Informatik II

Übung 2

FS 2019

Heutiges Programm

1. Feedback letzte Übung
2. Wiederholung Theorie
   - Analyse von Programmen
   - Weitere Sortieralgorithmen
3. Programmieraufgabe
   - Collections in Java

1. Feedback letzte Übung

2. Wiederholung Theorie

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Das Code-Fragment ruft `f()` $\Theta(n^2)$ mal auf: die äußere Schleife wird $n/9$ mal durchlaufen, und die innere Schleife ruft `f()` i mal auf.

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.
Die zweite innere Schleife ruft `f()` $\lfloor \log_2(n) \rfloor + 1$ mal auf, in Summe haben wir $\Theta(n \log(n))$ Aufrufe.

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n > 0$?

```
void g(int n){
        if (n>1){
                g(n/2);
        }
        else{
                f();
        }
}
```

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n > 0$?

```
void g(int n){
        if (n>1){
                g(n/2);
        }
        else{
                f();
        }
}
```

Es findet nur ein einziger Aufruf an f() statt, am Ende der Rekursion

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n > 0$?

```
void g(int n){
        if (n>1){
                g(n-1);
        }
        f();
}
```

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n > 0$?

```
void g(int n){
        if (n>1){
                g(n−1);
        }
        f();
}
```

Rekurrenz

$$T(n) = \begin{cases} T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases}$$

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n > 0$?

```
void g(int n){
        if (n>1){
                g(n−1);
        }
        f();
}
```

Rekurrenz

$$T(n) = \begin{cases} T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n)$$

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Rekurrenz

$$T(n) = \begin{cases} 2T(n/2) & n > 1 \\ 1 & n = 1 \end{cases}$$

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Rekurrenz

$$T(n) = \begin{cases} 2T(n/2) & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n)$$

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

Rekurrenz

$$T(n) = \begin{cases} 2T(n/2) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

Analyse

Wie oft wird `f()` in `g(n)` aufgerufen, abhängig von $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

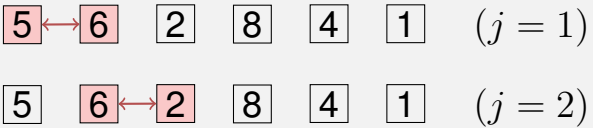
Rekurrenz

$$T(n) = \begin{cases} 2T(n/2) + n & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n \log n)$$

Bubble Sort

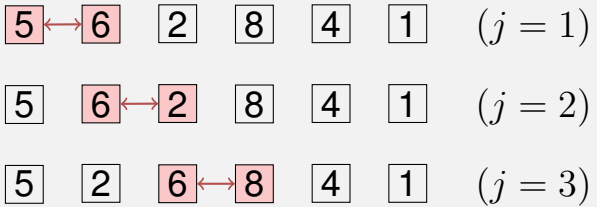
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen

Bubble Sort



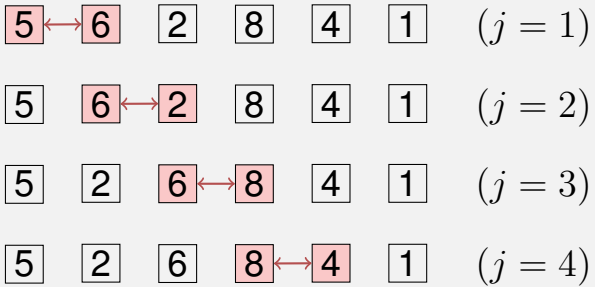
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen

Bubble Sort



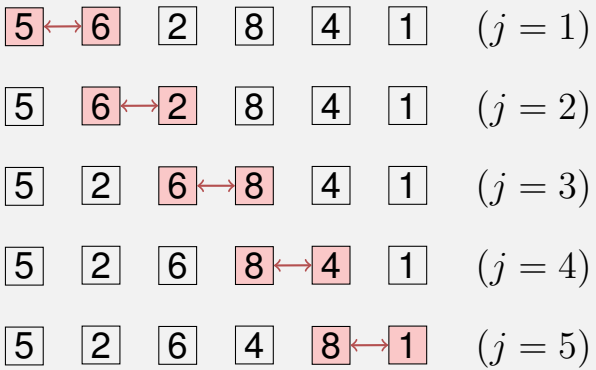
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen

