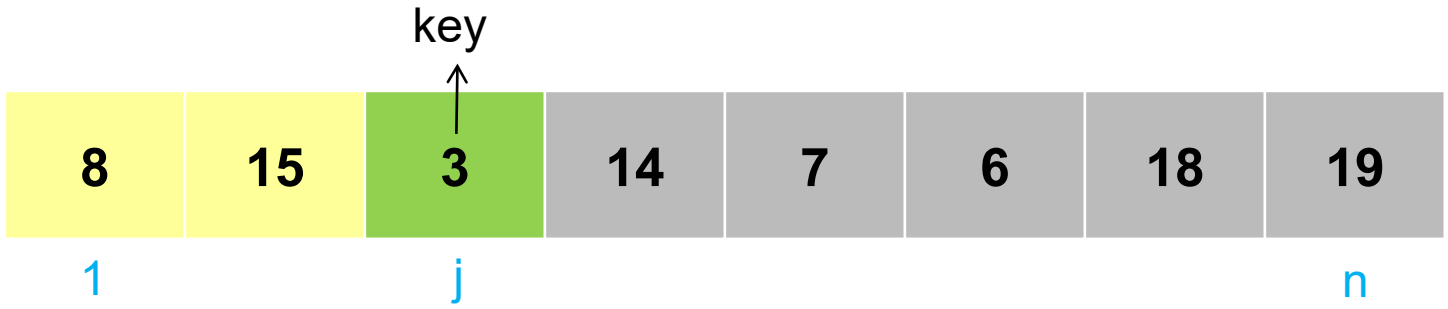
- Eingabegröße n
- length(A) = n



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
```

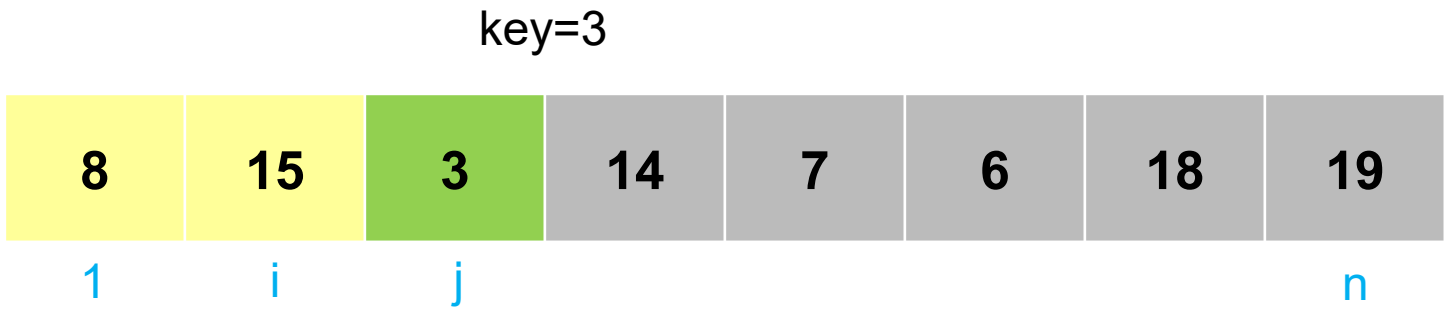
- Eingabegröße n
- length(A) = n



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
```

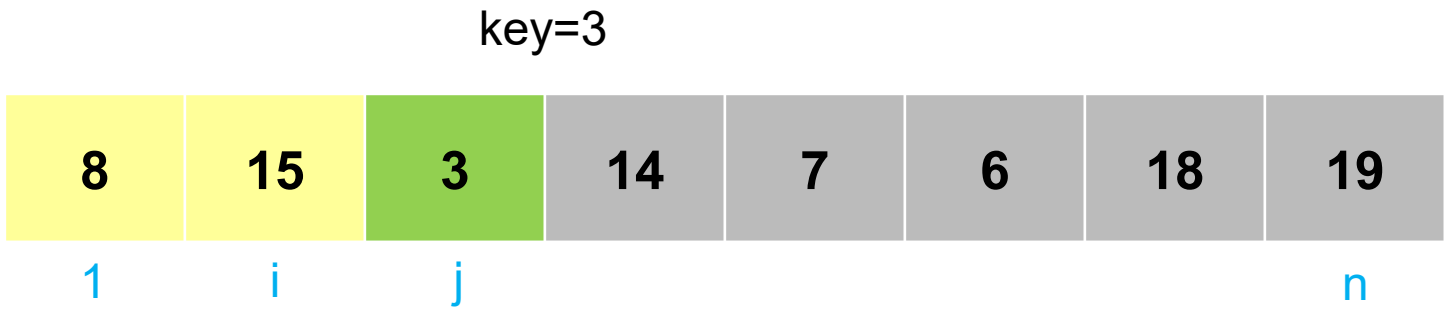
- Eingabegröße n
- length(A) = n



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
```

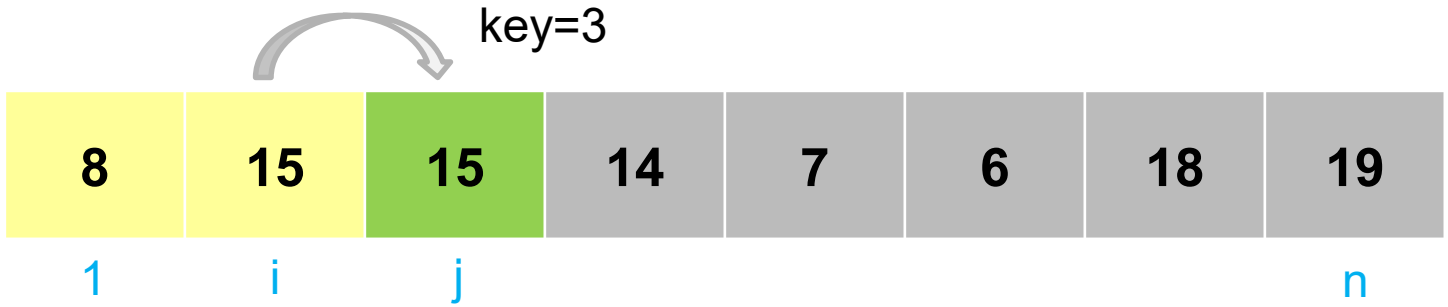
- Eingabegröße n
- length(A) = n



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
```

