Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Organisatorisches

Vorlesung DAP2

- Dienstag 12-14 c.t.
- Donnerstag 14-16 c.t.
- Vorlesungswebseite über Lehrstuhl 2 erreichbar
- Passwort für Folien: DaP2017!
- Benutzername: student

Zu meiner Person

- Christian Sohler
- Fachgebiet: Komplexitätstheorie und effiziente Algorithmen
- Lehrstuhl 2, Informatik
- Raum 306

Organisatorisches

Übungen

- Mittwoch: 8-10 (3), 10-12 (4), 12-14 (5), 16-18 (3)
- Donnerstag: 8-10 (3), 10-12 (2), 16-18 (6)
- Freitag: 12-14 (3), 14-16 (3)

- Zum Teil mehrere parallele Gruppen (Anzahl in Klammern)
- Anmeldung über AsSESS
- Anmeldung ab heute 14 Uhr
- Anmeldeschluss: Mittwoch 20 Uhr
- Änderungen der Übungsgruppe: Bis Montag 10 Uhr
(email an amer.krivosija@tu-dortmund.de)

Organisatorisches

Übungen

- Übungsblatt erscheint Freitags und enthält Präsenzübungen und Heimübungen
- Die erste Übung ist eine Präsenzübung
- Zu Hause soll nur der Heimübungsteil bearbeitet werden
- Abgabe Heimübung: Freitag 12 Uhr Briefkästen Übergang OH 12/14
- Zulassung zur Klausur (Studienleistung Übung; Teil 1):
  - 50% der Heimübungspunkte
- Max. 3 Personen pro Übungsblatt
- Die regelmäßige Teilnahme an den Übungen wird im Hinblick auf die Tests und die Klausur dringend empfohlen

Organisatorisches

Übungen Praktikum (außer ETIT und IKT)

- Für Studierende des Bachelorstudiengangs Informatik verpflichtend
- Bachelor Elektrotechnik/Informationstechnik und Informations- und Kommunikationstechnik hat eigenes Praktikum
- Das Praktikum wird in **Java** durchgeführt
- Termine:
- Dienstag: 8-10 (1), 10-12 (2), 14-16 (3), 16-18 (2)
- Mittwoch: 10-12 (1), 12-14 (2), 14-16 (1), 16-18 (1)
- Donnerstag: 8-10 (2), 10-12 (3), 12-14 (1), 16-18 (1)
- Freitag: 12-14 (2), 14-16 (2)
- Anmeldung über AsSESS (ab Donnerstag 8 Uhr; Anmeldeschluss Fr. 20 Uhr; Änderungen bis Montag 10 Uhr)

Organisatorisches

Übungen Praktikum Bachelor ETIT und IKT

- Eigenes Praktikum
- Für Bachelor ETIT ist das Praktikum Wahlpflicht
- Für Bachelor IKT ist das Praktikum verpflichtend
- Das Praktikum wird in **C/C++** durchgeführt
- Teilnahme am Java Praktikum der Informatik **nicht** zulässig
- Bis 24.4., 12 Uhr müssen sich die ETIT und IKT Studierenden über das LSF (Veranstaltungsnummer 080011) registriert haben
- LSF Link ist über den Lehrstuhl Kommunikationstechnik erreichbar
- Dort gibt es eigene Übungsaufgaben
- Termine: Montags ab 24.4. 14 Uhr

Organisatorisches

Übungen Praktikum

- Heimübungen, Präsenzübungen
- 50% der Punkte bei den Präsenzaufgaben
- 50% der Punkte bei den Heimaufgaben; Heimaufgaben müssen auch am Rechner präsentiert werden
- Bei Feiertagen -> andere Übung

Sonstiges

- Schüler -> Amer Krivosija
- Poolräume können außerhalb der Veranstaltungszeiten immer genutzt werden
- Weitere Informationen auf der Praktikumswebseite (erreichbar von der Vorlesungswebseite)

Organisatorisches

Lernraumbetreuung

- Wird zeitnah bekanntgegeben

Organisatorisches

Tests

- 1. Test: 13.06.
- 2. Test: wird noch bekanntgegeben (ca. 2 Wochen später als 1. Test)
- Einer der beiden Test muss mit 50% der Punkte bestanden werden (Studienleistung Übung; Teil 2)

Organisatorisches

Bei Fragen

- Meine Sprechzeiten: Montag 11-12 Uhr oder einfach nach der Vorlesung
- Fragen zum Prüfungsausschuss nur in meiner Sprechstunde

Organisatorische Fragen zur Vorlesung und Übung an

- Amer Krivosija (amer.krivosija@tu-dortmund.de)

Organisatorische Fragen zum Praktikum an

- Nils Kriege (nils.kriege@tu-dortmund.de)

- Außerdem INPUD Forum

Organisatorisches

Klausurtermine

- 7.8. 8-11 Uhr
- 25.9. 12-15 Uhr
- (Klausurlänge ist 180 Minuten)

Weitere Infos

- Vorlesungsseite
  http://ls2-www.cs.tu-dortmund.de/lehre/sommer2017/dap2/
- Oder von der Startseite des LS 2 -> Teaching -> DAP2

Einige Hinweise/Regeln

Klausur

- Eine Korrelation mit den Übungsaufgaben ist zu erwarten

Laptops

- Sind in der Vorlesung nicht zugelassen

Literatur

Skripte

- Kein Vorlesungsskript

Bücher und verwendete Literatur

- Cormen, Leisserson, Rivest, Stein: Introduction to Algorithms, MIT Press
- Kleinberg, Tardos: Algorithm Design, Addison Wesley