Bubble Sort



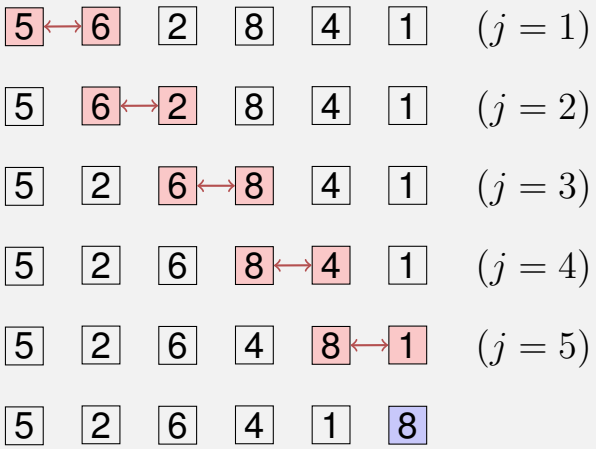
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen

Bubble Sort



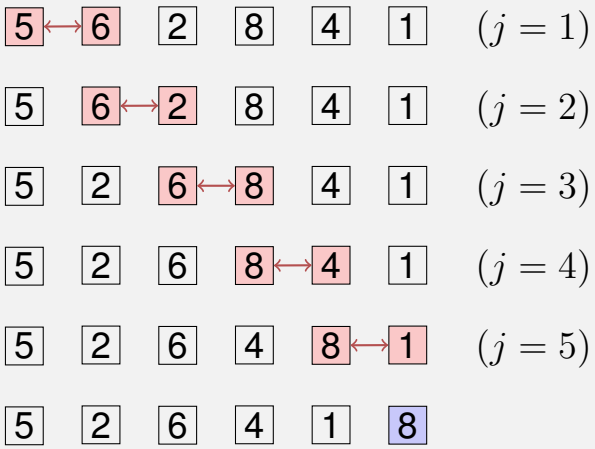
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen

Bubble Sort



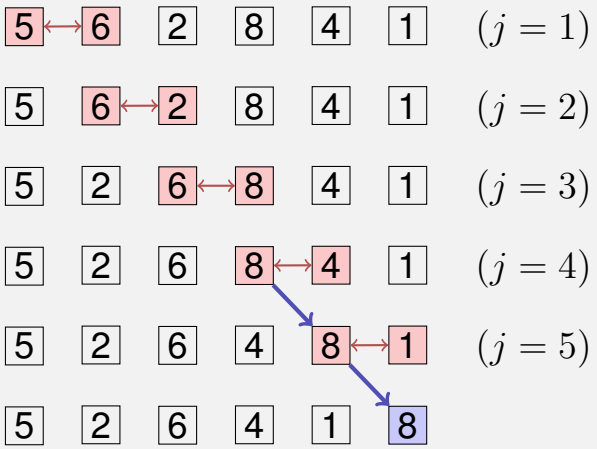
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen

Bubble Sort



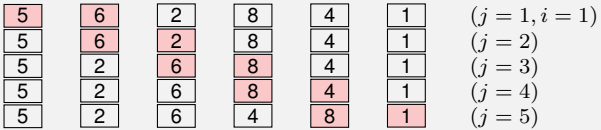
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen
- Nicht sortiert! ☹.

Bubble Sort



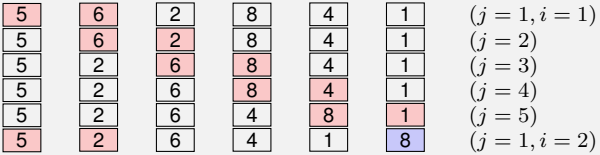
- Vertausche benachbarte Elemente, wenn sie in falscher Reihenfolge stehen
- Nicht sortiert! ☹.
- Aber das grösste Element wandert ganz nach rechts.
  $\Rightarrow$ Iteriere ☺

Bubble Sort



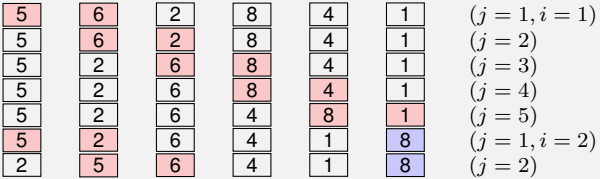
- Wende das Verfahren iterativ an.