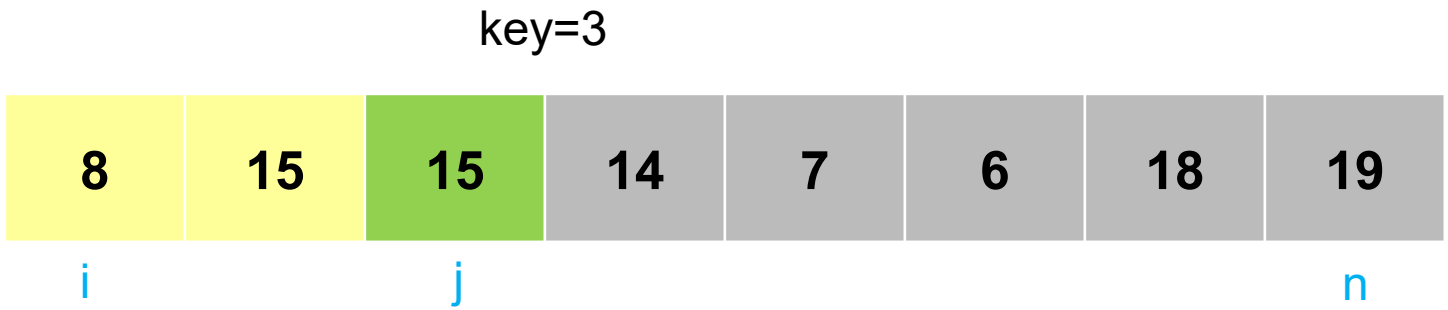
- Eingabegröße n
- length(A) = n



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
```

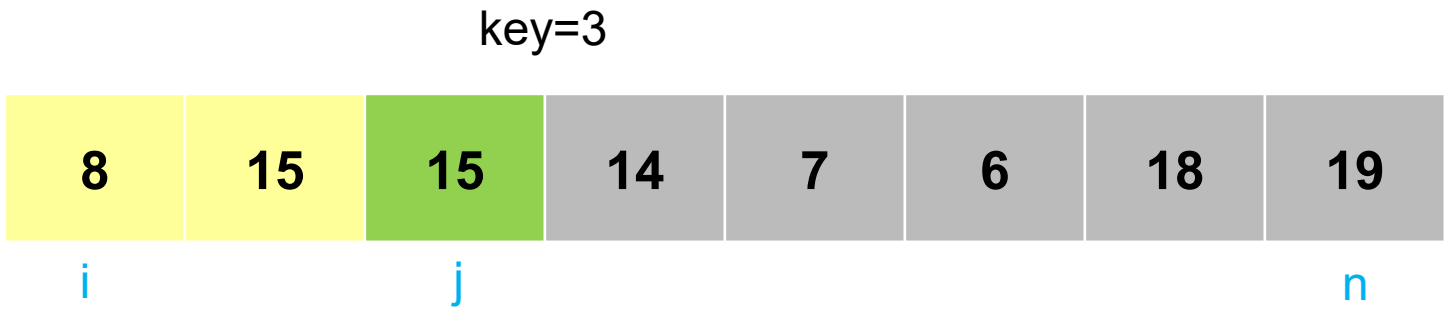
- Eingabegröße n
- length(A) = n



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
```

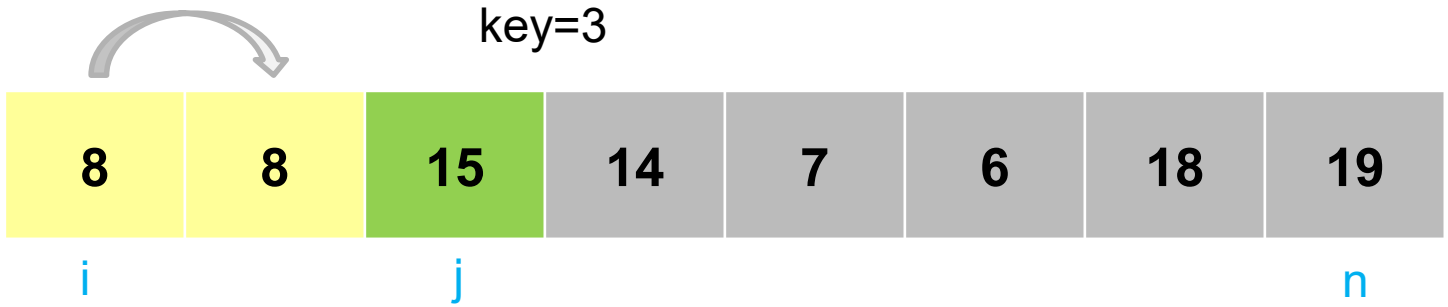
- Eingabegröße n
- length(A) = n



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
```

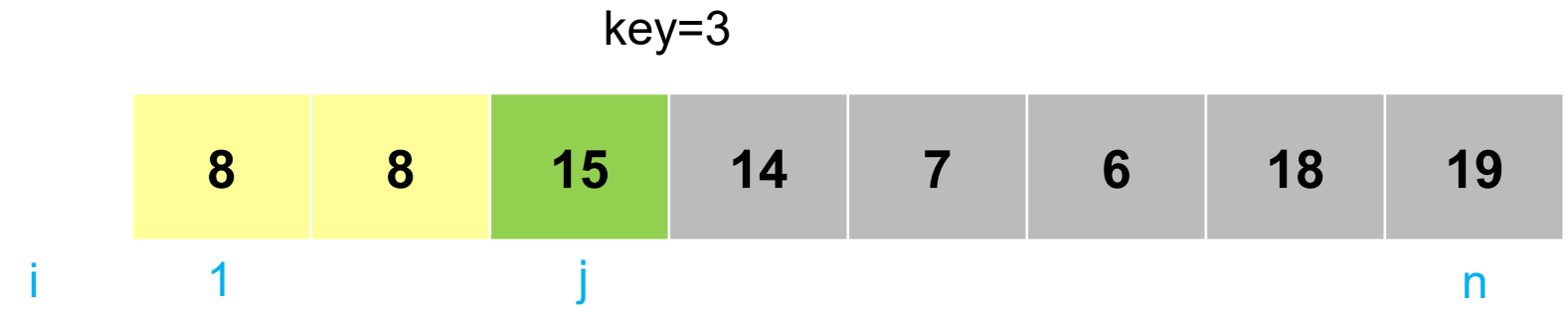
- Eingabegröße n
- length(A) = n



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
```