WWW

- Kurs „Introduction to Algorithms“ am MIT. Online Material (Folien, Video und Audio Files!)
- http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/

Lernziele

- *Bewertung von Algorithmen und Datenstrukturen*
  - Laufzeitanalyse
  - Speicherbedarf
  - Korrektheitsbeweise

- *Kenntnis grundlegender Algorithmen und Datenstrukturen*
  - Sortieren
  - Suchbäume
  - Neu: Verarbeitung sehr großer Datenmengen (Big Data)
  - Graphalgorithmen

- *Kenntnis grundlegender Entwurfsmethoden*
  - Teile und Herrsche
  - gierige Algorithmen
  - dynamische Programmierung

Lernziele

- *Unterschiede zu DAP1*
- DAP 1 behandelt Algorithmik aus der Perspektive der Softwaretechnik
- DAP 2 legt die theoretischen Grundlagen zur Algorithmenanalyse

Motivation

Beispiele für algorithmische Probleme, die z.T. mit Hilfe komplexer mathematischer Methoden gelöst werden

- Internetsuchmaschinen
- Berechnung von Bahnverbindungen
- Optimierung von Unternehmensabläufen
- Datenkompression
- Computer Spiele
- Datenanalyse (Big Data)

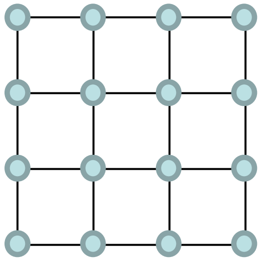
Alle diese Bereiche sind (immer noch) Stoff aktueller Forschung im Bereich Datenstrukturen und Algorithmen

Motivation

Problembeschreibung

- Ein $m \times n$-Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17×17 Gitter (289$ Problem)

- Beispiel:
  (4×4 Gitter)



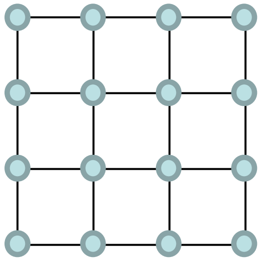
- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

Motivation

Problembeschreibung

- Ein $m \times n$-Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17×17 Gitter $\boxed{\text{Gelöst!}}$

- Beispiel:
  (4×4 Gitter)



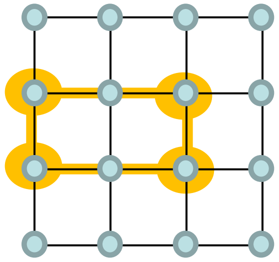
- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

Motivation

Problembeschreibung

- Ein $m \times n$-Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17×17 Gitter **Gelöst!**

- Beispiel: (4×4 Gitter)
- Die vier unterlegten Knoten dürfen z.B. nicht alle dieselbe Farbe haben



- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

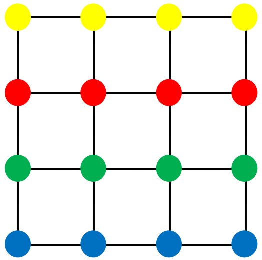
Motivation

Problembeschreibung

- Ein $m \times n$-Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17×17 Gitter

Gelöst!

- Beispiel:
  (4×4 Gitter)



4x4 Gitter ist 4-färbbar!
Geht es besser?

- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

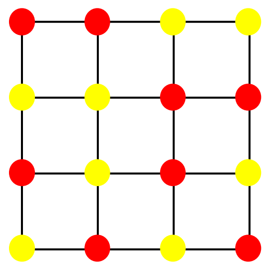
Motivation

Problembeschreibung

- Ein $m \times n$-Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17×17 Gitter

Gelöst!

- Beispiel:
  (4×4 Gitter)



Ja! 4x4 Gitter ist 2-färbbar!

- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

Motivation

17×17 *Problem*

- Es ist z.Z. nicht möglich, das 17×17 Problem mit einem Rechner zu lösen
- Warum ist dieses Problem so schwer zu lösen?
- Es gibt sehr viele Färbungen!

Fragen/Aufgaben

- Können wir die Laufzeit eines Algorithmus vorhersagen?
- Können wir bessere Algorithmen finden?

1. Teil der Vorlesung – Grundlagen der Algorithmenanalyse

Inhalt

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

Was ist ein Algorithmus?

Algorithmus

- Ein Algorithmus ist ein wohldefiniertes eindeutiges Berechnungsverfahren, das eine Eingabe in eine Ausgabe umformt
- Dabei besteht ein Algorithmus aus einer Sequenz von grundlegenden Berechnungsschritten

Berechnungsproblem

- Beschreibt eindeutig eine gewünschte Relation zwischen Eingabe und Ausgabe
- Ein Algorithmus kann als ein Verfahren zum Lösen eines Berechnungsproblems angesehen werden

Programm vs. Algorithmus

Programm

- Ein Programm ist eine Umsetzung eines Algorithmus in eine bestimmte Programmiersprache

Algorithmenentwurf

Anforderungen

- Korrektheit
- Effizienz (Laufzeit, Speicherplatz)

Entwurf umfasst

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

Algorithmenentwurf

Warum mathematische Korrektheitsbeweise?

- Fehler können fatale Auswirkungen haben (Steuerungssoftware in Flugzeugen, Autos, AKWs)
- Fehler können selten auftreten („Austesten“ funktioniert nicht)

Der teuerste algorithmische Fehler?

