Informatik II

Übung 4

FS 2020

Heutiges Programm

1. Survey Productive Failure Case Study 1
2. Feedback letzte Übung
3. Wiederholung Theorie
   - Analyse von Programmen
   - Analyse von Rekurrenzen
4. Divide & Conquer Sortieralgorithmen

3. 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();
        }
}
```

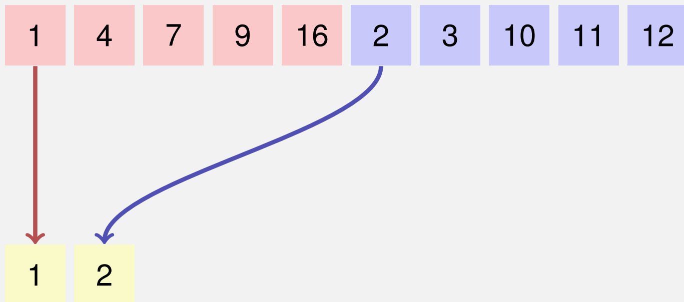
Rekurrenz

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

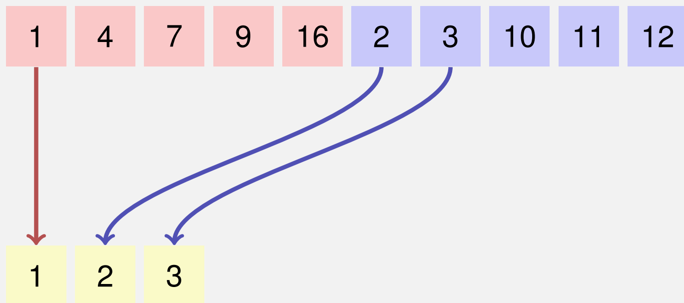
Merge

Merge

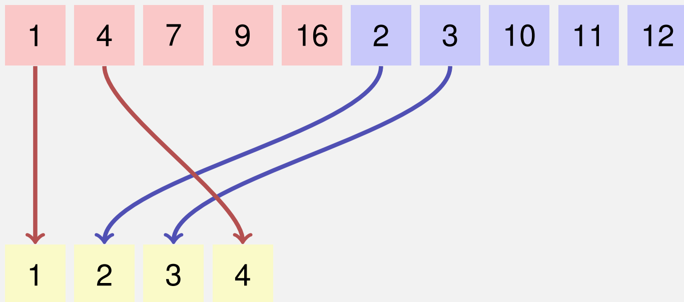
Merge



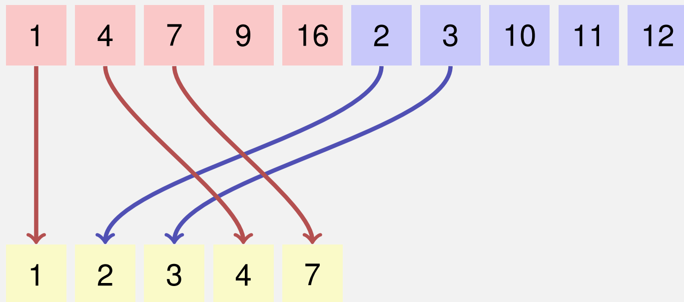
Merge



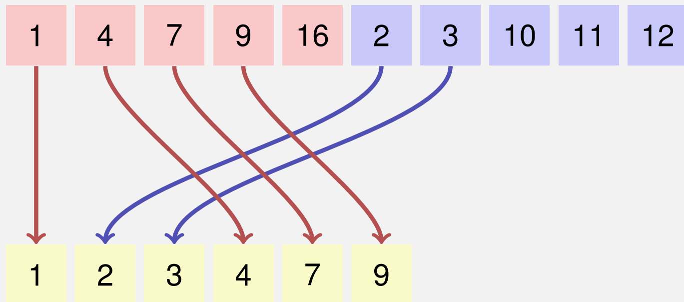
Merge



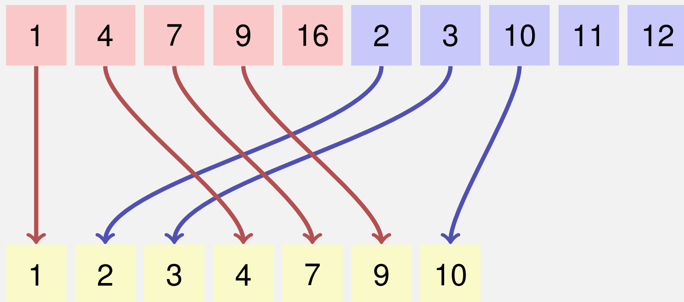
Merge



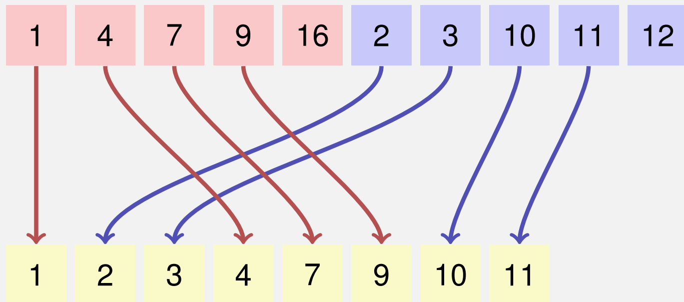
Merge



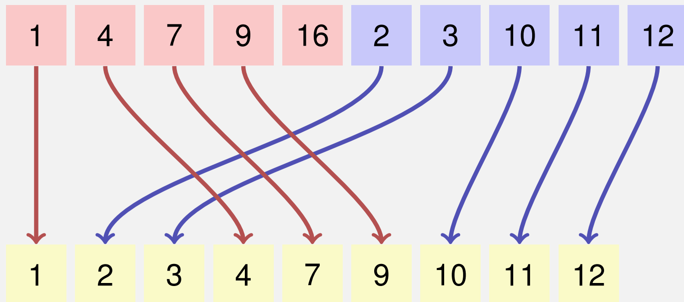
Merge



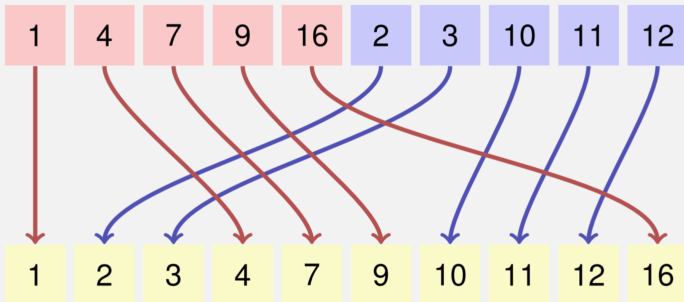
Merge



Merge



Merge



Algorithmus `Merge`(A, l, m, r)

