2.3 Sortieren

Quick-Sort

- Idee
  - Divide & Conquer
  - Wähle ein Element R aus
  - Teile Folge in zwei Sub-Folgen $R_L \leqslant R$ und $R_R \geqslant R$
  - Sortiere R_L und R_R durch rekursiven Aufruf
  - Setze Teilfolgen zusammen: $R_L \oplus R \oplus R_R$

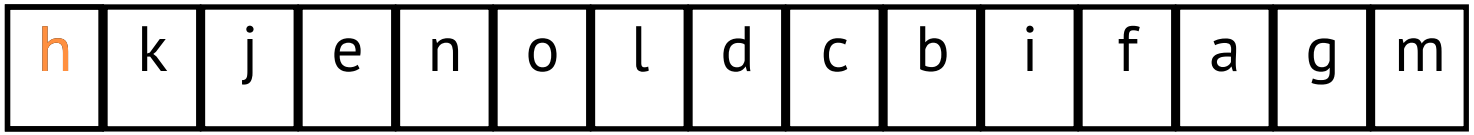
Quick-Sort

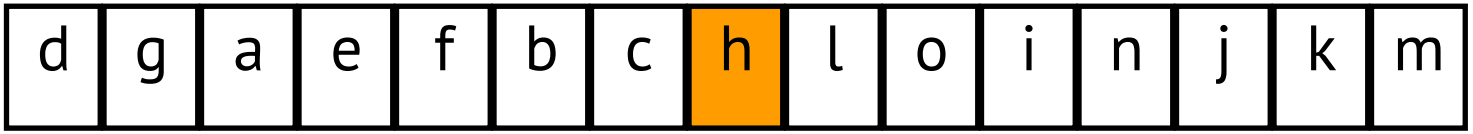
h	k	j	e	n	o	l	d	c	b	i	f	a	g	m

Quick-Sort

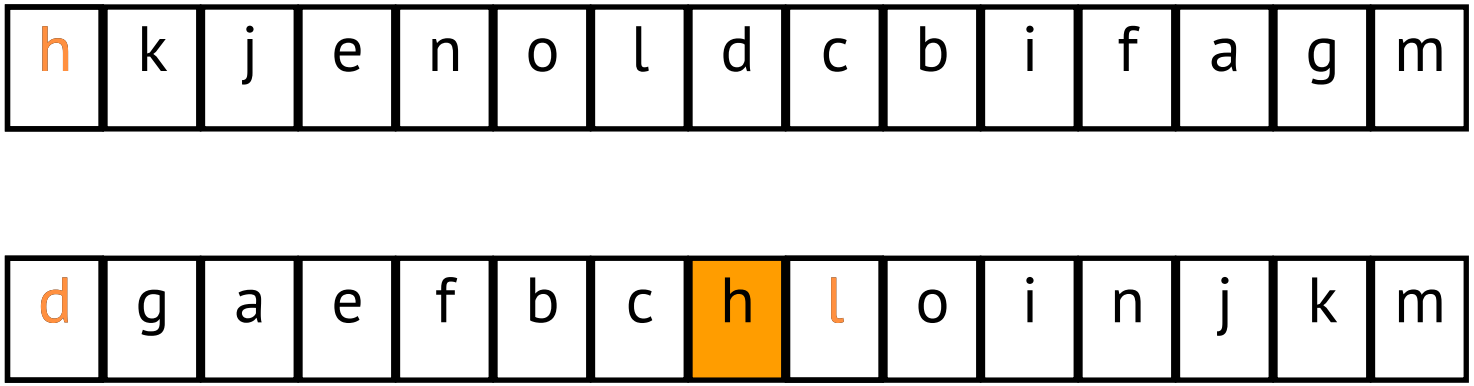
h	k	j	e	n	o	l	d	c	b	i	f	a	g	m

Quick-Sort

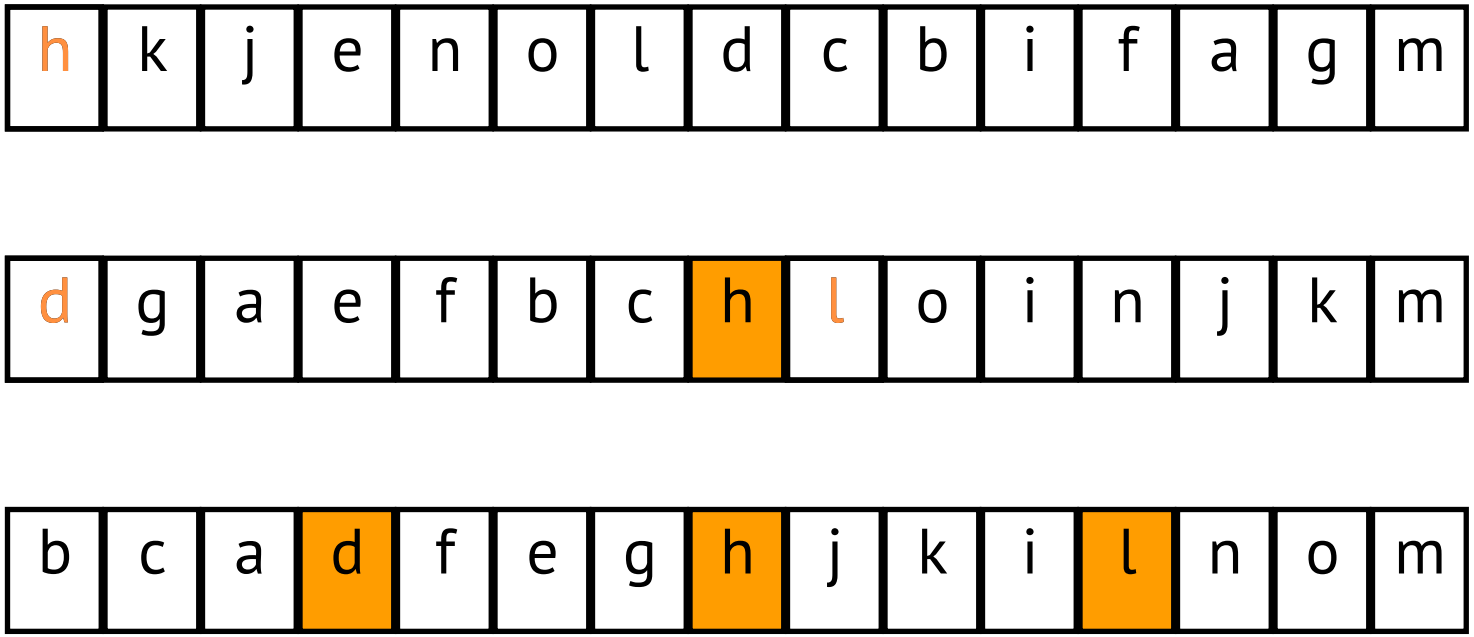




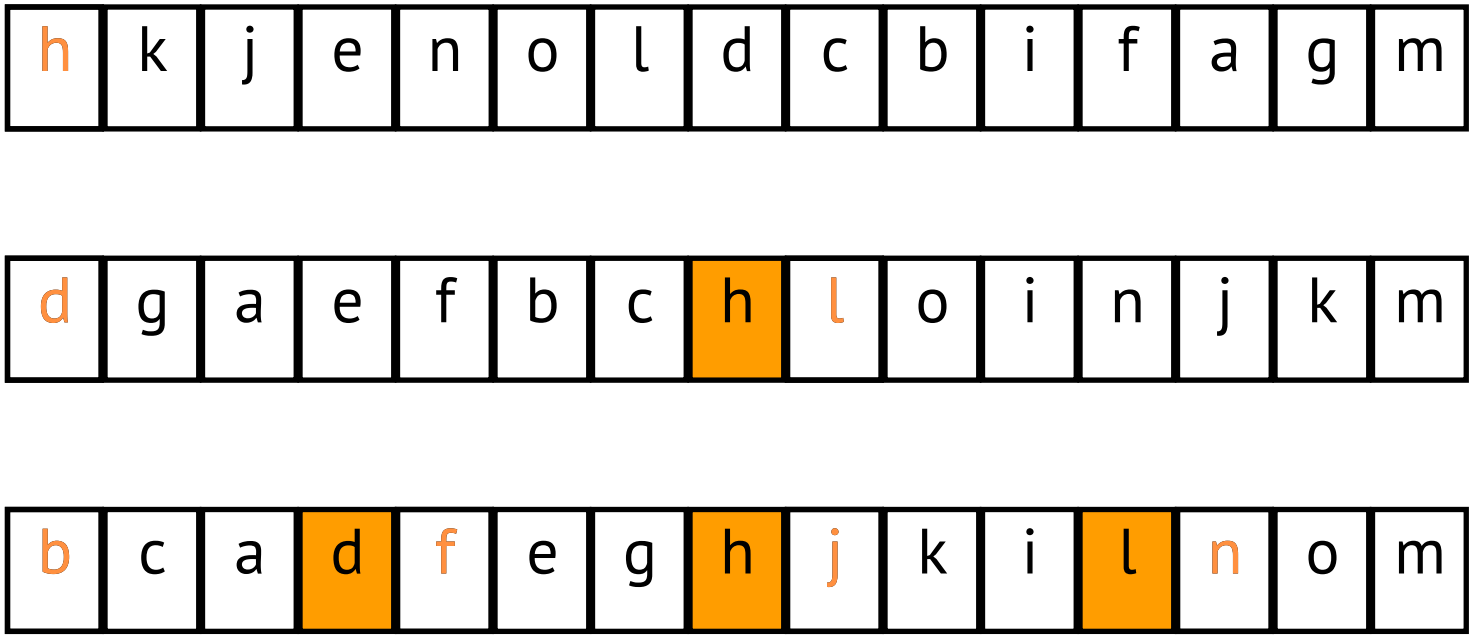
Quick-Sort



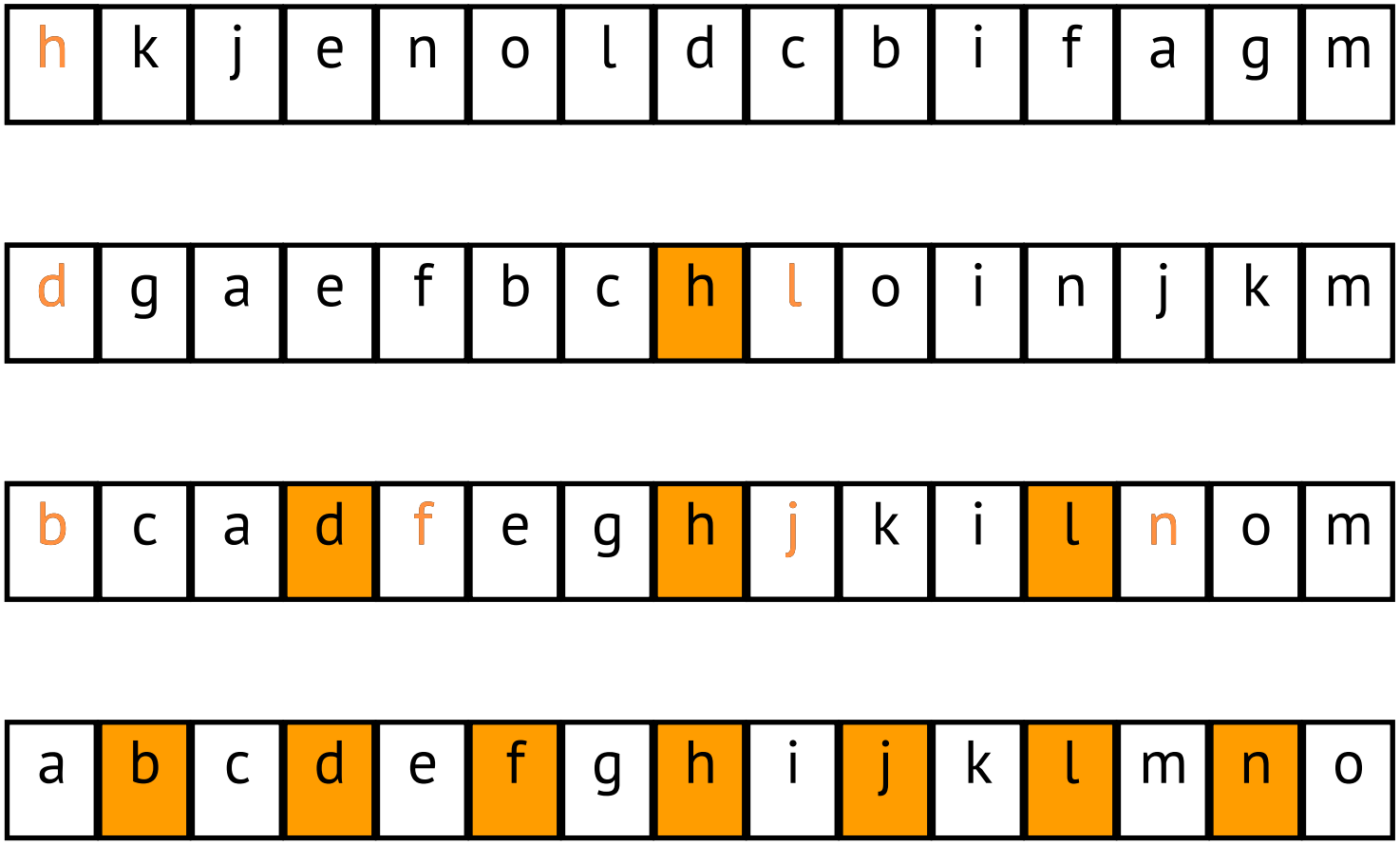
Quick-Sort



Quick-Sort



Quick-Sort


h k j e n o l d c b i f a g m
d g a e f b c h l o i n j k m
b c a d f e g h j k i l n o m
a b c d e f g h i j k l m n o


A
C
G

Quick-Sort: Algorithmus

- QuickSort(A[l..r])
  if l < r then
    m ← Partition(A[l..r])
    QuickSort(A[l..m−1])
    QuickSort(A[m+1..r])

- Aufruf mit: QuickSort(A[1..n])

Quick-Sort: Algorithmus

- Partition(A[l..r])

```
i ← l−1; j ← r
while i < j do
    i ← i+1
    while A[ i ] < A[ r ] do
        i ← i+1
    j ← j−1
    while A[ j ] > A[ r ] do
        j ← j−1
    swap(A[ i ], A[ j ])
swap(A[ i ], A[ j ])
swap(A[ i ], A[ r ])
return i
```

Quick-Sort: Algorithmus

- Achtung: Verwende Sentinel, um korrekte Terminierung der inneren Schleife zu garantieren.
- Aufruf mit

