



Übungslektion 7 – Suchen und Sortieren

Informatik II

1. / 2. April 2025

Heutiges Programm

Verbesserter Insertion Sort

Asymptotische Laufzeit Rekursiver Funktionen

Divide & Conquer Sortieralgorithmen

Gruppenübung

Hausaufgaben

1. Verbesserter Insertion Sort

Letztes Mal: Insertion Sort

5 2 8 4 1

Website?

- **Schleifeninvariante:** Vor Iteration i sind Elemente in $li[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $li[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

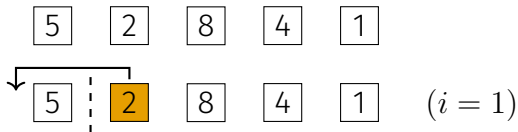
Letztes Mal: Insertion Sort

5 2 8 4 1

5 2 8 4 1 ($i = 1$)

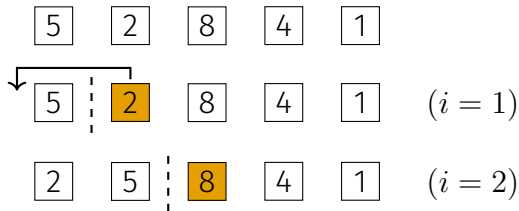
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $li[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $li[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



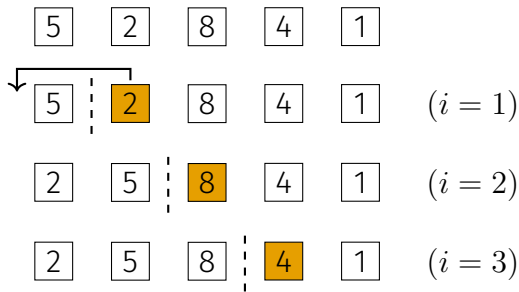
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



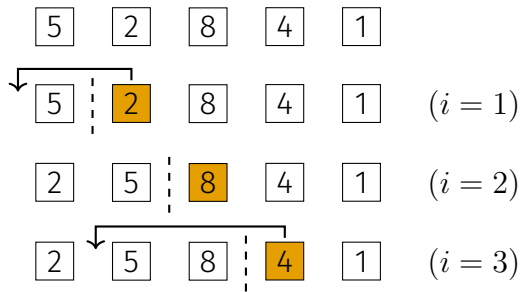
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



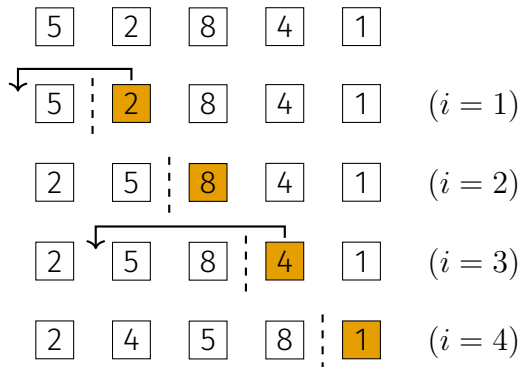
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



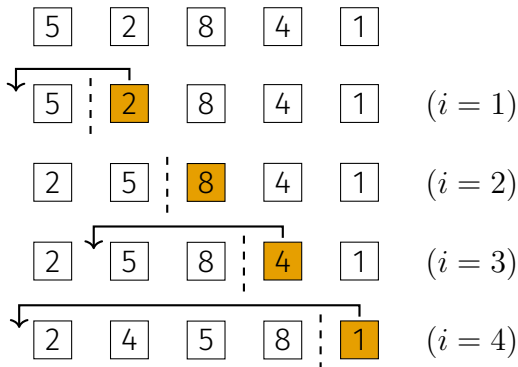
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



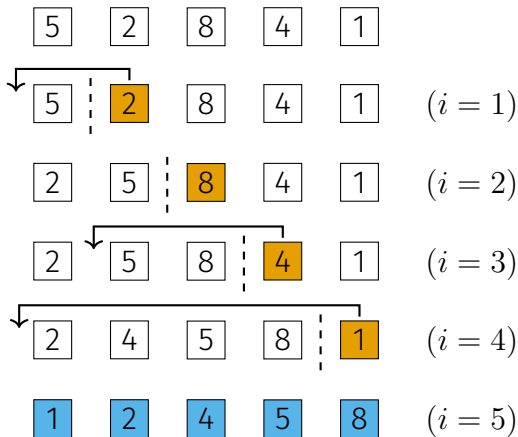
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert.
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort

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

Output: Sortiertes Array A

```
1 for i in range(1, len(A)):  
2     j = i  
3     while j > 0 and A[j-1] > A[j]:  
4         A[j], A[j-1] = A[j-1], A[j]  
5         j = j-1
```

■ Anzahl Vergleiche im schlechtesten Fall?