Input: Array A der Länge n, Indizes $1 \leq l \leq m \leq r \leq n$.
$A[l, \dots, m]$, $A[m+1, \dots, r]$ sortiert

Output: $A[l, \dots, r]$ sortiert

1 $B \leftarrow$ new Array$(r - l + 1)$
2 $i \leftarrow l$; $j \leftarrow m + 1$; $k \leftarrow 1$
3 **while** $i \leq m$ and $j \leq r$ **do**
4   **if** $A[i] \leq A[j]$ **then** $B[k] \leftarrow A[i]$; $i \leftarrow i + 1$
5   **else** $B[k] \leftarrow A[j]$; $j \leftarrow j + 1$
6     $k \leftarrow k + 1$;
7 **while** $i \leq m$ **do** $B[k] \leftarrow A[i]$; $i \leftarrow i + 1$; $k \leftarrow k + 1$
8 **while** $j \leq r$ **do** $B[k] \leftarrow A[j]$; $j \leftarrow j + 1$; $k \leftarrow k + 1$
9 **for** $k \leftarrow l$ **to** r **do** $A[k] \leftarrow B[k - l + 1]$

Algorithmus (Rekursives 2-Wege) $\texttt{Mergesort}(A, l, r)$

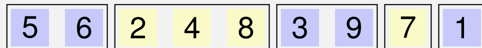
Input: Array A der Länge n. $1 \leq l \leq r \leq n$
Output: $A[l, \dots, r]$ sortiert.