Bubble Sort



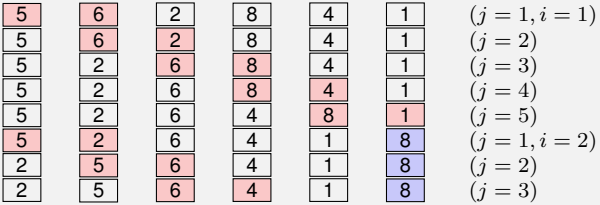
- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$,

Bubble Sort



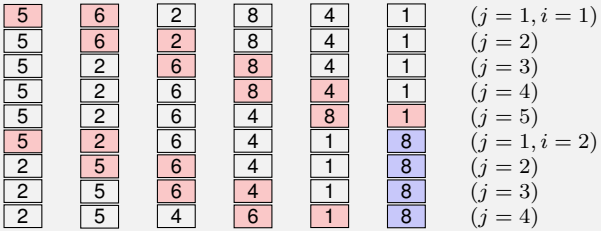
- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$, dann $A[1, \ldots, n-1]$,

Bubble Sort



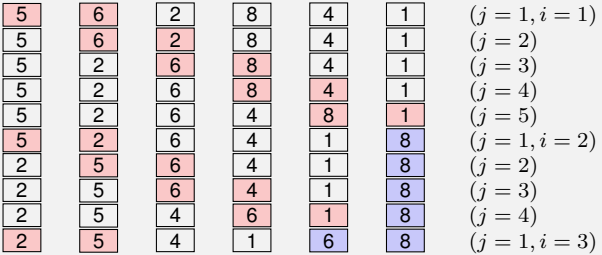
- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$, dann $A[1, \ldots, n-1]$,

Bubble Sort



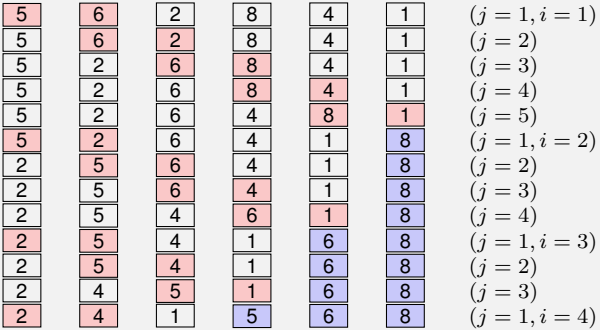
- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$, dann $A[1, \ldots, n-1]$,

Bubble Sort



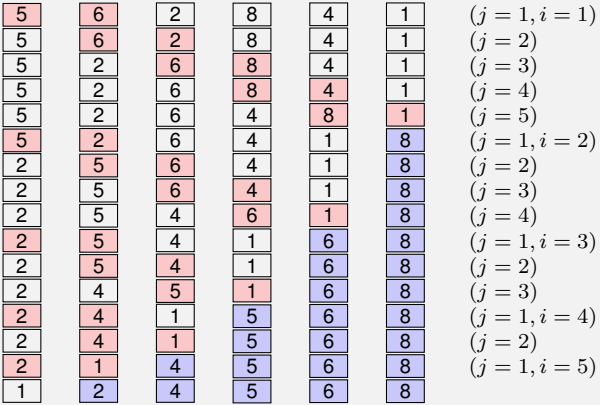
- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$, dann $A[1, \ldots, n-1]$, dann $A[1, \ldots, n-2]$,

Bubble Sort



- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$, dann $A[1, \ldots, n-1]$, dann $A[1, \ldots, n-2]$,

Bubble Sort



- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$, dann $A[1, \ldots, n-1]$, dann $A[1, \ldots, n-2]$, etc.

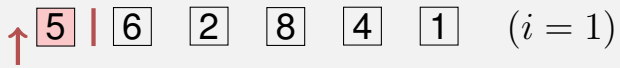
Algorithmus: Bubblesort

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sortiertes Array A

```
for i ← 1 to n − 1 do
    for j ← 1 to n − i do
        if A[j] > A[j + 1] then
            swap(A[j], A[j + 1]);
```

Sortieren durch Einfügen

Sortieren durch Einfügen

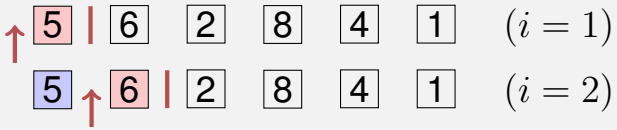


- Iteratives Vorgehen: $i = 1...n$

Sortieren durch Einfügen

- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.