Kahoot 1

Letztes Mal: Insertion Sort

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

Output: Sortiertes Array A

```
1  for i in range(1,len(A)):  
2      j = i  
3      while j>0 and A[j-1] > A[j]:  
4          A[j], A[j-1] = A[j-1], A[j]  
5          j = j-1
```

■ Anzahl Vergleiche im schlechtesten Fall? $\Theta(n^2)$

Letztes Mal: Insertion Sort

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

Output: Sortiertes Array A

```
1  for i in range(1, len(A)):  
2      j = i  
3      while j > 0 and A[j-1] > A[j]:  
4          A[j], A[j-1] = A[j-1], A[j]  
5          j = j-1
```

- Anzahl Vergleiche im schlechtesten Fall? $\Theta(n^2)$
- Anzahl Vertauschungen im schlechtesten Fall?

Kahoot 2

Letztes Mal: Insertion Sort

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

Output: Sortiertes Array A

```
1  for i in range(1,len(A)):
2      j = i
3      while j>0 and A[j-1] > A[j]:
4          A[j], A[j-1] = A[j-1], A[j]
5          j = j-1
```

- Anzahl Vergleiche im schlechtesten Fall? $\Theta(n^2)$
- Anzahl Vertauschungen im schlechtesten Fall? $\Theta(n^2)$

Letztes Mal: Insertion Sort

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

Output: Sortiertes Array A

```
1 for i in range(1, len(A)):
2     j = i
3     while j > 0 and A[j-1] > A[j]: vgl
4         A[j], A[j-1] = A[j-1], A[j] swap
5         j = j-1 ← rückwärts swap
```



ij

↑
j
Swap where?

- Anzahl Vergleiche im schlechtesten Fall? $\Theta(n^2)$
- Anzahl Vertauschungen im schlechtesten Fall? $\Theta(n^2)$
- **Frage:** Wie können wir die Anzahl Vergleiche für den schlechtesten Fall verbessern?

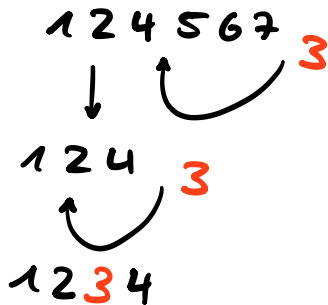
↪ alles rechts davon ist bereits sortiert!
also binary search für einfügen

Letztes Mal: Insertion Sort

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

Output: Sortiertes Array A

```
1 for i in range(1, len(A)):  
2     j = i  
3     while j > 0 and A[j-1] > A[j]:  
4         A[j], A[j-1] = A[j-1], A[j]  
5         j = j-1
```



- Anzahl Vergleiche im schlechtesten Fall? $\Theta(n^2)$
- Anzahl Vertauschungen im schlechtesten Fall? $\Theta(n^2)$
- **Frage:** Wie können wir die Anzahl Vergleiche für den schlechtesten Fall verbessern?

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

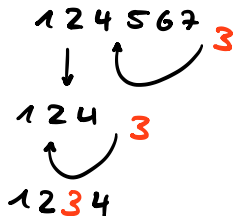
Binärer Insertion Sort

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

Output: Sortiertes Array A

```
1      for i in range(1, len(A)):  
2          x = A[i]  
3          p = BinarySearchIndex(A, 0, i-1, x)  
4          for j in range(i-1, p-1, -1):  
5              A[j+1] = A[j]  
6          A[p] = x
```

→ wie funktioniert
Binary search?



Binärer Insertion Sort

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

Output: Sortiertes Array A

```
1      for i in range(1, len(A)):
2          x = A[i]
3          p = BinarySearchIndex(A, 0, i-1, x)
4          for j in range(i-1, p-1, -1):
5              A[j+1] = A[j]
6          A[p] = x
```

■ Anzahl Vergleiche im schlechtesten Fall?