- Pentium bug (>400 Mio $)
- Enormer Image Schaden
- Trat relativ selten auf

Algorithmenentwurf

Warum Laufzeit/Speicherplatz optimieren?

- Riesige Datenmengen durch Vernetzung (Internet)
- Datenmengen wachsen schneller als Rechenleistung und Speicher
- Physikalische Grenzen
- Schlechte Algorithmen versagen häufig bereits bei kleinen und mittleren Eingabegrößen

Beschreibung von Algorithmen: Pseudocode

- Ziel: Wir wollen von syntaktischen Besonderheiten der Programmiersprachen abstrahieren
- Beschreibungssprache ähnlich wie C, Java, Pascal, etc…
- Hauptunterschied: Wir benutzen immer die klarste und präziseste Beschreibung
- Manchmal kann auch ein vollständiger Satz die beste Beschreibung sein
- Wir ignorieren Aspekte wie
  - Modularität
  - Fehlerbehandlung

Berechnungsproblem: 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 $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Beispiel:

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

Insertion Sort

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

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

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
   key ← A[j]
   i ← j − 1
   while i > 0 and A[i] > key do
      A[i + 1] ← A[i]
      i ← i − 1
   A[i + 1] ← key
```

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

Pseudocode

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

Insertion Sort

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

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

Pseudocode

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

Insertion Sort

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

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

Pseudocode

- Zuweisungen durch ←

Insertion Sort

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

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

Pseudocode

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

Insertion Sort

InsertionSort(Array A)

$\textbf{for}\ j \leftarrow 2\ \textbf{to}\ \text{length}[A]\ \textbf{do}$

$\quad \text{key} \leftarrow A[j]$

$\quad i \leftarrow j - 1$

$\quad \textbf{while}\ i > 0 \text{ and } A[i] > \text{key}\ \textbf{do}$

$\quad\quad A[i+1] \leftarrow A[i]$

$\quad\quad i \leftarrow i - 1$

$\quad A[i+1] \leftarrow \text{key}$

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

Pseudocode

- Keine Typdeklaration, wenn Typ klar aus dem Kontext

Insertion Sort

InsertionSort(Array A)

$\textbf{for}\ j \leftarrow 2\ \textbf{to}\ \mathrm{length}[A]\ \textbf{do}$
$\quad \mathrm{key} \leftarrow A[j]$
$\quad i \leftarrow j - 1$
$\quad \textbf{while}\ i > 0\ \mathrm{and}\ A[i] > \mathrm{key}\ \textbf{do}$
$\quad\quad A[i+1] \leftarrow A[i]$
$\quad\quad i \leftarrow i - 1$
$\quad A[i+1] \leftarrow \mathrm{key}$

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

Pseudocode

- Zugriff auf Feldelemente mit [.]

Insertion Sort

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

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

Pseudocode

- Verbunddaten sind typischerweise als Objekte organisiert
- Ein Objekt besteht aus Attributen instanziiert durch Attributwerte
- Beispiel: Feld wird als Objekt mit Attribut Länge betrachtet

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ key

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

Pseudocode

- Beispiel: Objekt ist Graph G mit Knotenmenge V
- Auf den Attributwert V von Graph G wird mit $V[G]$ zugegriffen

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \text{length}[A] \textbf{ do}$
  $\text{key} \leftarrow A[j]$
  $i \leftarrow j - 1$
  $\textbf{while } i > 0 \text{ and } A[i] > \text{key} \textbf{ do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow \text{key}$

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

Pseudocode

- Objekte werden als Zeiger referenziert, d.h. für alle Attribute f eines Objektes x bewirkt $y \leftarrow x$, dass gilt: $f[y] = f[x]$.

Insertion Sort

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

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

Pseudocode

- Blockstruktur durch Einrücken

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  A[i + 1] ← key
```

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

Pseudocode

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

Insertion Sort

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

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

Pseudocode

- Prozeduren „call-by-value“ ; jede aufgerufene Prozedur erhält neue Kopie der übergebenen Variable
- Die lokalen Änderungen sind nicht global sichtbar
- Bei Objekten wird nur der Zeiger kopiert (lokale Änderungen am Objekt global sichtbar)

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  A[i + 1] ← key
```

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

Pseudocode

- Rückgabe von Parametern durch **return**

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i + 1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i + 1] \leftarrow$ key

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

Pseudocode

- Kommentare durch ➢

Insertion Sort

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

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

Beispiel

8	**15**	**3**	**14**	**7**	**6**	**18**	**19**

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
   key ← A[j]
   i ← j − 1
   while i > 0 and A[i] > key do
      A[i + 1] ← A[i]
      i ← i − 1
   A[i + 1] ← key
```

- Eingabegröße n
- $\text{length}[A] = n$

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

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ key

- Eingabegröße n
- length$[A] = n$

	key						
	↑						
8	**15**	**3**	**14**	**7**	**6**	**18**	**19**
1	j						n

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $\boxed{i \leftarrow j - 1}$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ key

- Eingabegröße n
- length$[A] = n$

	key $\uparrow$						
8	15	3	14	7	6	18	19
i	j						n

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  A[i + 1] ← key
```

- Eingabegröße n
- $\text{length}[A] = n$

key

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

Insertion Sort

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

- Eingabegröße n
- $\text{length}[A] = n$

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

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

- Eingabegröße n
- $\text{length}[A] = n$

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

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  A[i + 1] ← key
```