```
if l < r then
    m ← ⌊(l + r)/2⌋           // Mittlere Position
    Mergesort(A, l, m)        // Sortiere vordere Hälfte
    Mergesort(A, m + 1, r)    // Sortiere hintere Hälfte
    Merge(A, l, m, r)         // Verschmelzen der Teilfolgen
```

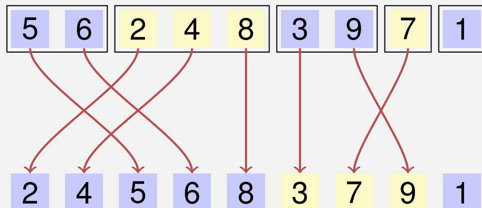
Natürliches 2-Wege Mergesort

5 6 2 4 8 3 9 7 1

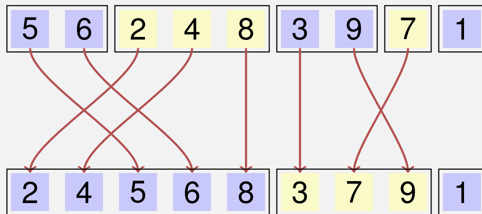
Natürliches 2-Wege Mergesort



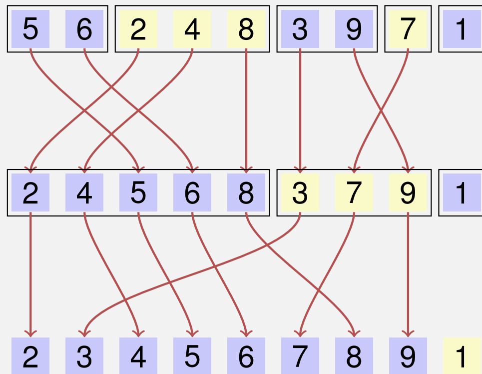
Natürliches 2-Wege Mergesort



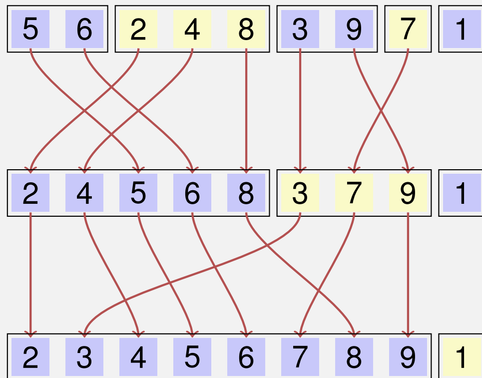
Natürliches 2-Wege Mergesort



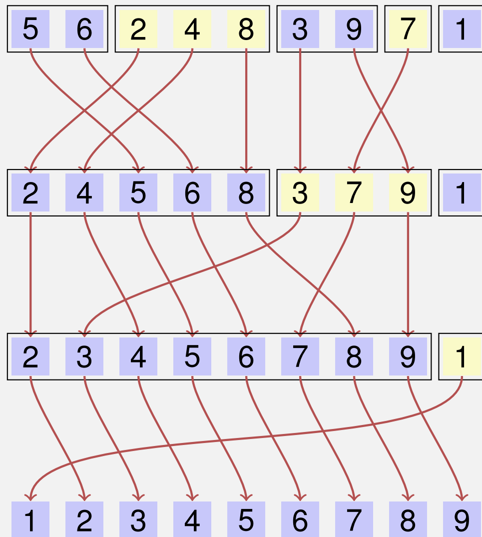
Natürliches 2-Wege Mergesort



Natürliches 2-Wege Mergesort



Natürliches 2-Wege Mergesort



Algorithmus `NaturalMergesort`(A)

Input: Array A der Länge $n > 0$
Output: Array A sortiert

repeat
  $r \leftarrow 0$
  while $r < n$ **do**
    $l \leftarrow r + 1$
    $m \leftarrow l$; **while** $m < n$ **and** $A[m+1] \geq A[m]$ **do** $m \leftarrow m + 1$
    if $m < n$ **then**
      $r \leftarrow m + 1$; **while** $r < n$ **and** $A[r+1] \geq A[r]$ **do** $r \leftarrow r + 1$
      Merge(A, l, m, r);
    else
      $r \leftarrow n$
until $l = 1$

Algorithmus Partition(A, l, r, p)

Input: Array A, welches den Pivot p in $A[l, \ldots, r]$ mindestens einmal enthält.
Output: Array A partitioniert in $A[l, \ldots, r]$ um p. Rückgabe der Position von p.