Kahoot 3

$$\sum_{i=1}^{n-1} a \cdot \log(i) = a \log((n-1)!) \in \Theta(n \cdot \log(n))$$

Binärer Insertion Sort

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

Output: Sortiertes Array A

```
1      for i in range(1, len(A)):
2          x = A[i]
3          p = BinarySearchIndex(A, 0, i-1, x)
4          for j in range(i-1, p-1, -1):
5              A[j+1] = A[j]
6          A[p] = x
```

■ Anzahl Vergleiche im schlechtesten Fall?

$$\sum_{i=1}^{n-1} a \cdot \log i = a \log((n-1)!) \in \Theta(n \log n)$$

Binärer Insertion Sort

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

Output: Sortiertes Array A

```
1      for i in range(1, len(A)):
2          x = A[i]
3          p = BinarySearchIndex(A, 0, i-1, x)
4          for j in range(i-1, p-1, -1):
5              A[j+1] = A[j]
6          A[p] = x
```

- Anzahl Vergleiche im schlechtesten Fall?

$$\sum_{i=1}^{n-1} a \cdot \log i = a \log((n-1)!) \in \Theta(n \log n)$$

- Anzahl Kopiervorgänge im schlechtesten Fall?

immer noch n^2 , suchen zwar schneller aber einfügen n^2

Kahoot 4

$\Theta(n^2)$

Binärer Insertion Sort

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

Output: Sortiertes Array A

```
1      for i in range(1, len(A)):
2          x = A[i]
3          p = BinarySearchIndex(A, 0, i-1, x)
4          for j in range(i-1, p-1, -1):
5              A[j+1] = A[j]
6          A[p] = x
```

■ Anzahl Vergleiche im schlechtesten Fall?

$$\sum_{i=1}^{n-1} a \cdot \log i = a \log((n-1)!) \in \Theta(n \log n)$$

■ Anzahl Kopiervorgänge im schlechtesten Fall? $\sum_{i=1}^{n-1} i \in \Theta(n^2)$

2. Asymptotische Laufzeit Rekursiver Funktionen

Übung 1

Machen lassen

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ -Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 #pre: n is a positive integer
2 def g(n):
3     count = 0
4     while n // (2 ** count) >= 1:
5         f()
6         count += 1
```

iteration
misst iterationen

$$\frac{n}{2^c} \geq 1$$

$$n > 2^c \quad // \log_2$$

$$\log_2(n) > c \\ \rightarrow \Theta(\log n)$$

Antwort 1

```
1 #pre: n is a positive integer
2 def g(n):
3     count = 0
4     while n // (2 ** count) >= 1:
5         f()
6         count += 1
```

- Das Programm ruft die Funktion f , solange $n/2^{count} \geq 1$ ist.
- Diese Ungleichung gilt genau dann, wenn $\log n \geq count$.
- Die Variable $count$ misst die Anzahl der Iterationen der while Schleife, und dabei auch die Anzahl Aufrufe der Funktion f .
- Darum ist diese Anzahl $\log n$.
- Das bedeutet, dass diese Anzahl $\Theta(\log n)$ ist.

Übung 2

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ -Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4         f()
5         g(n // 2)
```

Antwort 2

Zeigen

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4         f() → 1
5         g(n // 2) → T(n/2)
```

Teleskopieren:

$$T(n) = 1 + T(n/2)$$

↑ ?

$$T(n/2) = 1 + T(n/4)$$

↑ ?

$$T(n/4) = 1 + T(n/8)$$

⋮

$$T(n/2^k) = 1 + T\left(\frac{n}{2^{k+1}}\right)$$

Sei $T(n)$ die Anzahl der Aufrufe der Funktion f .

fortsetzen bis $\frac{n}{2^k} \geq 1$

$$n \geq 2^k$$

$$k \leq \underline{\log(n)} \quad \Theta(\log n)$$

Antwort 2

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4         f()
5         g(n // 2)
```

Sei $T(n)$ die Anzahl der Aufrufe der Funktion f . Wenn $n \geq 1$, es gilt:
 $T(n) = 1 + T(n/2) = 1 + 1 + T(n/2^2) = k + T(n/2^k)$.

- f ist aufgerufen, solange $n/2^k \geq 1$ ist.
- Diese Ungleichung gilt genau dann, wenn $\log n \geq k$.
- Das bedeutet, dass die Anzahl der Aufrufe der Funktion f ist $\log n$ liegt, und dabei diese Anzahl $\Theta(\log n)$ ist.