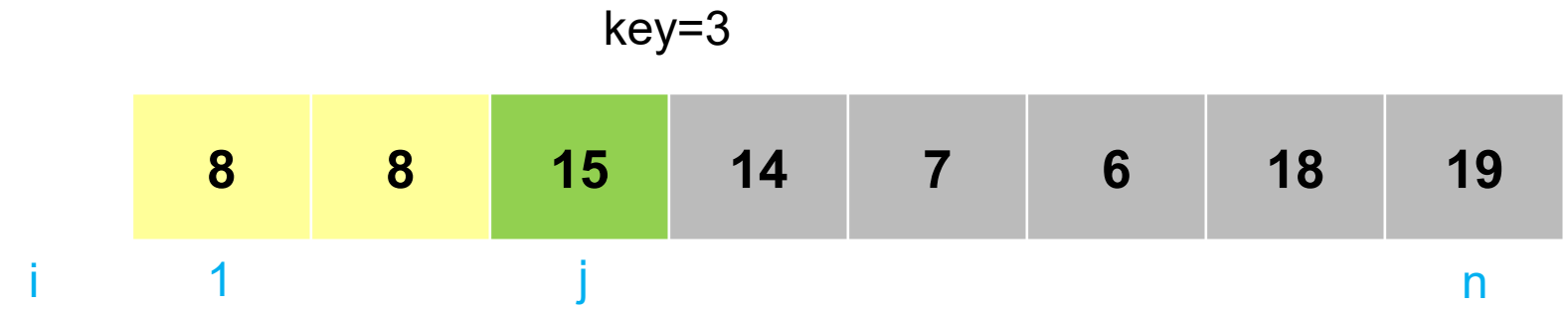
- Eingabegröße n
- length(A) = n



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
```

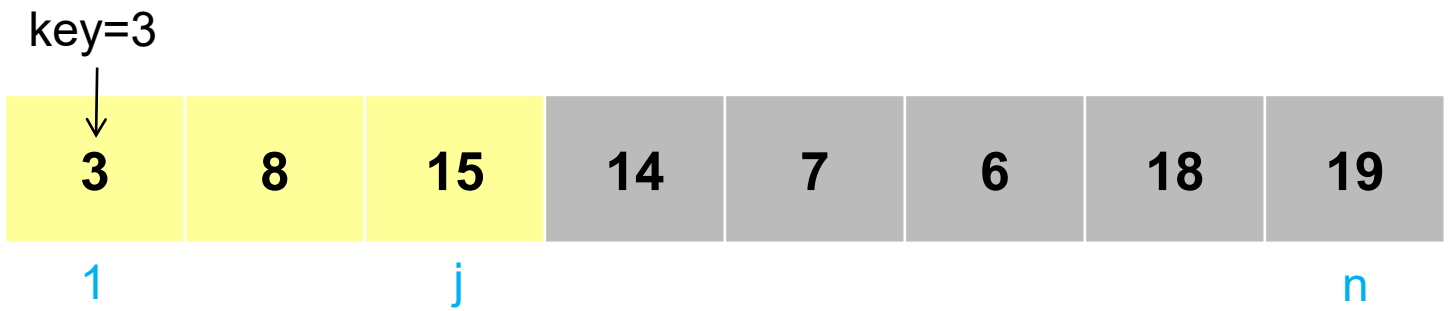
- Eingabegröße n
- length(A) = n



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
```

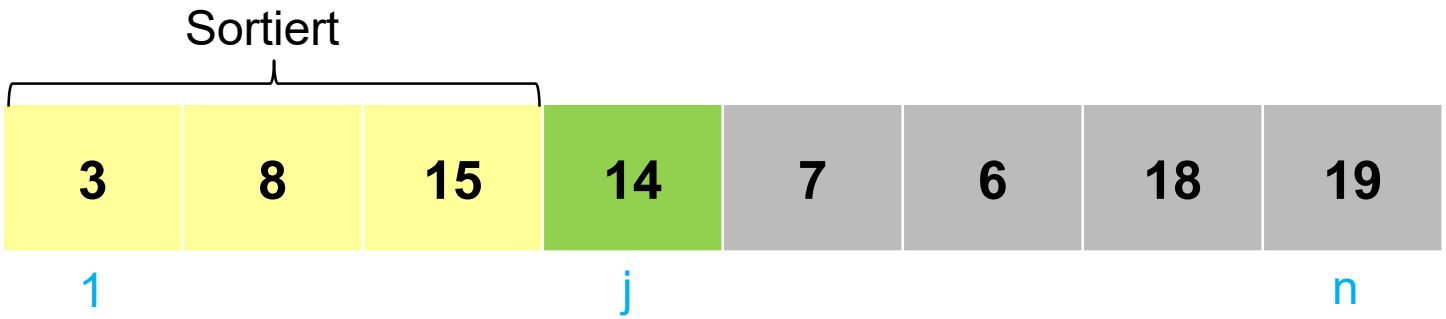
- Eingabegröße n
- length(A) = n



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
```

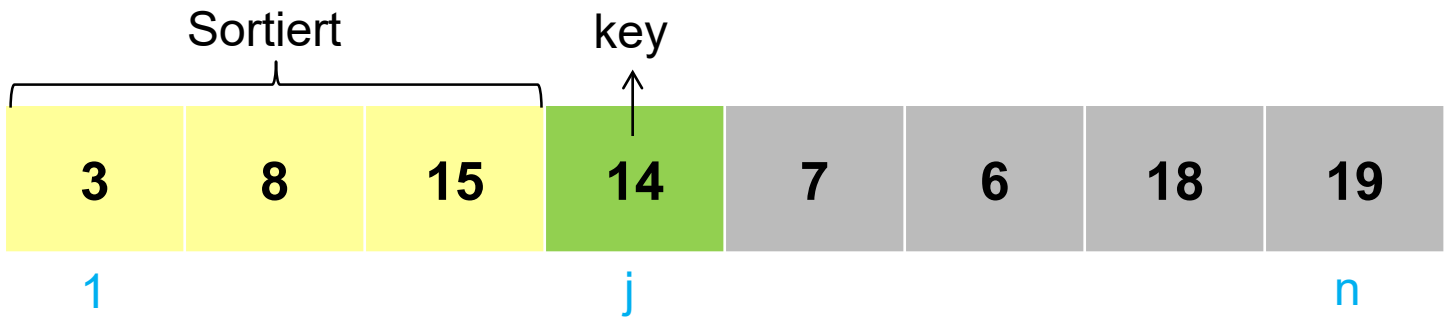
- Eingabegröße n
- length(A) = n



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
```