- Eingabegröße n
- $\text{length}[A] = n$

		key					
8	**15**	**3**	**14**	**7**	**6**	**18**	**19**
1		j					n

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  A[i + 1] ← key
```

- Eingabegröße n
- $\mathrm{length}[A] = n$

$\mathrm{key} = 3$

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

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ 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)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $\boxed{A[i+1] \leftarrow A[i]}$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ key

- Eingabegröße n
- length$[A] = n$

key $= 3$

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

Insertion Sort

InsertionSort(Array A)

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

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

- Eingabegröße n
- length$[A] = n$

key $= 3$

8	**15**	**15**	**14**	**7**	**6**	**18**	**19**
i		j					n

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \text{length}[A] \textbf{ do}$
  $\text{key} \leftarrow A[j]$
  $i \leftarrow j-1$
  $\textbf{while } i > 0 \text{ and } A[i] > \text{key } \textbf{do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i-1$
  $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- $\text{length}[A] = n$

$\text{key} = 3$

8	**8**	**15**	**14**	**7**	**6**	**18**	**19**
i		j					n

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j-1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i-1$
  $A[i+1] \leftarrow$ key

- Eingabegröße n
- length$[A] = n$

key $= 3$

8	**8**	**15**	**14**	**7**	**6**	**18**	**19**
1		j					n

i

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  A[i + 1] ← key
```

- Eingabegröße n
- $\text{length}[A] = n$

$\text{key} = 3$

8	**8**	**15**	**14**	**7**	**6**	**18**	**19**
1		j					n

i

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $\boxed{A[i+1] \leftarrow \text{key}}$

- Eingabegröße n
- length$[A] = n$

key $= 3$

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

Insertion Sort

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

- Eingabegröße n
- $\text{length}[A] = n$

$$\underbrace{\begin{array}{|c|c|c|}\hline 3 & 8 & 15 \\ \hline\end{array}}_{\text{Sortiert}}$$

3	8	15	14	7	6	18	19
1			j				n

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \text{length}[A] \textbf{ do}$
  $\boxed{\text{key} \leftarrow A[j]}$
  $i \leftarrow j - 1$
  $\textbf{while } i > 0 \text{ and } A[i] > \text{key } \textbf{do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- $\text{length}[A] = n$

Sortiert

key

3	8	15	14	7	6	18	19
1			j				n

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \mathrm{length}[A] \textbf{ do}$
  $\mathrm{key} \leftarrow A[j]$
  $i \leftarrow j - 1$
  $\textbf{while } i > 0 \text{ and } A[i] > \mathrm{key} \textbf{ do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow \mathrm{key}$

- Eingabegröße n
- $\mathrm{length}[A] = n$

Sortiert

$\mathrm{key} = 14$

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

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \mathrm{length}[A] \textbf{ do}$

  $\mathrm{key} \leftarrow A[j]$

  $i \leftarrow j-1$

  $\textbf{while } i > 0 \text{ and } A[i] > \mathrm{key} \textbf{ do}$

    $A[i+1] \leftarrow A[i]$

    $i \leftarrow i-1$

  $A[i+1] \leftarrow \mathrm{key}$

- Eingabegröße n
- $\mathrm{length}[A] = n$

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

Sortiert

$\mathrm{key} = 14$

3	**8**	**15**	**14**	**7**	**6**	**18**	**19**
1		$j-1$	j				n

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ key

- Eingabegröße n
- length$[A] = n$

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

Sortiert

key $= 14$

3	**8**	**15**	**14**	**7**	**6**	**18**	**19**
1		$j-1$	j				n

Insertion Sort

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

- Eingabegröße n
- $\mathrm{length}[A] = n$

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

$\mathrm{key} = 14$

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

Insertion Sort

InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  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

$\text{key} = 14$

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

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \text{length}[A] \textbf{ do}$
  $\text{key} \leftarrow A[j]$
  $i \leftarrow j - 1$
  $\textbf{while } i > 0 \text{ and } A[i] > \text{key} \textbf{ do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow \text{key}$

- 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

$\text{key} = 14$

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

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \mathrm{length}[A] \textbf{ do}$
  $\mathrm{key} \leftarrow A[j]$
  $i \leftarrow j - 1$
  $\textbf{while } i > 0 \text{ and } A[i] > \mathrm{key} \textbf{ do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow \mathrm{key}$

- Eingabegröße n
- $\mathrm{length}[A] = n$

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

- Speichere key in Lücke

Sortiert

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

Insertion Sort

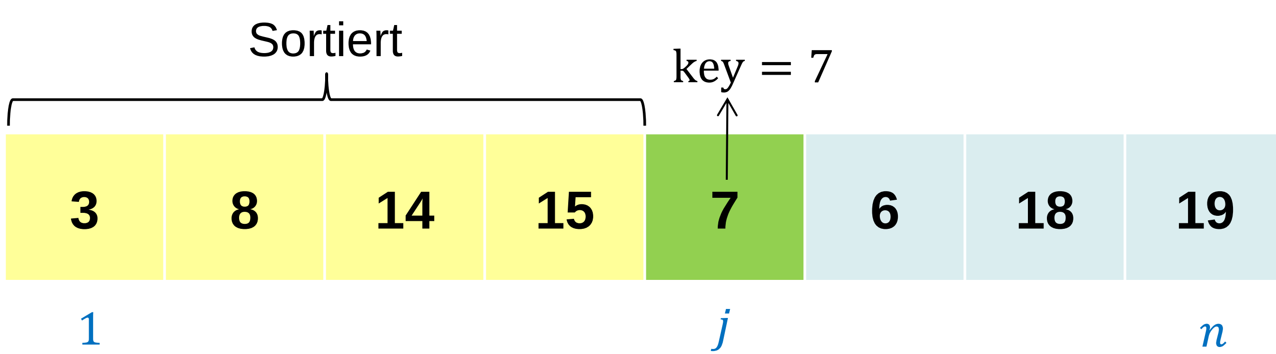
InsertionSort(Array A)