```
while l ≤ r do
    while A[l] < p do
        l ← l + 1
    while A[r] > p do
        r ← r − 1
    swap(A[l], A[r])
    if A[l] = A[r] then
        l ← l + 1
return l-1
```

Illustration Partition

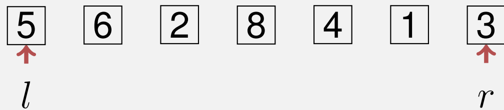


Illustration Partition

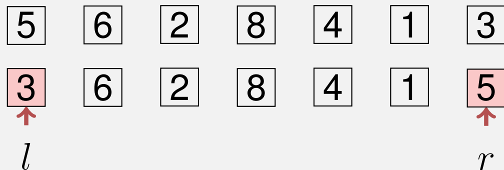


Illustration Partition

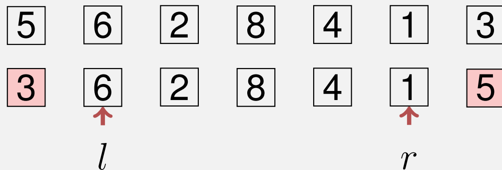


Illustration Partition

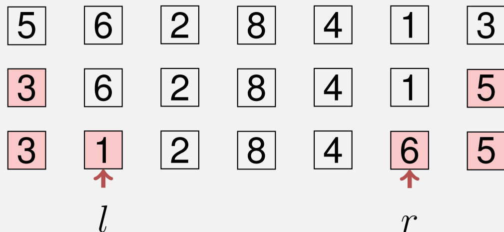


Illustration Partition

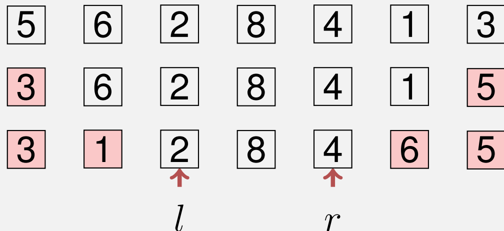


Illustration Partition

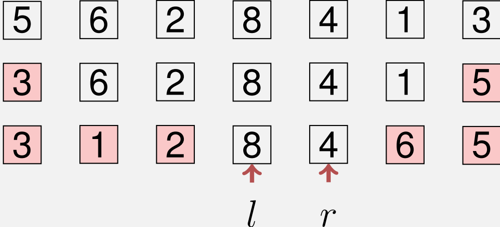


Illustration Partition

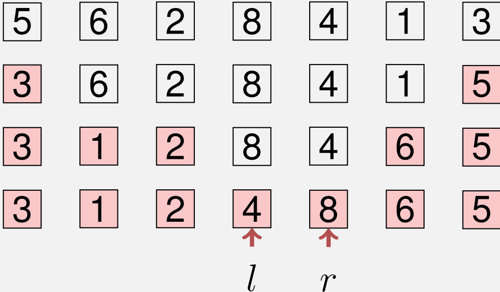
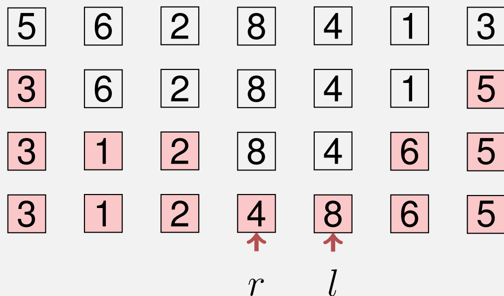


Illustration Partition



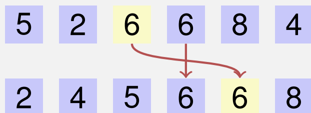
Algorithmus `Quicksort`(A, l, r)

Input: Array A der Länge n. $1 \leq l \leq r \leq n$.
Output: Array A, sortiert in $A[l, \ldots, r]$.

if $l < r$ **then**
  Wähle Pivot $p \in A[l, \ldots, r]$
  $k \leftarrow$ **`Partition`**(A, l, r, p)
  `Quicksort`$(A, l, k-1)$
  `Quicksort`$(A, k+1, r)$

Stabile und in-situ-Sortieralgorithmen

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

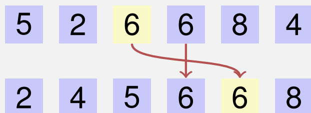


nicht stabil

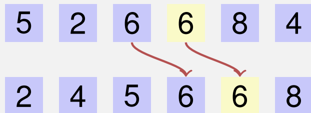
[1] Jeder Sortieralgorithmus kann durch Hinzufügen der alten Position im Array zum Sortierkriterium stabil gemacht werden.

Stabile und in-situ-Sortieralgorithmen

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



nicht stabil

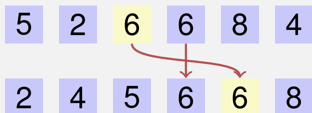


stabil

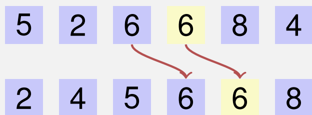
[1] Jeder Sortieralgorithmus kann durch Hinzufügen der alten Position im Array zum Sortierkriterium stabil gemacht werden.

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	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$*
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	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$*
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$

* nicht-trivial (und in der Vorlesung nicht hergeleitet)

Fragen oder Anregungen?