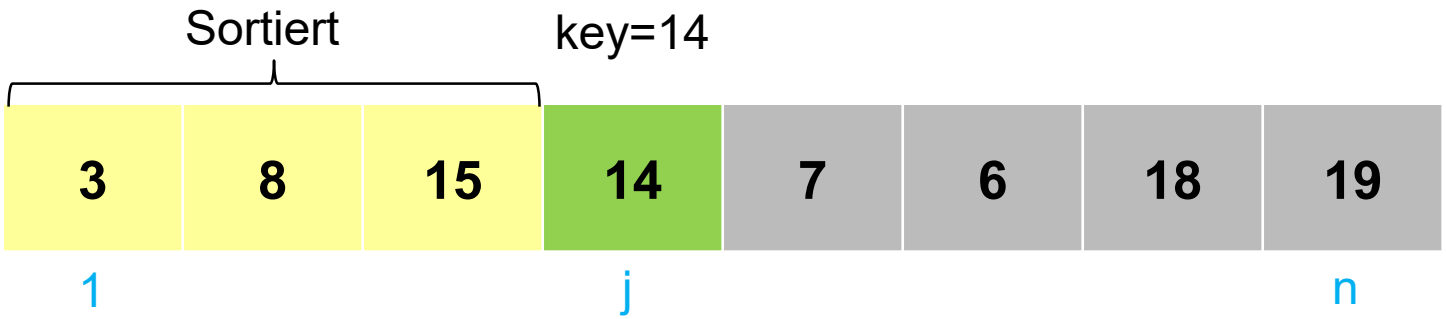
- Eingabegröße n
- length(A) = n



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

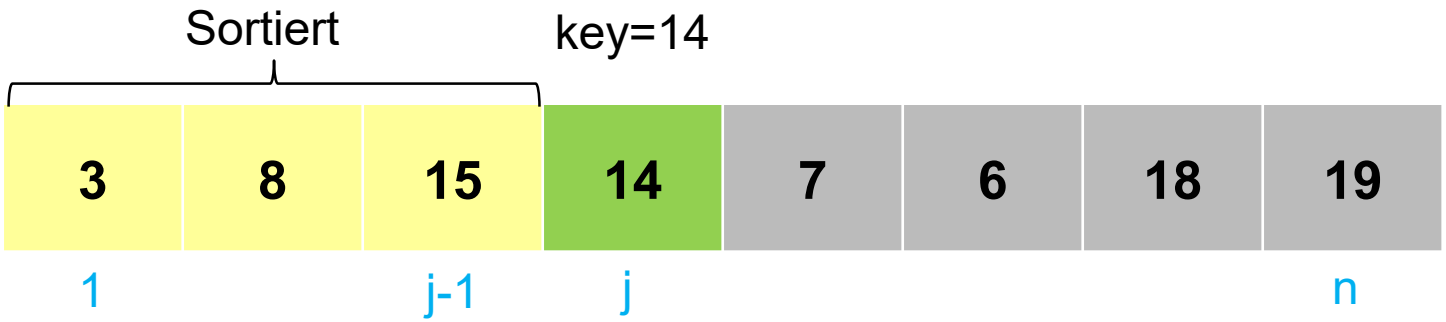


Insertion Sort

```
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

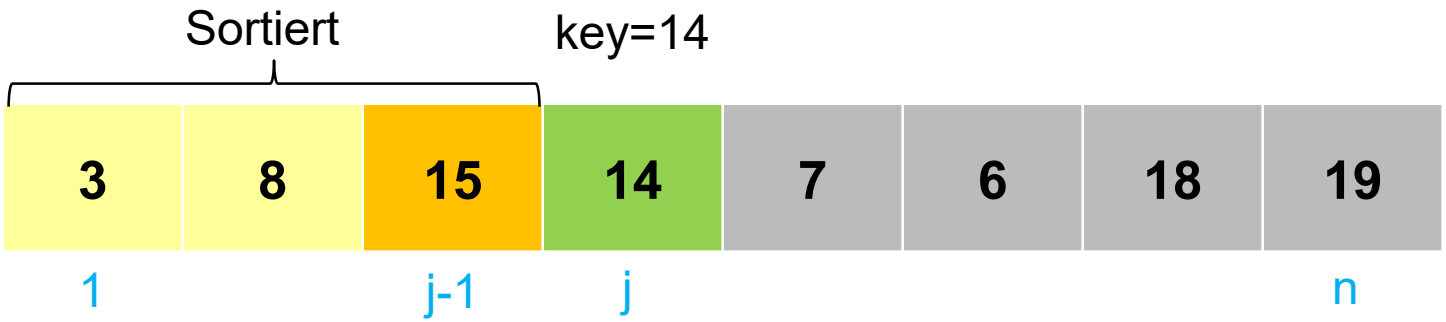


Insertion Sort

```
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



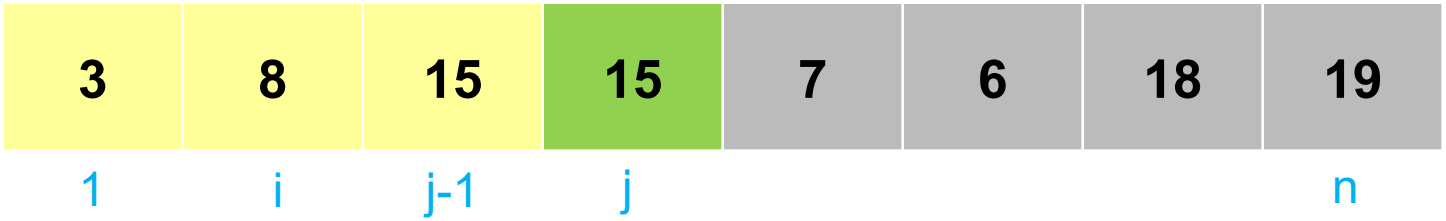
Insertion Sort

```
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

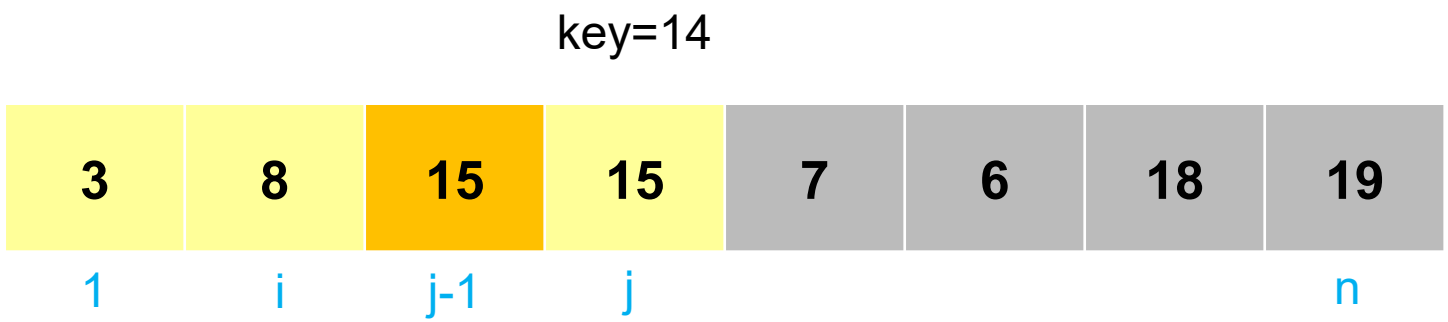


Insertion Sort