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

- 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

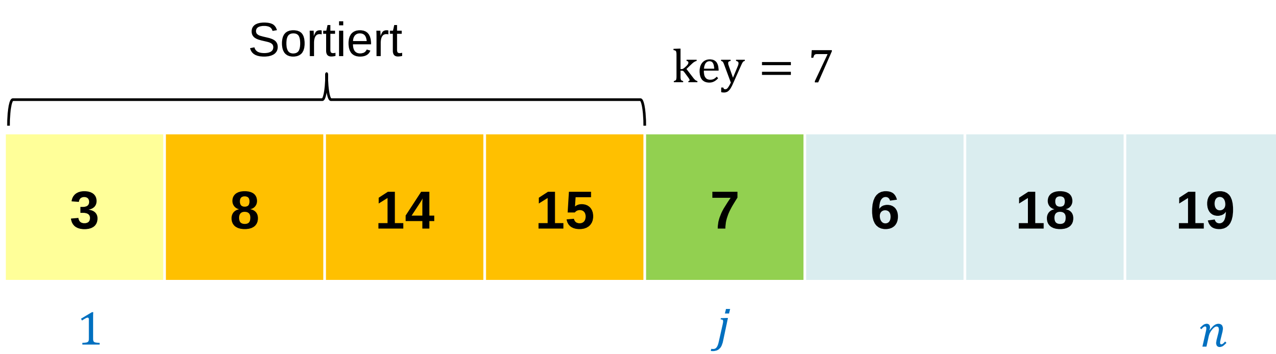
InsertionSort(Array A)

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

- 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)

$\textbf{for } j \leftarrow 2 \textbf{ to } \mathrm{length}[A] \textbf{ do}$
  $\mathrm{key} \leftarrow A[j]$
  $i \leftarrow j - 1$
  $\textbf{while } i > 0 \text{ and } A[i] > \mathrm{key} \textbf{ do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow \mathrm{key}$

- Eingabegröße n
- $\mathrm{length}[A] = n$

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

- Speichere key in Lücke

$\mathrm{key} = 7$

3	**8**	**8**	**14**	**15**	**6**	**18**	**19**
1				j			n

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ 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

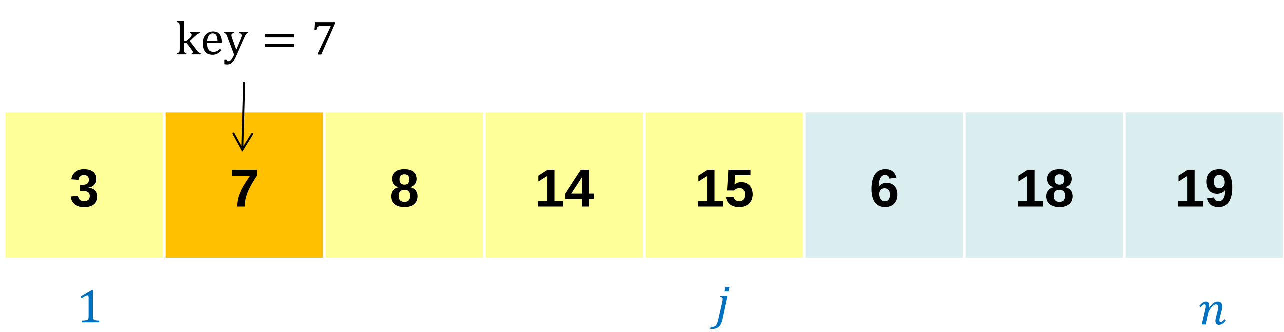
key $= 7$

3	**8**	**8**	**14**	**15**	**6**	**18**	**19**
1				j			n

Insertion Sort

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



Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ 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