  A[0] ← – ∞

  QuickSort(A[1..n])
- while A[i] < A[r] do

  i←i+1
- while A[j] > A[r] do

  j←j–1

Bricht spätestens für i = r ab

Bricht spätestens für j = l – 1 ab

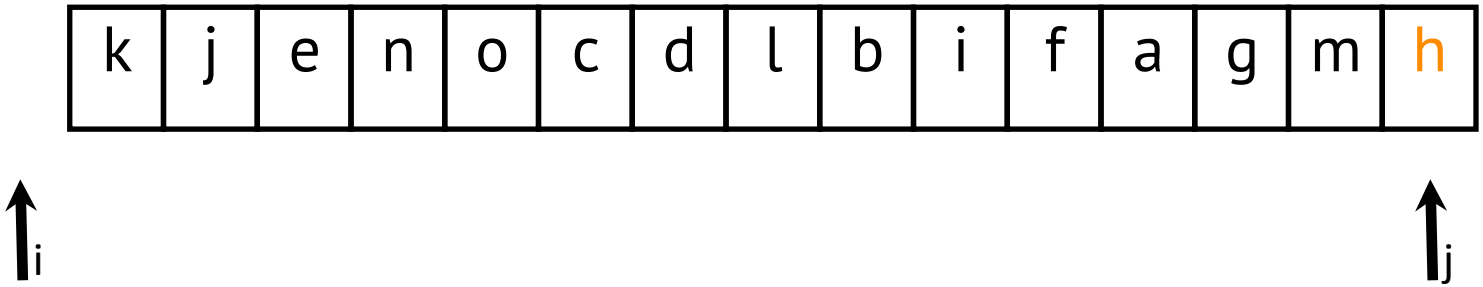
Quick-Sort: Beispiel

k	j	e	n	o	c	d	l	b	i	f	a	g	m	h

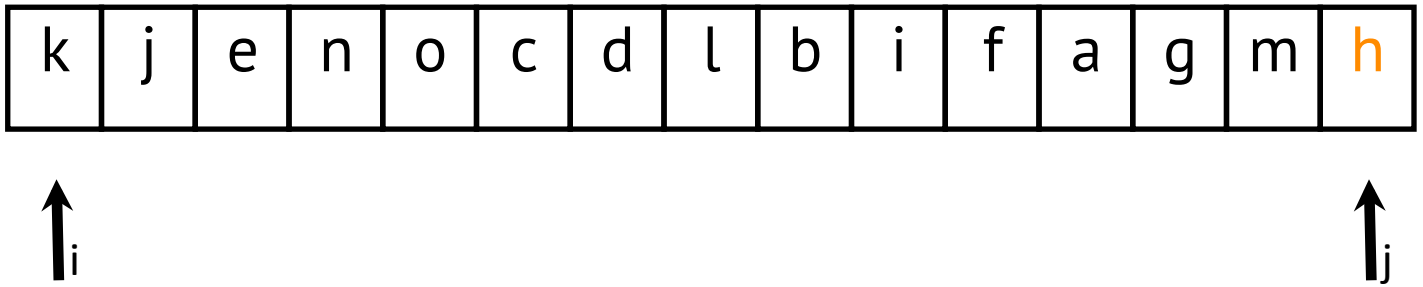
Quick-Sort: Beispiel

k	j	e	n	o	c	d	l	b	i	f	a	g	m	h

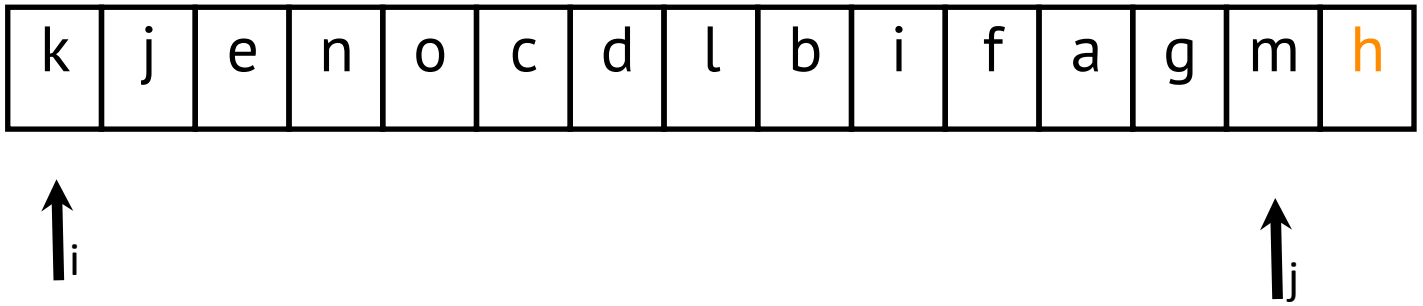
Quick-Sort: Beispiel



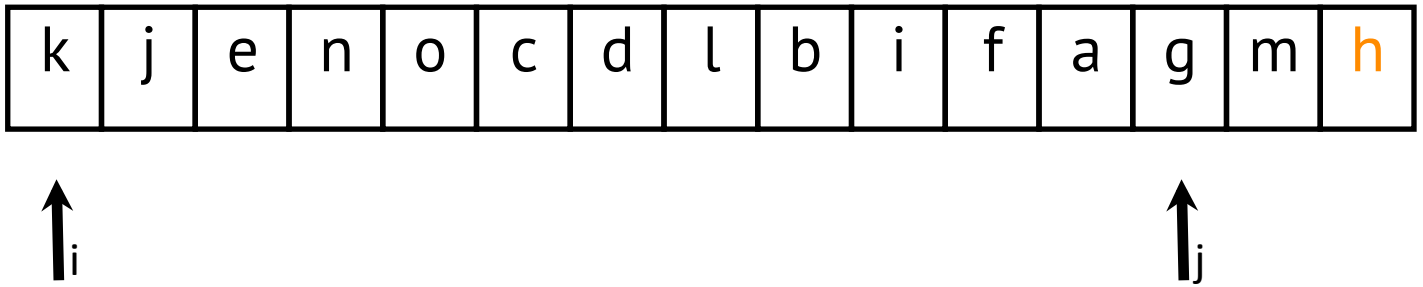
Quick-Sort: Beispiel



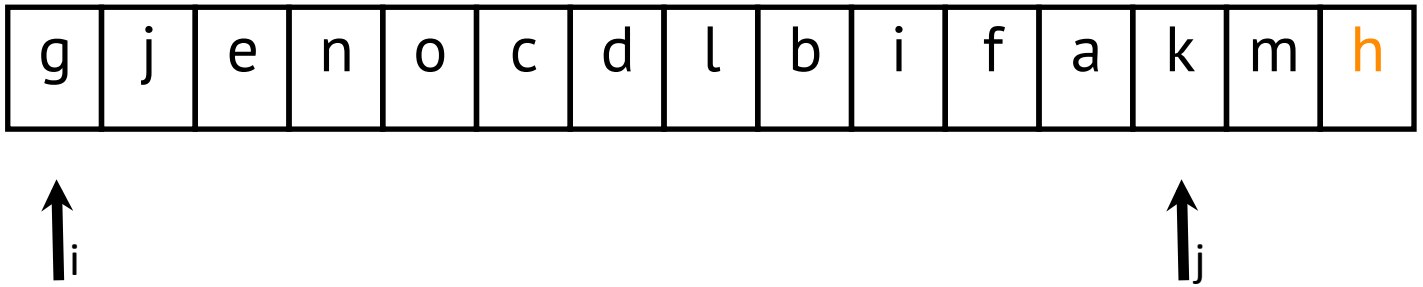
Quick-Sort: Beispiel



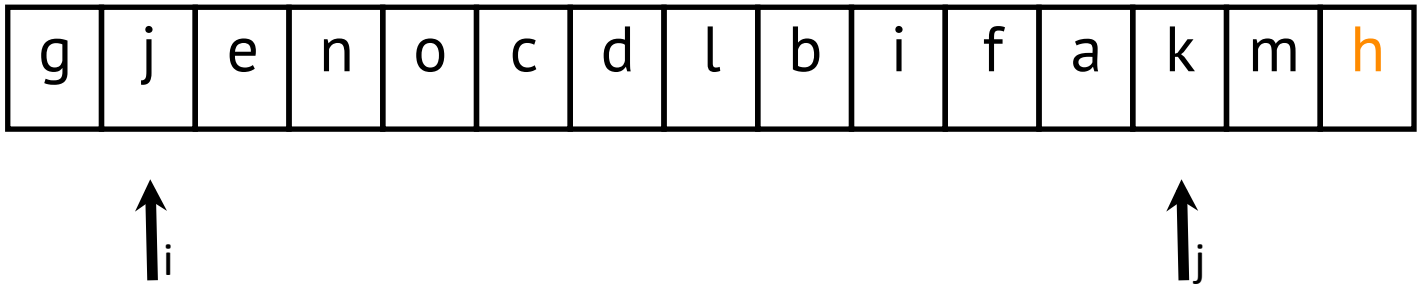
Quick-Sort: Beispiel



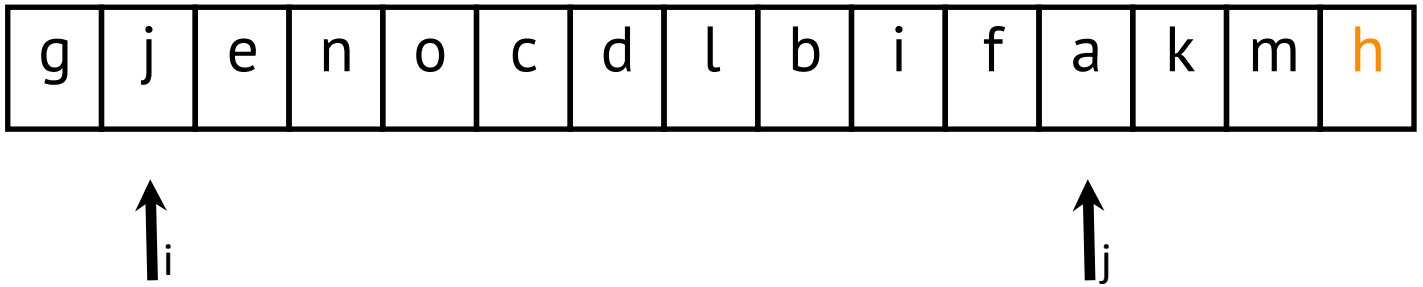
Quick-Sort: Beispiel



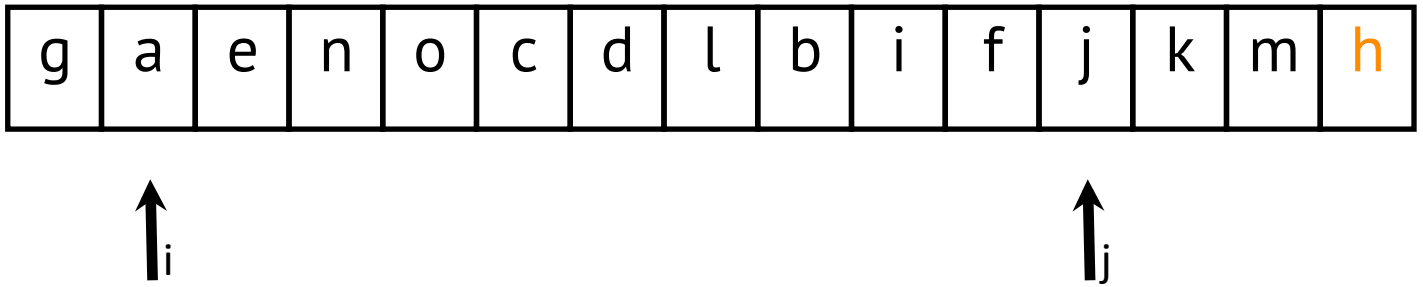
Quick-Sort: Beispiel



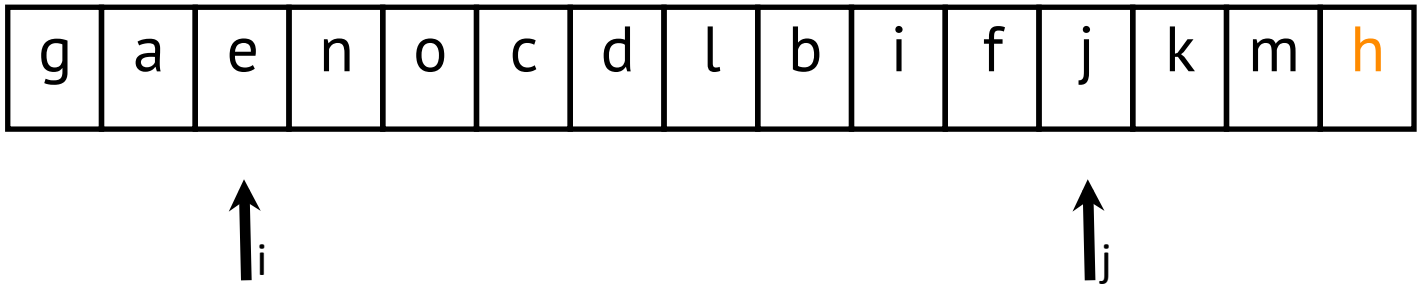
Quick-Sort: Beispiel



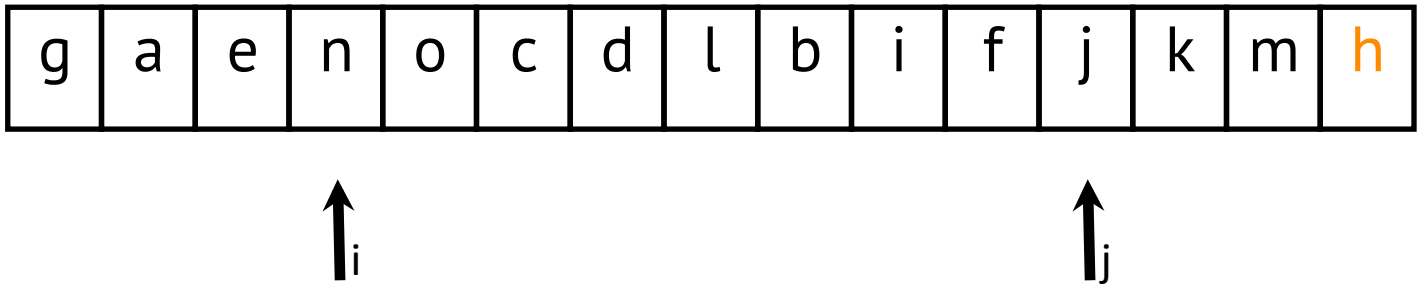
Quick-Sort: Beispiel



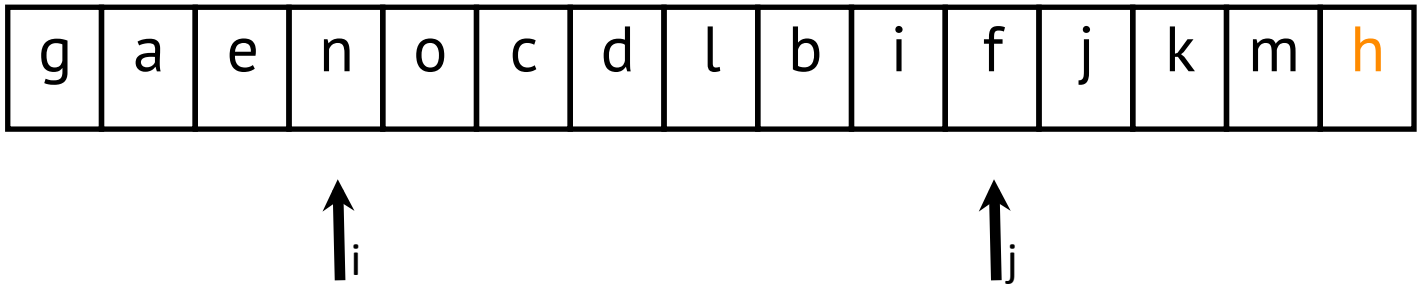
Quick-Sort: Beispiel



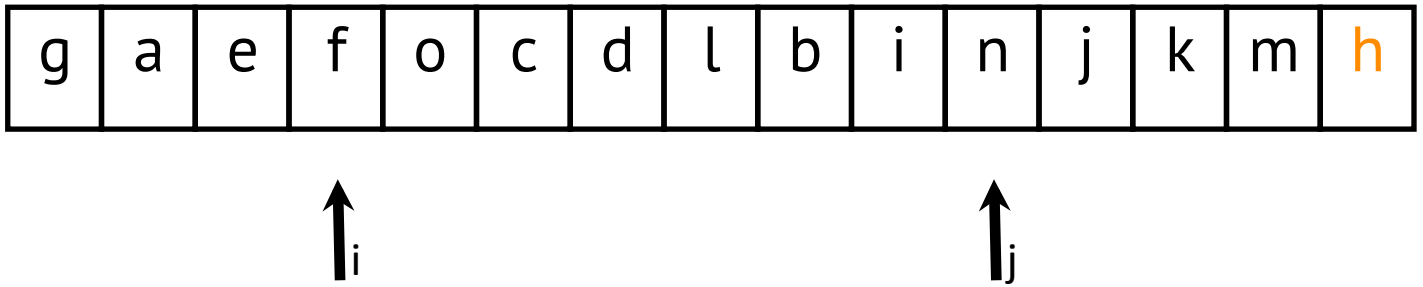
Quick-Sort: Beispiel



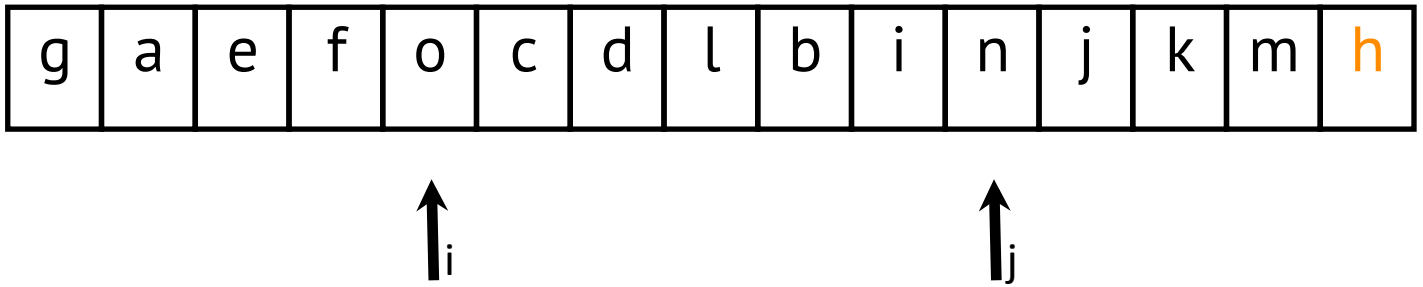
Quick-Sort: Beispiel



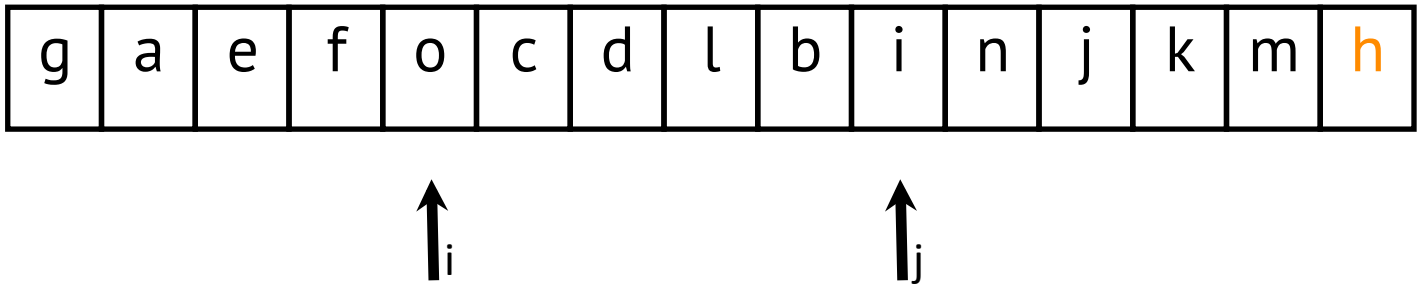
Quick-Sort: Beispiel



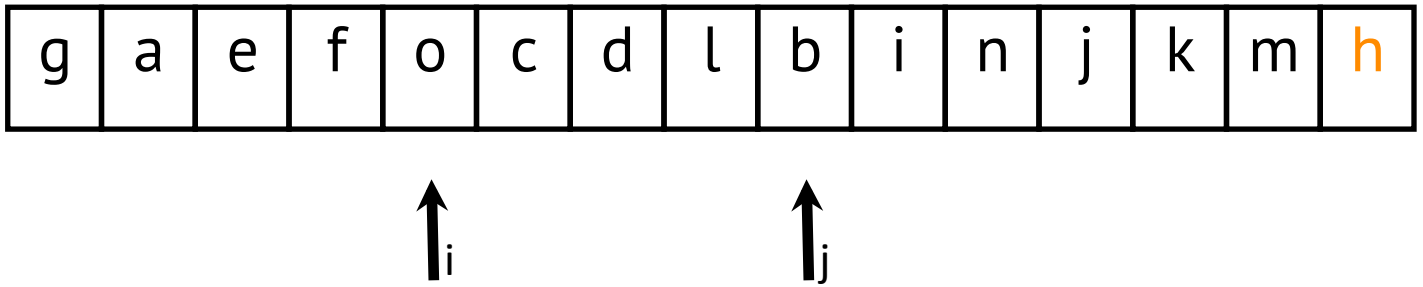
Quick-Sort: Beispiel



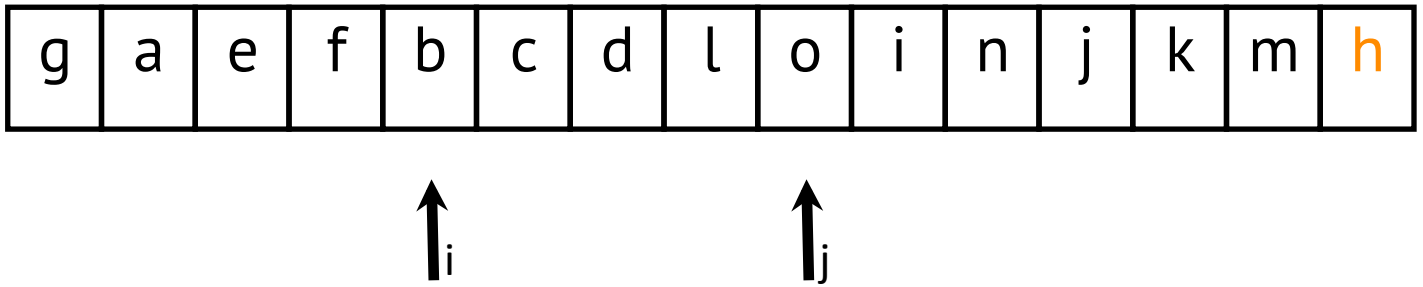
Quick-Sort: Beispiel



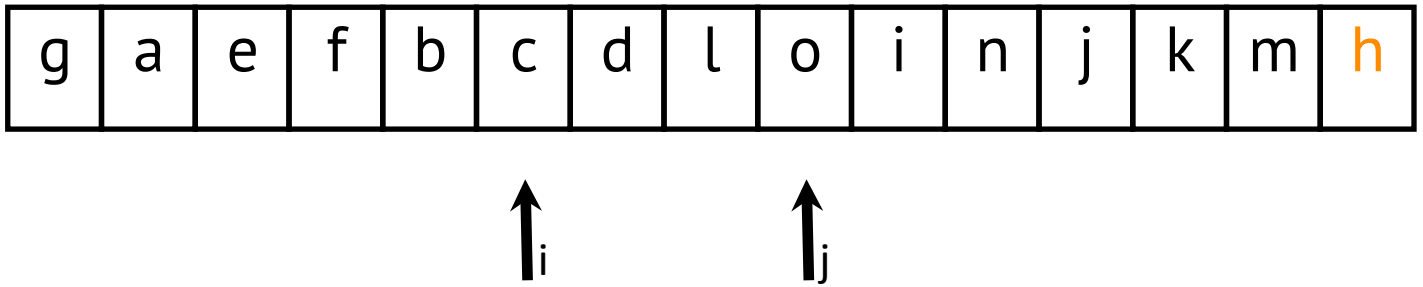
Quick-Sort: Beispiel



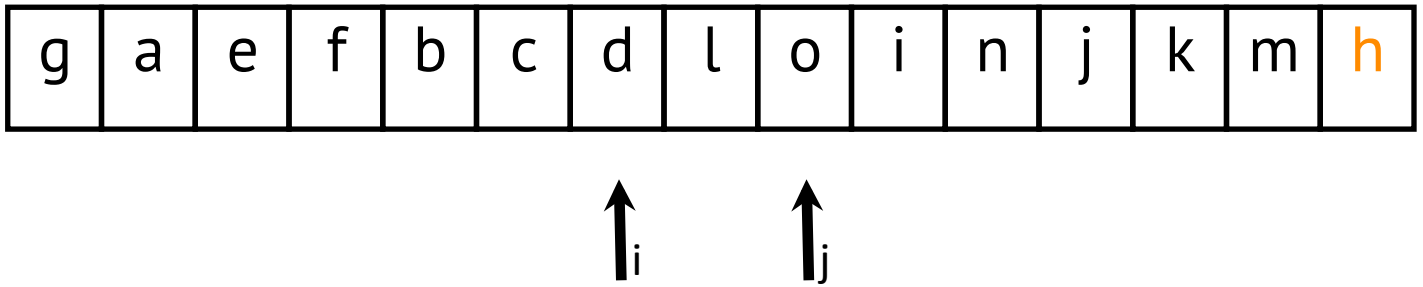
Quick-Sort: Beispiel



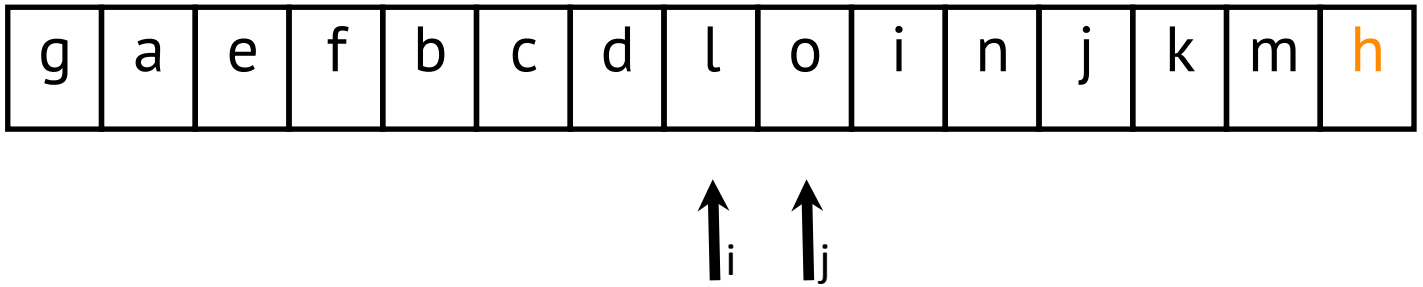
Quick-Sort: Beispiel



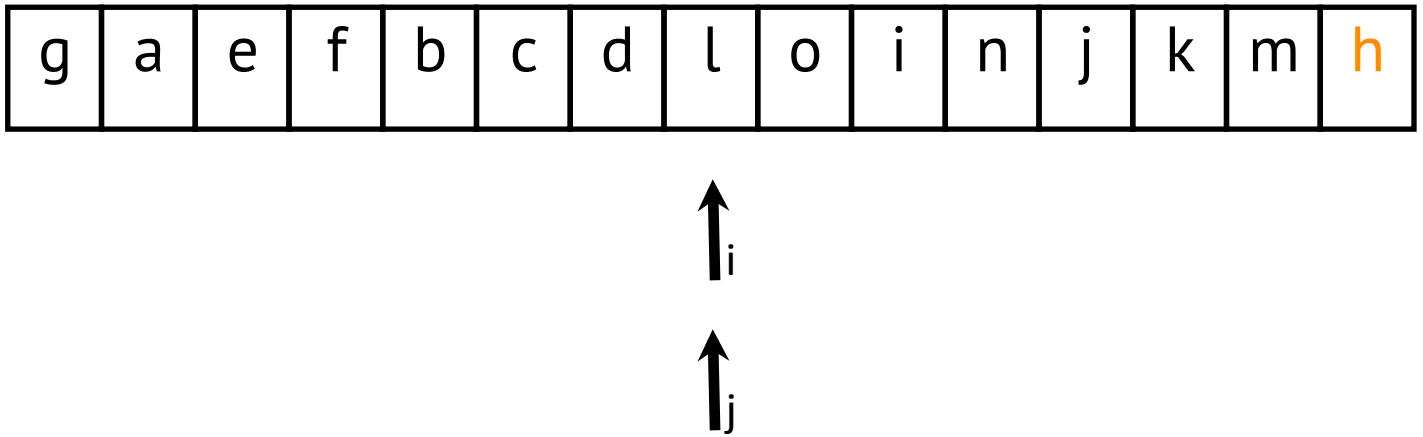
Quick-Sort: Beispiel



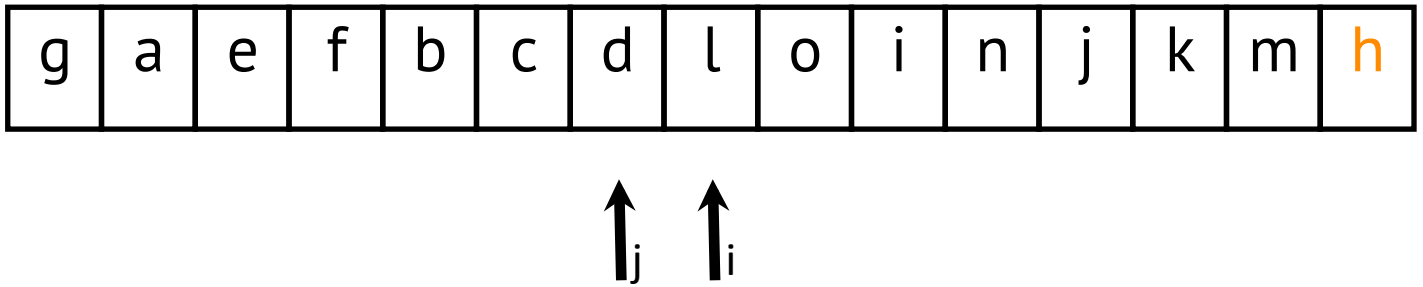
Quick-Sort: Beispiel



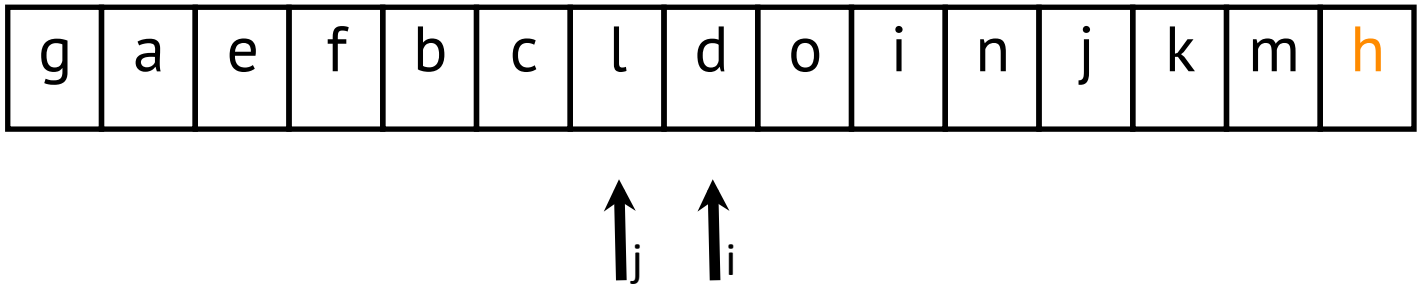
Quick-Sort: Beispiel



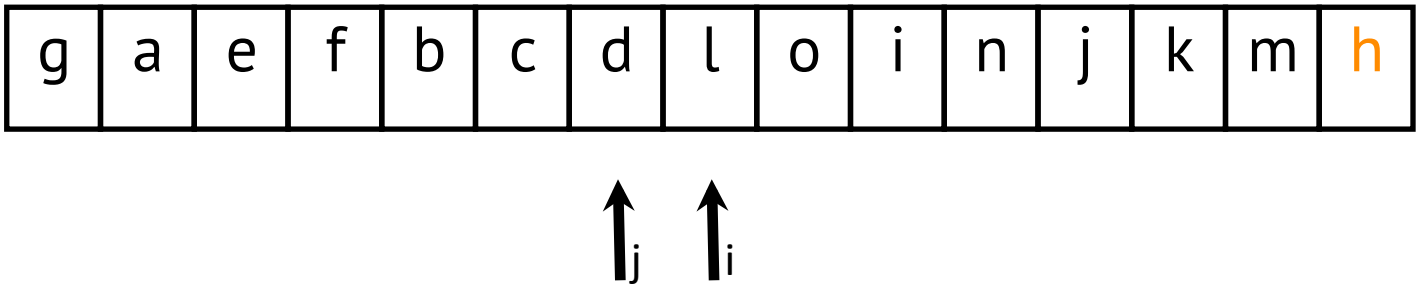
Quick-Sort: Beispiel



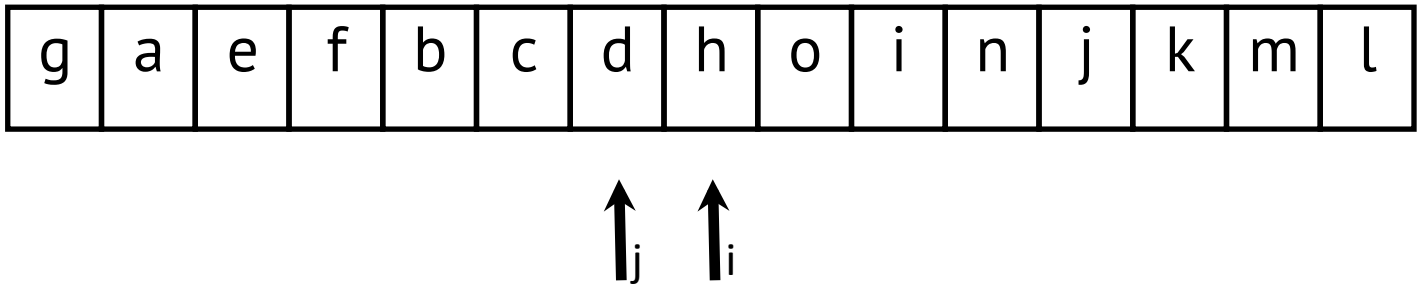
Quick-Sort: Beispiel



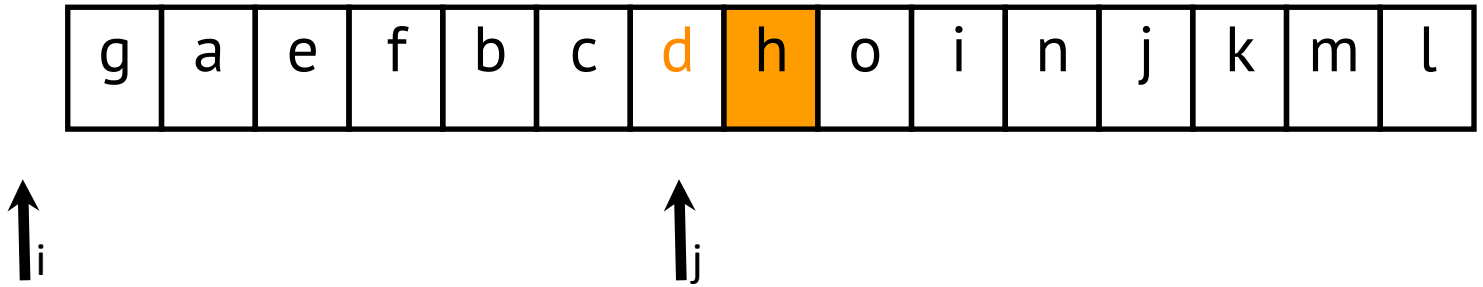
Quick-Sort: Beispiel