```
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

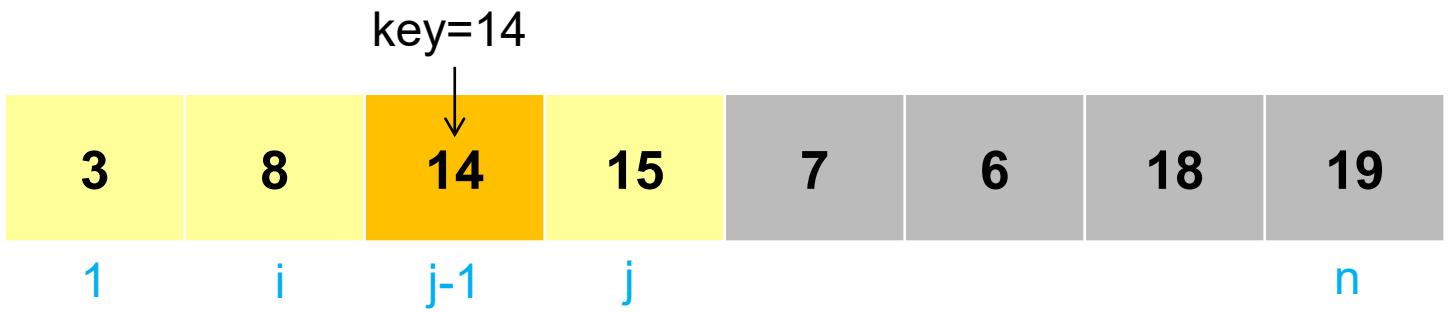


Insertion Sort

```
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
- Speichere key in Lücke

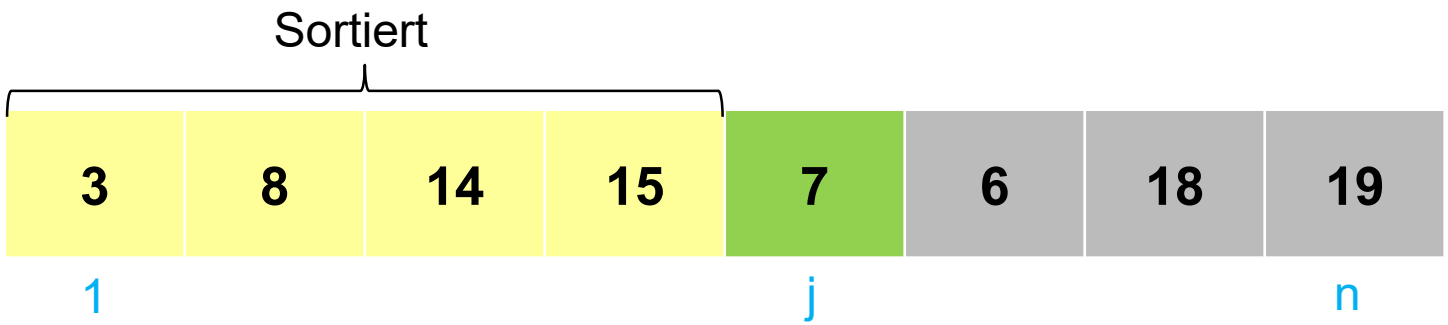


Insertion Sort

```
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

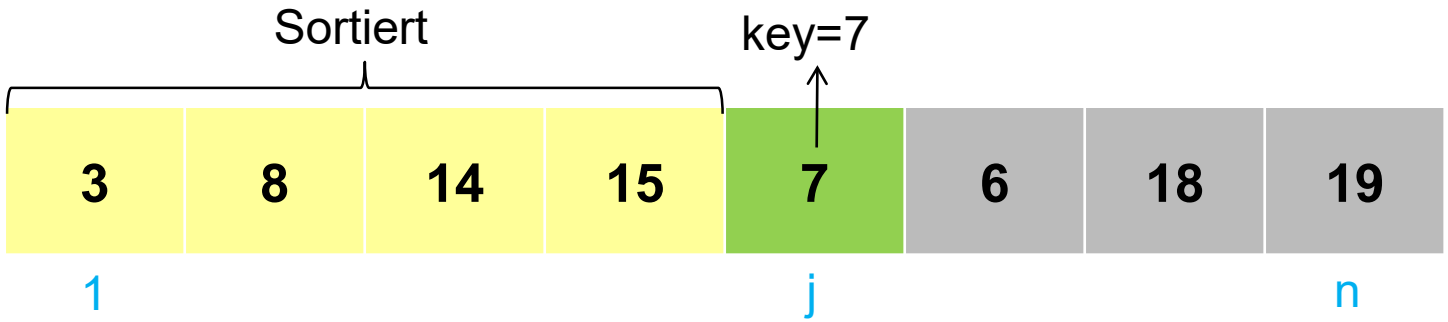


Insertion Sort

```
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

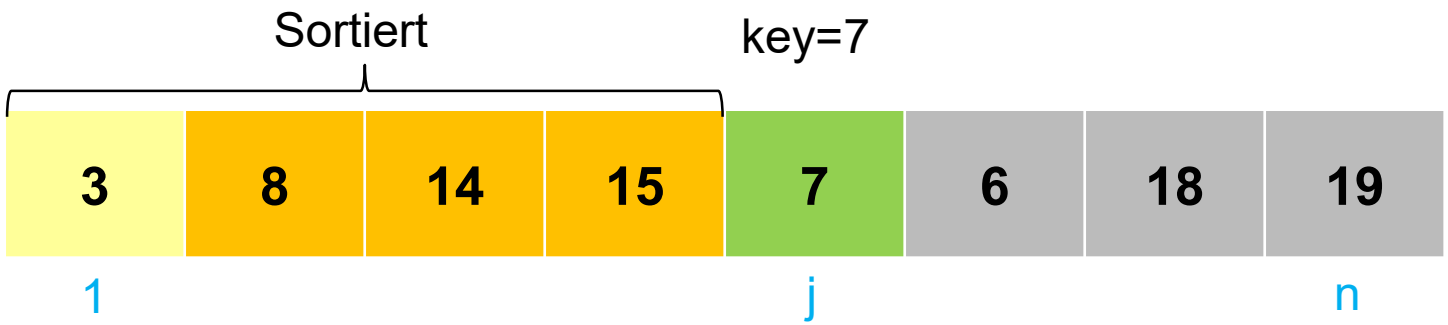


Insertion Sort