Sortieren durch Einfügen

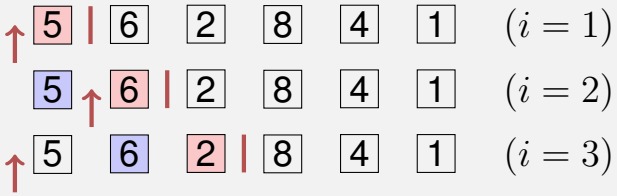


- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen,

Sortieren durch Einfügen

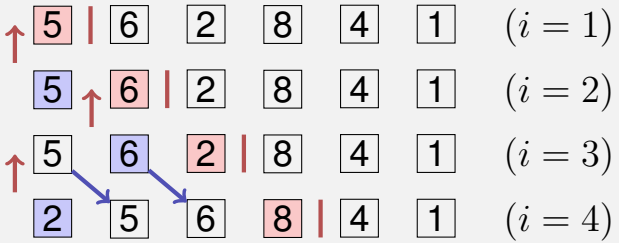
- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen,

Sortieren durch Einfügen



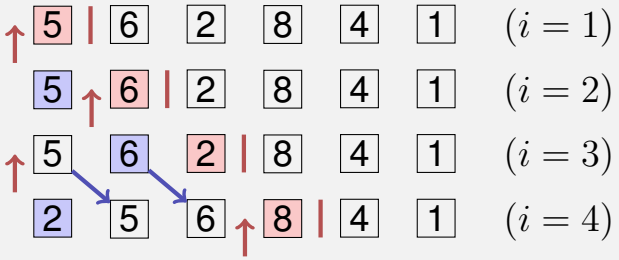
- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen,

Sortieren durch Einfügen



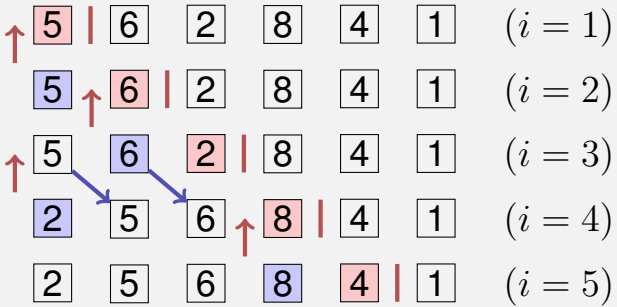
- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



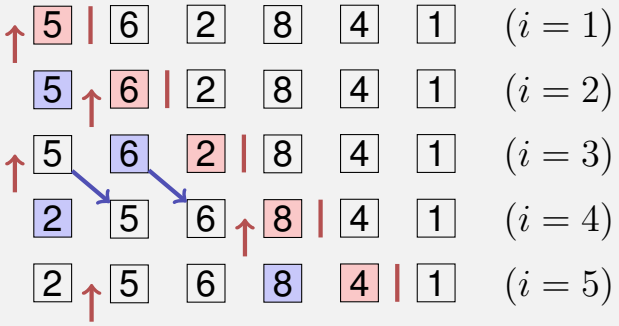
- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



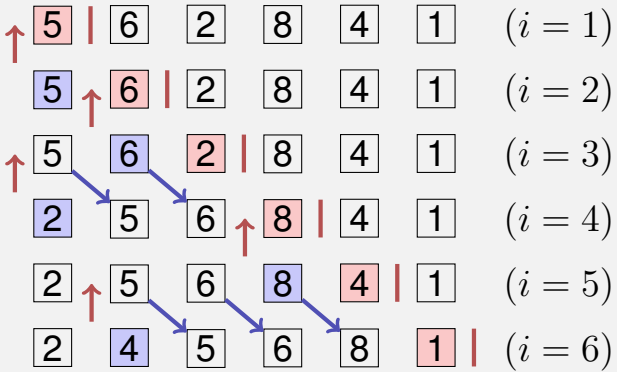
- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen

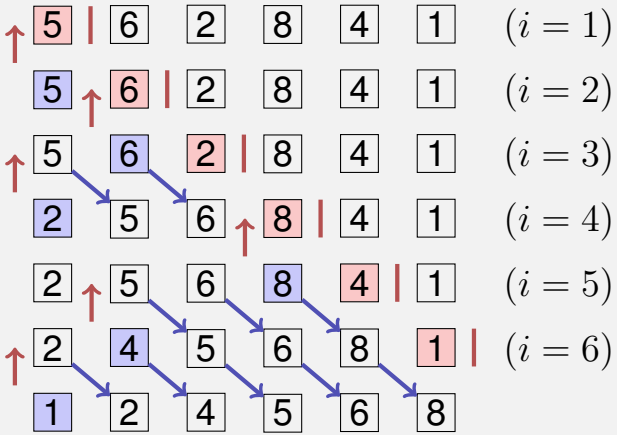


- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen

- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen

? Welchen Nachteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

Sortieren durch Einfügen

? Welchen Nachteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

! Im schlechtesten Fall viele Elementverschiebungen.