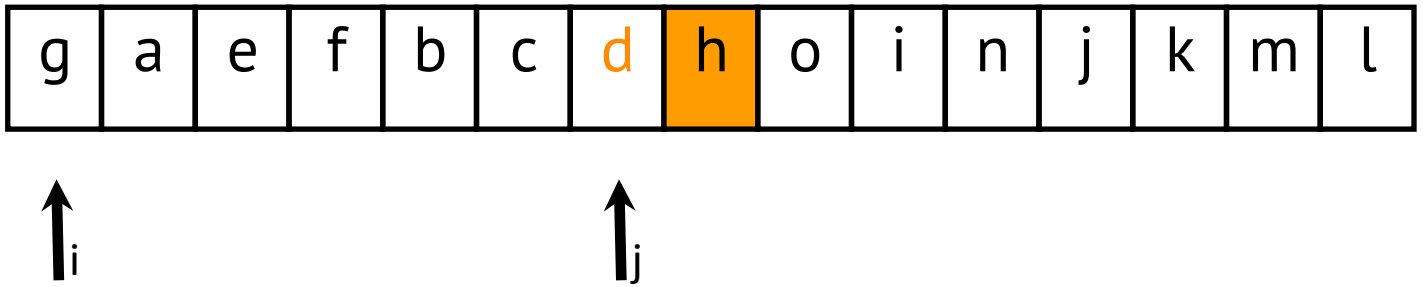
Quick-Sort: Beispiel



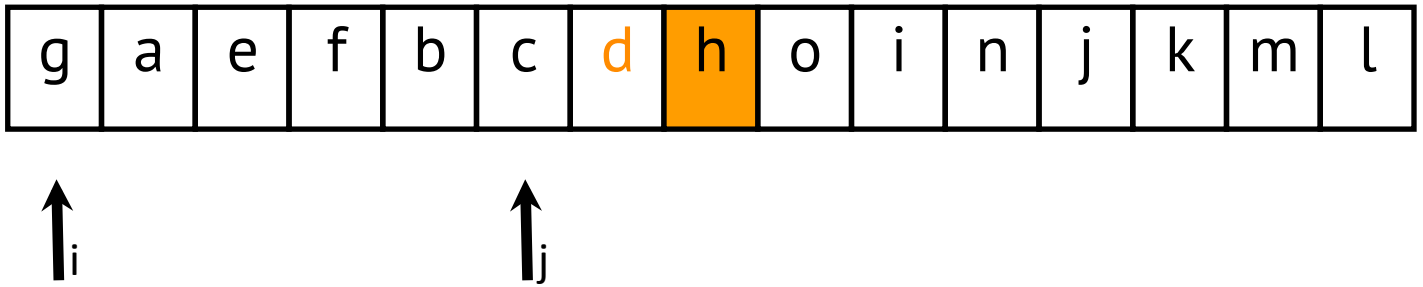
Quick-Sort: Beispiel



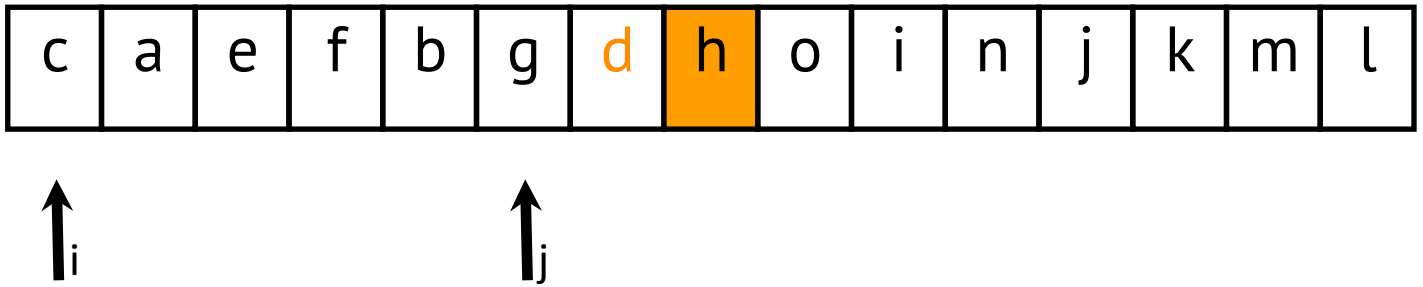
Quick-Sort: Beispiel



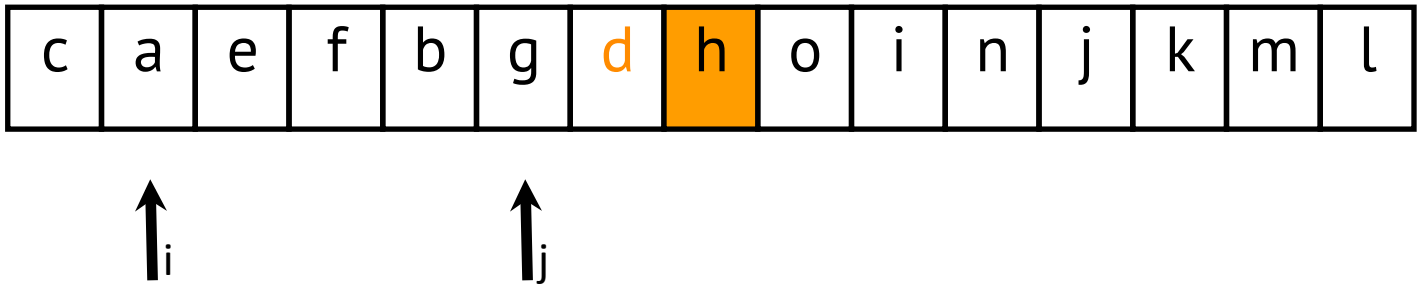
Quick-Sort: Beispiel



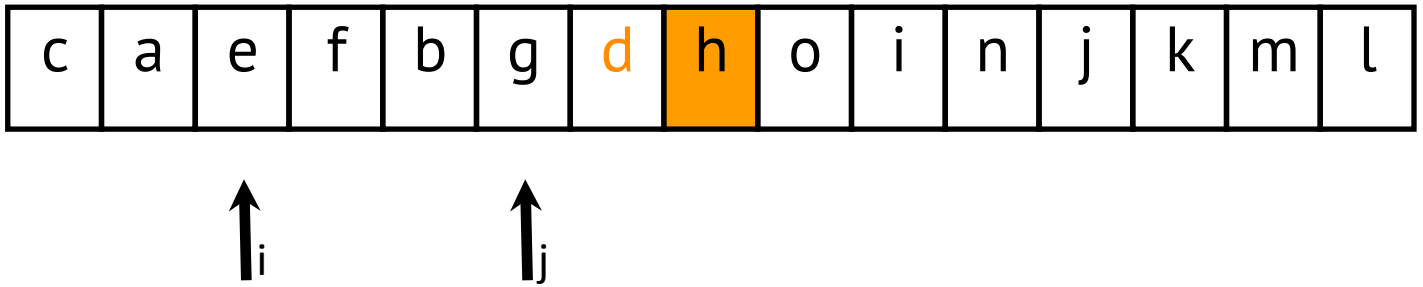
Quick-Sort: Beispiel



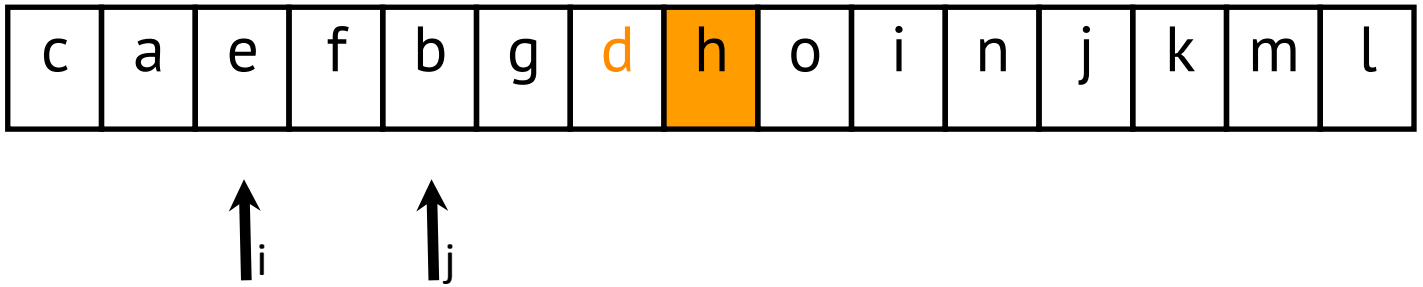
Quick-Sort: Beispiel



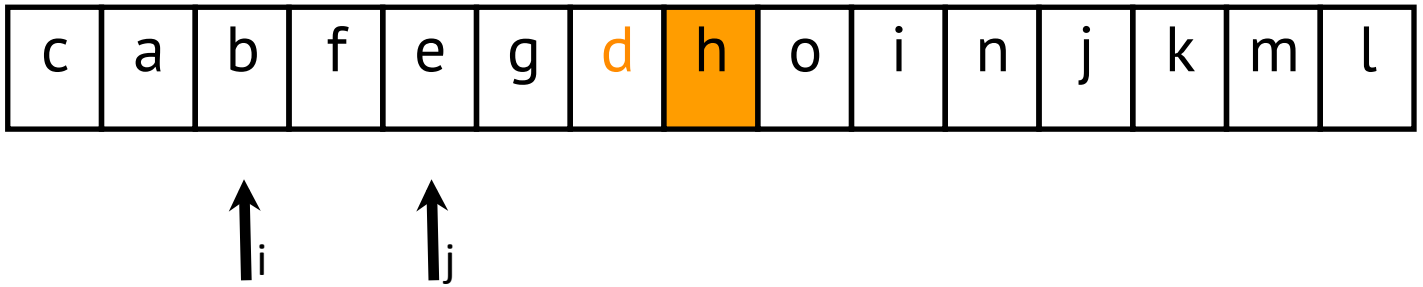
Quick-Sort: Beispiel



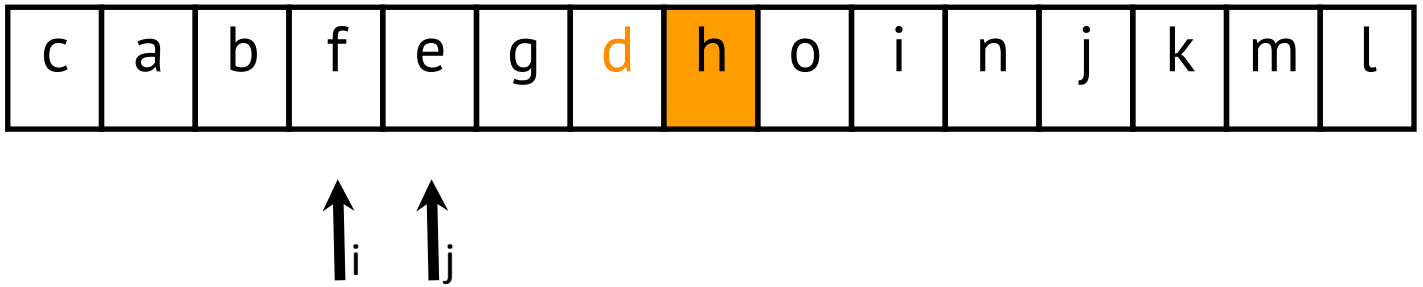
Quick-Sort: Beispiel



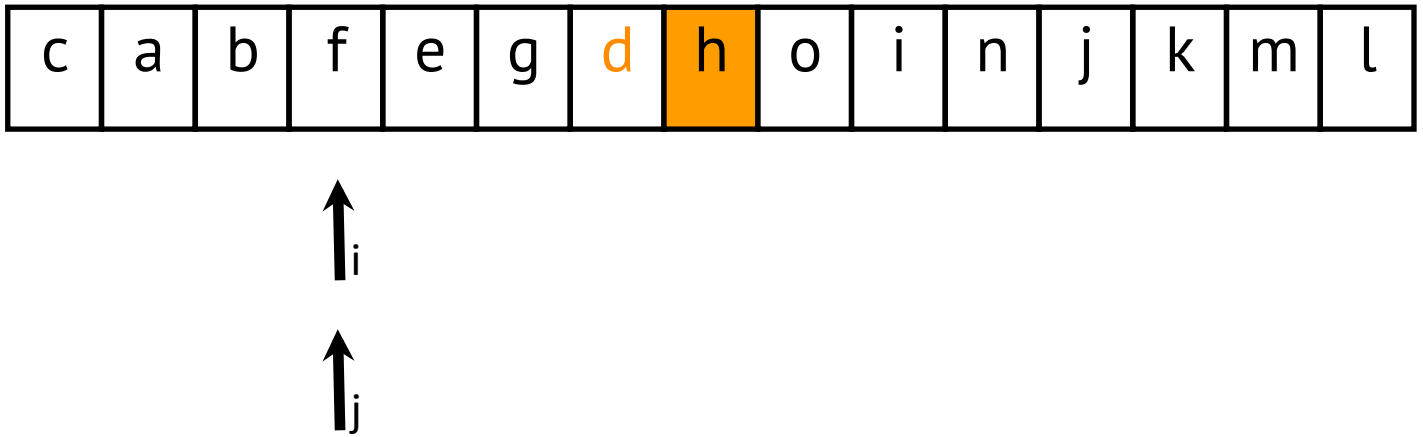
Quick-Sort: Beispiel



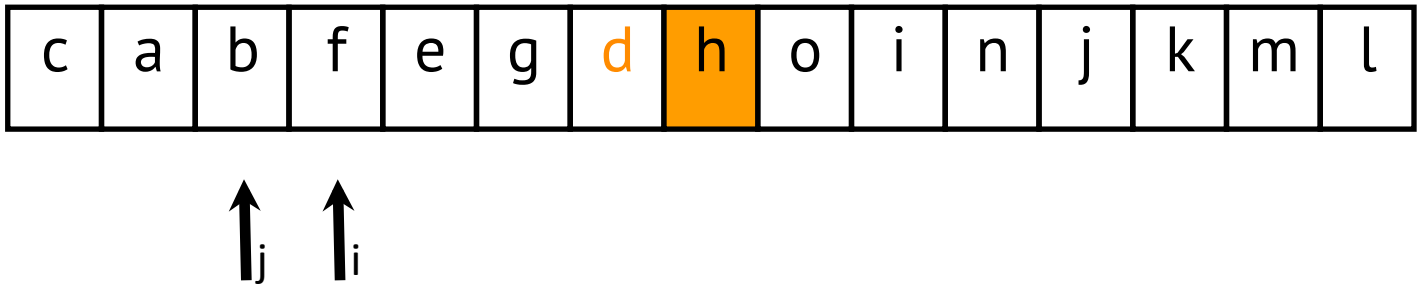
Quick-Sort: Beispiel



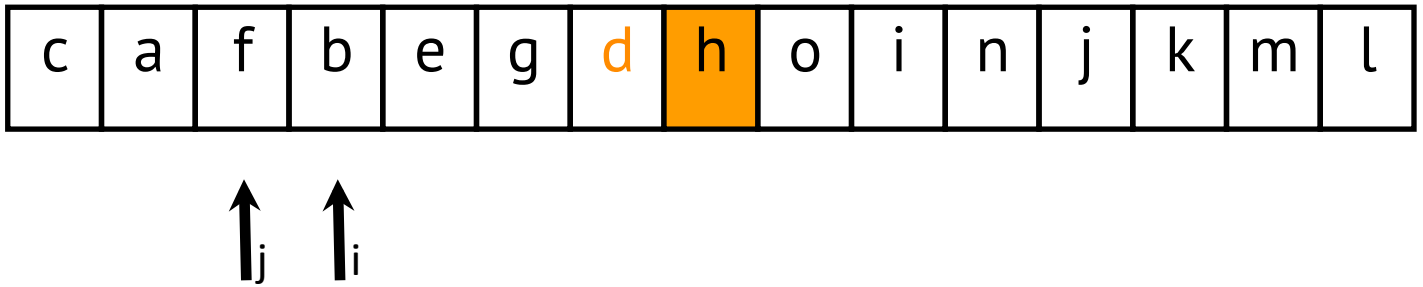
Quick-Sort: Beispiel



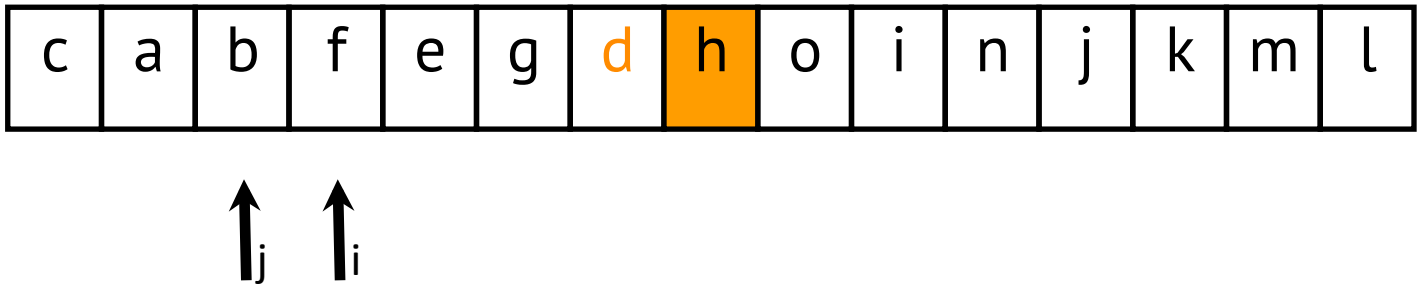
Quick-Sort: Beispiel



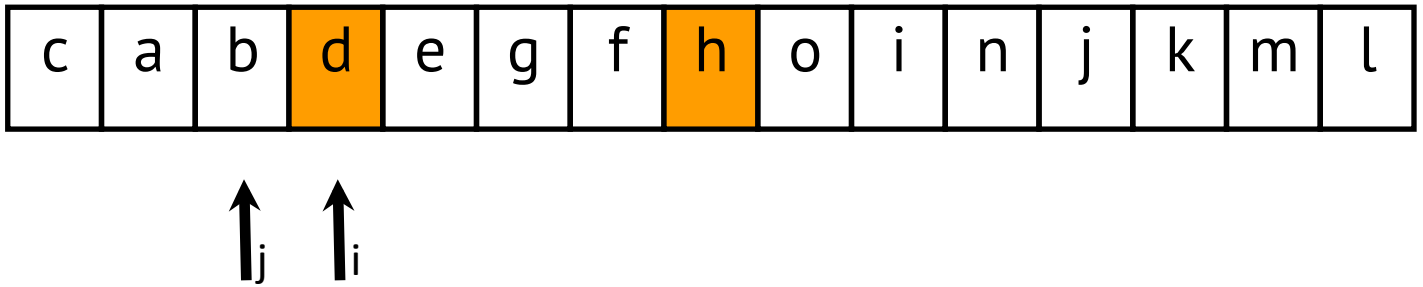
Quick-Sort: Beispiel



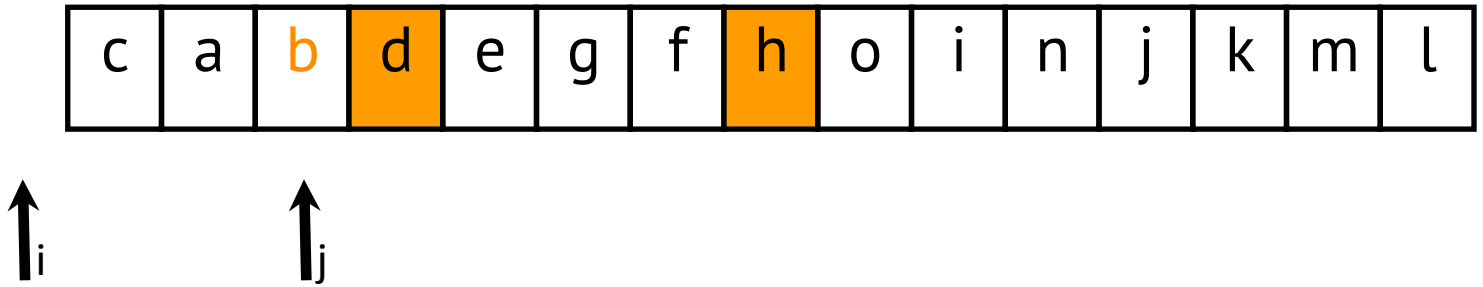
Quick-Sort: Beispiel



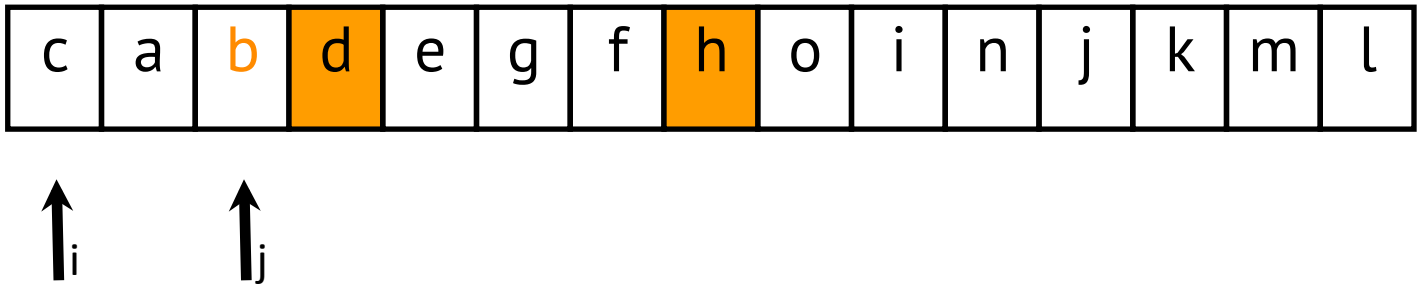
Quick-Sort: Beispiel



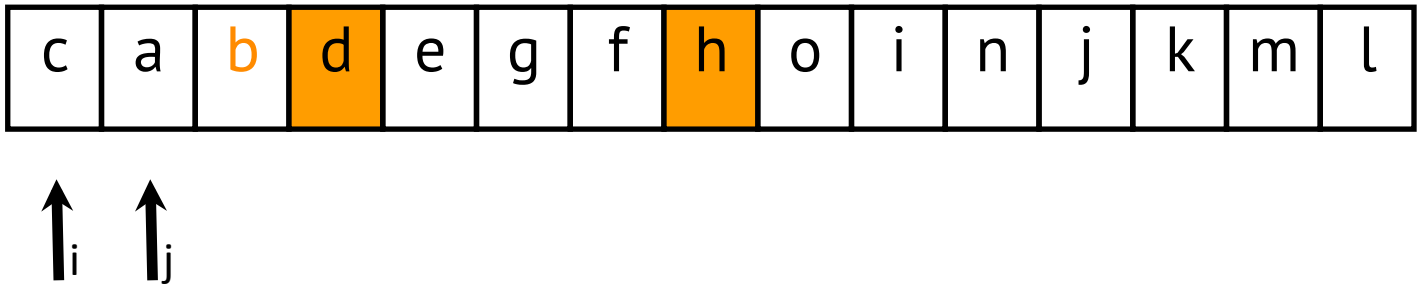
Quick-Sort: Beispiel



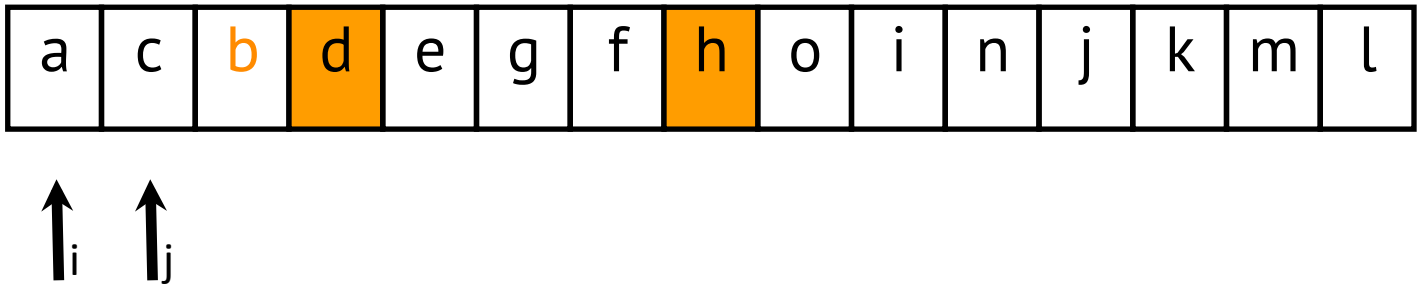
Quick-Sort: Beispiel



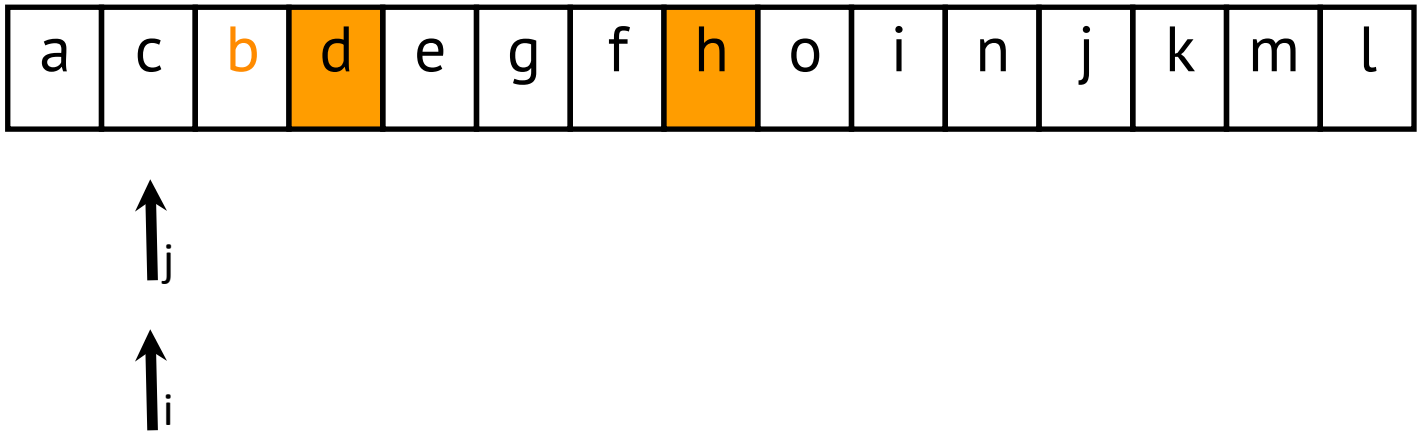
Quick-Sort: Beispiel



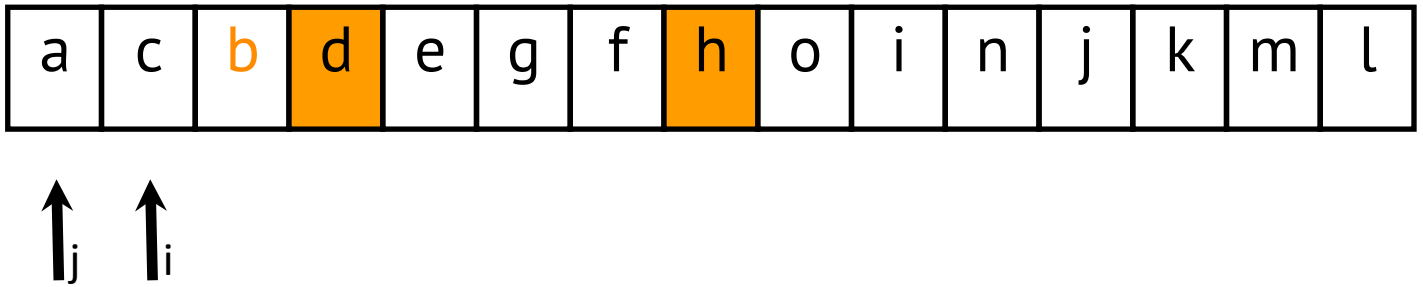
Quick-Sort: Beispiel



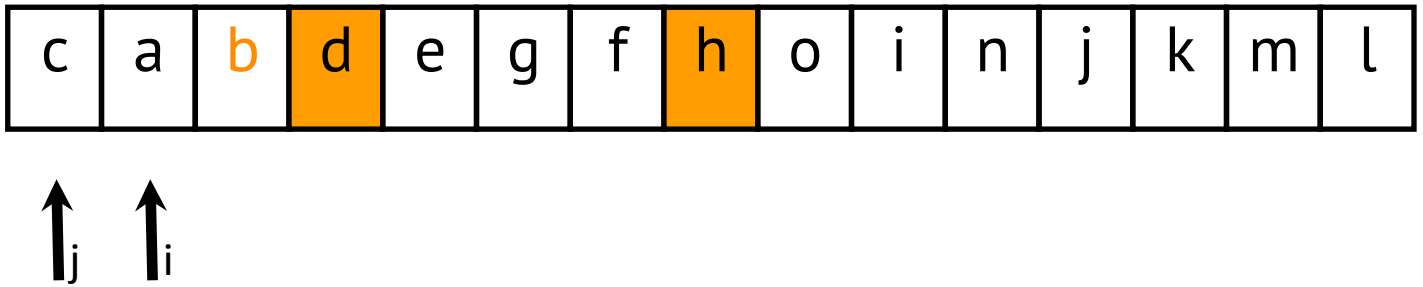
Quick-Sort: Beispiel



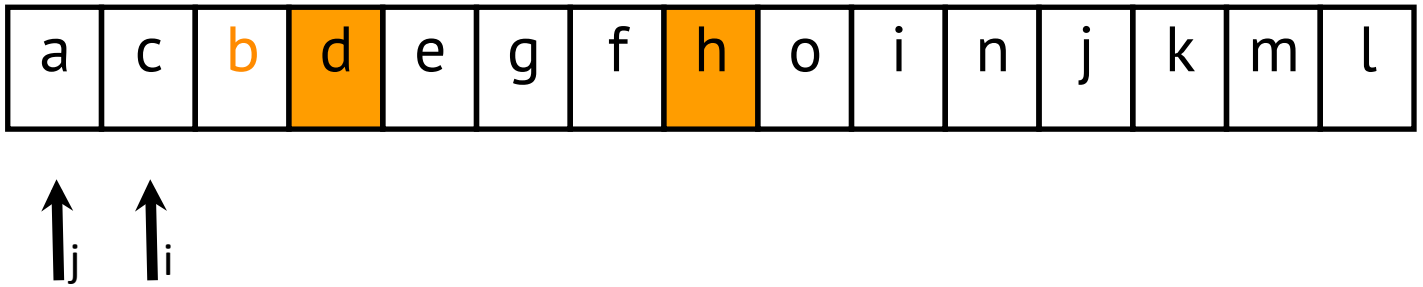
Quick-Sort: Beispiel



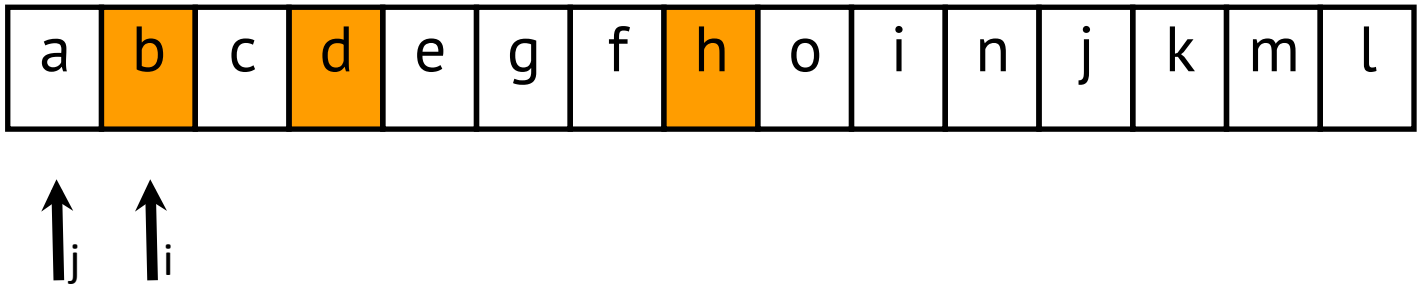
Quick-Sort: Beispiel



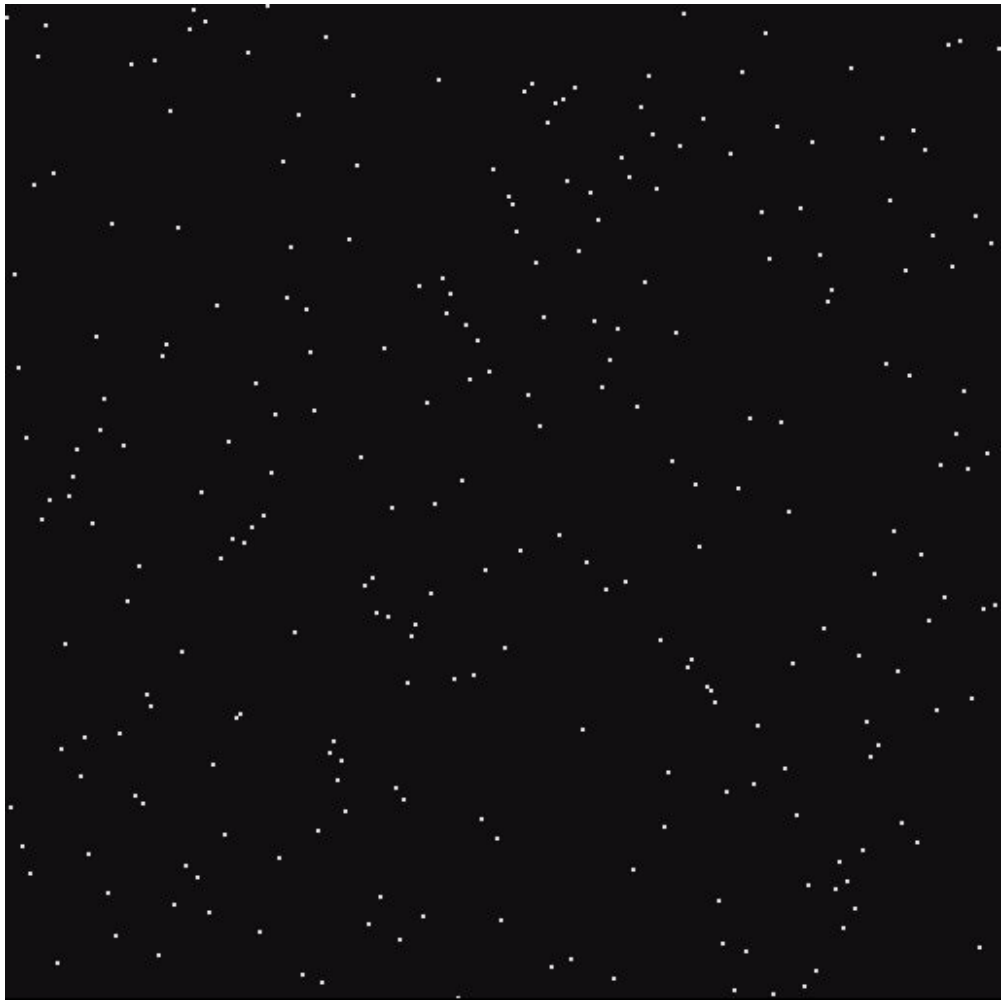
Quick-Sort: Beispiel



Quick-Sort: Beispiel



Quick-Sort: Beispiel



Quick-Sort: Aufwand

- Aufwand
  - Best Case:
    $T(n) \approx 2 \times T(n/2) + O(n) \Rightarrow O(n \times \log n)$
    (Folge zerfällt immer in zwei gleich große Teile)
  - Worst Case:
    $T(n) \approx T(n-1) + O(n) \Rightarrow O(n^2)$
    (Folge ist bereits auf- oder absteigend sortiert)
  - Average Case: ...