```
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

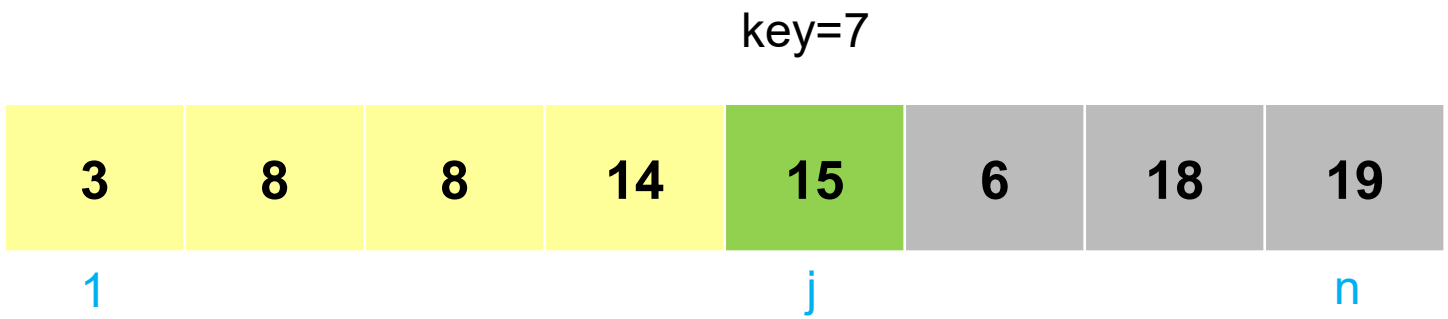


Insertion Sort

```
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

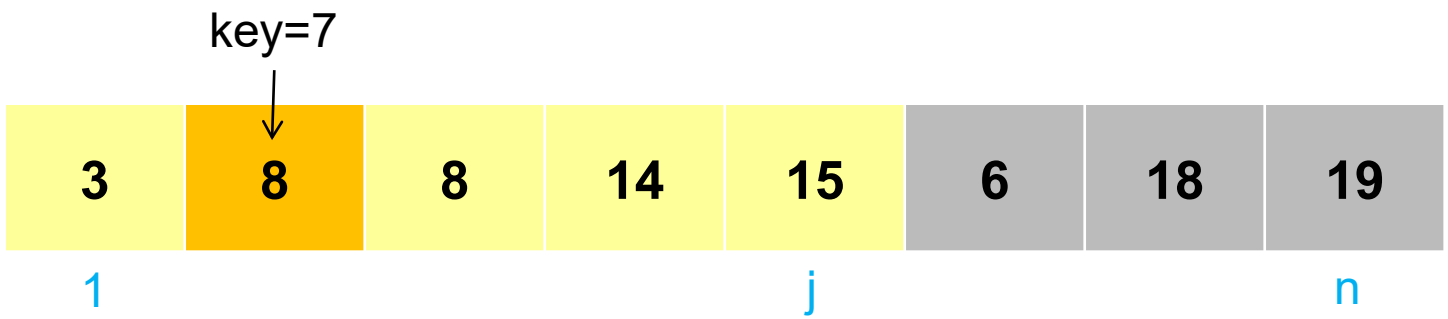


Insertion Sort

```
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

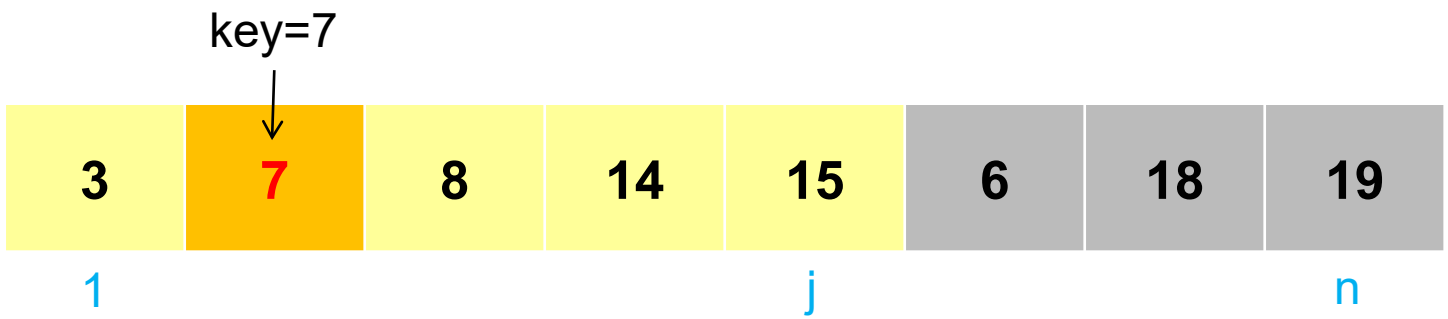


Insertion Sort

```
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

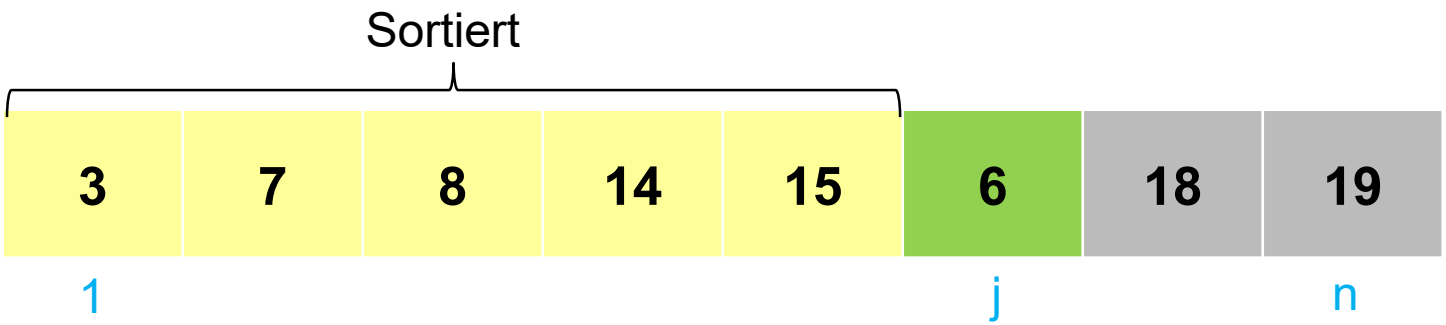


Insertion Sort