Besser: Teleskopieren mit Master Theorem

→ jlotzer

Step 1: Rewrite the recursive Function $T(n)$ into following mathematical form:

$$T(n) = \underbrace{a \cdot T\left(\frac{n}{b}\right)}_{\text{recursive call(s)}} + \underbrace{\theta(n^k \cdot \log^p(n))}_{\text{additional runtime in each call of } T(n), \text{ (example: for-loop)}}$$

Tipp: determine $\theta()$ directly and do Koeffizientenvergleich with the Formula above

Step 2: Runtime is one of these cases:

1. if $a > b^k$, then $T(n) = \theta(n^{\log_b(a)})$
2. if $a = b^k$, and
 - a) if $p > -1$, then $T(n) = \theta(n^{\log_b(a)} \cdot \log^{p+1}(n))$
 - b) if $p = -1$, then $T(n) = \theta(n^{\log_b(a)} \cdot \log(\log(n)))$
 - c) if $p < -1$, then $T(n) = \theta(n^{\log_b(a)})$
3. if $a < b^k$, and
 - a) if $p \geq 0$, then $T(n) = \theta(n^k \cdot \log^p(n))$
 - b) if $p < 0$, then $T(n) = \theta(n^k)$

Antwort 2

Mach

```

1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:      icall
4         f()          a=1
5         g(n // 2)    b=2

```

k: additional runtime: 0
k=0
p=0

Sei $T(n)$ die Anzahl der Aufrufe der Funktion f .

$$1) T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta\left[n^k \cdot \log^p(n)\right]$$

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \Theta\left(n^0 \cdot \log^0(n)\right)$$

$$2) a \stackrel{?}{=} b^k$$

$$1 = 2^0 \rightarrow \text{Case 2: } p \stackrel{?}{=} -1$$

$$0 > -1 : \text{Case 2a}$$

$$\Theta\left(n^{\overbrace{\log_2 1}^0} \cdot \log^{0+1}(n)\right)$$

$$= \Theta\left(n^0 \cdot \log(n)\right) = \Theta(\log n)$$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \theta(n^k \cdot \log^p(n))$$

1. if $a > b^k$, then $T(n) = \theta(n^{\log_b(a)})$
2. if $a = b^k$, and
 - a) if $p > -1$, then $T(n) = \theta(n^{\log_b(a)} \cdot \log^{p+1}(n))$
 - b) if $p = -1$, then $T(n) = \theta(n^{\log_b(a)} \cdot \log(\log(n)))$
 - c) if $p < -1$, then $T(n) = \theta(n^{\log_b(a)})$
3. if $a < b^k$, and
 - a) if $p \geq 0$, then $T(n) = \theta(n^k \cdot \log^p(n))$
 - b) if $p < 0$, then $T(n) = \theta(n^k)$

Übung 3

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ -Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4          for i in range(n): | n
5             f()
6             g(n // 2) | .... etwas/2
```

$n \cdot \log n$?

Nicht ganz! falsche
Intuition

Antwort 3 Teleskopieren normal...

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1: |
4         n | for i in range(n):
5             + f()
6         T(n/2) g(n // 2)
```

Sei $T(n)$ die Anzahl der Aufrufe der Funktion f . Wenn $n \geq 1$, es gilt:

Stopp bei $T(0) = 0$

$$T(n) = n + T(n/2)$$

$$T(n/2) = (n/2) + T(n/4)$$

$$T(n/4) = (n/4) + T(n/8)$$

$$T\left(\frac{n}{2^k}\right) = \left(\frac{n}{2^k}\right) + T\left(\frac{n}{2^{k+1}}\right)$$

Fortsetzen bis $\frac{n}{2^k} \geq 1$

riesen rechnung

⋮

Antwort 3

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4         for i in range(n): | k →
5             f()              p = 0
6         g(n // 2) → 1 call a = 1
                        ↪ b = 2
```

f() wird einmal pro T(n) aufgerufen k=1

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta\left[n^k \cdot \log^p(n)\right]$$

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \Theta(n^1 \cdot \log^0)$$

$$\begin{aligned} a &= 1 \\ b &= 2 \\ k &= 1 \\ p &= 0 \end{aligned}$$

$$\left. \begin{aligned} a &= 1 \\ b &= 2 \\ k &= 1 \\ p &= 0 \end{aligned} \right\} \text{ZTF: } 1 < 2^1 \rightarrow \text{fall 3)} \\ \hookrightarrow 3a) : p=0 \\ \Theta(n^k \cdot \log^p) \\ = \underline{\underline{\Theta(n^1)}}$$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta(n^k \cdot \log^p(n))$$

1. if $a > b^k$, then $T(n) = \Theta(n^{\log_b(a)})$
2. if $a = b^k$, and
 - a) if $p > -1$, then $T(n) = \Theta(n^{\log_b(a)} \cdot \log^{p+1}(n))$
 - b) if $p = -1$, then $T(n) = \Theta(n^{\log_b(a)} \cdot \log(\log(n)))$
 - c) if $p < -1$, then $T(n) = \Theta(n^{\log_b(a)})$
3. if $a < b^k$, and
 - a) if $p \geq 0$, then $T(n) = \Theta(n^k \cdot \log^p(n))$
 - b) if $p < 0$, then $T(n) = \Theta(n^k)$

Antwort 3

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4         for i in range(n):
5             f()
6         g(n // 2)
```