Quick-Sort: Aufwand

- Aufwand
  - Average case

$$T(n) \approx O(n) + \frac{1}{n}\left(\sum_{i=0}^{n-1} T(i) + T(n-1-i)\right)$$

$$= O(n) + \frac{2}{n}\sum_{i=0}^{n-1} T(i)$$

Quick-Sort: Aufwand

$$nT(n) = cn^2 + 2\sum_{i=0}^{n-1} T(i)$$

$$(n-1)T(n-1) = c(n-1)^2 + 2\sum_{i=0}^{n-2} T(i)$$

$$nT(n) - (n-1)T(n-1) = cn^2 - c(n-1)^2 + 2T(n-1)$$

$$nT(n) = 2cn - c + (n+1)T(n-1)$$

Quick-Sort: Aufwand

$$
\begin{aligned}
T(n) &= c' + \tfrac{n+1}{n} T(n-1) \\
&= c' + \tfrac{n+1}{n} c' + \tfrac{n+1}{n-1} T(n-2) \\
&= c' + \tfrac{n+1}{n} c' + \tfrac{n+1}{n-1} c' + \tfrac{n+1}{n-2} T(n-3) \\
&= c' \left(\tfrac{n+1}{n+1} + \tfrac{n+1}{n} + \tfrac{n+1}{n-1} + \cdots + \tfrac{n+1}{1} \right) \\
&= c'(n+1) \sum_{i=1}^{n+1} \tfrac{1}{i} \\
&\leq c''(n+1) \log(n+1) \\
&= O(n \log n)
\end{aligned}
$$

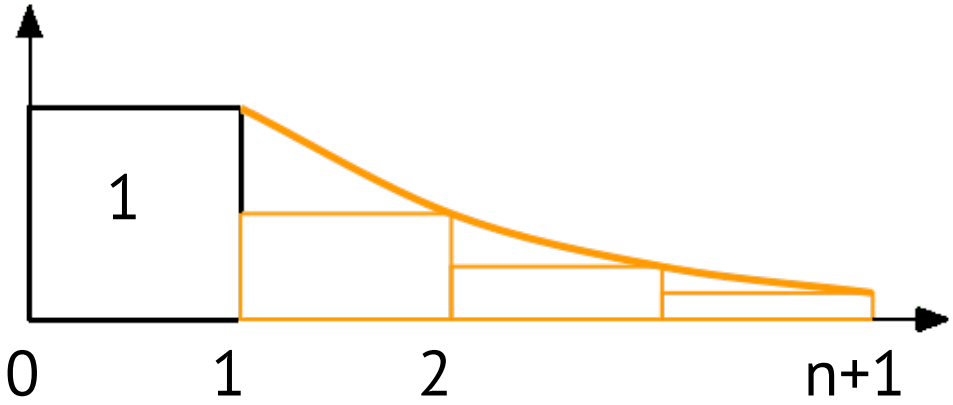
Quick-Sort: Aufwand

$$\sum_{i=1}^{n+1} \frac{1}{i} \leq 1 + \int_1^{n+1} \frac{1}{x} dx$$

$$= 1 + ln(x)|_1^{n+1}$$

$$= 1 + ln(n+1)$$

$$= O(\log(n+1))$$



Quick-Sort: Aufwand

- Average Case: $T(n) = O(n \times \log n)$
- Heuristiken zur Verbesserung der Performanz
  - Pivot-Element: (A[l]+A[r])/2, …
  - Bei kleinen Folgen auf Insertion-Sort wechseln
    (oder auch nicht: Cache-Kohärenz)
  - Iterative Implementierung mit Stack

Merge-Sort

- Idee
  - Quick-Sort ist schnell, wenn die beiden Teilfolgen nach Partition() ungefähr die gleiche Größe haben.
  - Teile die Folge **ohne** Vorsortieren in zwei gleiche Teile
  - Sortiere die Teilfolgen
  - Kombiniere die Teilergebnisse

Merge-Sort (rekursiv)

- MergeSort(A[l..r])
  if l < r then
    m ← (l+r)/2
    MergeSort(A[l..m])
    MergeSort(A[m+1..r])
    Merge(A[l..m,m+1..r])

vgl. mit Quick-Sort

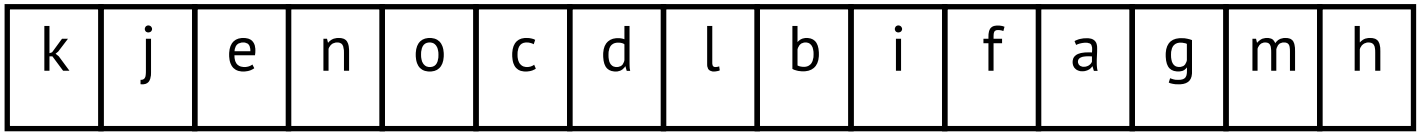
- Aufruf mit: MergeSort(A,1,n)

Merge-Sort (rekursiv)

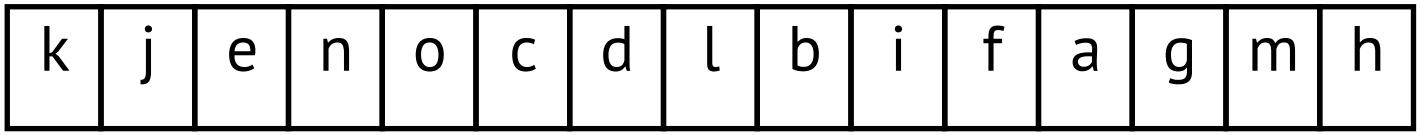
- Merge(A[l..m,m+1..r])

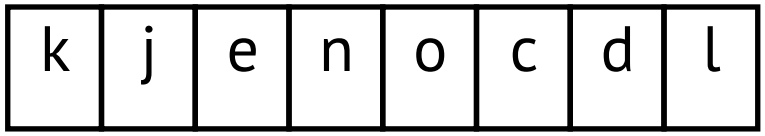
```
new B[1..r−l+1]; i ← 1; j ← l; k ← m+1
while j ⩽ m and k ⩽ r do
    if A[j] ⩽ A[k] then
        B[i++] ← A[j++]
    else
        B[i++] ← A[k++]
while j ⩽ m do B[i++] ← A[j++]
while k ⩽ r  do B[i++] ← A[k++]
for i ← l to r do
    A[ i ] ← B[ i−l+1 ]
```