```
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

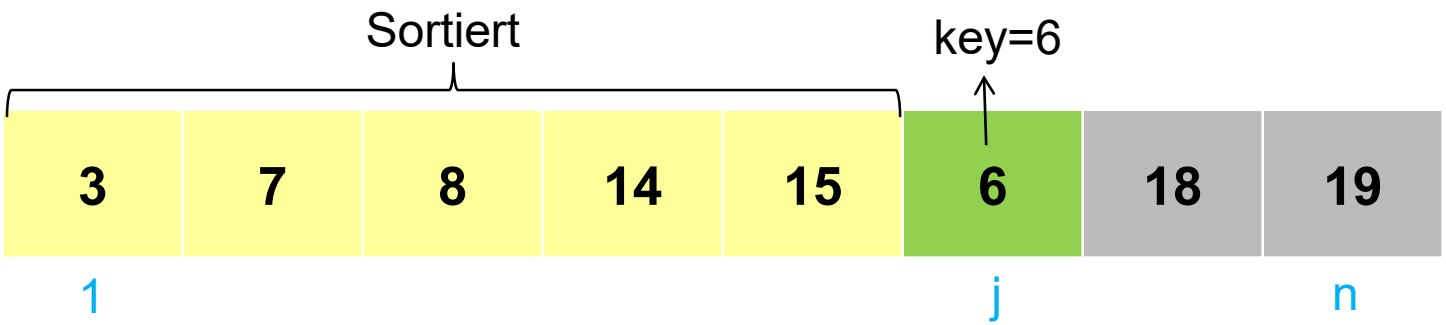


Insertion Sort

```
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

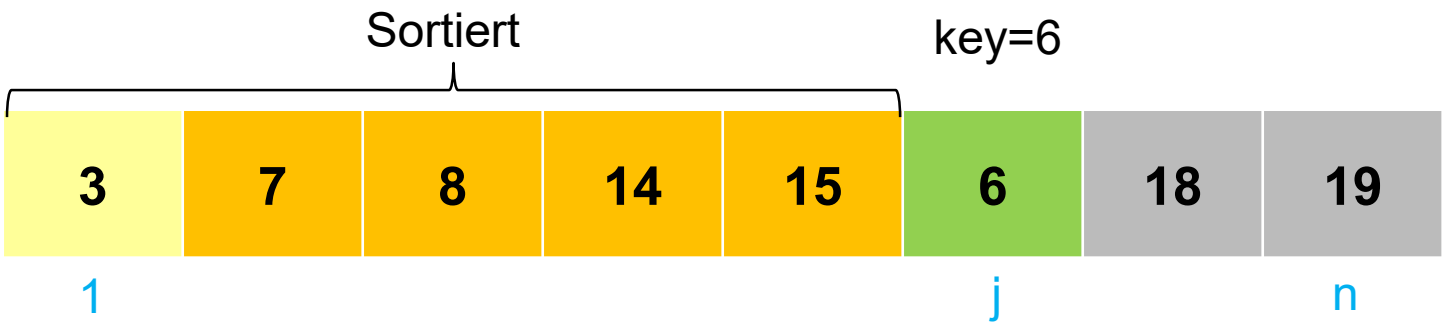


Insertion Sort

```
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



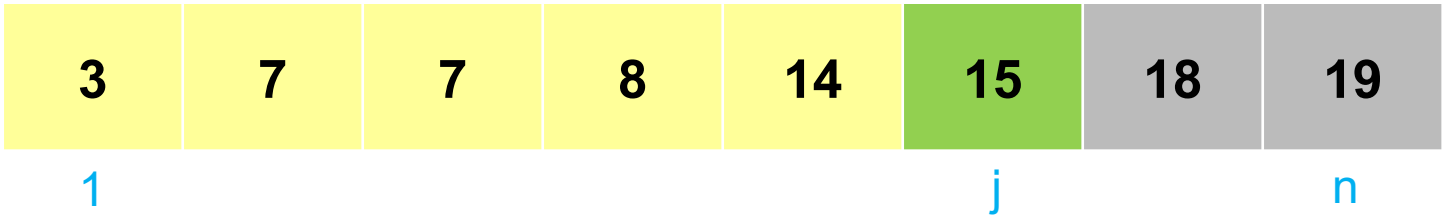
Insertion Sort

```
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

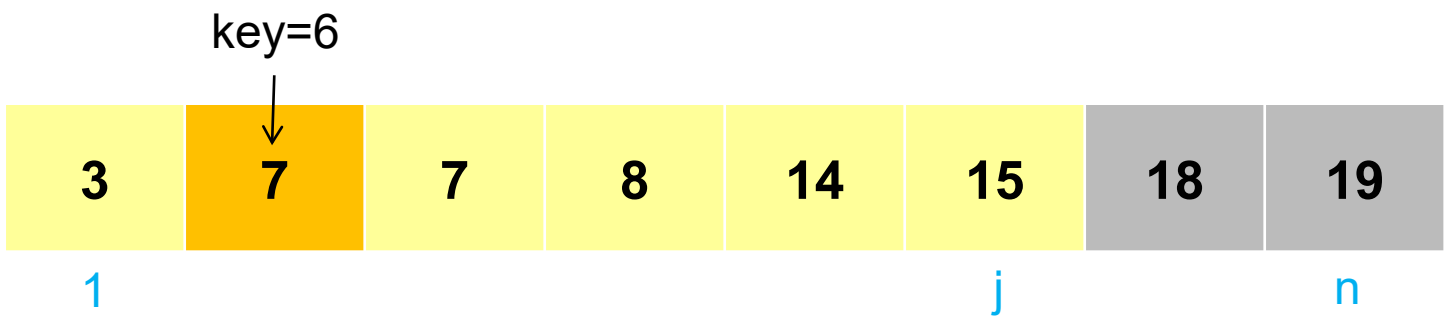


Insertion Sort