Sortiert

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

Insertion Sort

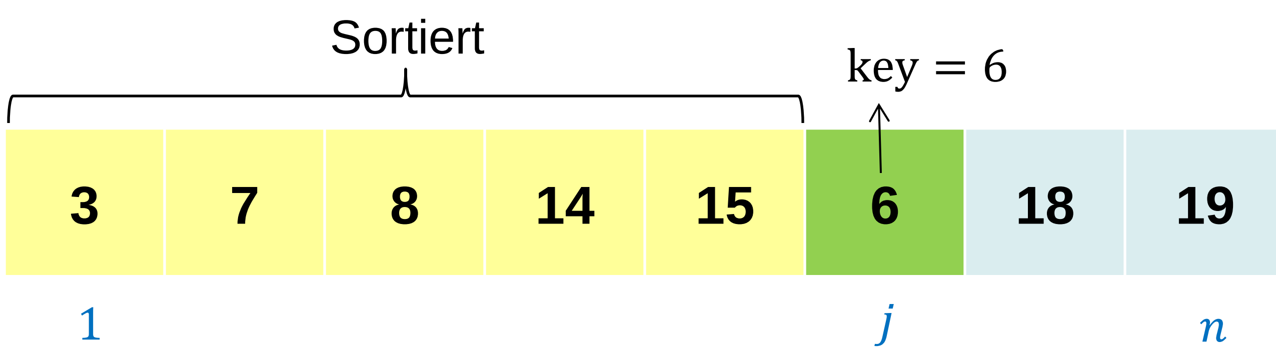
InsertionSort(Array A)

```
for j ← 2 to length[A] do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i ← i − 1
  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



Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ 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

$\underbrace{3 \quad 7 \quad 8 \quad 14 \quad 15}_{\text{Sortiert}}$ key $= 6$

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

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \text{length}[A] \textbf{ do}$
$\quad \text{key} \leftarrow A[j]$
$\quad i \leftarrow j - 1$
$\quad \textbf{while } i > 0 \text{ and } A[i] > \text{key} \textbf{ do}$
$\quad\quad A[i+1] \leftarrow A[i]$
$\quad\quad i \leftarrow i - 1$
$\quad A[i+1] \leftarrow \text{key}$

- 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

$\text{key} = 6$

3	**7**	**7**	**8**	**14**	**15**	**18**	**19**
1					j		n

Insertion Sort

InsertionSort(Array A)

$\textbf{for } j \leftarrow 2 \textbf{ to } \text{length}[A] \textbf{ do}$
  $\text{key} \leftarrow A[j]$
  $i \leftarrow j - 1$
  $\textbf{while } i > 0 \text{ and } A[i] > \text{key} \textbf{ do}$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow \text{key}$

- 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

$\text{key} = 6$

3	**7**	**7**	**8**	**14**	**15**	**18**	**19**
1					j		n

Insertion Sort

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

- Eingabegröße n
- $\mathrm{length}[A] = n$

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

- Speichere key in Lücke

$\mathrm{key} = 6$

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

Insertion Sort

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ 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

Sortiert

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

Insertion Sort

InsertionSort(Array A)

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

- 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)
for j ← 2 to length[A] do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    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

Sortiert

3	**6**	**7**	**8**	**14**	**15**	**18**	**19**

1 n

Laufzeitanalyse

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

- 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

Fragestellung

Wie kann man die Laufzeit eines Algorithmus vorhersagen?

Laufzeitanalyse

Laufzeit hängt ab von

- Größe der Eingabe (Parameter n)
- Art der Eingabe
  (Insertionsort ist schneller auf sortierten Eingaben)

Analyse

- Parametrisiere Laufzeit als Funktion der Eingabegröße
- Finde obere Schranken (Garantien) an die Laufzeit

Laufzeitanalyse

Worst-Case Analyse

- Für jedes n definiere Laufzeit
  $T(n)$ = Maximum über alle Eingaben der Größe n
- Garantie für jede Eingabe
- Standard

Average-Case Analyse

- Für jedes n definiere Laufzeit
  $T(n)$ = Durchschnitt über alle Eingaben der Größe n
- Hängt von Definition des Durchschnitts ab (wie sind die Eingaben verteilt)

Laufzeitanalyse

Laufzeit hängt auch ab von

- Hardware
  (Prozessor, Cache, Pipelining)
- Software
  (Betriebssystem, Programmiersprache, Compiler)

Aber

- Analyse soll unabhängig von Hard- und Software gelten

Laufzeitanalyse

Maschinenmodell

- Eine Pseudocode-Instruktion braucht einen Zeitschritt
- Wird eine Instruktion r-mal aufgerufen, werden r Zeitschritte benötigt
- Formales Modell: Random Access Machines (RAM Modell)

Idee

- Ignoriere rechnerabhängige Konstanten
- Betrachte Wachstum von $T(n)$ für $n \to \infty$

„Asymptotische Analyse“

Laufzeitanalyse

InsertionSort(Array A)

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

Was ist die Eingabegröße?

Laufzeitanalyse

InsertionSort(Array A)

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

Was ist die Eingabegröße?
Die Länge des Feldes A

Laufzeitanalyse

InsertionSort(Array A)

for $j \leftarrow 2$ **to** length$[A]$ **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$ key **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow$ key

Zeit:

n

Laufzeitanalyse

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

Laufzeitanalyse

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