Merge-Sort: Beispiel

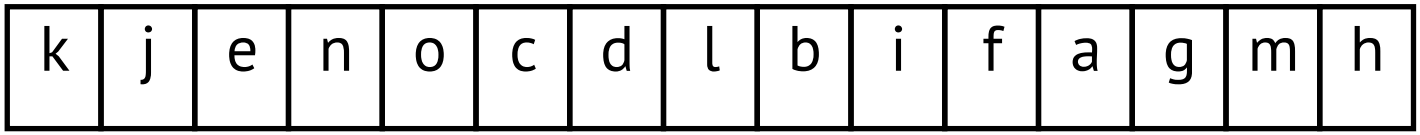


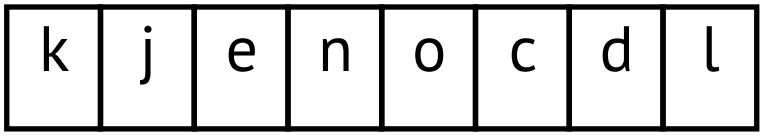
Merge-Sort: Beispiel

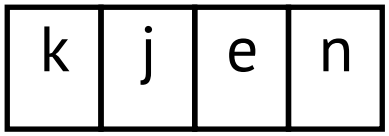




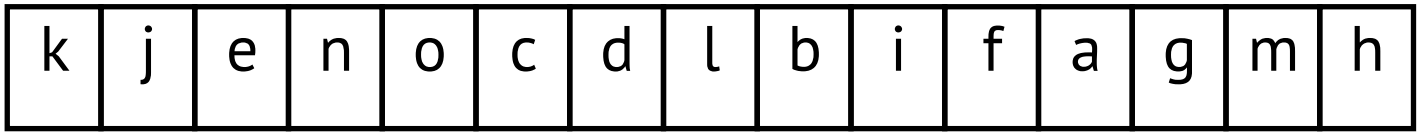
Merge-Sort: Beispiel

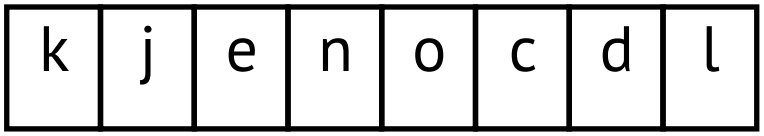


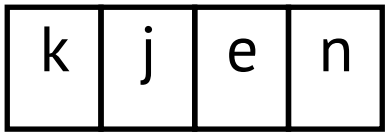


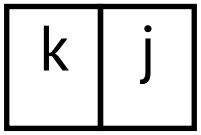


Merge-Sort: Beispiel

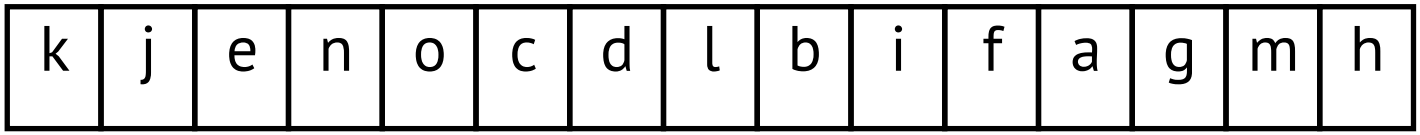


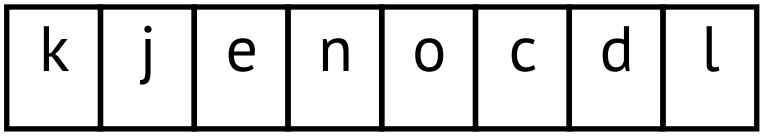


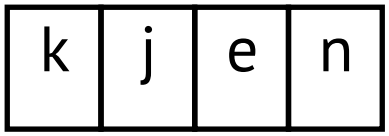


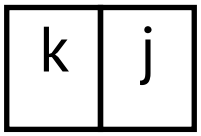


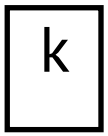
Merge-Sort: Beispiel



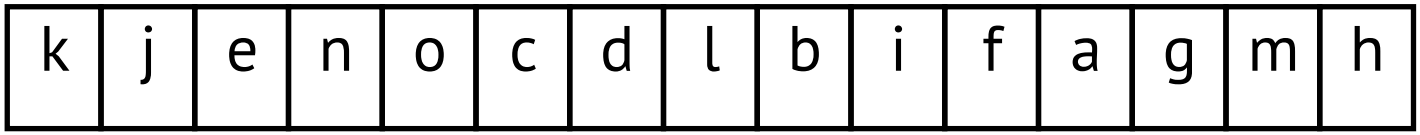


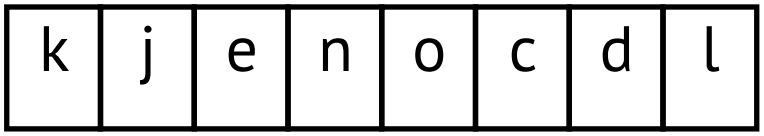


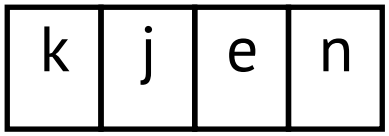


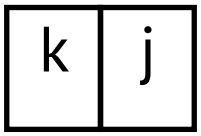


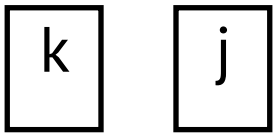
Merge-Sort: Beispiel



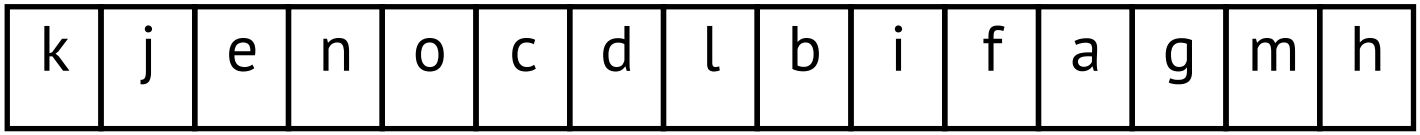


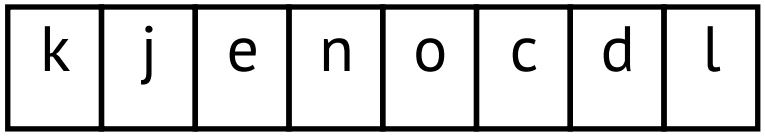


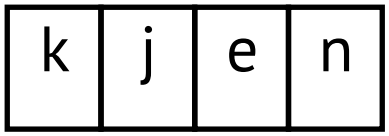


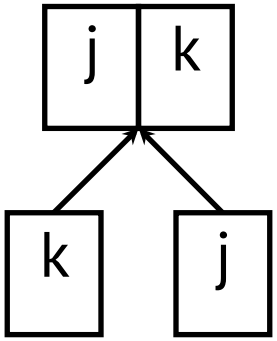


Merge-Sort: Beispiel

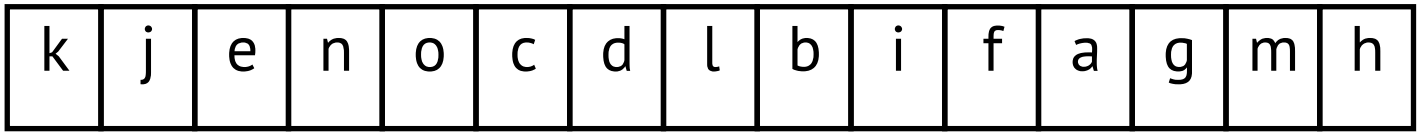


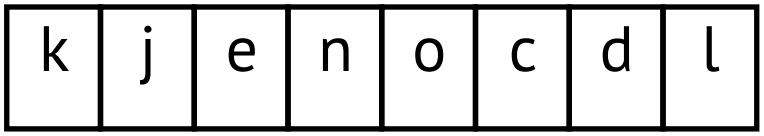


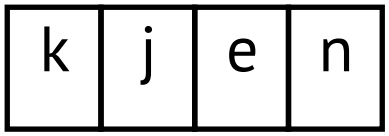


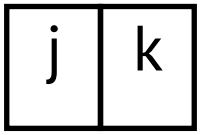


Merge-Sort: Beispiel

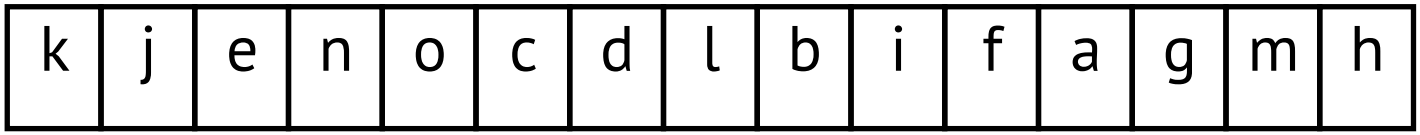


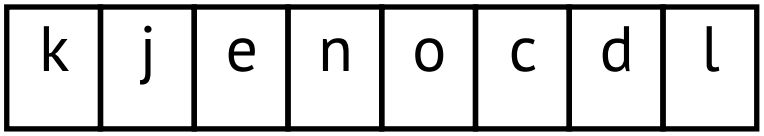


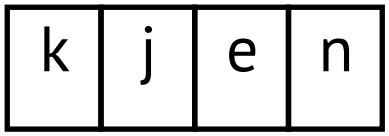


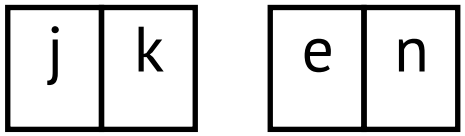


Merge-Sort: Beispiel

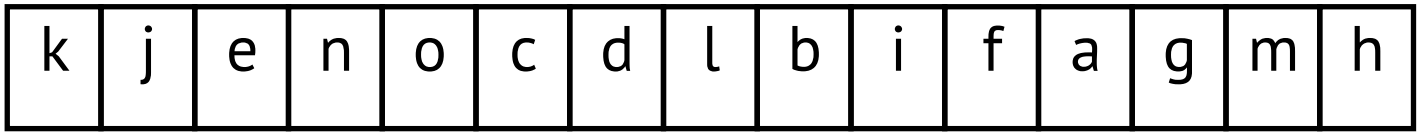


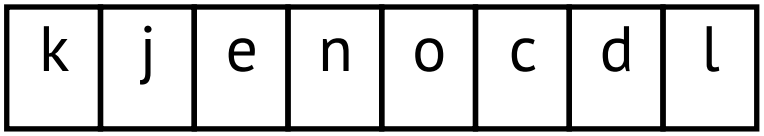


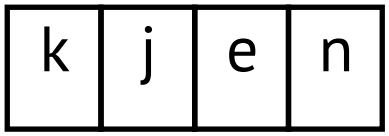


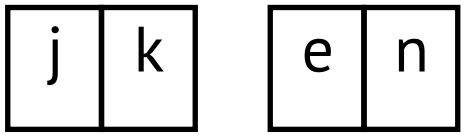


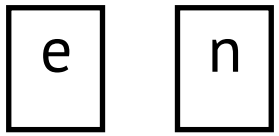
Merge-Sort: Beispiel



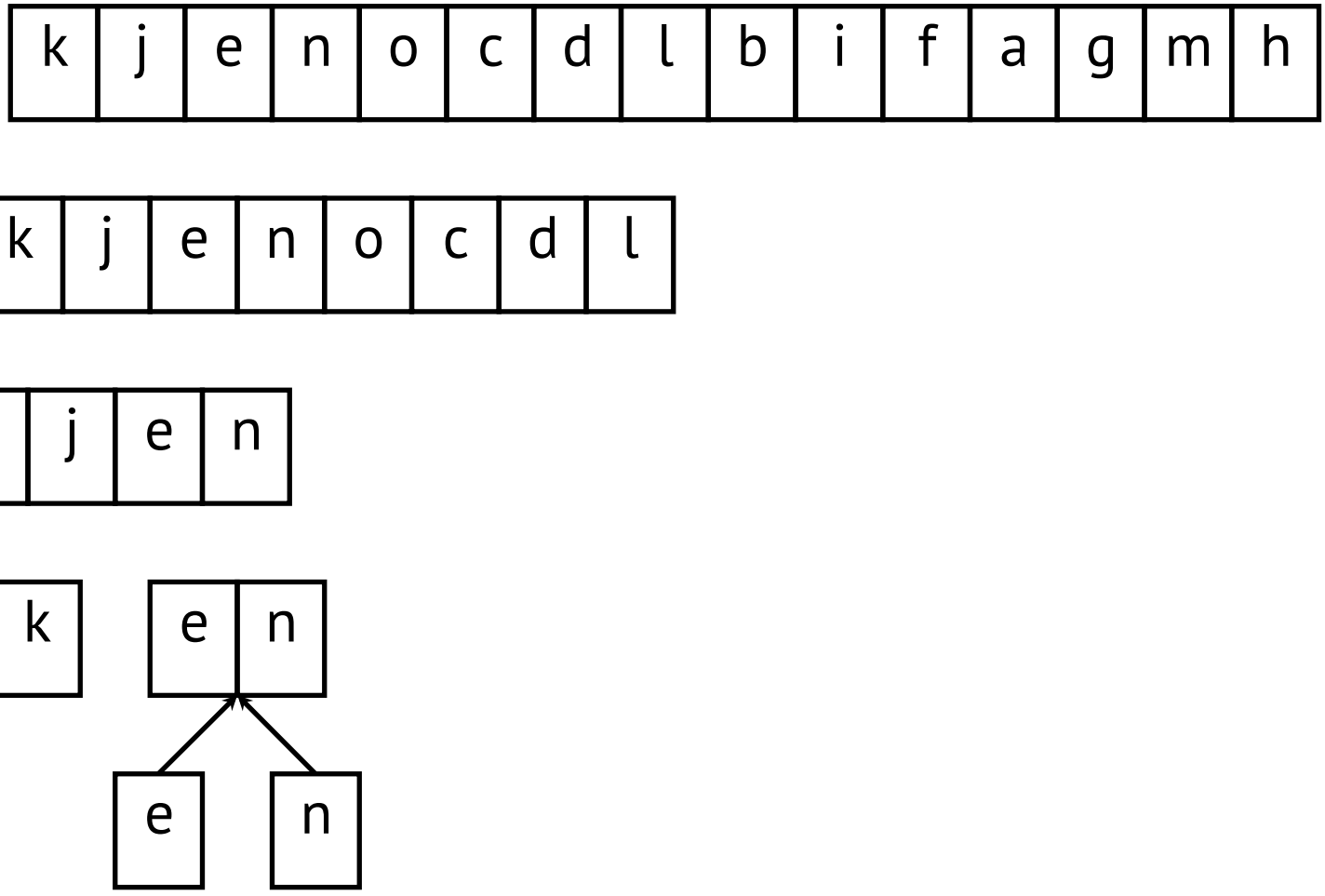




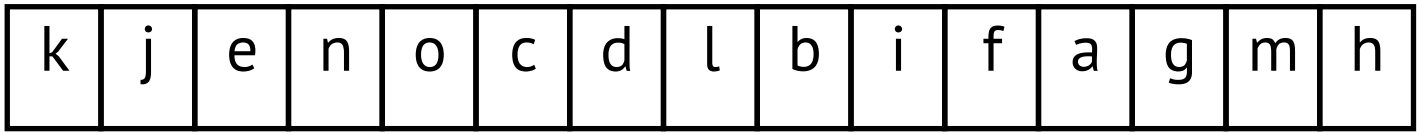


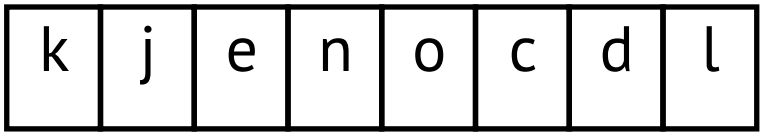


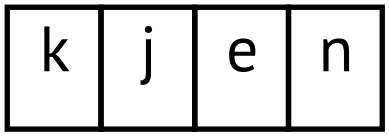
Merge-Sort: Beispiel

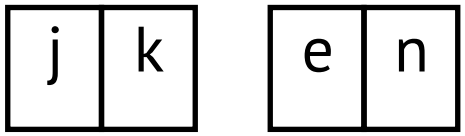


Merge-Sort: Beispiel

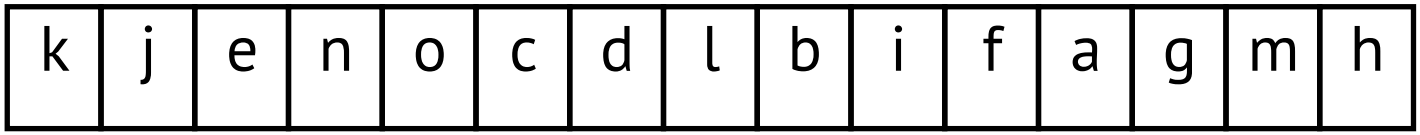


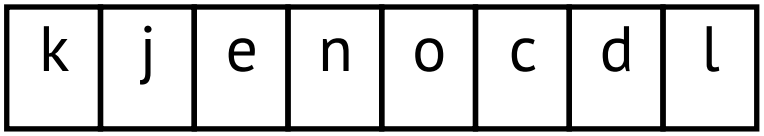


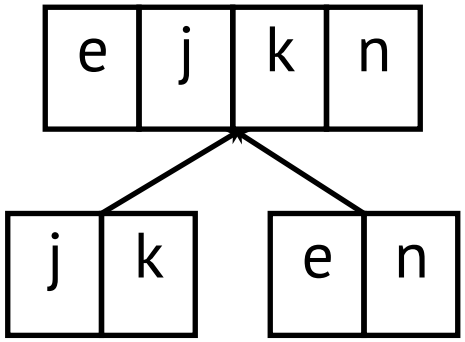




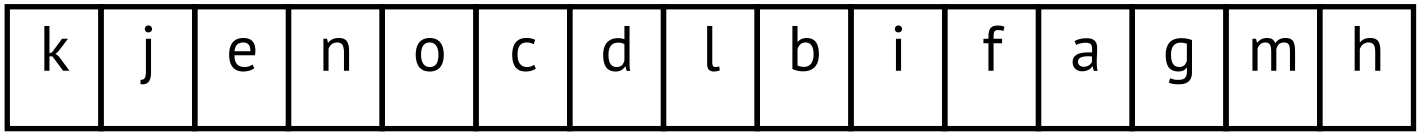
Merge-Sort: Beispiel

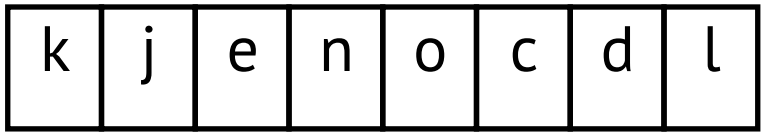


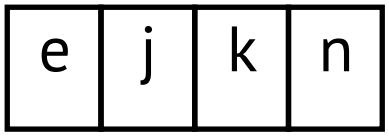




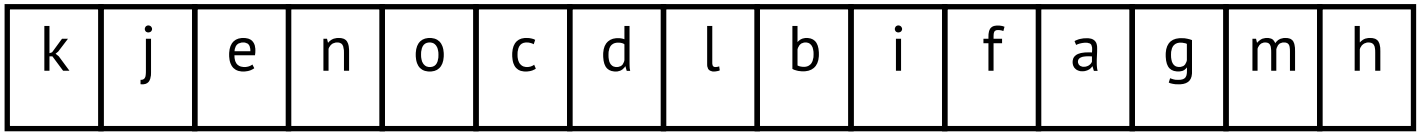
Merge-Sort: Beispiel

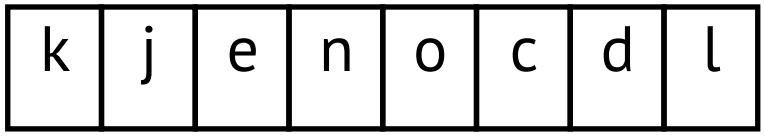


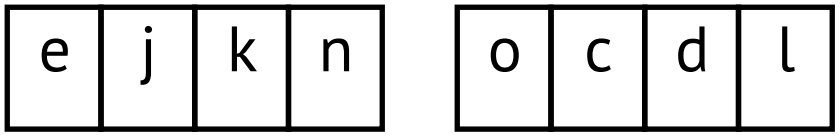




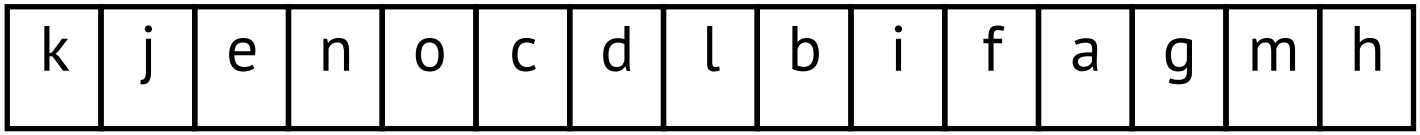
Merge-Sort: Beispiel

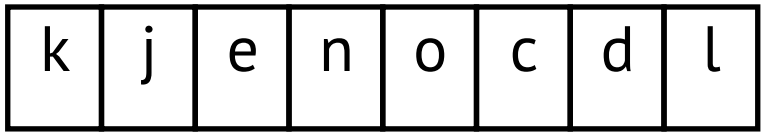


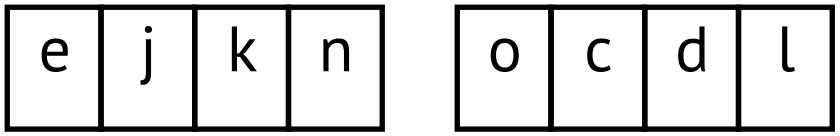


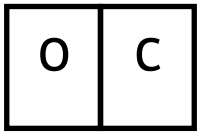


Merge-Sort: Beispiel

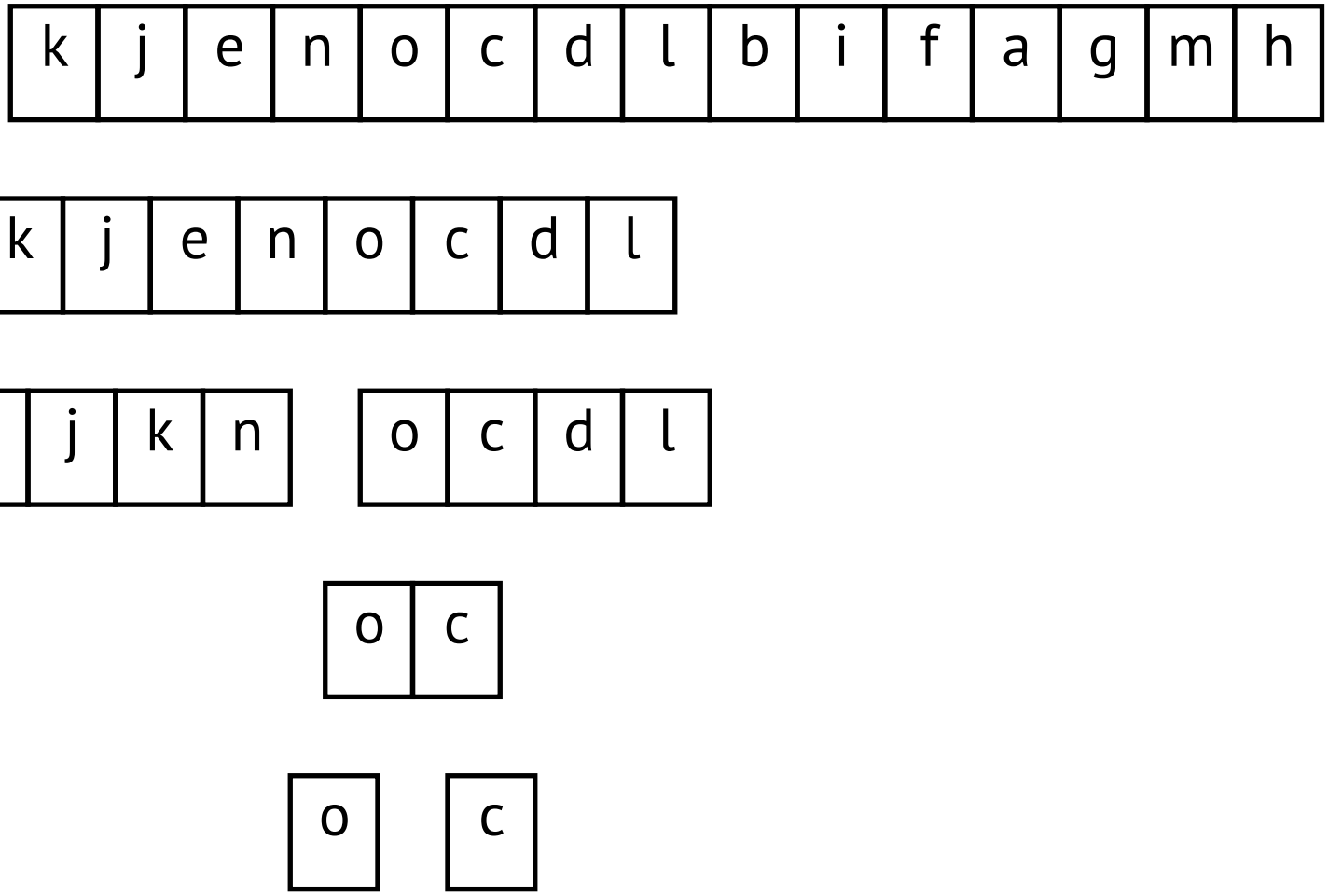




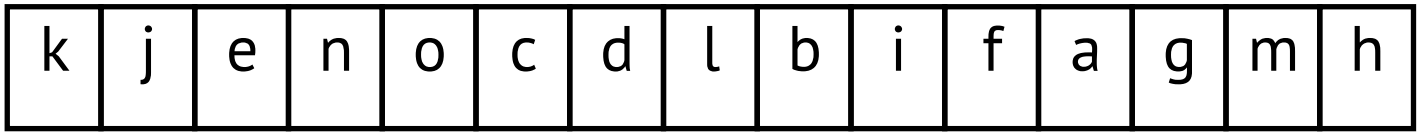


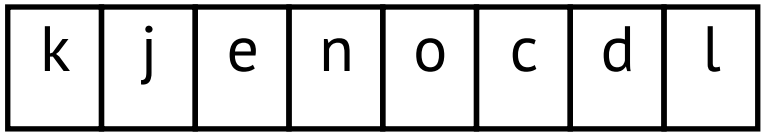


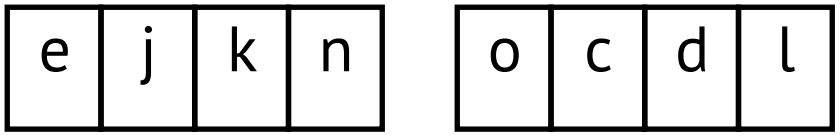
Merge-Sort: Beispiel

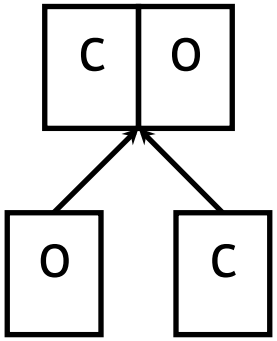


Merge-Sort: Beispiel

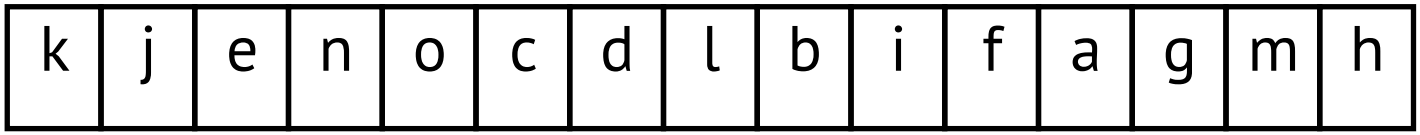


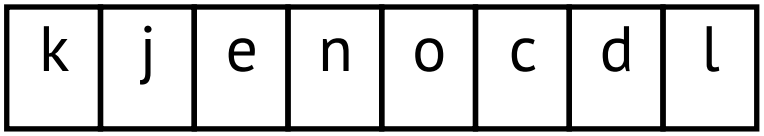


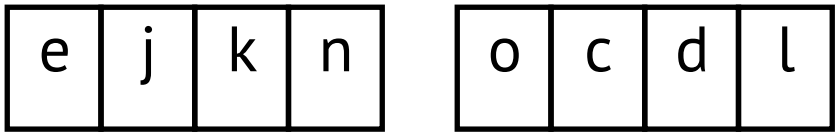


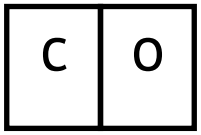


Merge-Sort: Beispiel

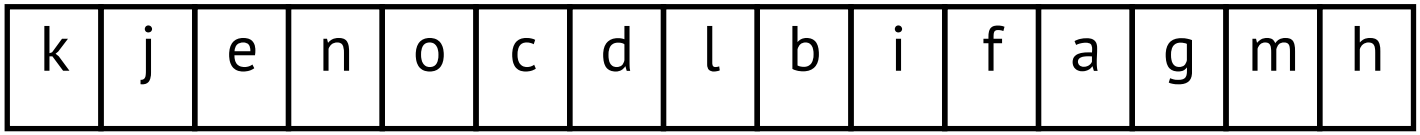


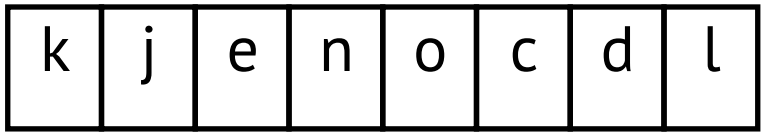


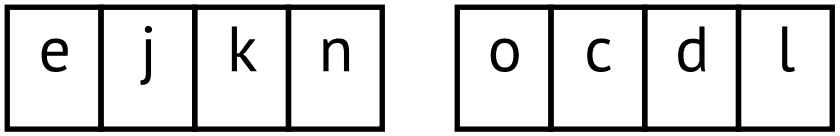


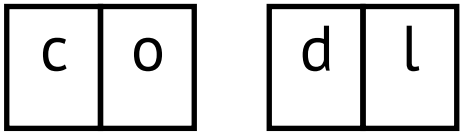


Merge-Sort: Beispiel

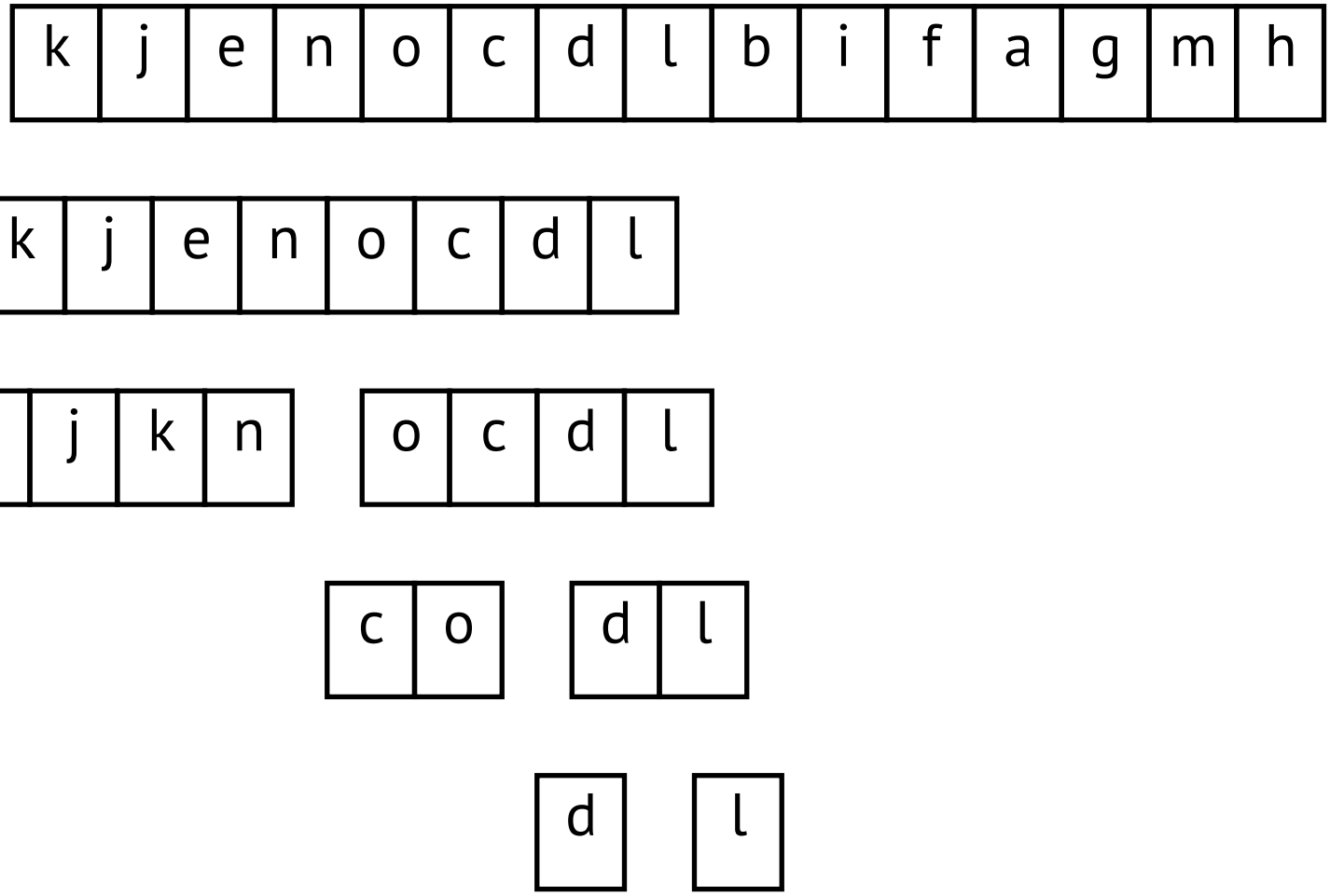




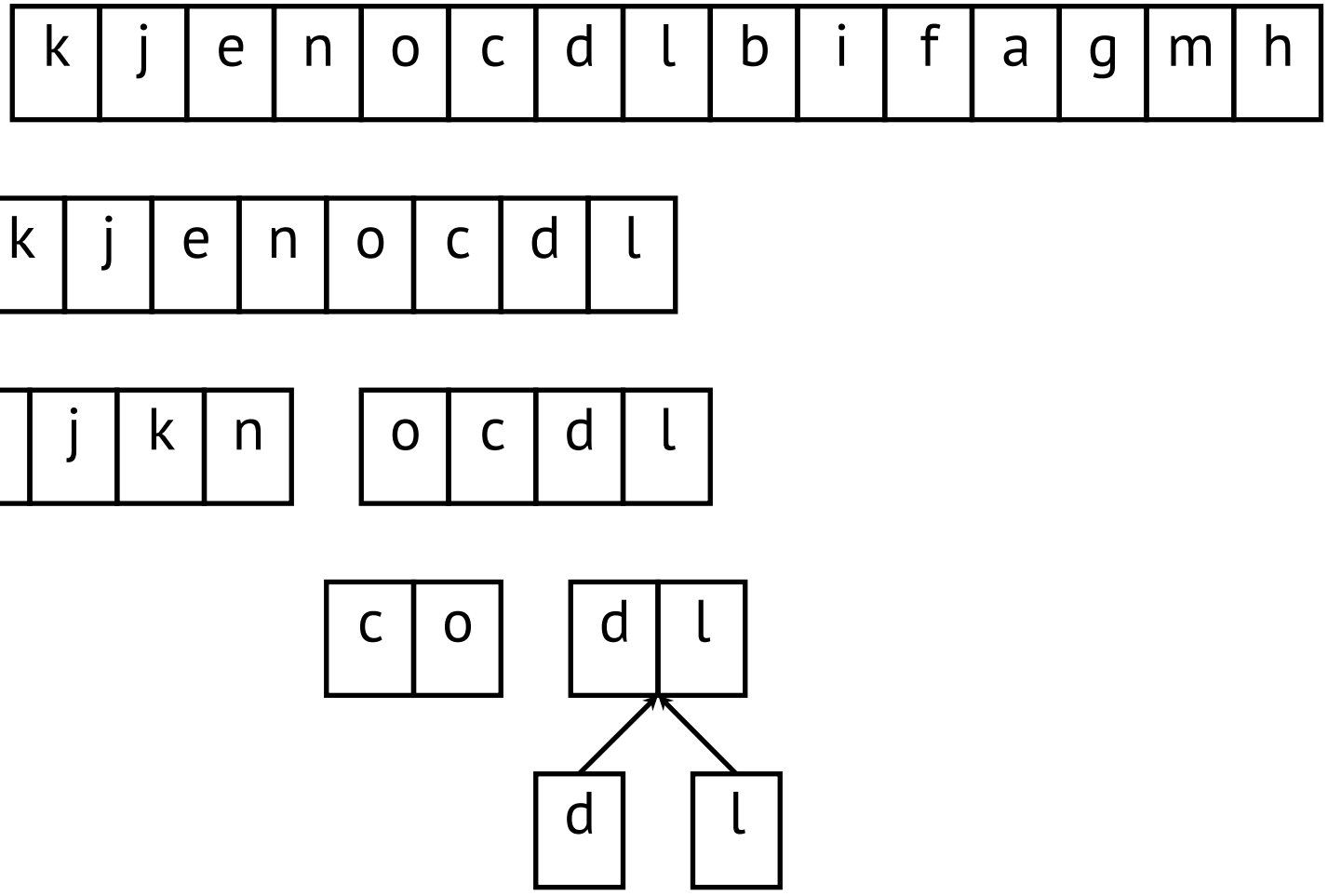




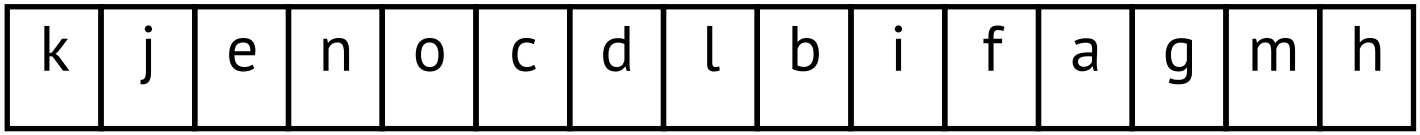
Merge-Sort: Beispiel

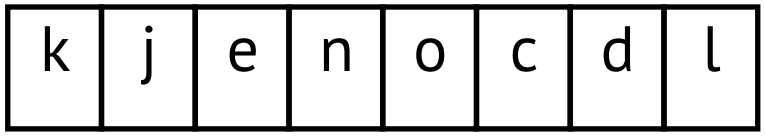


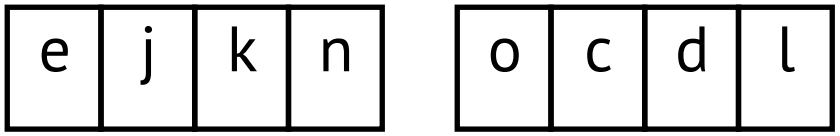
Merge-Sort: Beispiel

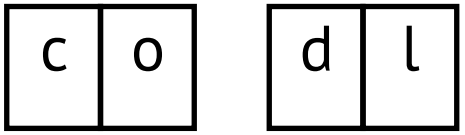


Merge-Sort: Beispiel

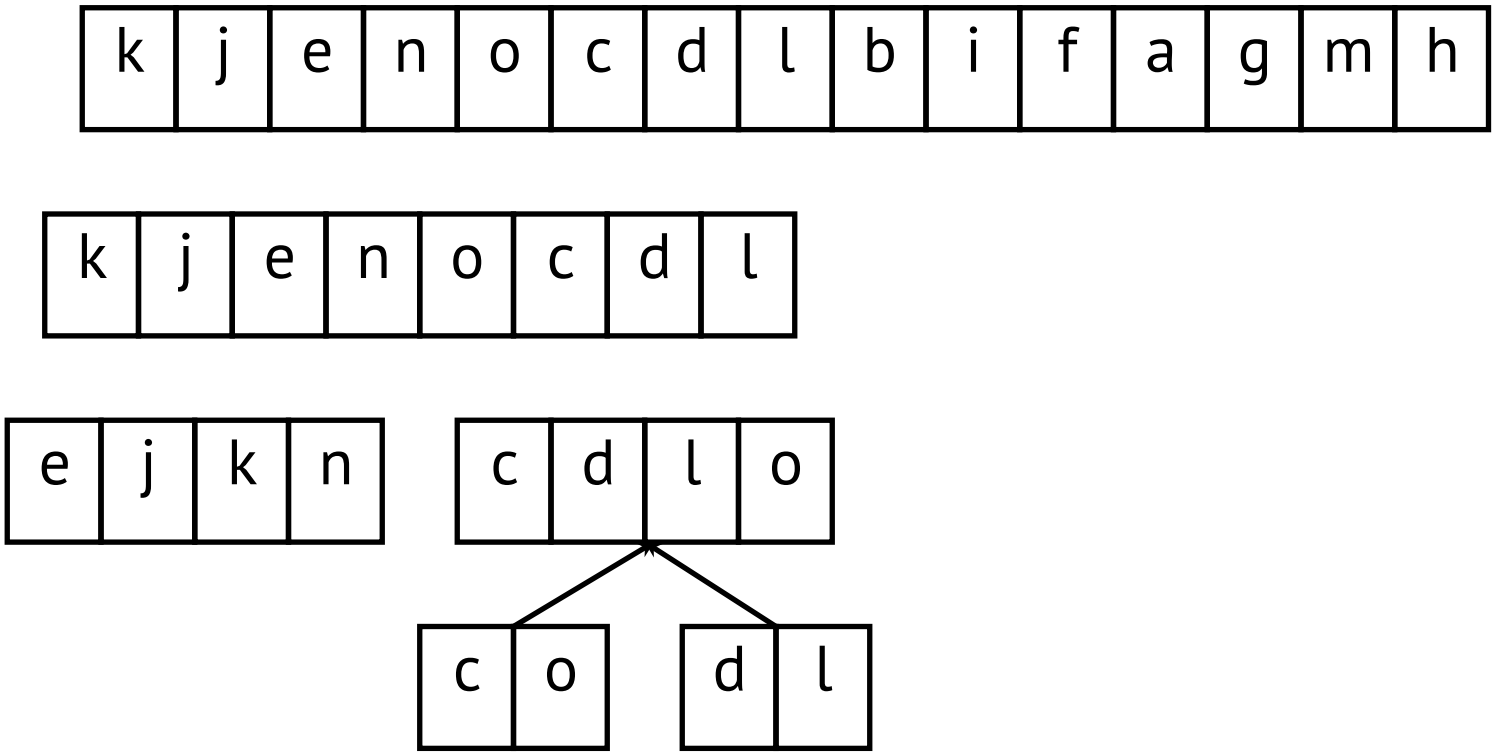




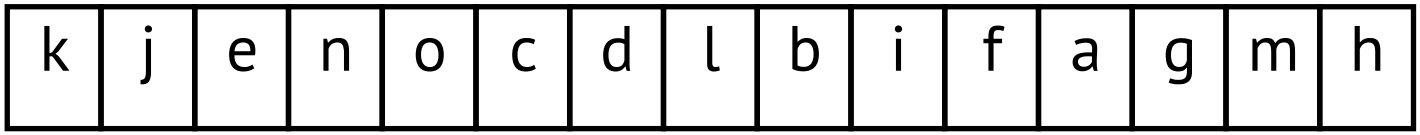


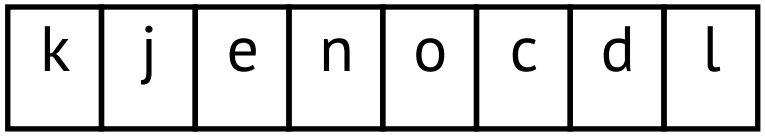


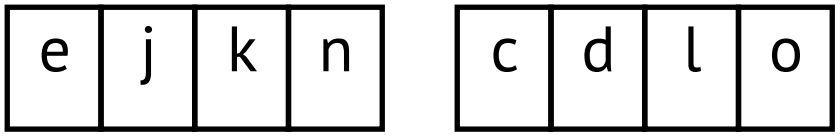
Merge-Sort: Beispiel



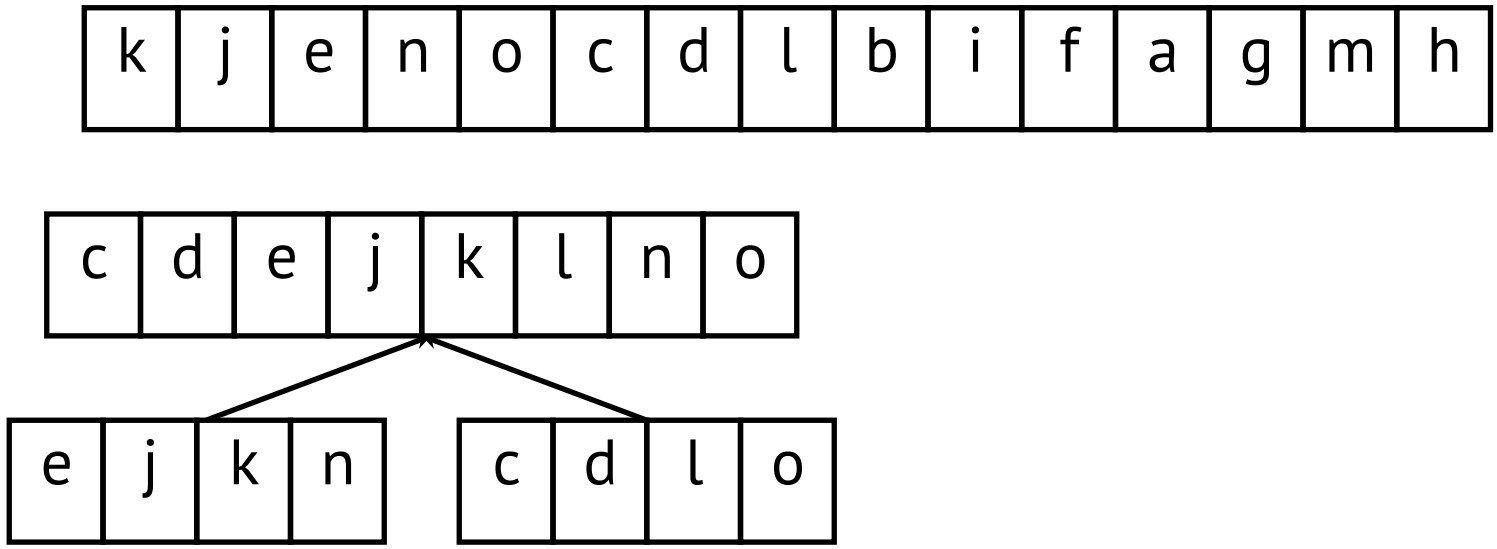
Merge-Sort: Beispiel



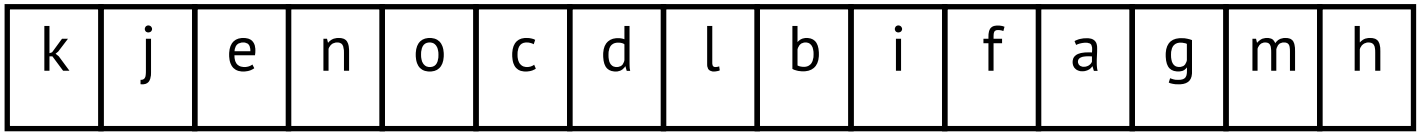


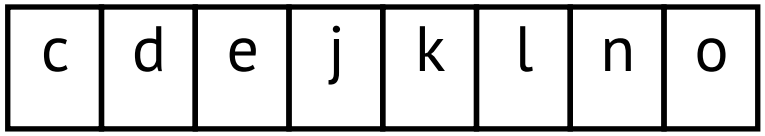


Merge-Sort: Beispiel



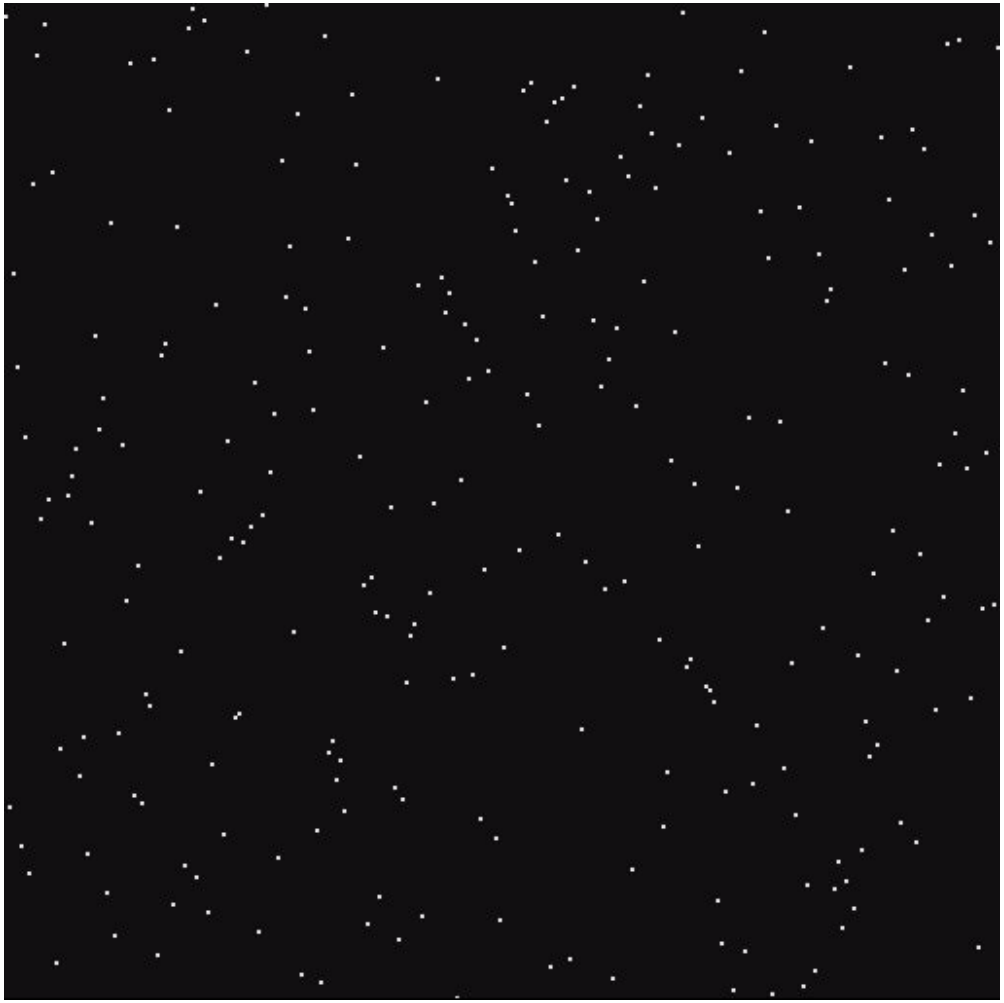
Merge-Sort: Beispiel





usw.

Merge-Sort (rekursiv)



Merge-Sort: Aufwand

- Das Mischen zweier Folgen der Länge L_1 und L_2 erfordert höchstens L_1+L_2 Vergleiche und genau $2\times(L_1+L_2)$ Kopier-operationen (i++ in jedem Schleifendurchlauf)
- Summe aller Folgenlängen auf jeder Rekursionsstufe = n
- Anzahl der Rekursionsstufen = log(n)
- Best = Worst = Average Case = $O(n\times\log n)$
- Aber: Doppelter Speicherbedarf !