```
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

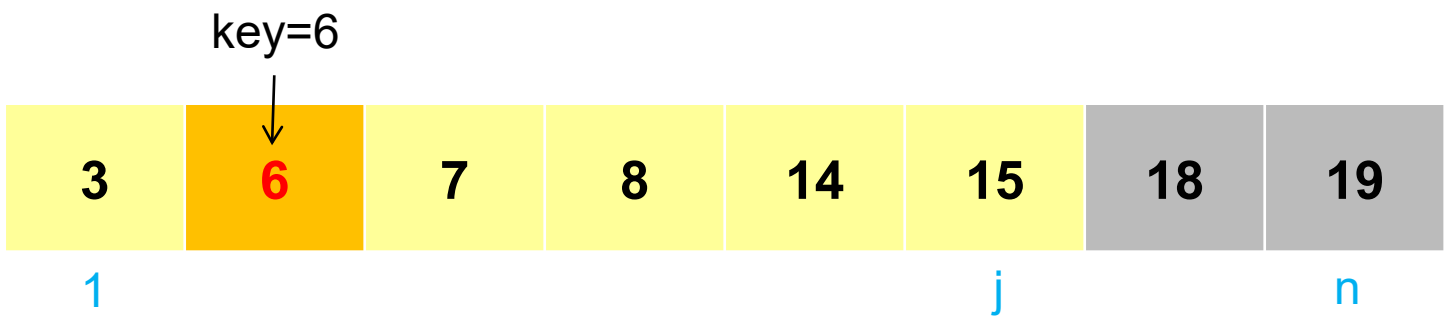


Insertion Sort

```
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

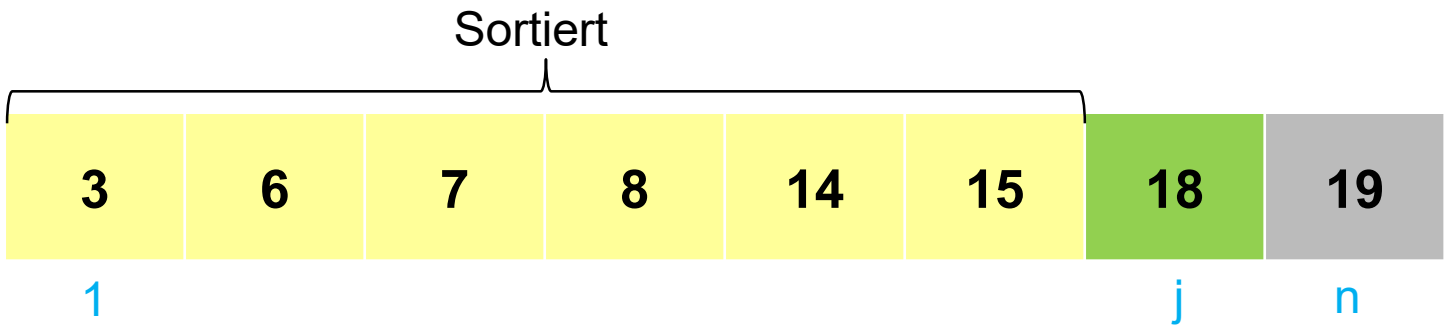


Insertion Sort

```
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

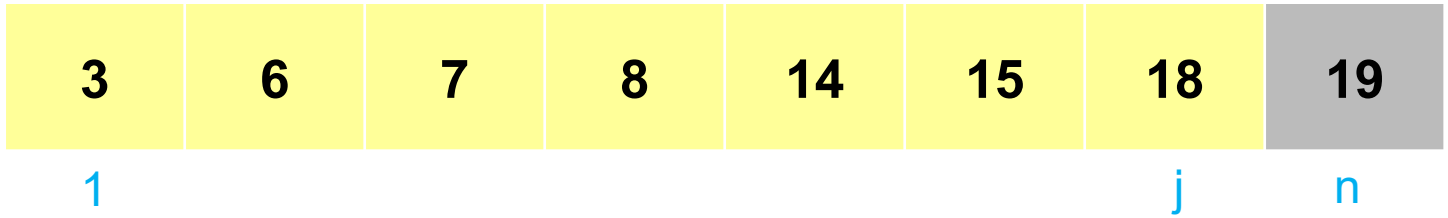


Insertion Sort

```
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

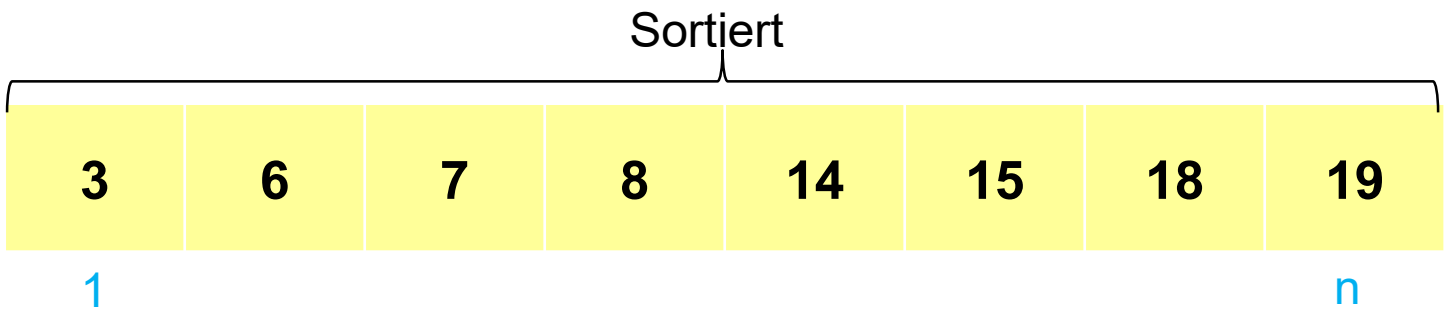


Insertion Sort

```
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 visualisiert

https://www.youtube.com/watch?v=ROalU379l3U

Ausblick: Laufzeitanalyse und Korrektheit

```
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?

Ausblick: Sortieralgorithmen im Vergleich

https://www.youtube.com/watch?v=ZZuD6iUe3Pc

Ausblick

VL 0 „Organisation und Inhalt": Ablauf der Vorlesung, Termine

VL 1 „Algorithmen, Pseudocode, Sortieren |“: Insertion Sort

VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort

VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortierverfahren

VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue

VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen

VL 6 „Teile und Herrsche |“: Einführung der algorithmischen Methode, Merge Sort

VL 7 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise

VL 8 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C

VL 9 „Prioritätenschlangen/Halden/Heaps“; Heap Sort, Binärer Heap, Heap Operationen

VL 10 „Fortgeschrittene Sortierverfahren“: Quick Sort, Radix Sort

VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung

VL 12 „Teile und Herrsche I|“: Generalisierung des algorithmischen Prinzips, Mastertheorem

VL 13 „Q & A“: Offene Vorlesung/Wiederholung