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
for $j \leftarrow 2$ **to** length$[A]$ **do**	n
key $\leftarrow A[j]$	$n-1$
$i \leftarrow j-1$	$n-1$
while $i > 0$ and $A[i] >$ key **do**	$n-1 + \sum t_k$
$A[i+1] \leftarrow A[i]$	
$i \leftarrow i-1$	
$A[i+1] \leftarrow$ key	

t_k: Anzahl Wiederholungen der **while**-Schleife bei Laufindex $j = k$

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
for $j \leftarrow 2$ **to** length$[A]$ **do**	n
key $\leftarrow A[j]$	$n-1$
$i \leftarrow j-1$	$n-1$
while $i > 0$ and $A[i] >$ key **do**	$n-1 + \sum t_k$
$A[i+1] \leftarrow A[i]$	$\sum t_k$
$i \leftarrow i-1$	
$A[i+1] \leftarrow$ key	

t_k: Anzahl Wiederholungen der **while**-Schleife bei Laufindex $j = k$

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** $j \leftarrow 2$ **to** length$[A]$ **do**	n
2. key $\leftarrow A[j]$	$n-1$
3. $i \leftarrow j-1$	$n-1$
4. **while** $i > 0$ and $A[i] >$ key **do**	$n-1 + \sum t_k$
5. $A[i+1] \leftarrow A[i]$	$\sum t_k$
6. $i \leftarrow i-1$	$\sum t_k$
7. $A[i+1] \leftarrow$ key	

t_k: Anzahl Wiederholungen der **while**-Schleife bei Laufindex $j = k$

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** $j \leftarrow 2$ **to** length$[A]$ **do**	n
2. key $\leftarrow A[j]$	$n-1$
3. $i \leftarrow j-1$	$n-1$
4. **while** $i > 0$ and $A[i] >$ key **do**	$n-1 + \sum t_k$
5. $A[i+1] \leftarrow A[i]$	$\sum t_k$
6. $i \leftarrow i-1$	$\sum t_k$
7. $A[i+1] \leftarrow$ key	$n-1$

t_k: Anzahl Wiederholungen der **while**-Schleife bei Laufindex $j = k$

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** $j \leftarrow 2$ **to** length$[A]$ **do**	n
2. key $\leftarrow A[j]$	$n-1$
3. $i \leftarrow j-1$	$n-1$
4. **while** $i > 0$ and $A[i] >$ key **do**	$n-1 + \sum t_k$
5. $A[i+1] \leftarrow A[i]$	$\sum t_k$
6. $i \leftarrow i-1$	$\sum t_k$
7. $A[i+1] \leftarrow$ key	$n-1$
	$5n-4+3\sum t_k$

t_k: Anzahl Wiederholungen der **while**-Schleife bei Laufindex $j=k$

Laufzeitanalyse

Worst-Case Analyse

- $t_k = k - 1$ für absteigend sortierte Eingabe (schlechtester Fall)

$$T(n) = 5n - 4 + 3 \cdot \sum_{k=2}^{n}(k-1) = 2n - 4 + 3 \cdot \sum_{k=1}^{n} k$$

Laufzeitanalyse

Worst-Case Analyse

- $t_k = k - 1$ für absteigend sortierte Eingabe (schlechtester Fall)

$$T(n) = 5n - 4 + 3 \cdot \sum_{k=2}^{n} (k-1) = 2n - 4 + 3 \cdot \sum_{k=1}^{n} k$$

$$= 2n - 4 + 3 \cdot \frac{n(n+1)}{2} = \frac{3n^2 + 7n - 8}{2}$$

Speicherplatzanalyse

Maschinenmodell (Speicherplatz)

- Rechner hat beliebig viele Speicherzellen zur Verfügung
- Die Speicherzellen sind mit natürlichen Zahlen nummeriert
- Jede Variable benötigt eine Speicherzelle
- Jedes Feld $A[1..k]$ benötigt k Speicherzellen
- Jede Referenz benötigt eine Speicherzelle
- Verbunddaten benötigen die Summe der Speicherzellen, die die einzelnen Daten des Verbunds benötigen

Speicherplatzanalyse

Maschinenmodell (Speicherplatz - Verwaltung)

- Das Betriebssystem (bzw. der Compiler) übernimmt die Zuordnung von Variablen zu ihren Speicherzellen
- Eine Referenz ist die Nummer der Speicherzelle, in der eine Variable oder ein Objekt abgespeichert ist
- Bei Verbunddaten verweist die Referenz auf die erste Speicherzelle

Speicherplatzanalyse

Worst-Case Analyse

- Für jedes n definiere Speicherplatz
  $S(n)$ = maximaler Speicherplatz über alle Eingaben der Größe n

Average-Case Analyse

- Für jedes n definiere Speicherplatz
  $S(n)$ = durchschnittlicher Speicherplatz über alle Eingaben der Größe n

Speicherplatzanalyse

InsertionSort(Array A)

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

Verwendete Variablen

- Integers i, j, key — 3
- Feld A — n

$$n + 3$$

Laufzeit- und Speicherplatzanalyse

Diskussion

- Die konstanten Faktoren sind wenig aussagekräftig, da wir bereits bei den einzelnen Befehlen konstante Faktoren ignorieren
- Je nach Rechnerarchitektur und genutzten Befehlen könnte also z.B. $3n + 4$ langsamer sein als $5n + 7$
- Betrachte nun Algorithmus A mit Laufzeit $100n$ und Algorithmus B mit Laufzeit $5n^2$
- Ist n klein, so ist Algorithmus B schneller
- Ist n groß, so wird das Verhältnis Laufzeit B / Laufzeit A beliebig groß
- Algorithmus B braucht also einen beliebigen Faktor mehr Laufzeit als A (wenn die Eingabe lang genug ist)
- Ähnliches gilt für Speicherplatz

Asymptotische Analyse