Merge-Sort (iterativ)

- Die Zerlegung verändert die Folge nicht.
- Die top-down Rekursion steuert nur die Reihenfolge beim Mischen.
- Fasse die unsortierte Liste als Folge von sortierten einelementigen Folgen auf.
- Mische die Teilfolgen bottom-up

Merge-Sort (iterativ)

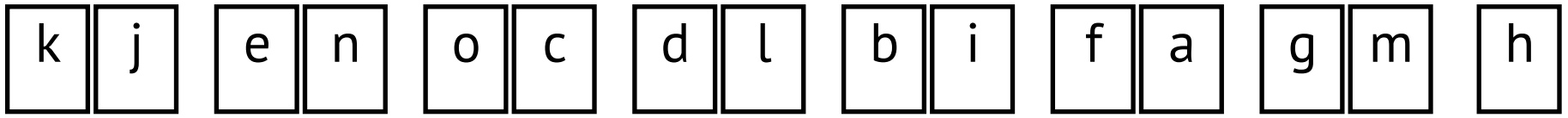
- MergeSort(A[1..n])

```
k ← 1
while k < n do
    i ← 1
    while i + k − 1 < n do
        l ← i
        m ← i + k − 1
        r ← Min(n, i + 2*k − 1)
        Merge(A[ l..m,m+1..r] )
        i ← i + 2*k
    k ← 2*k
```