Sei $T(n)$ die Anzahl der Aufrufe der Funktion f . Wenn $n \geq 1$, es gilt:

$$\begin{aligned} T(n) &= n + T(n/2) \\ &= n + n/2 + T(n/2^2) \\ &= n + n/2 + \dots + n/2^k + T(n/2^{k+1}) \\ &= n \sum_{j \leq k} 1/2^j + T(n/2^{k+1}) \\ &\leq 2n + T(n/2^{k+1}). \end{aligned}$$

- Die letzte Ungleichung gilt, weil $\forall k : \sum_{j \leq k} 1/2^j \leq 2$.
- Es gibt k , wofür $\lfloor n/2^{k+1} \rfloor = 0$, dann auch $T(\lfloor n/2^{k+1} \rfloor) = T(0) = 0$.
- Das bedeutet, dass $T(n) \leq 2n \in O(n)$.
- Analog dazu kann man zeigen, dass $T(n) \in \Omega(n)$, weil $g(n)$ ruft f zumindest n -mal.
- Darum, $T(n) \in \Theta(n)$.

Übung 4

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ -Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4         for i in range(n):
5             f()
6         g(n // 2)
7         g(n // 2)
```

Antwort 4

```
1 # pre: n is a positive integer
2 def g(n):
3     if n >= 1:
4         for i in range(n):  $k=1$ 
5             f()
6          $g(n // 2)$   $\left. \begin{array}{l} 2=b \\ 2=a \end{array} \right\}$ 
7          $g(n // 2)$ 