Idee (asymptotische Analyse)

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

Asymptotische Analyse

$\boldsymbol{O}$-Notation

- $\mathbf{O}\big(f(n)\big) = \{g(n): \exists c > 0, n_0 > 0$, so dass für alle $n \geq n_0$ gilt $g(n) \leq c \cdot f(n)\}$
- (wobei $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$)

Interpretation

- $g(n) \in \mathbf{O}\big(f(n)\big)$ bedeutet, dass $g(n)$ für $n \rightarrow \infty$ höchstens genauso stark wächst wie $f(n)$
- Beim Wachstum ignorieren wir Konstanten

Asymptotische Analyse

Beispiele

- $10n \in \mathbf{O}(n)$
- $10n \in \mathbf{O}(n^2)$
- $n^2 \notin \mathbf{O}(1000n)$
- $\mathbf{O}(1000n) = \mathbf{O}(n)$

Hierarchie

- $\mathbf{O}(\log n) \subseteq \mathbf{O}(\log^2 n) \subseteq \mathbf{O}(\log^c n) \subseteq \mathbf{O}(n^\varepsilon) \subseteq \mathbf{O}(\sqrt{n}) \subseteq \mathbf{O}(n)$
- $\mathbf{O}(n) \subseteq \mathbf{O}(n^2) \subseteq \mathbf{O}(n^c) \subseteq \mathbf{O}(2^n)$
- $\left(\text{für } c \geq 2 \text{ und } 0 < \varepsilon \leq \frac{1}{2}\right)$

Asymptotische Analyse

Gilt … ?

A) $n + n^2 \in \mathbf{O}(n^2)$

B) $\log(n^2) \in \mathbf{O}(\log n)$

C) $\log(n^2) \in \mathbf{O}(\log^2 n)$

D) $\sqrt{n} \in \mathbf{O}(\log n)$

Beispiele

- $10n \in \mathbf{O}(n)$
- $10n \in \mathbf{O}(n^2)$
- $n^2 \notin \mathbf{O}(1000n)$
- $\mathbf{O}(1000n) = \mathbf{O}(n)$

Hierarchie

- $\mathbf{O}(\log n) \subseteq \mathbf{O}(\log^2 n) \subseteq \mathbf{O}(\log^c n) \subseteq \mathbf{O}(n^\varepsilon) \subseteq \mathbf{O}(\sqrt{n}) \subseteq \mathbf{O}(n)$
- $\mathbf{O}(n) \subseteq \mathbf{O}(n^2) \subseteq \mathbf{O}(n^c) \subseteq \mathbf{O}(2^n)$
- $\left(\text{für } c \geq 2 \text{ und } 0 < \varepsilon \leq \frac{1}{2}\right)$

Asymptotische Analyse

Ω-Notation

- $\mathbf{\Omega}\big(f(n)\big) = \{g(n)\colon \exists c > 0, n_0 > 0$, so dass für alle $n \geq n_0$ gilt $g(n) \geq c \cdot f(n)\}$
- (wobei $f, g\colon \mathbb{N} \to \mathbb{R}^+$)

Interpretation

- $g(n) \in \mathbf{\Omega}\big(f(n)\big)$ bedeutet, dass $g(n)$ für $n \to \infty$ mindestens so stark wächst wie $f(n)$
- Beim Wachstum ignorieren wir Konstanten

Asymptotische Analyse

Beispiele

- $10n \in \mathbf{\Omega}(n)$
- $1000n \notin \mathbf{\Omega}(n^2)$
- $n^2 \in \mathbf{\Omega}(n)$
- $\mathbf{\Omega}(1000n) = \mathbf{\Omega}(n)$
- $f(n) = \mathbf{\Omega}\big(g(n)\big) \Leftrightarrow g(n) = \mathbf{O}(f(n))$

Asymptotische Analyse

Θ-Notation

- $g(n) \in \mathbf{\Theta}\big(f(n)\big) \Leftrightarrow g(n) = \mathbf{O}\big(f(n)\big)$ und $g(n) = \mathbf{\Omega}\big(f(n)\big)$

Beispiele

- $1000n \in \mathbf{\Theta}(n)$
- $10n^2 + 1000n \in \mathbf{\Theta}(n^2)$
- $n^{1-\sin(\pi n/2)} \notin \mathbf{\Theta}(n)$

Asymptotische Analyse

$\boldsymbol{o}$-Notation

- $\mathbf{o}(f(n)) \in \{g(n) \colon \forall c > 0\ \exists n_0 > 0,$ so dass für alle $n \geq n_0$ gilt $c \cdot g(n) < f(n)\}$
- (wobei $f, g \colon \mathbb{N} \to \mathbb{R}^+$)

$\boldsymbol{\omega}$-Notation

- $f(n) \in \boldsymbol{\omega}(g(n)) \Leftrightarrow g(n) \in \mathbf{o}(f(n))$

Asymptotische Analyse

Beispiele

- $n \in \mathbf{o}(n^2)$
- $n \notin \mathbf{o}(n)$

Eine weitere Interpretation

- Grob gesprochen sind $\mathbf{O}, \mathbf{\Omega}, \mathbf{\Theta}, \mathbf{o}, \boldsymbol{\omega}$ die „asymptotischen Versionen" von $\leq, \geq, =, <, >$ (in dieser Reihenfolge)

Schreibweise

- Wir schreiben häufig $f(n) = \mathbf{O}\big(g(n)\big)$ anstelle von $f(n) \in \mathbf{O}\big(g(n)\big)$

Asymptotische Analyse

Worst-Case Laufzeitanalyse (Insertion Sort)

- $t_k = k - 1$ für absteigend sortierte Eingabe (schlechtester Fall)

$$T(n) = 5n - 4 + 3 \cdot \sum_{k=2}^{n} (k-1) = 2n - 4 + 3 \cdot \sum_{k=1}^{n} k$$

$$= 2n - 4 + 3 \cdot \frac{n(n+1)}{2} = \frac{3n^2+7n-8}{2} = \mathbf{\Theta}(n^2)$$

Worst-Case Speicherplatzanalyse

- $n + 3 = \mathbf{\Theta}(n)$ Speicherplatz

Zusammenfassung

Rechenmodell

- Abstrahiert von maschinennahen Einflüssen wie Cache, Pipelining, Prozessor, etc.
- Jede Pseudocodeoperation braucht einen Zeitschritt
- Jedes Datum benötigt eine Speicherzelle

Asymptotische Analyse

- Normalerweise Worst-Case, manchmal Average-Case (sehr selten auch Best-Case)
- Asymptotische Analyse für $n \to \infty$
- Ignorieren von Konstanten → $\mathbf{O}$-Notation