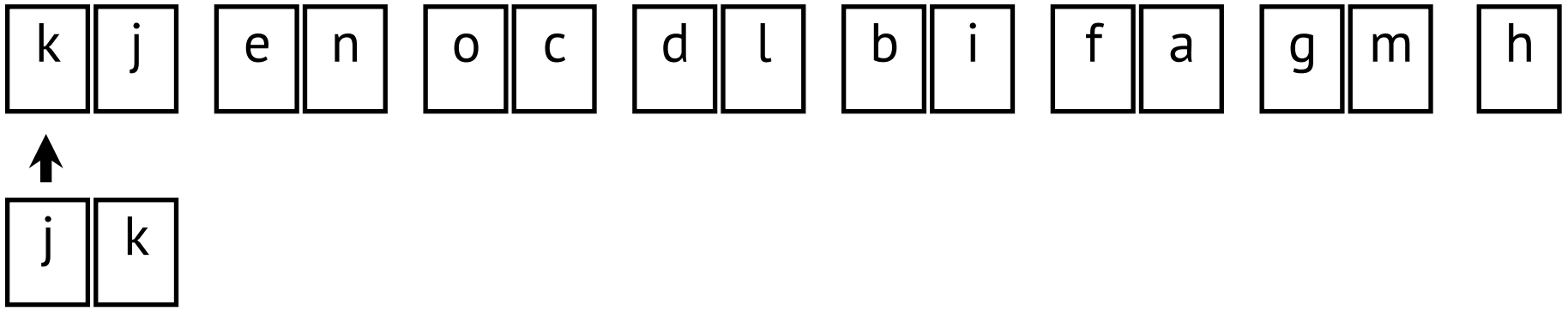
Merge-Sort (iterativ): Beispiel

k	j	e	n	o	c	d	l	b	i	f	a	g	m	h

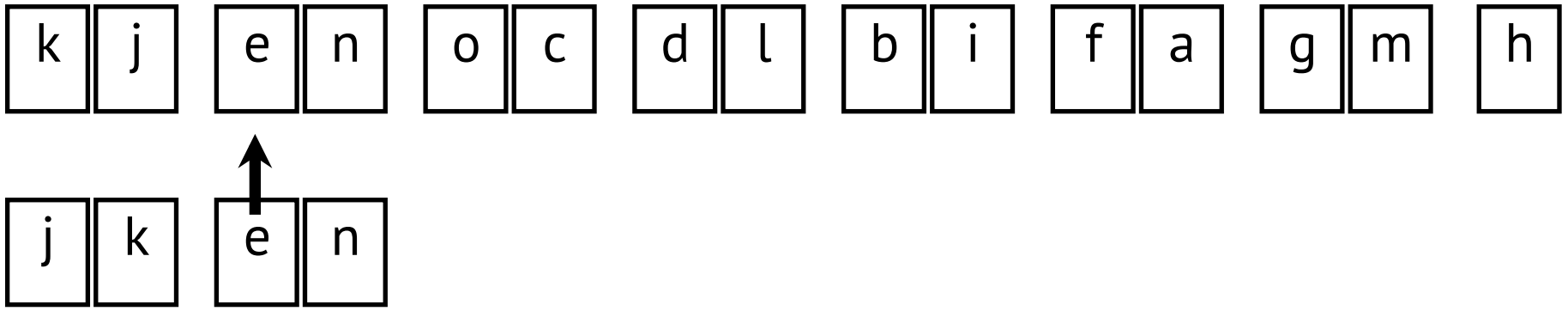
Merge-Sort (iterativ): Beispiel



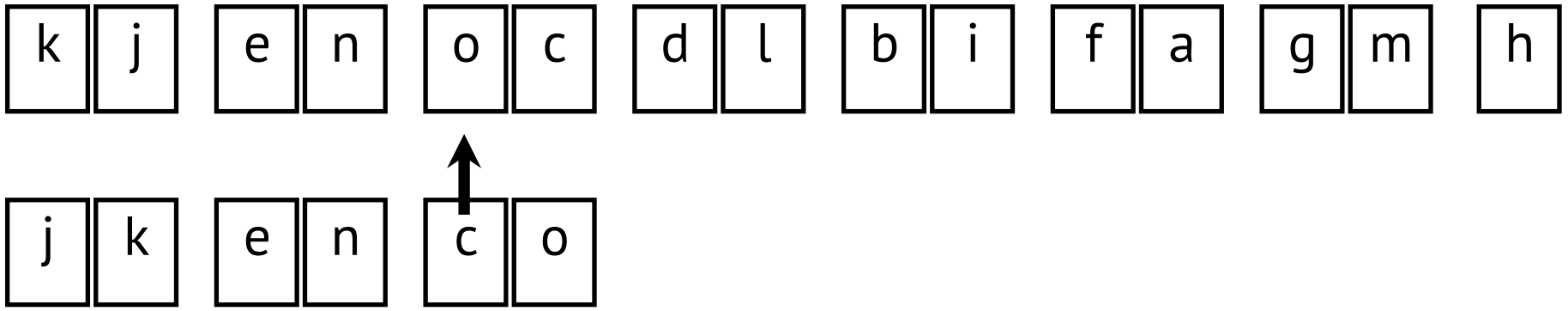
Merge-Sort (iterativ): Beispiel



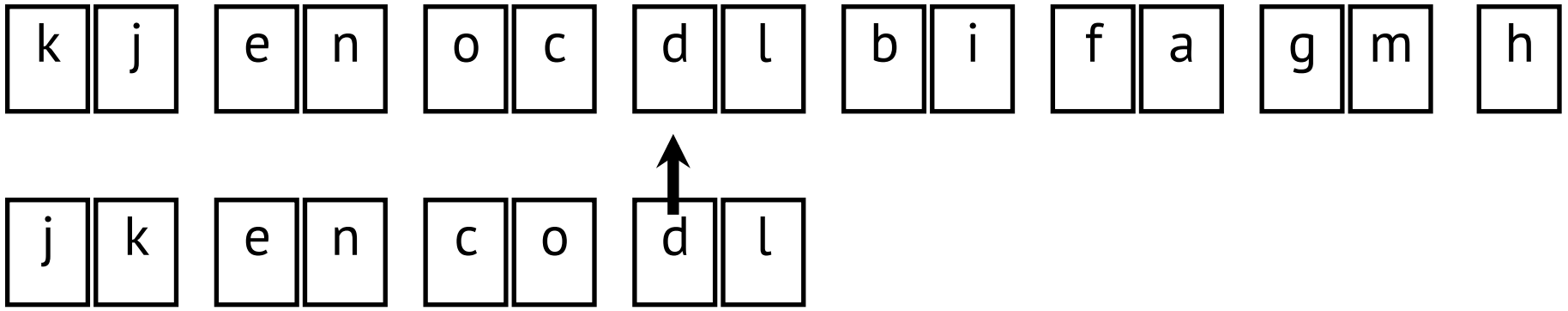
Merge-Sort (iterativ): Beispiel



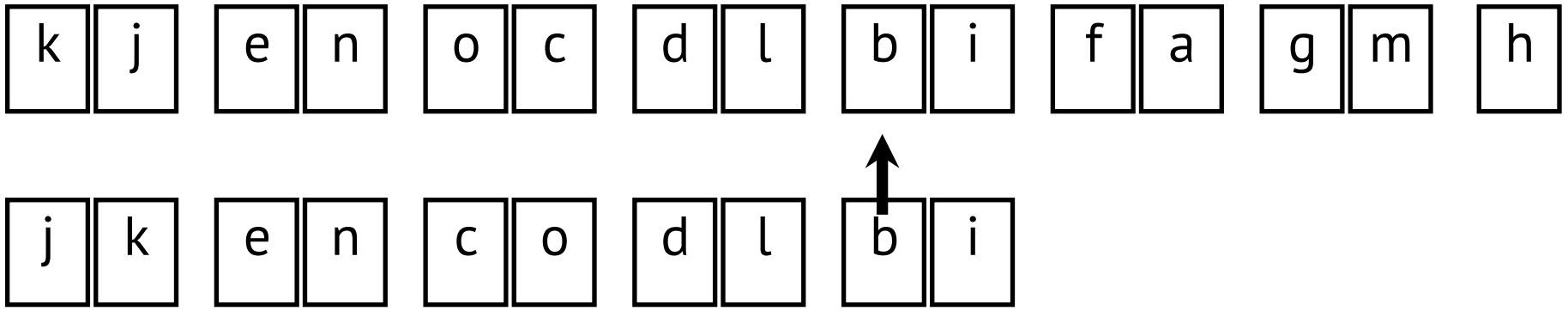
Merge-Sort (iterativ): Beispiel



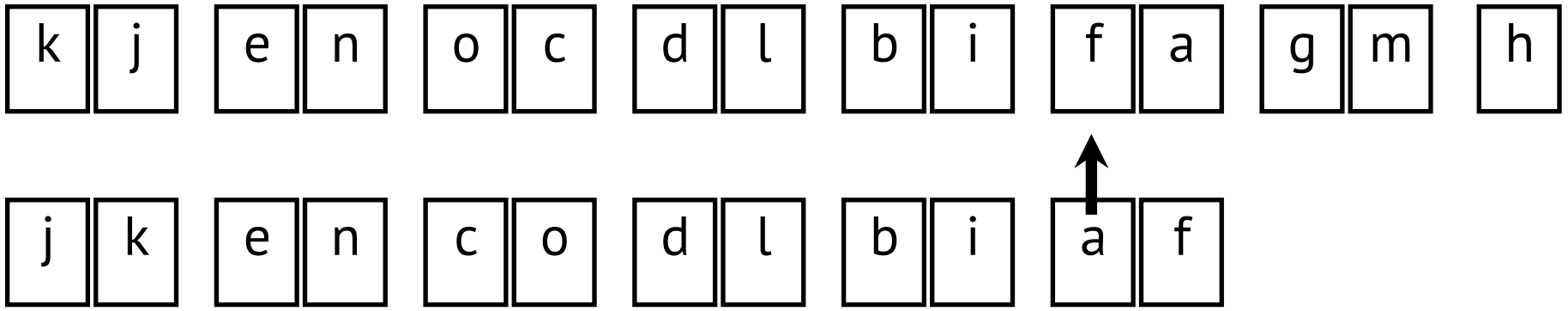
Merge-Sort (iterativ): Beispiel



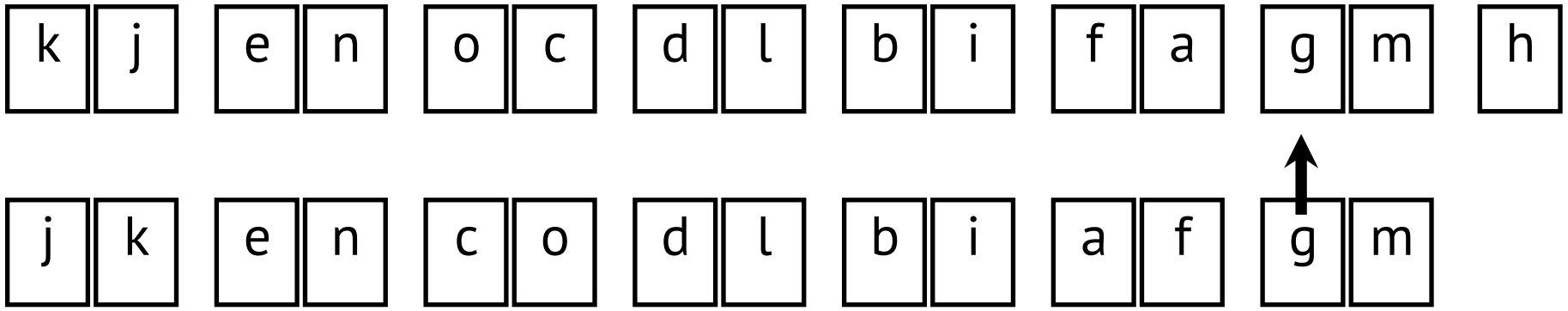
Merge-Sort (iterativ): Beispiel



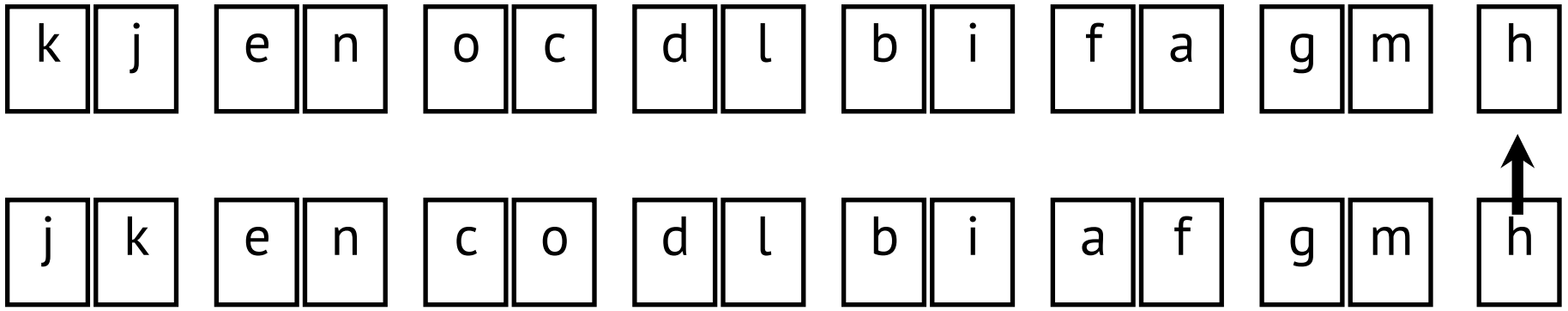
Merge-Sort (iterativ): Beispiel



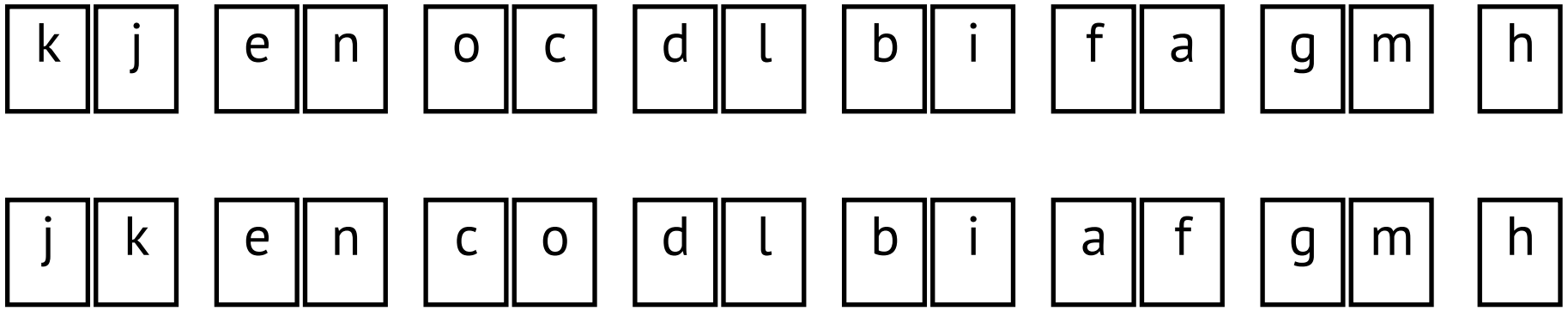
Merge-Sort (iterativ): Beispiel



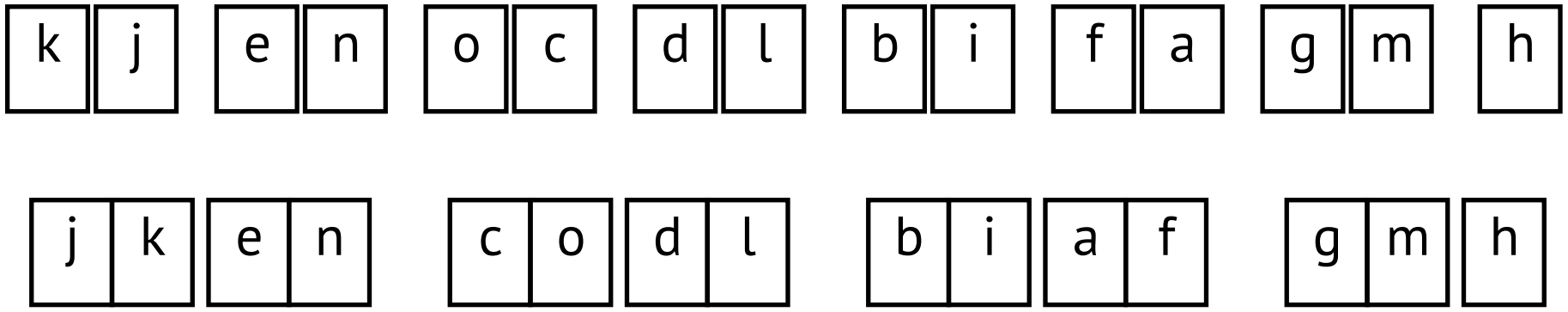
Merge-Sort (iterativ): Beispiel



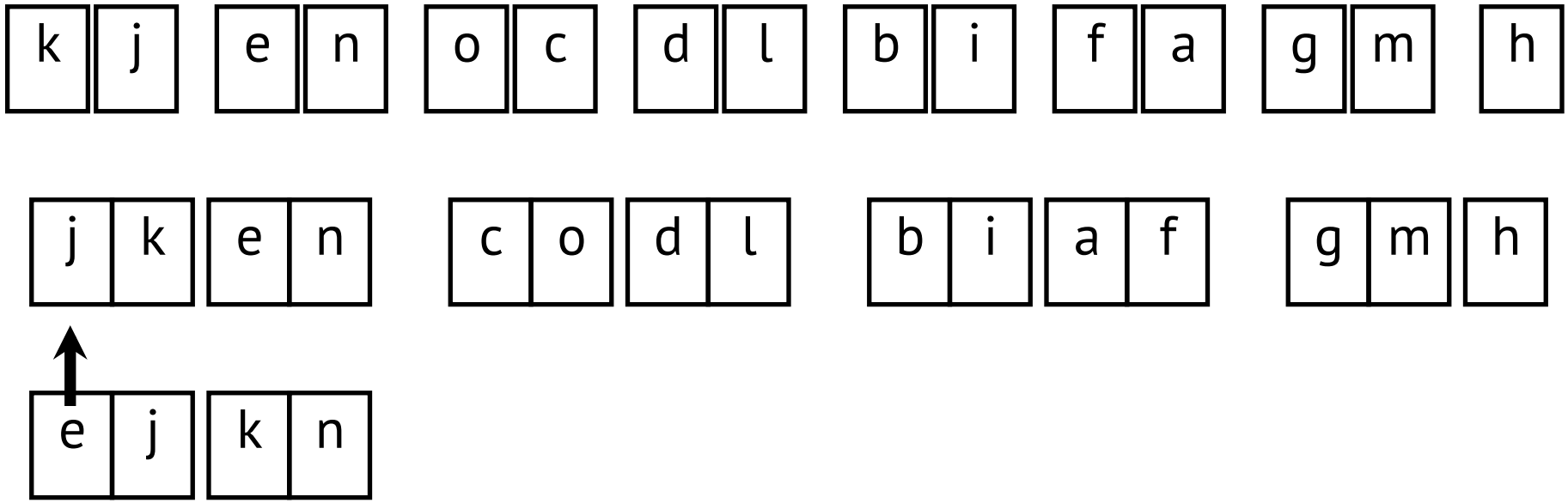
Merge-Sort (iterativ): Beispiel



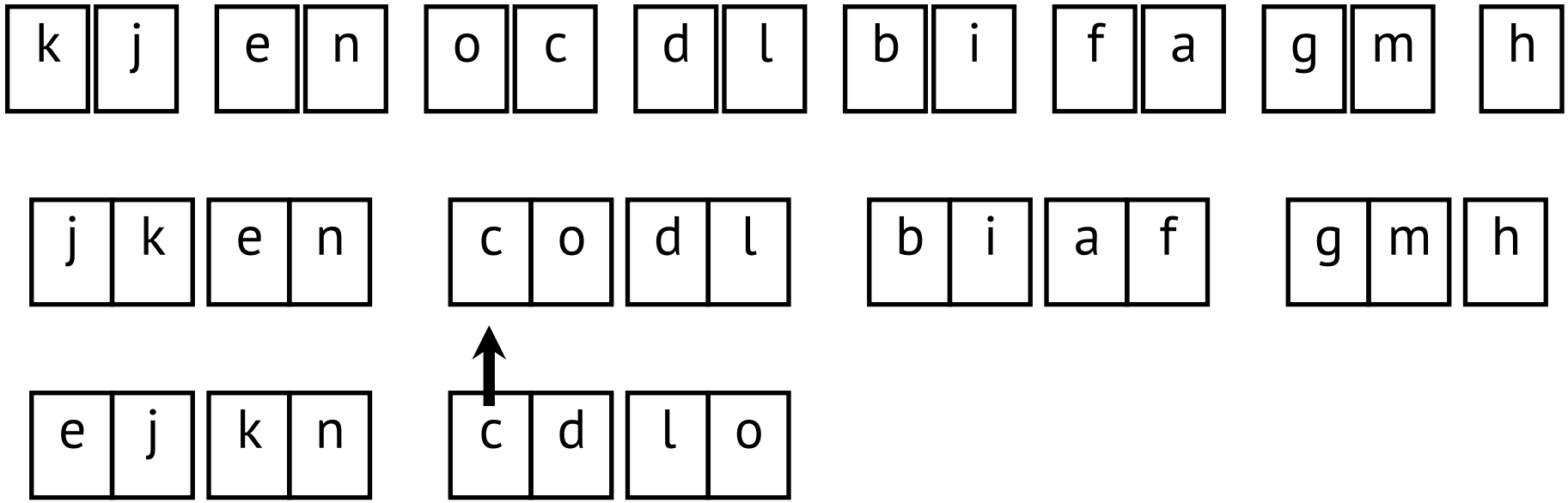
Merge-Sort (iterativ): Beispiel



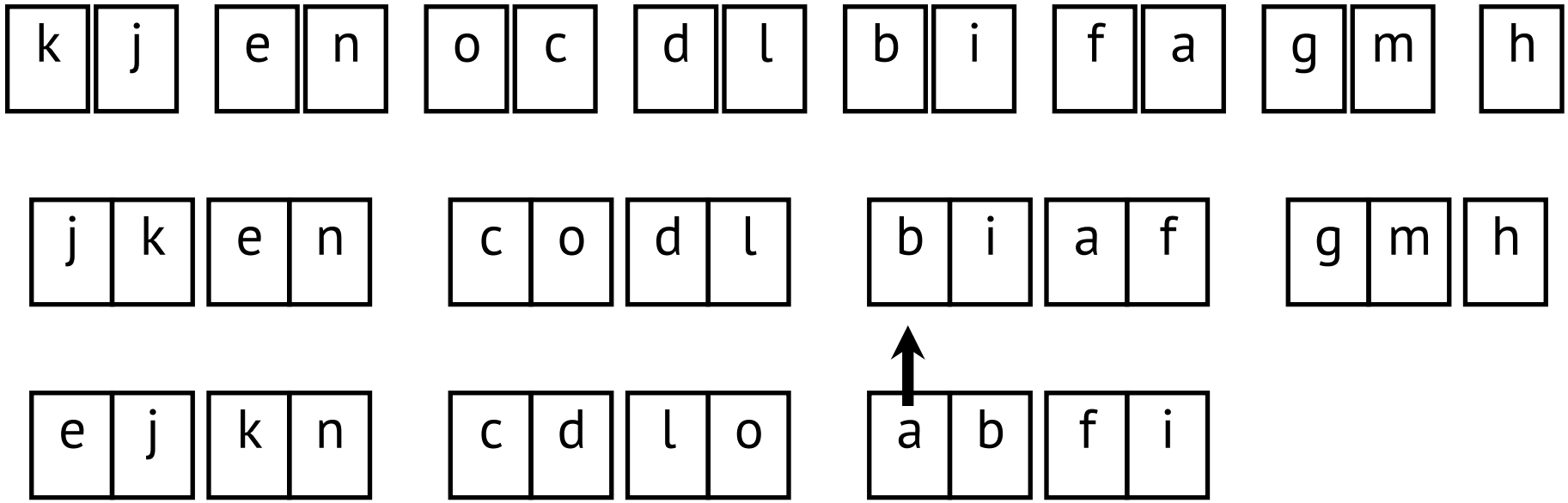
Merge-Sort (iterativ): Beispiel



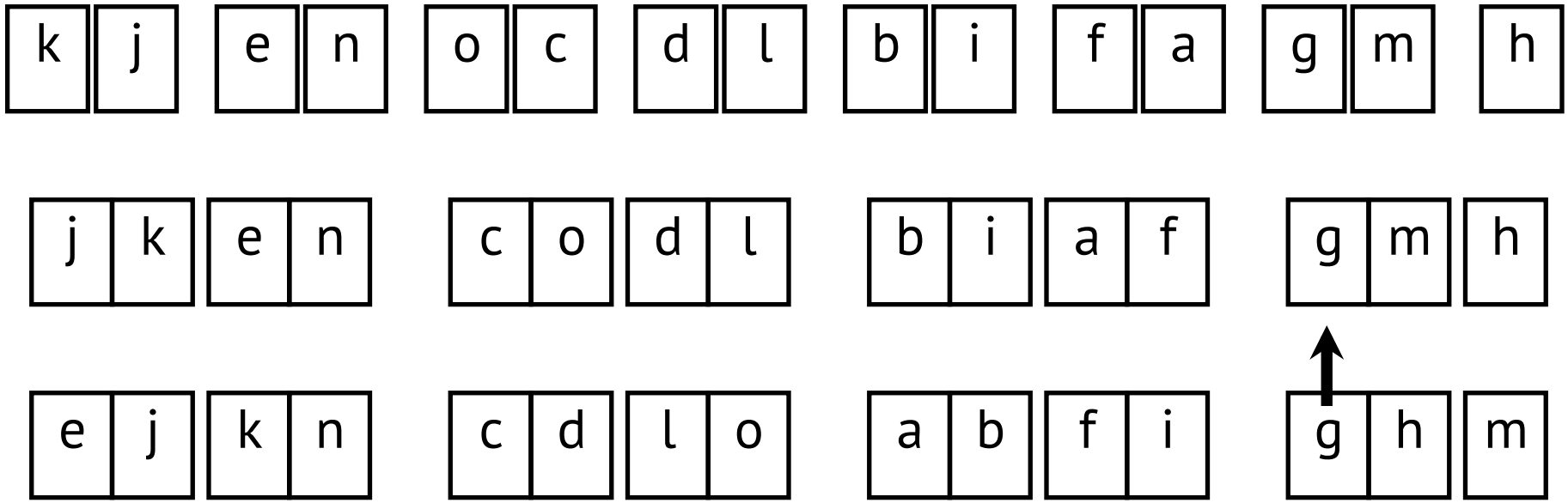
Merge-Sort (iterativ): Beispiel



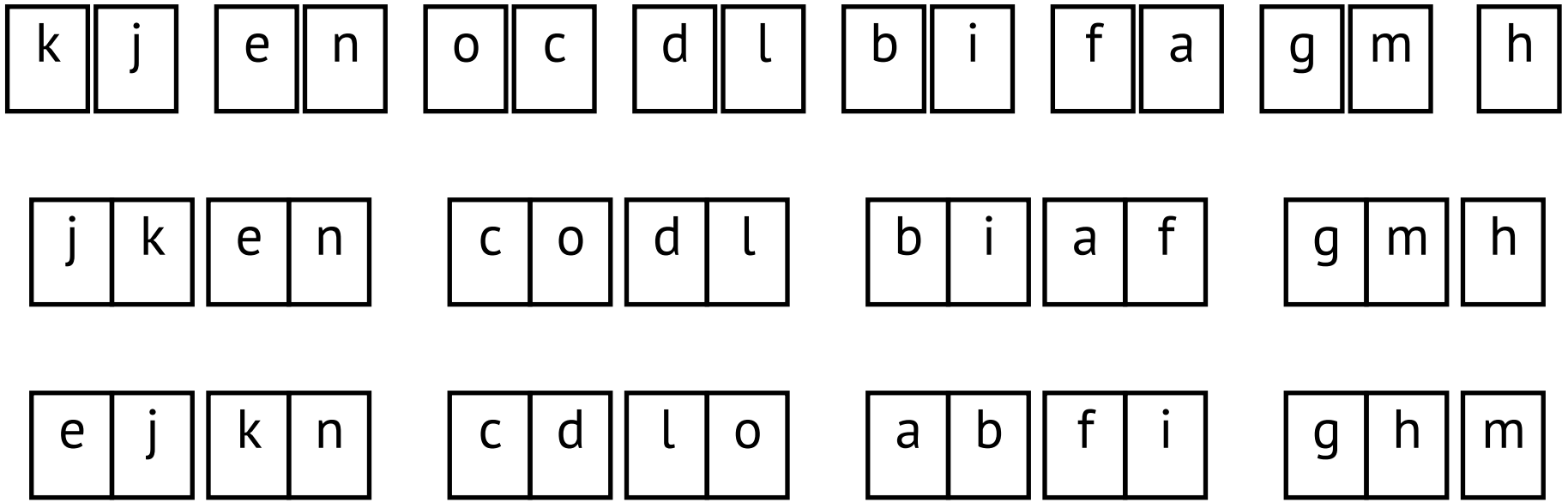
Merge-Sort (iterativ): Beispiel



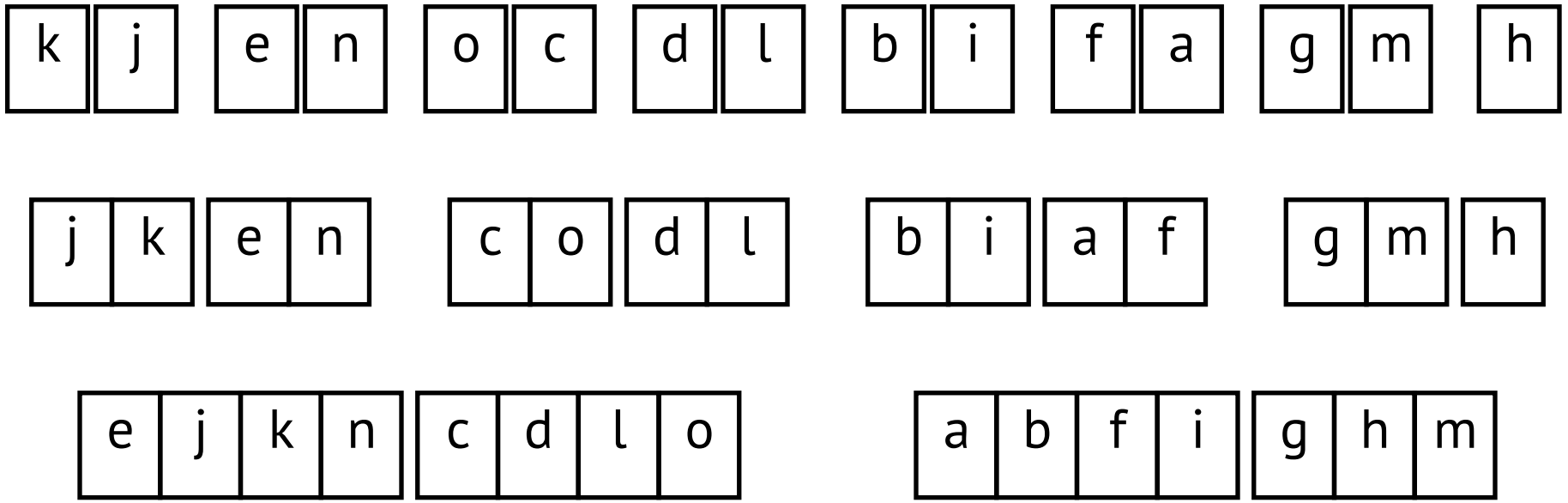
Merge-Sort (iterativ): Beispiel



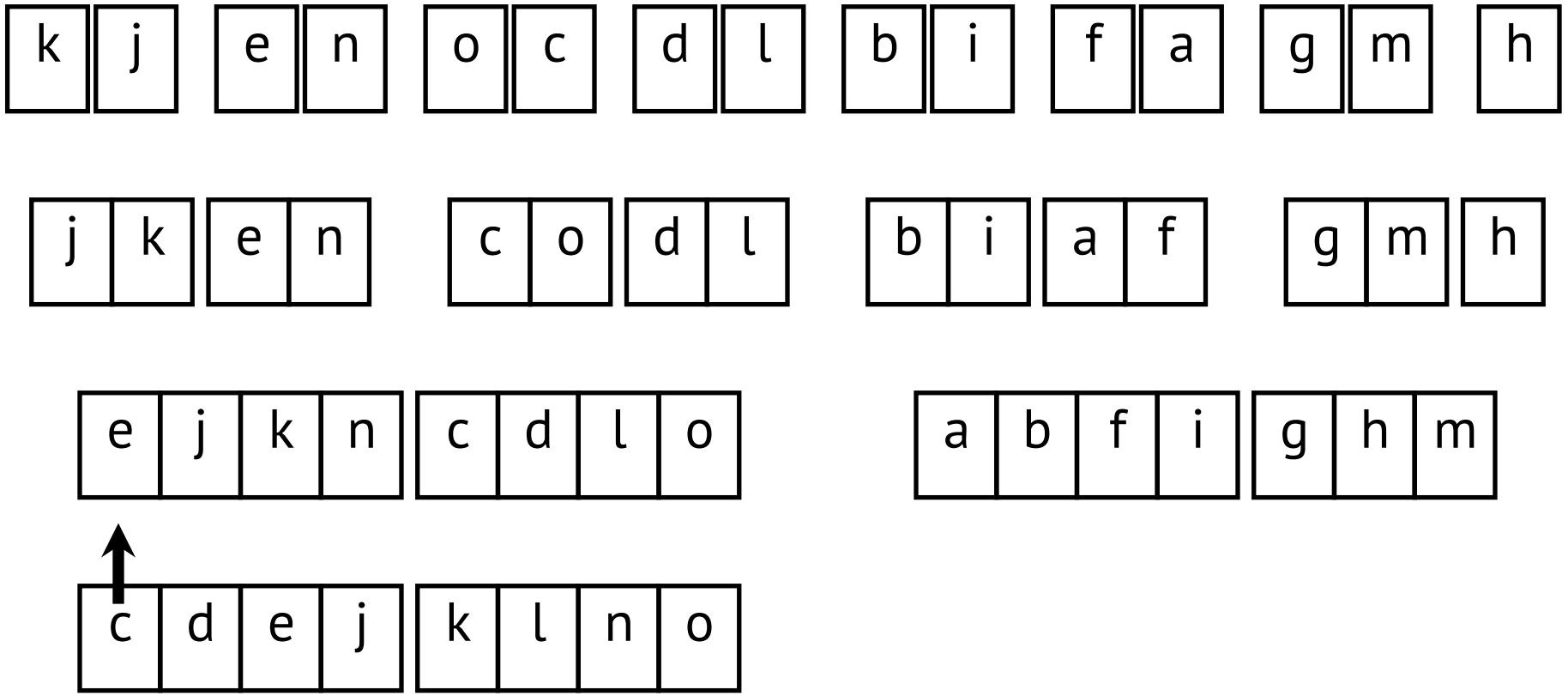
Merge-Sort (iterativ): Beispiel



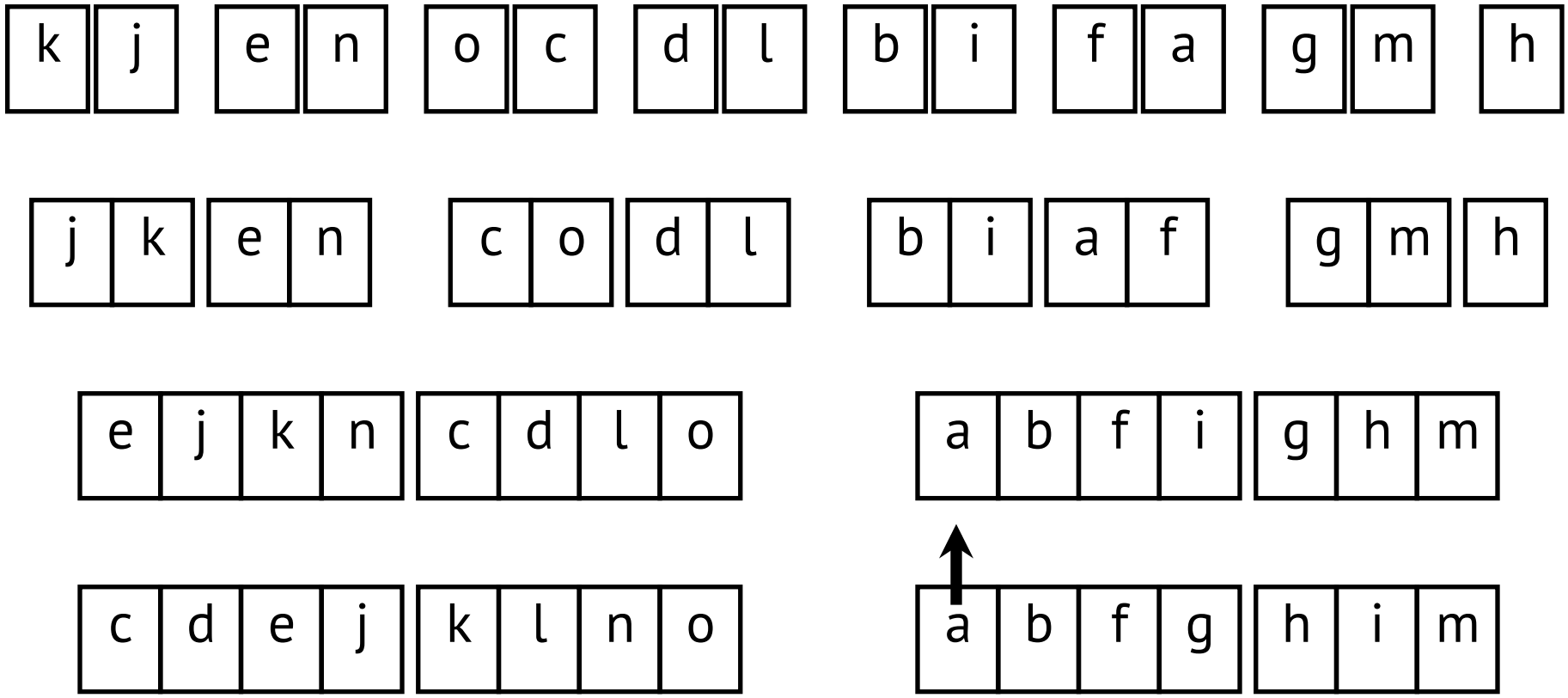
Merge-Sort (iterativ): Beispiel



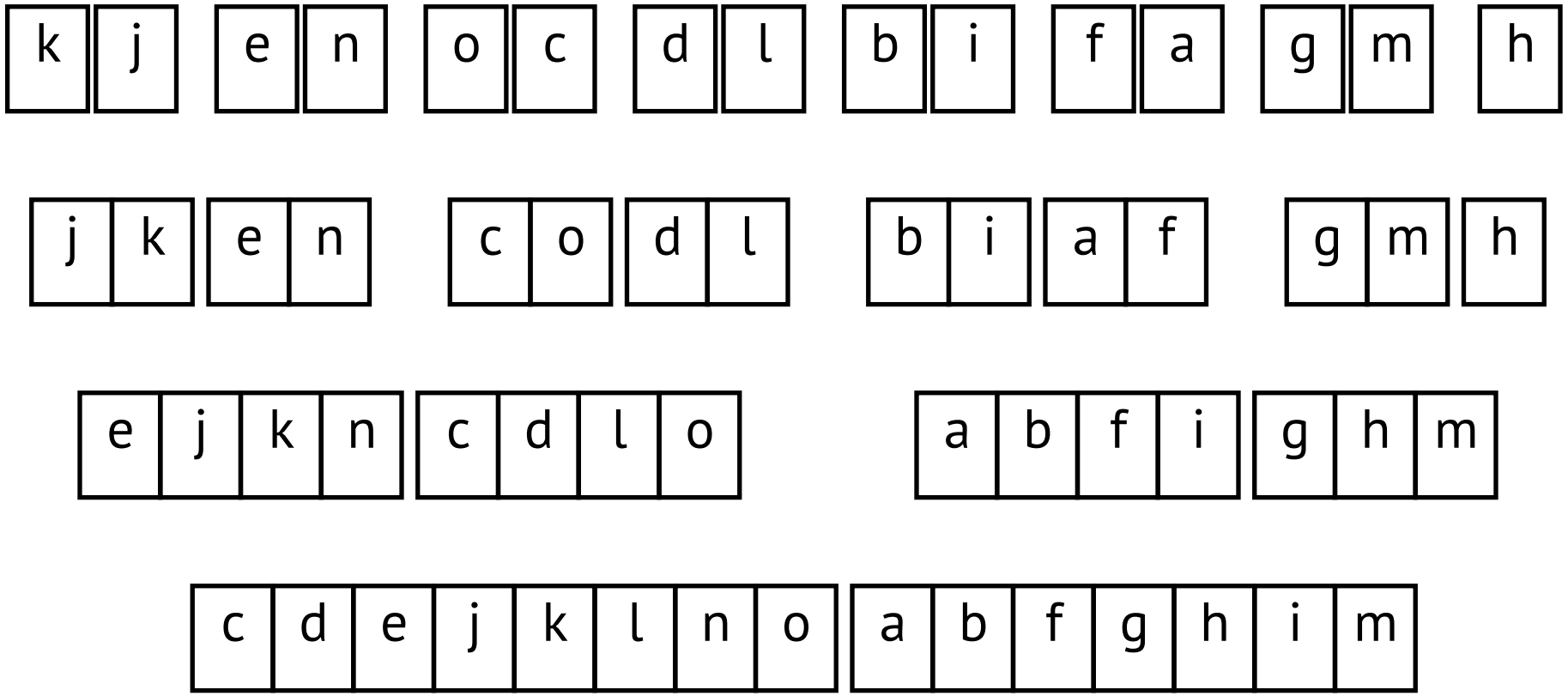
Merge-Sort (iterativ): Beispiel



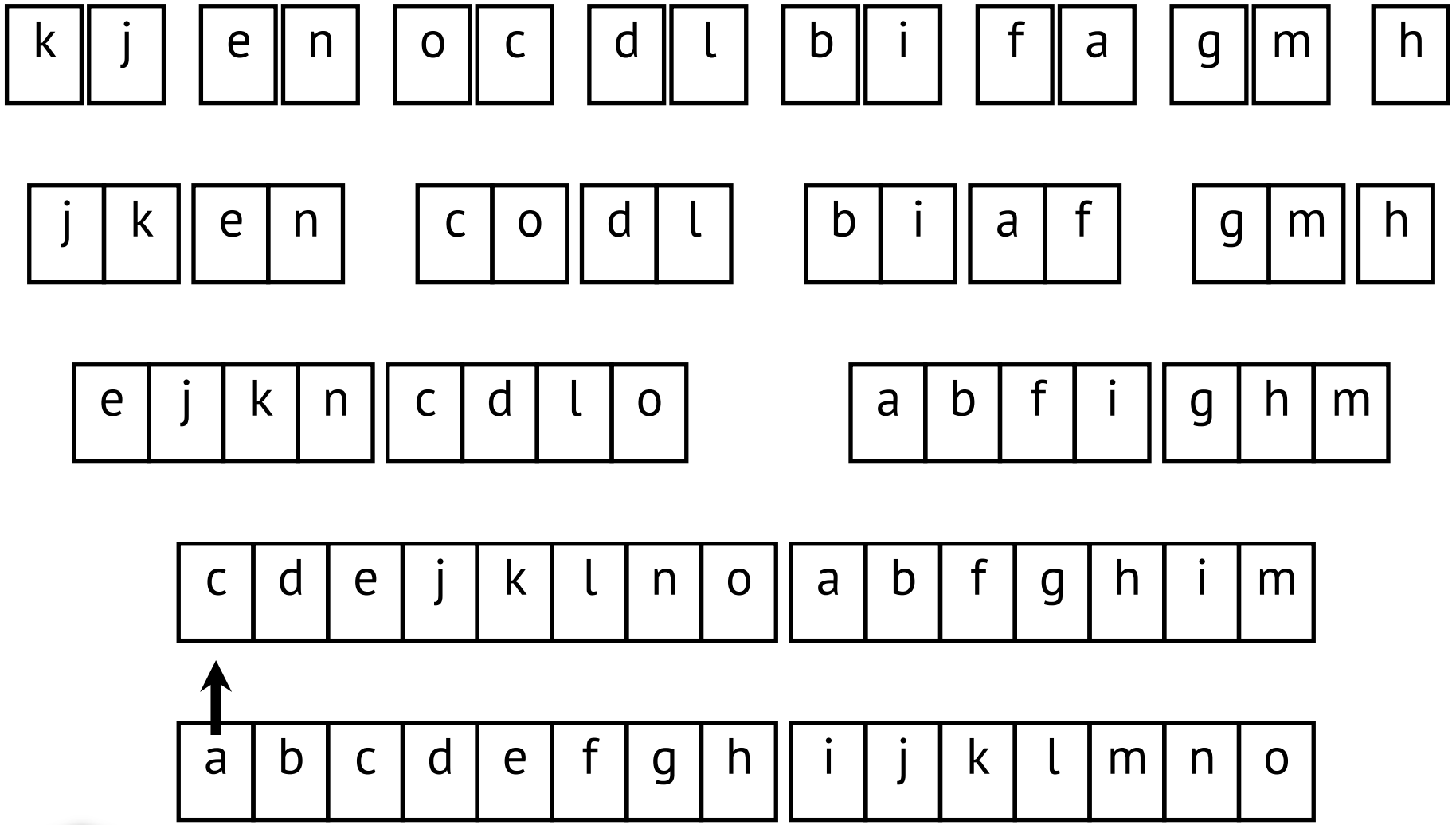
Merge-Sort (iterativ): Beispiel



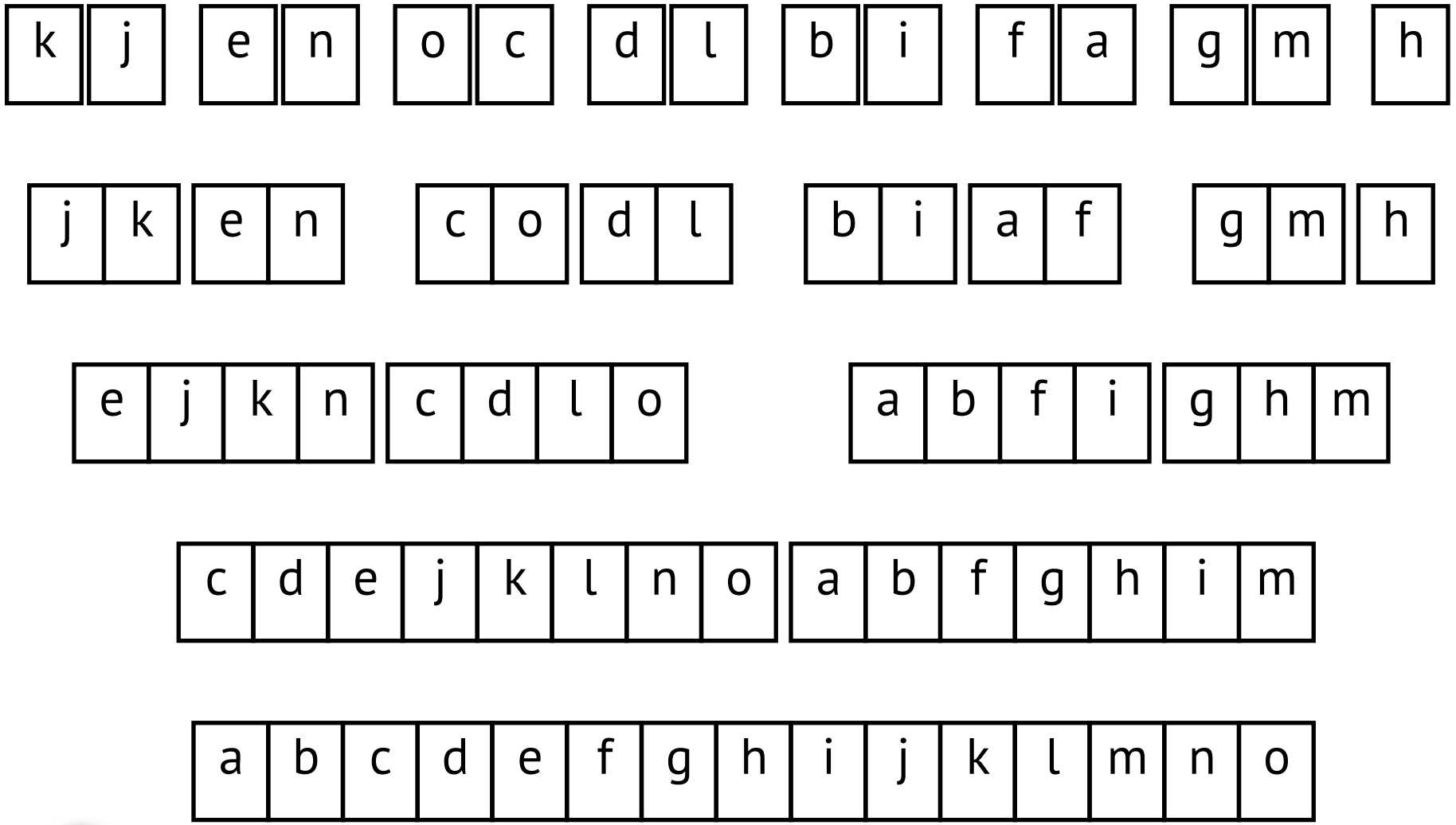
Merge-Sort (iterativ): Beispiel



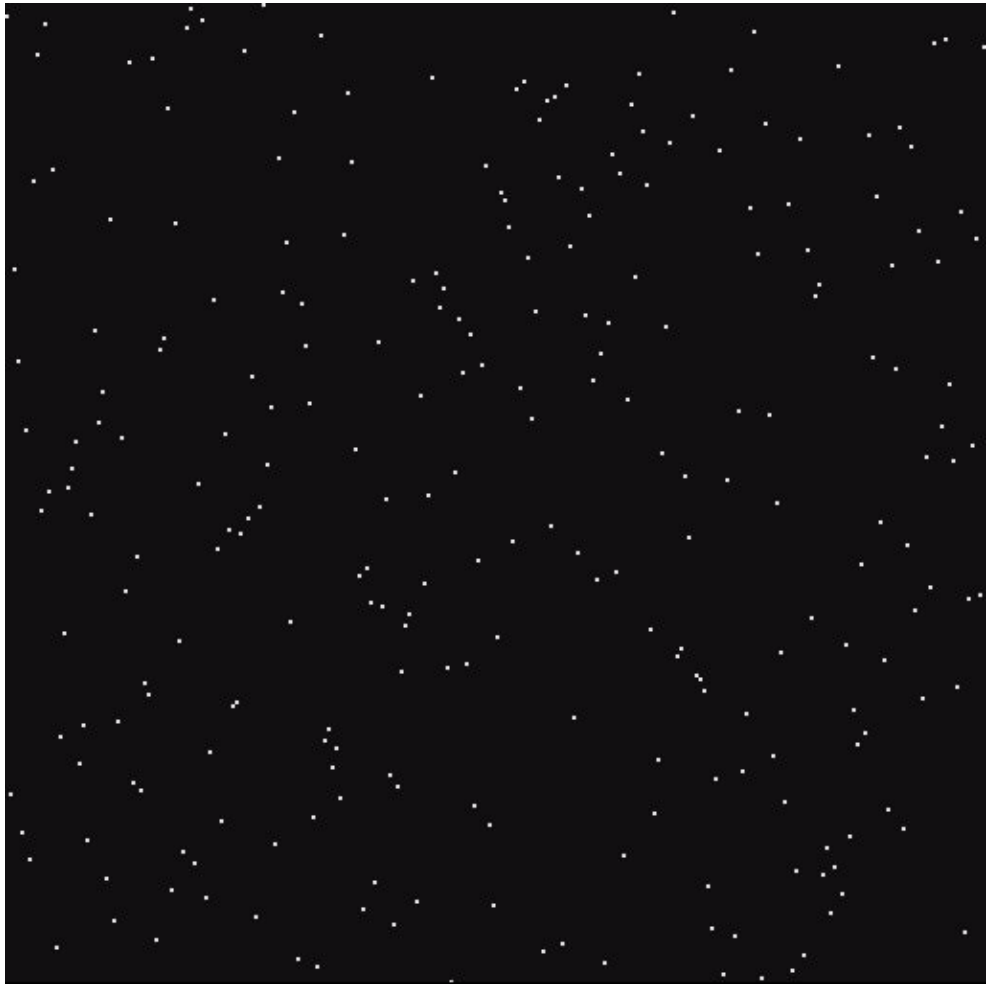
Merge-Sort (iterativ): Beispiel



Merge-Sort (iterativ): Beispiel



Merge-Sort (iterativ)



Heap-Sort

- Idee
  - Bei Selection-Sort entsteht der $O(n^2)$-Aufwand durch die lineare Suche nach dem kleinsten (oder größten) Element in der Rest-Liste.
  - Mit einer Prioritätsschlange (Heap) kann man das kleinste Element schneller finden ($O(\log n)$-Aufwand zum Wiederherstellen der Heap-Bedingung).

Heap-Definition (reprise)

- (Links-)vollständiger Binärbaum in Array-Darstellung
  - Vaterknoten A[i] hat die Nachfolger A[2×i] und A[2×i+1]
- Heap-Bedingung
  - Node ⩾ max(Left-Subtree) und Node ⩾ max(Right-Subtree)
  - A[i] ⩾ A[2×i] und A[i] ⩾ A[2×i+1]

Heap-Definition (reprise)

- (Links-)vollständiger Binärbaum in Array-Darstellung
  - Vaterknoten A[i] hat die Nachfolger A[2×i] und A[2×i+1]
- Heap-Bedingung
  - Node ⩾ max(Left-Subtree) und Node ⩾ max(Right-Subtree)
  - $A[\lfloor i/2 \rfloor] \geqslant A[\ i\ ]$

Heap-Definition (reprise)

- Ein Array A hat die Heap-Eigenschaft ab Index p, wenn $A[\lfloor i/2 \rfloor] \geqslant A[\ i\ ]$ ab Index $2 \times p$ gilt (weil dann $\lfloor i/2 \rfloor = p$).
- Insbesondere: jedes beliebige Array $A[1..n]$ hat die Heap-Eigenschaft ab Index $\lfloor n/2 \rfloor + 1$.

Heap-Sort

- Konvertiere ein beliebiges Array in einen Heap ($A[\lfloor i/2 \rfloor] \geqslant A[\, i \,]$).
- Entferne sukzessive das Top-Element und stelle die Heap-Eigenschaft wieder her (vgl. Deq()-Operation für Prioritäts-schlangen in Abschnitt 1.4).
- Heap liefert sortierte Elemente.

Heap-Sort

- Sei A[1..n] ein Heap,Top-Element A[1]
- Nach dem Entfernen des Top-Elementes wird A[n] nach A[1] kopiert und durch „Versickern“ die Heap-Eigenschaft wieder hergestellt.
- Danach benutzt der Heap noch die Array-Elemente A[1..n−1].
- A[n] ist also frei und das entfernte Top-Element kann dort abgelegt werden.

Heap-Sort: Algorithmus

- ConvertToHeap(A[1..n])
  for i = $\lfloor n/2 \rfloor$ downto 1 do
    Sink(A[1..n],i)

- Wenn die Schleife bis i=p durchgelaufen ist, hat das Array A ab Index p die Heap-Eigenschaft.

Heap-Sort: Algorithmus

- Sink(A[1..n],i)

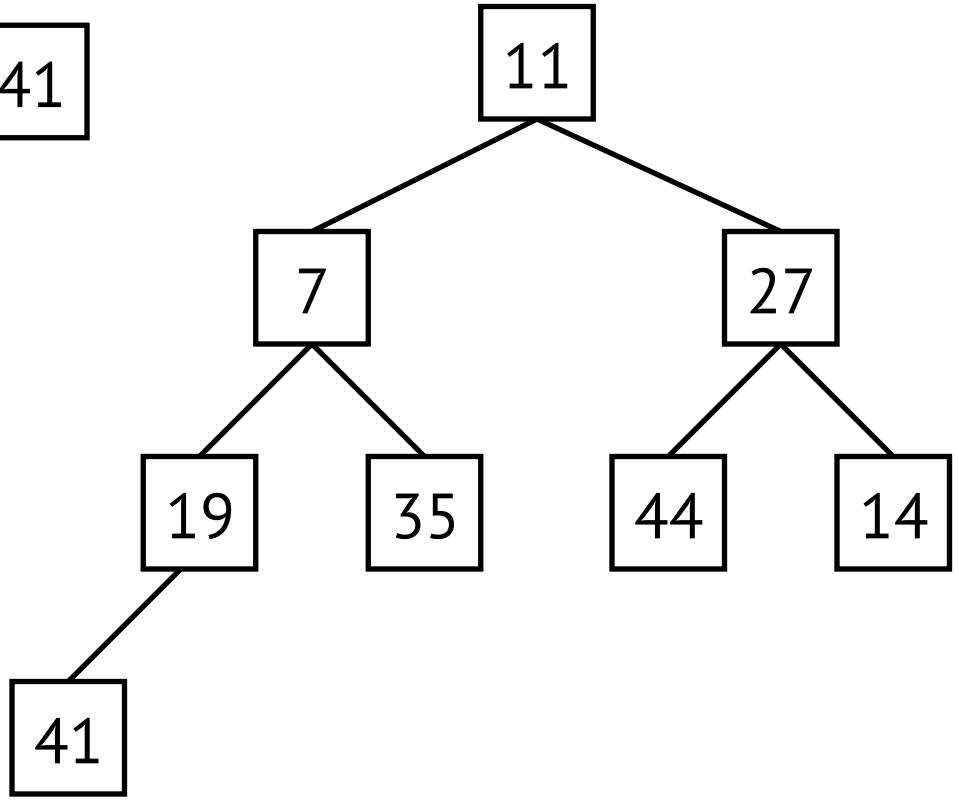
```
while i ≤ ⌊n/2⌋ do
    j ← 2*i
    if j < n and A[ j+1] > A[ j ] then
        j ← j+1
    if A[ j ] > A[ i ] then
        Swap(A[ j ],A[ i ])
        i ← j
    else
        i ← n
```

Heap-Sort: Algorithmus

- HeapSort(A[1..n])
  ConvertToHeap(A[1..n])
  for i ← n downto 1 do
    Swap(A[1],A[i])
    Sink(A[1..i−1],1)