```

$$1) T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta[n^k \cdot \log^p(n)]$$
$$T(n) = 2 T\left(\frac{n}{2}\right) + \Theta[n^1 \log^0(n)]$$

$$2) a \stackrel{?}{=} b^k$$

$$2 = 2^1 \rightarrow \text{case } 2) \quad p = 0 > -1$$

$$2a): \Theta(n^{\log_2(2)} \cdot \log^{0+1}(n))$$
$$= \Theta(n \log(n))$$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \theta(n^k \cdot \log^p(n))$$

1. if $a > b^k$, then $T(n) = \theta(n^{\log_b(a)})$
2. if $a = b^k$, and
 - a) if $p > -1$, then $T(n) = \theta(n^{\log_b(a)} \cdot \log^{p+1}(n))$
 - b) if $p = -1$, then $T(n) = \theta(n^{\log_b(a)} \cdot \log(\log(n)))$
 - c) if $p < -1$, then $T(n) = \theta(n^{\log_b(a)})$
3. if $a < b^k$, and
 - a) if $p \geq 0$, then $T(n) = \theta(n^k \cdot \log^p(n))$
 - b) if $p < 0$, then $T(n) = \theta(n^k)$

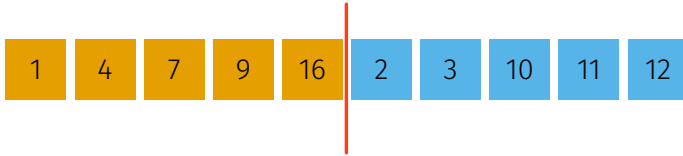
Antwort 4

<pre>1 # pre: n is a positive integer 2 def g(n): 3 if n >= 1: 4 for i in range(n): 5 f() 6 g(n // 2) 7 g(n // 2)</pre>	Sei $T(n)$ die Anzahl der Aufrufe der Funktion f. Wenn $n \geq 1$, es gilt: $\begin{aligned} T(n) &= n + 2 T(n/2) \\ &= n + 2(n/2) + 2^2 T(n/2^2) \\ &= n + 2(n/2) + \dots + 2^k(n/2^k) \\ &\quad + 2^{k+1}T(n/2^{k+1}) \\ &= n(k+1) + 2^{k+1} T(n/2^{k+1}). \end{aligned}$