? Welchen Vorteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

Sortieren durch Einfügen

? Welchen Nachteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

! Im schlechtesten Fall viele Elementverschiebungen.

? Welchen Vorteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

! Der Suchbereich (Einfügebereich) ist bereits sortiert.
Konsequenz: binäre Suche möglich.

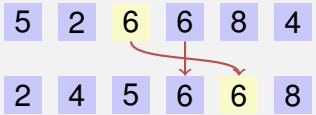
Algorithmus: Sortieren durch Einfügen

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sortiertes Array A

for $i \leftarrow 2$ **to** n **do**
    $x \leftarrow A[i]$
    $p \leftarrow \text{BinarySearch}(A[1...i-1], x)$; // Kleinstes $p \in [1, i]$ mit $A[p] \geq x$
    for $j \leftarrow i-1$ **downto** p **do**
        $A[j+1] \leftarrow A[j]$
    $A[p] \leftarrow x$

Stabile und in-situ-Sortieralgorithmen

- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht. [1]

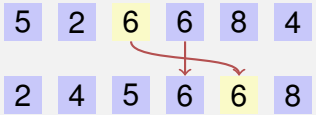


nicht stabil

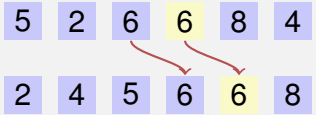
[1] Jeder Sortieralgorithmus kann durch Hinzufügen der alten Position im Array zum Sortierkriterium stabil gemacht werden.

Stabile und in-situ-Sortieralgorithmen

- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht. [1]



nicht stabil

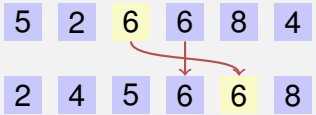


stabil

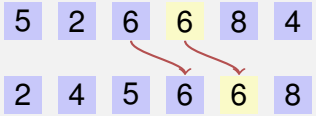
[1] Jeder Sortieralgorithmus kann durch Hinzufügen der alten Position im Array zum Sortierkriterium stabil gemacht werden.

Stabile und in-situ-Sortieralgorithmen

- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht. [1]



nicht stabil



stabil

- In-situ-Algorithmen brauchen nur konstant viel zusätzlichen Speicher. (Mergesort ist nicht in-situ)

[1] Jeder Sortieralgorithmus kann durch Hinzufügen der alten Position im Array zum Sortierkriterium stabil gemacht werden.

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort				

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$		

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl				

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$		

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen				

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$		

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort				

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$		

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$		

Quiz: Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$

3. Programmieraufgabe

Daten Organisieren

- Datenstrukturen, die wir kennen
  - Arrays – Sequenzen fixer Grösse
  - Strings – Buchstabensequenzen
  - Verkettete Listen (bisher: für festen Elementtyp selbstgemacht)
- Allgemeines Container Konzept in Java
  - ArrayList auf generischem Elementtyp – dynamischer als Arrays
  - LinkedList, HashMaps, Sets, Maps, ...

Generische Liste in Java: `java.util.List`

```
import java.util.ArrayList;
import java.util.List;
...

// Liste von Strings
List<String> list = new ArrayList<String>();

list.add("abc");
list.add("xyz");
list.add(1,"123"); // Fuege 123 an Position 1 ein
System.out.println(list.get(0)); // abc

for (String s: list) // For auf Iterator der Liste
       System.out.println(s); // abc 123 xyz
```

Liste von Integers

- Java Generics (z.B. Container) können nur auf Objekten operieren
- Fundamentaltypen `int`, `float` (etc.) sind keine Objekte
- Java bietet Wrapperklassen für Fundamentaltypen an, z.B. typ `Integer`
- Java macht *autoboxing* und packt einen Fundamentaltyp automatisch in eine Wrapperklasse, wo nötig.

Liste von Integers

```
import java.util.ArrayList;
import java.util.List;
...

// Lists of integers
List<Integer> list = new ArrayList<Integer>();

list.add(3);
list.add(4);
list.add(1,5); // Fuege 5 an Position 1 ein
System.out.println(list.get(0)); // 3

for (int i: list){ // For auf Iterator der Liste
        System.out.println(i); // 3 5 4
}
```

Fragen oder Anregungen?