--	---

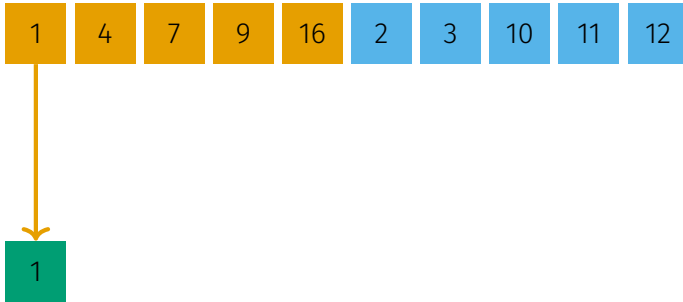
- f ist aufgerufen, solange $n/2^k \geq 1$ ist.
- Diese Ungleichung gilt genau dann, wenn $\log n \geq k$.
- Das bedeutet, dass $k = \log n$. Darum ist
$$T(n) = n(\log n + 1) + 2n T(1/2) = n(\log n + 1) + 2n T(0) = n(\log n + 1).$$
- Darum ist die Anzahl Aufrufe der Funktion f zwischen $\Theta(n \log n)$.

3. Divide & Conquer Sortieralgorithmen

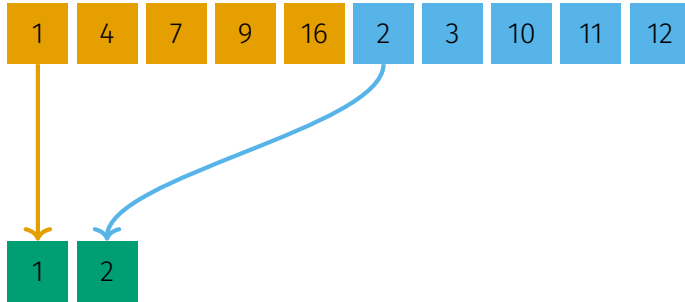
Merge



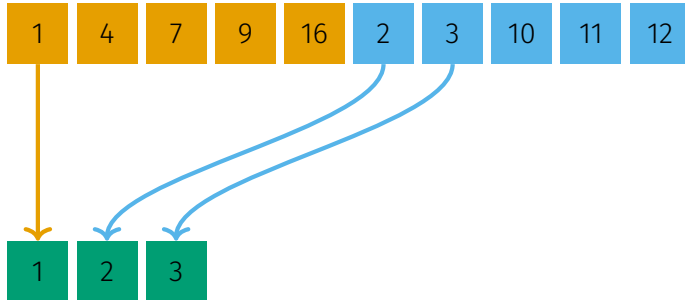
Merge



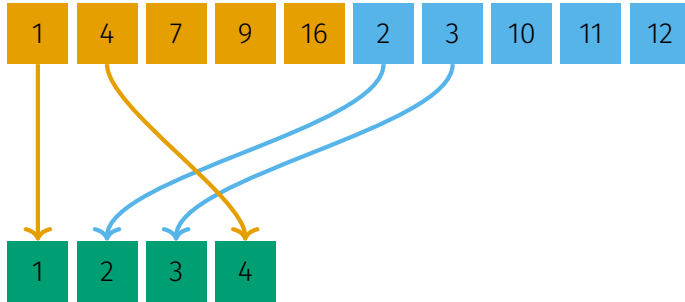
Merge



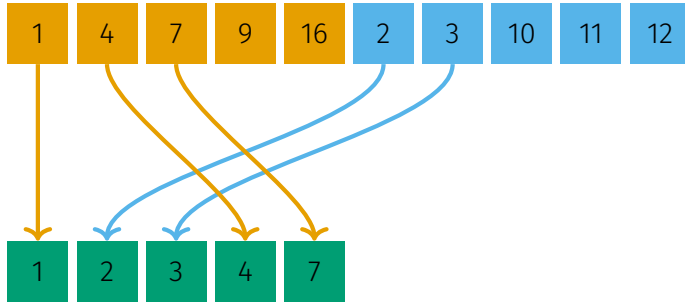
Merge



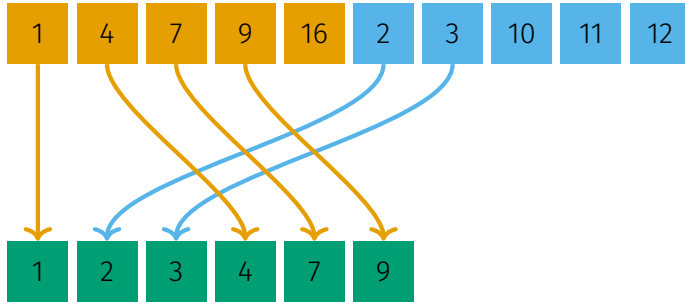
Merge



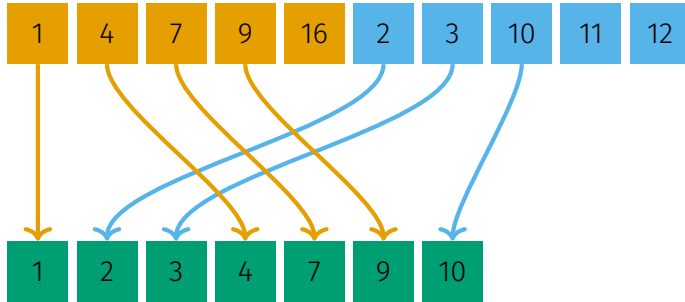
Merge



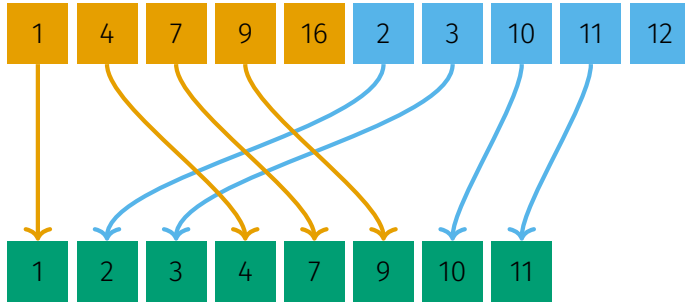
Merge



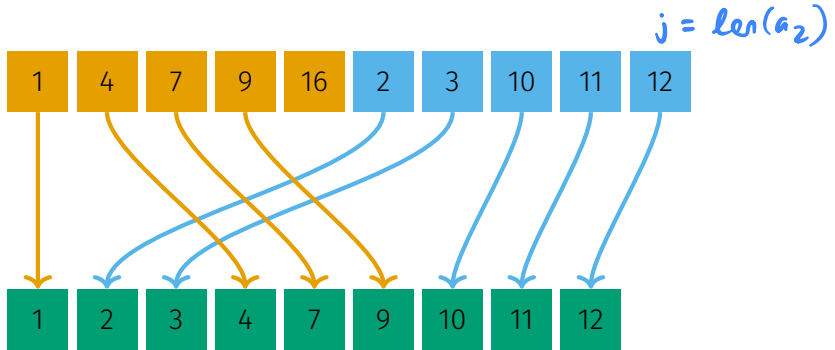
Merge



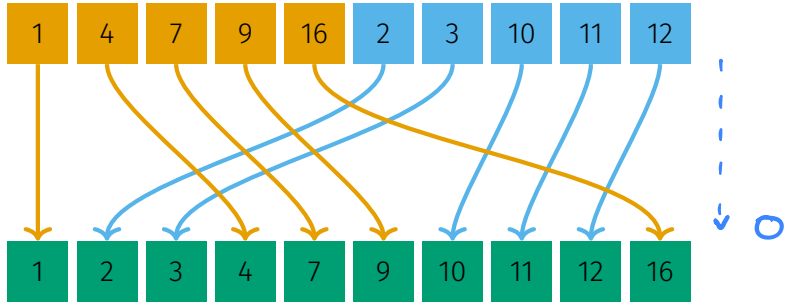
Merge



Merge

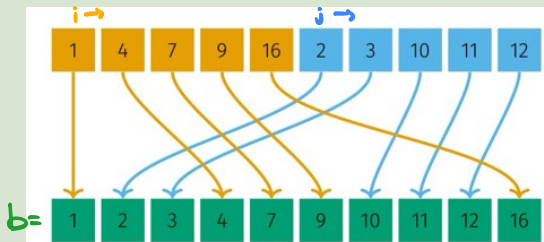


Merge



Algorithmus merge

```
1 def merge(a1, a2):
2     b, i, j = [], 0, 0
3     while i < len(a1) and j < len(a2):
4         if a1[i] < a2[j]:
5             b.append(a1[i])
6             i += 1
7         else:
8             b.append(a2[j])
9             j += 1
10    b += a1[i:] | rest
11    b += a2[j:]
12    return b
```



Algorithmus merge_sort

```
1 def merge_sort(a):  
2     if len(a) <= 1:  
3         return a  
4     else:  
5         sorted_a1 = merge_sort(a[:len(a) // 2])  
6         sorted_a2 = merge_sort(a[len(a) // 2:])  
7         return merge(sorted_a1, sorted_a2)
```

→ [hackerearth.com](https://www.hackerearth.com)

Natürlicher 2-Wege Mergesort

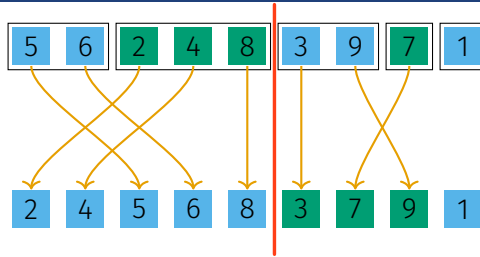
Suche : sortierte
Teilfolgen



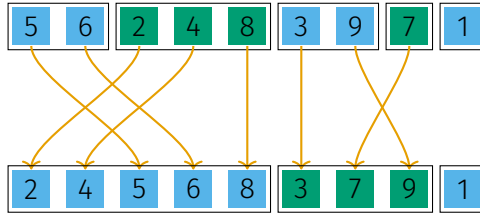
Natürlicher 2-Wege Mergesort



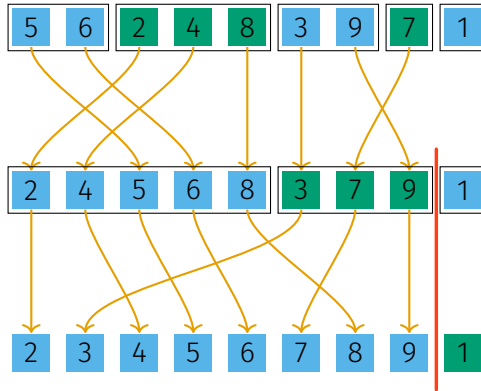
Natürlicher 2-Wege Mergesort



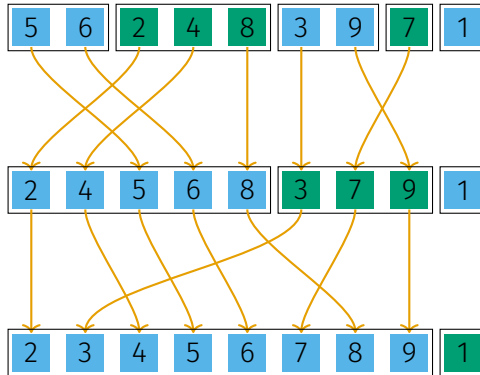
Natürlicher 2-Wege Mergesort



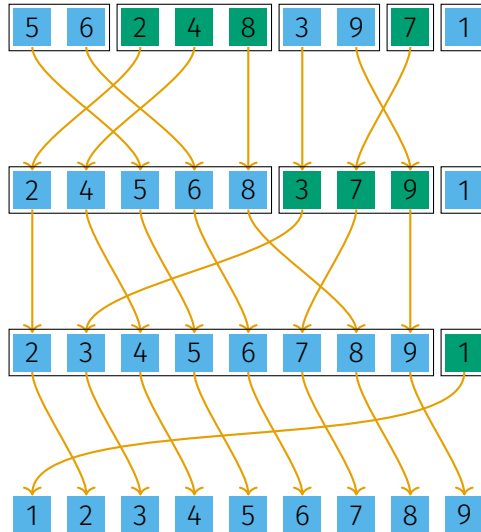
Natürlicher 2-Wege Mergesort



Natürlicher 2-Wege Mergesort



Natürlicher 2-Wege Mergesort



Algorithmus NaturalMergesort(A)

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

```
1  def NaturalMergesort(A):
2      l = -1
3      while l != 0:
4          r = 0
5          while r < len(A)-1:
6              l = r
7              m = l
8              while m < len(A)-1 and A[m+1] >= A[m]:
9                  m = m+1
10             if m < len(A)-1:
11                 r = m+1
12                 while r < len(A)-1 and A[r+1] >= A[r]:
13                     r = r+1
14                 Merge(A,l,m,r)
15             else:
16                 r = len(A)-1
17             r+=1
```

4. Gruppenübung

Gruppenübung: Timsort

<https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Timsort, eine Kombination aus Insertion Sort und Merge Sort, ist der Standard-Sortieralgorithmus in Python. Mehr dazu in der Aufgabenbeschreibung auf Code Expert.

→ extra 30 xp !!

5. Hausaufgaben

Übung 6: Search and Sort

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/exercises>

Übung 6: Search and Sort

- Eierwerfen
- Vergleich von Sortieralgorithmen
- Verbesserter Insertion Sort → check ZF : Binary Search?
- Invarianten der Suchalgorithmen

Abgabedatum: Montag 07.04.2025, 20:00 CET

KEINE HARDCODIERUNG

Fragen oder Anregungen?