Heap-Sort: Beispiel

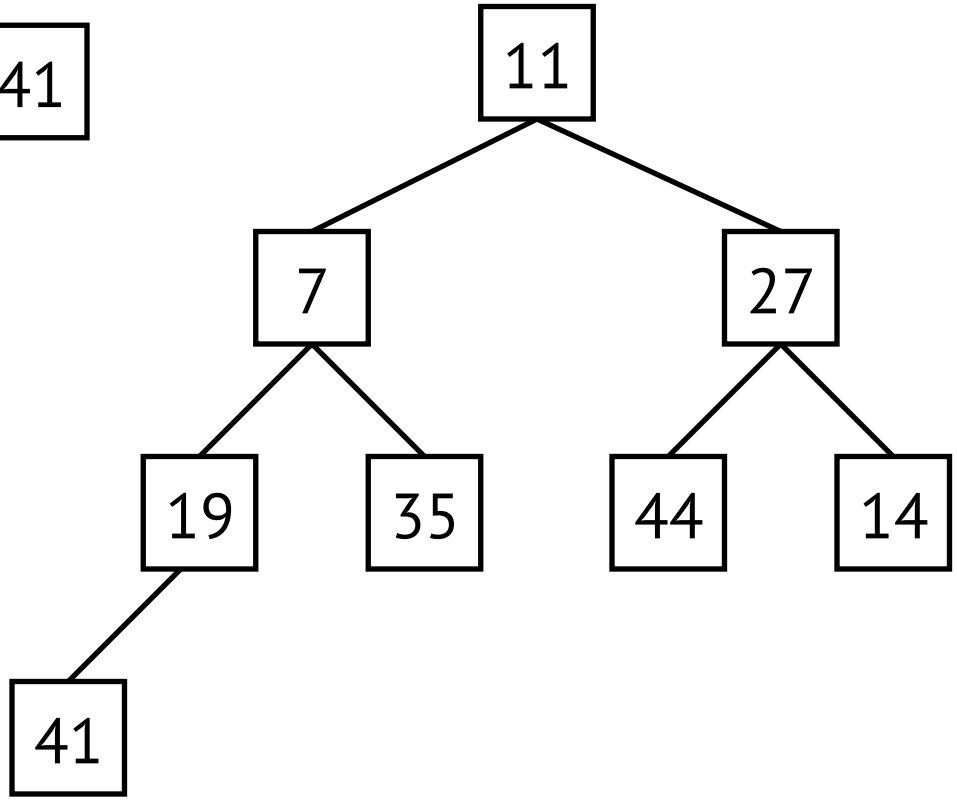
11	7	27	19	35	44	14	41



Heap-Sort: Beispiel

Phase I: Aufbau des Heaps

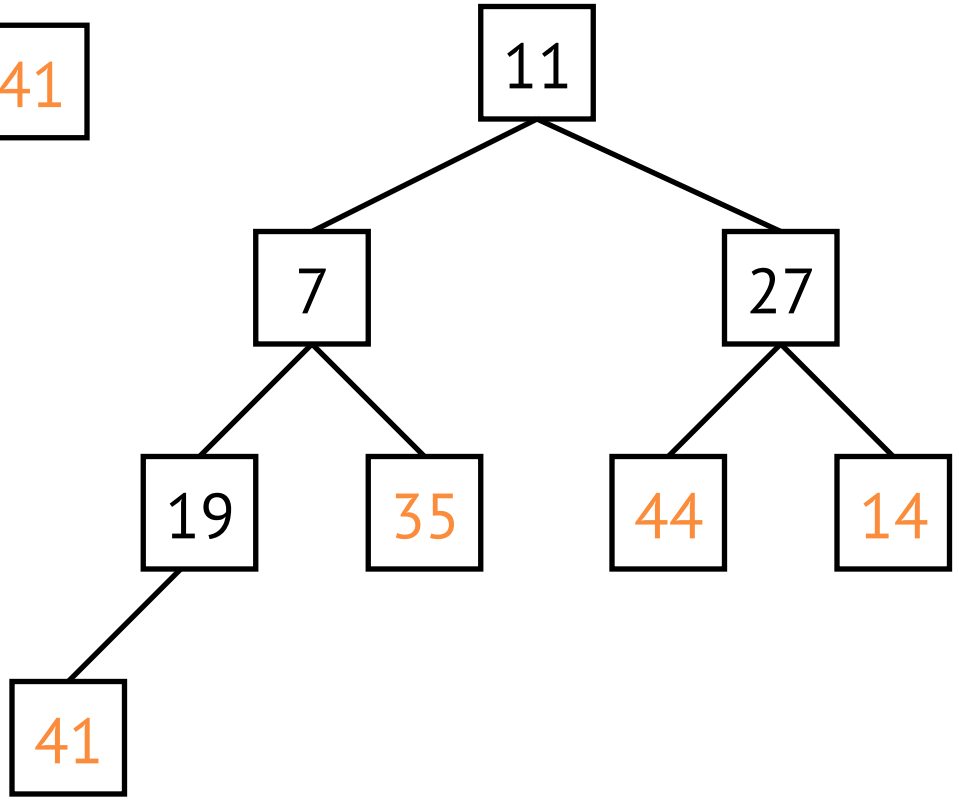
11	7	27	19	35	44	14	41



Heap-Sort: Beispiel

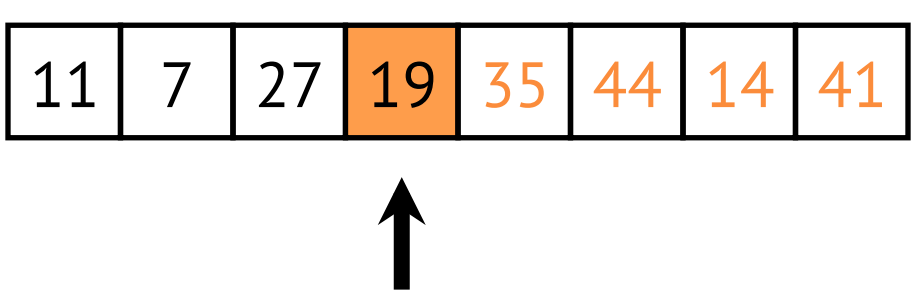
Phase I: Aufbau des Heaps

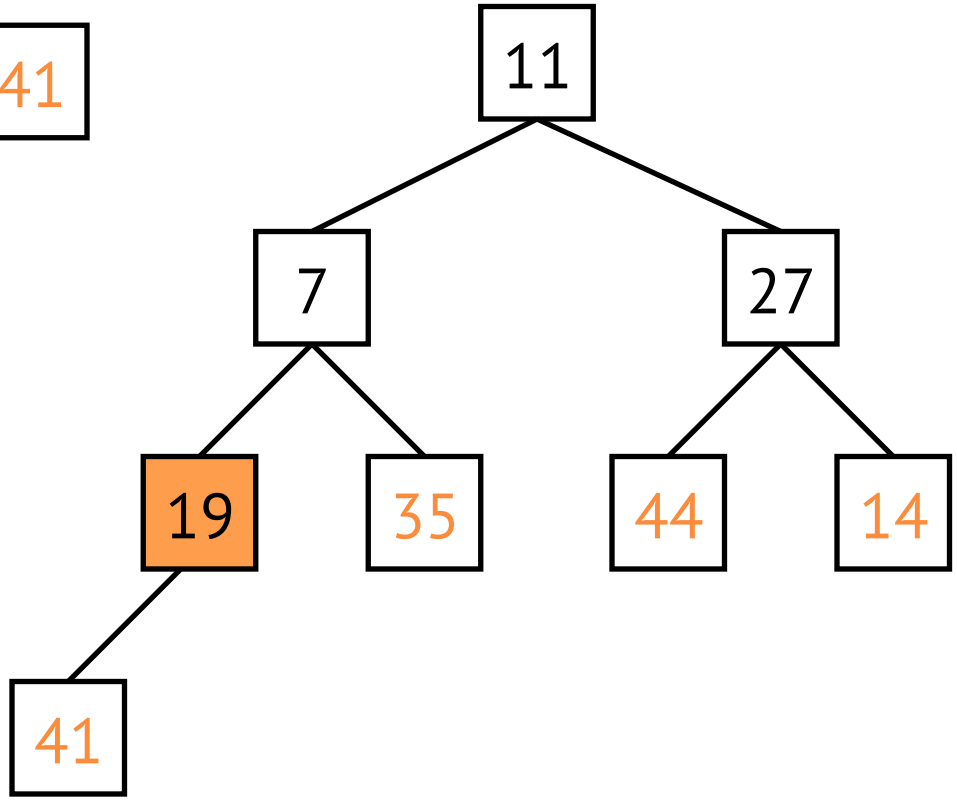
11	7	27	19	35	44	14	41



Heap-Sort: Beispiel

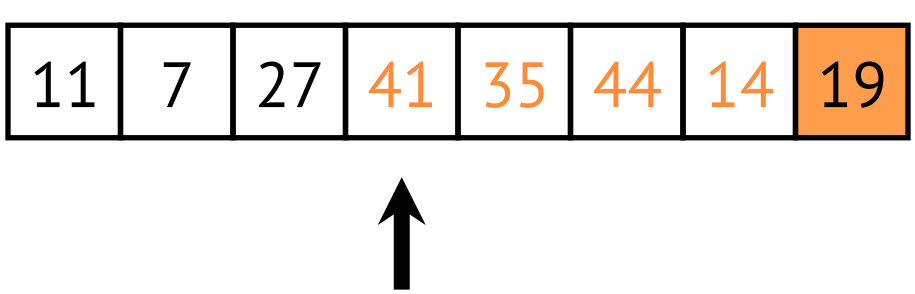
Phase I: Aufbau des Heaps

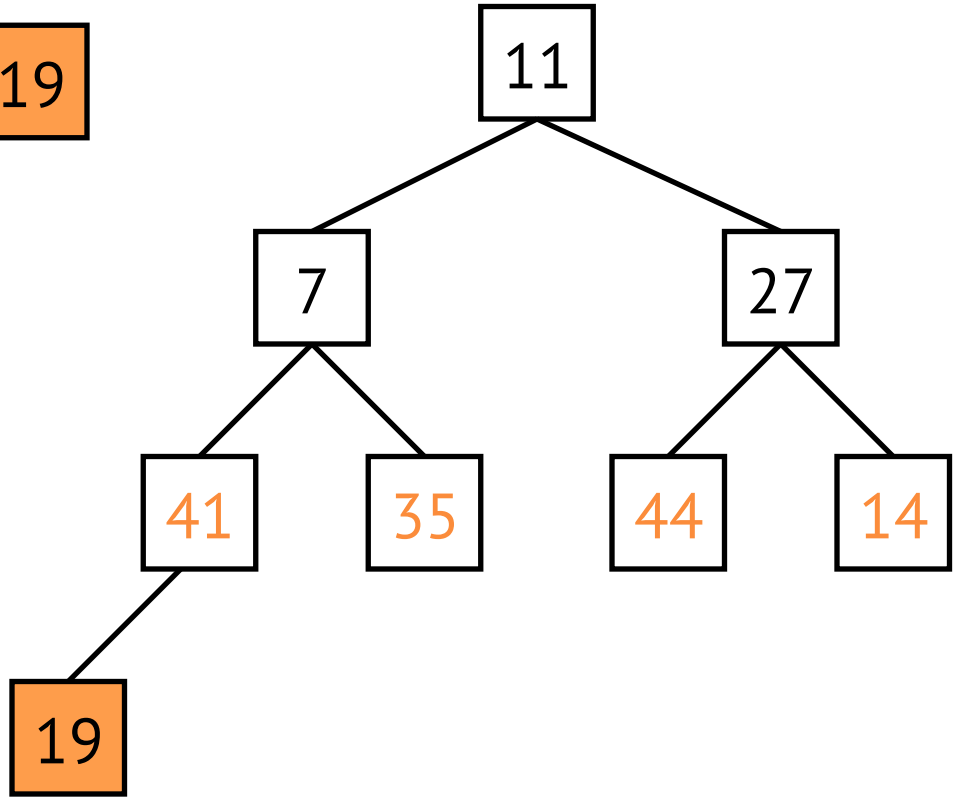




Heap-Sort: Beispiel

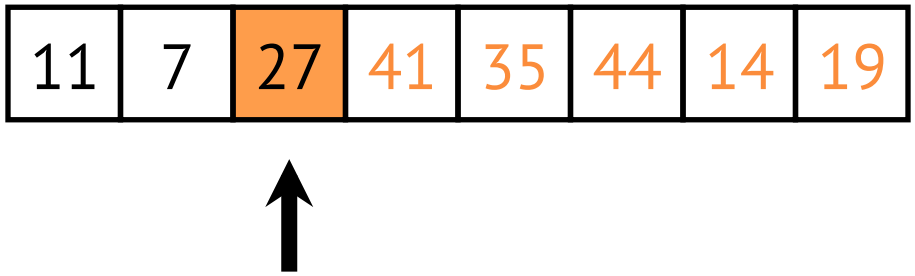
Phase I: Aufbau des Heaps

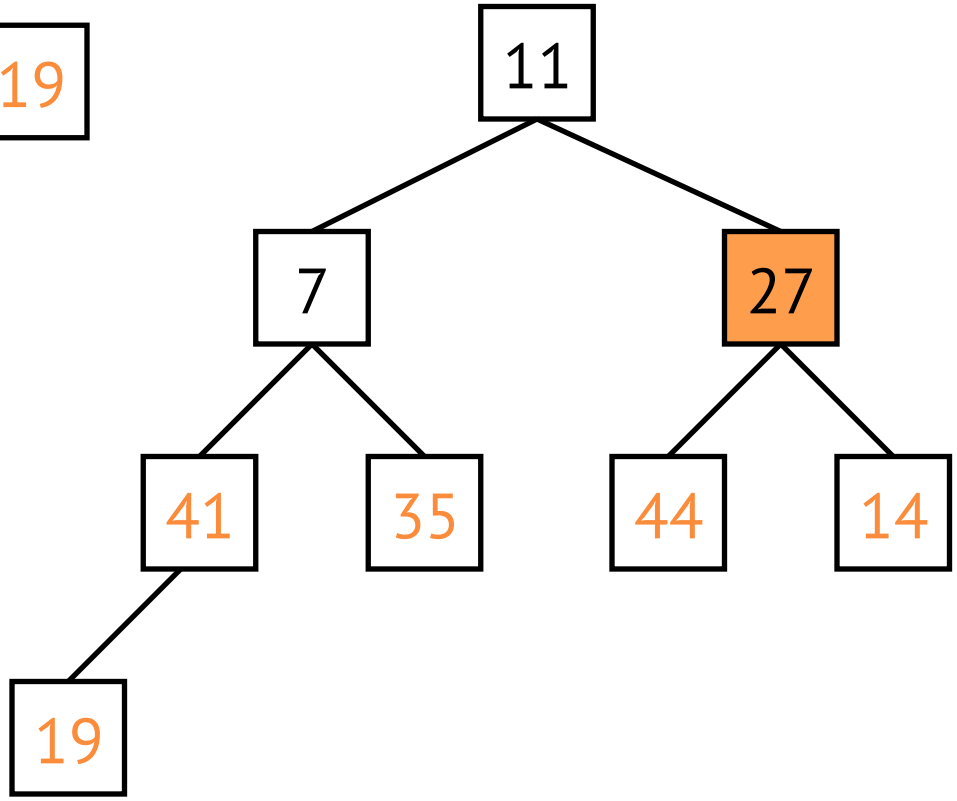




Heap-Sort: Beispiel

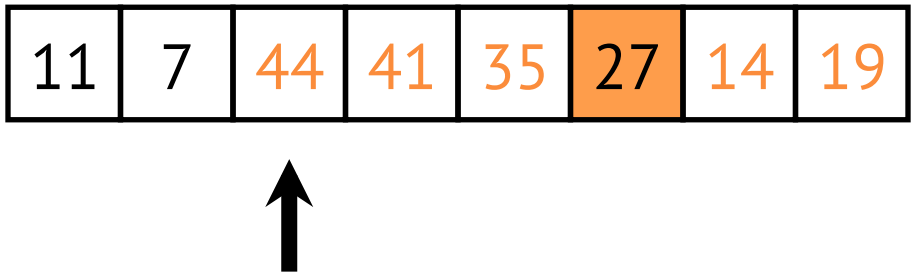
Phase I: Aufbau des Heaps

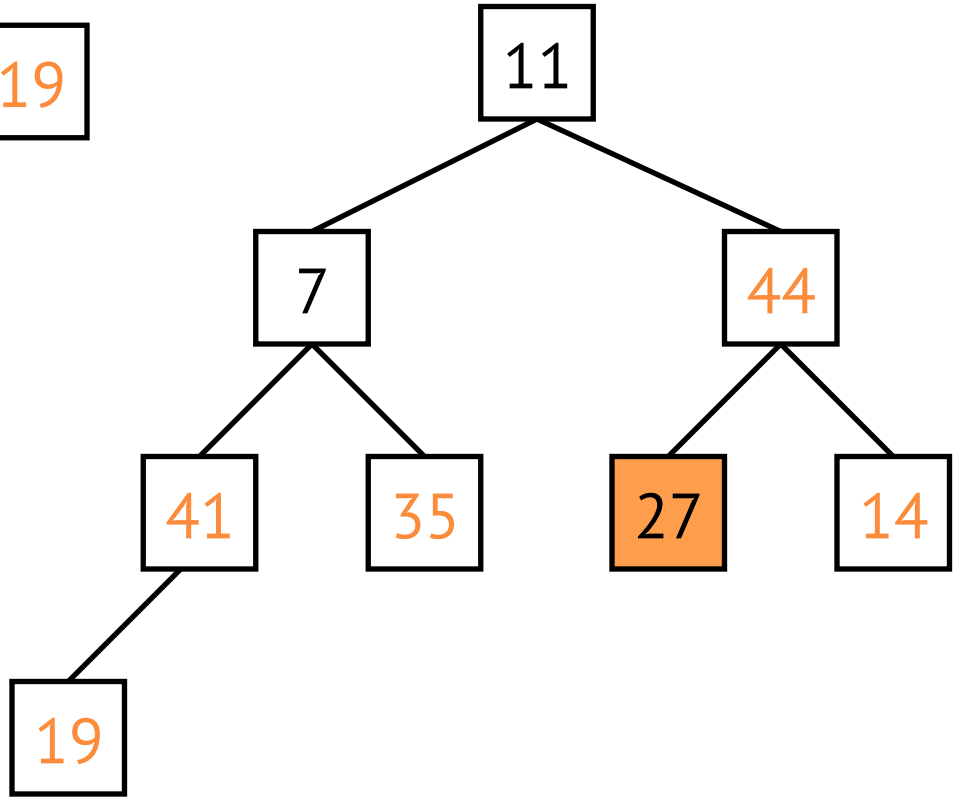




Heap-Sort: Beispiel

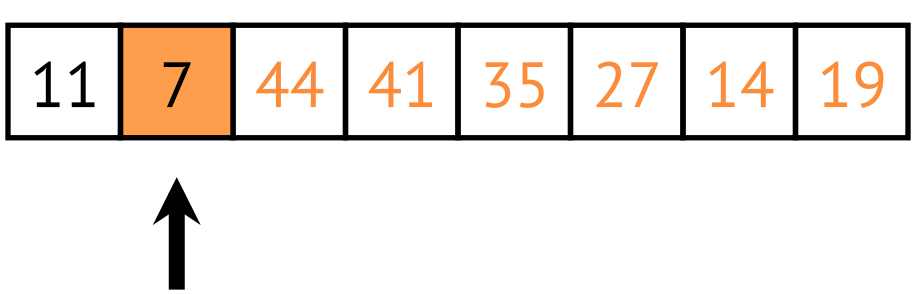
Phase I: Aufbau des Heaps

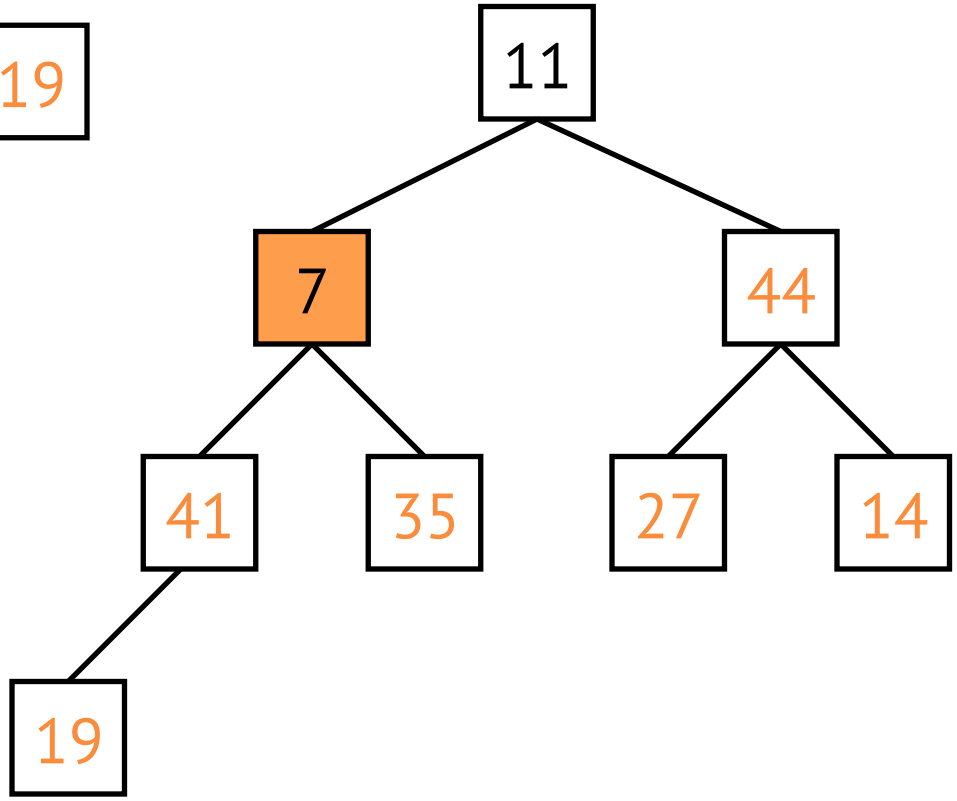




Heap-Sort: Beispiel

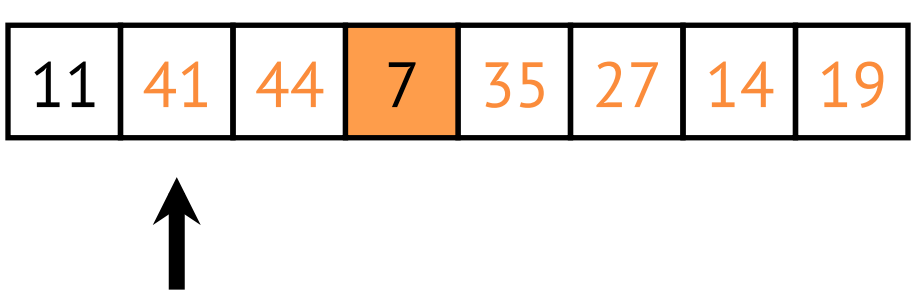
Phase I: Aufbau des Heaps

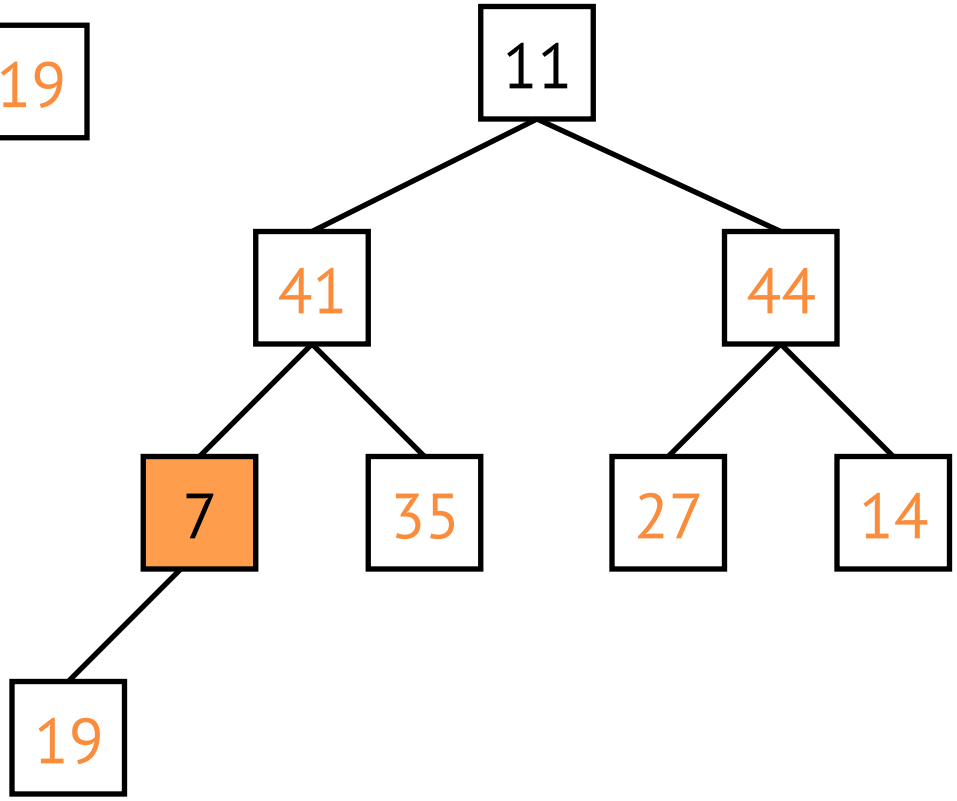




Heap-Sort: Beispiel

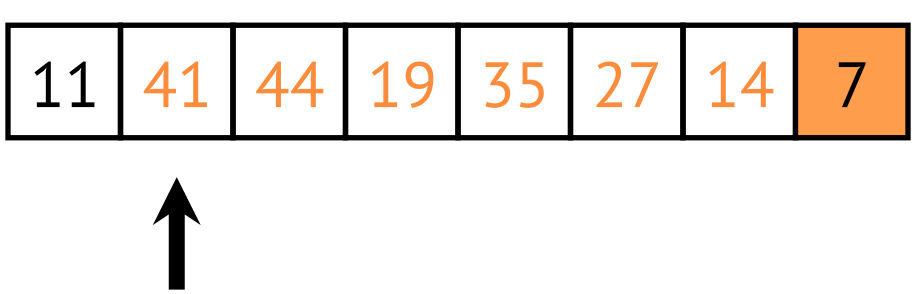
Phase I: Aufbau des Heaps

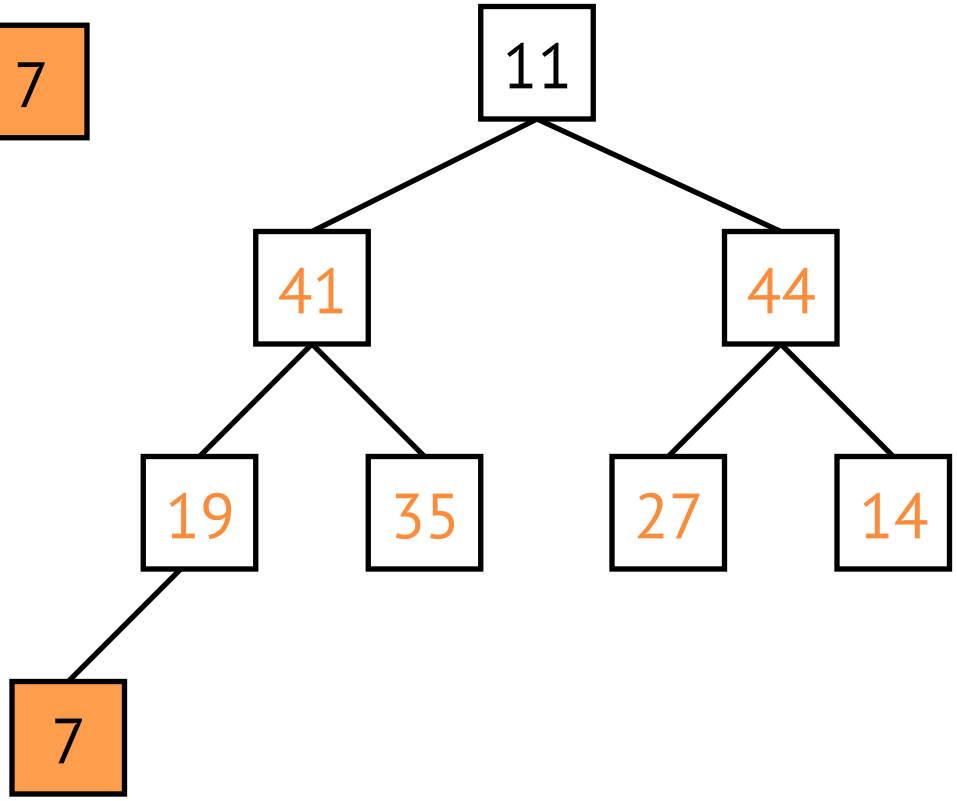




Heap-Sort: Beispiel

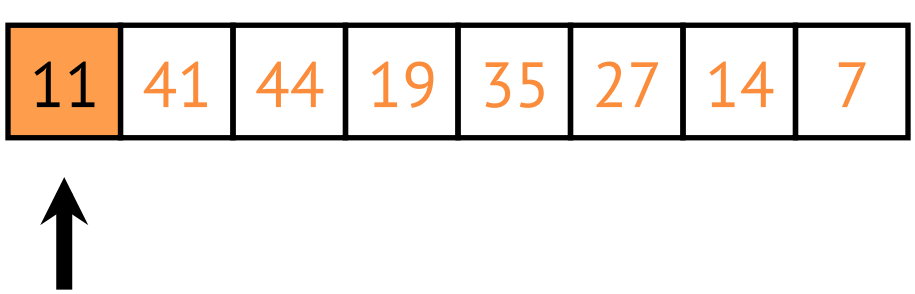
Phase I: Aufbau des Heaps

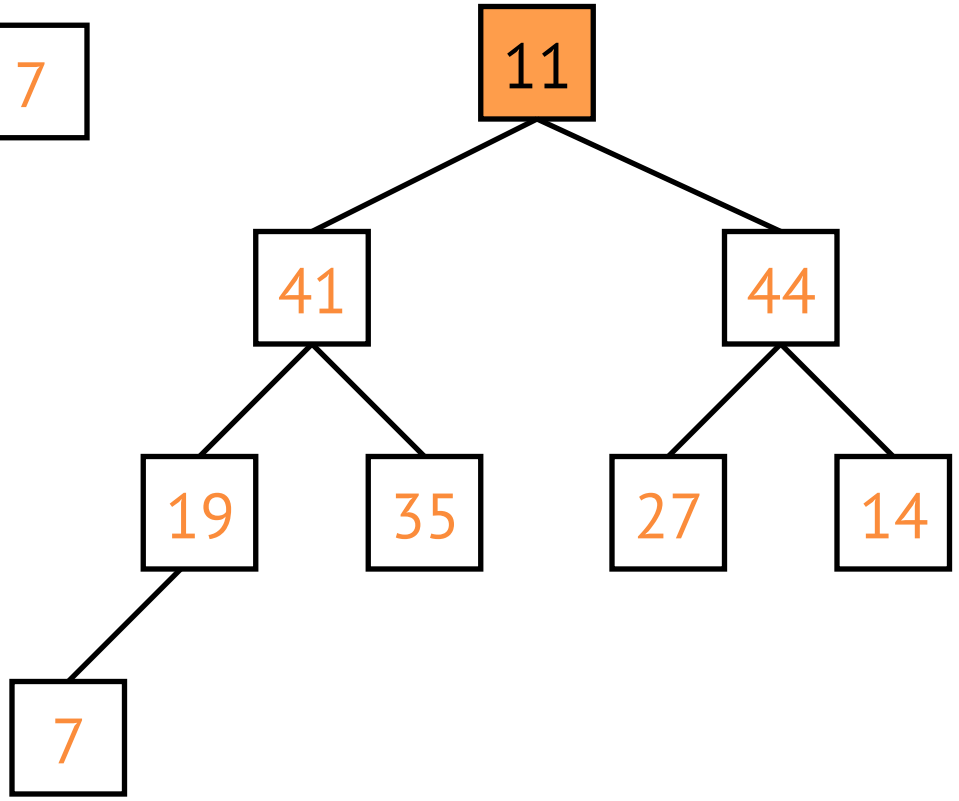




Heap-Sort: Beispiel

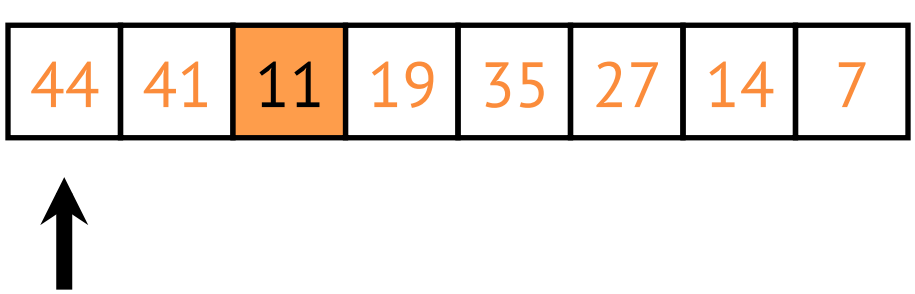
Phase I: Aufbau des Heaps

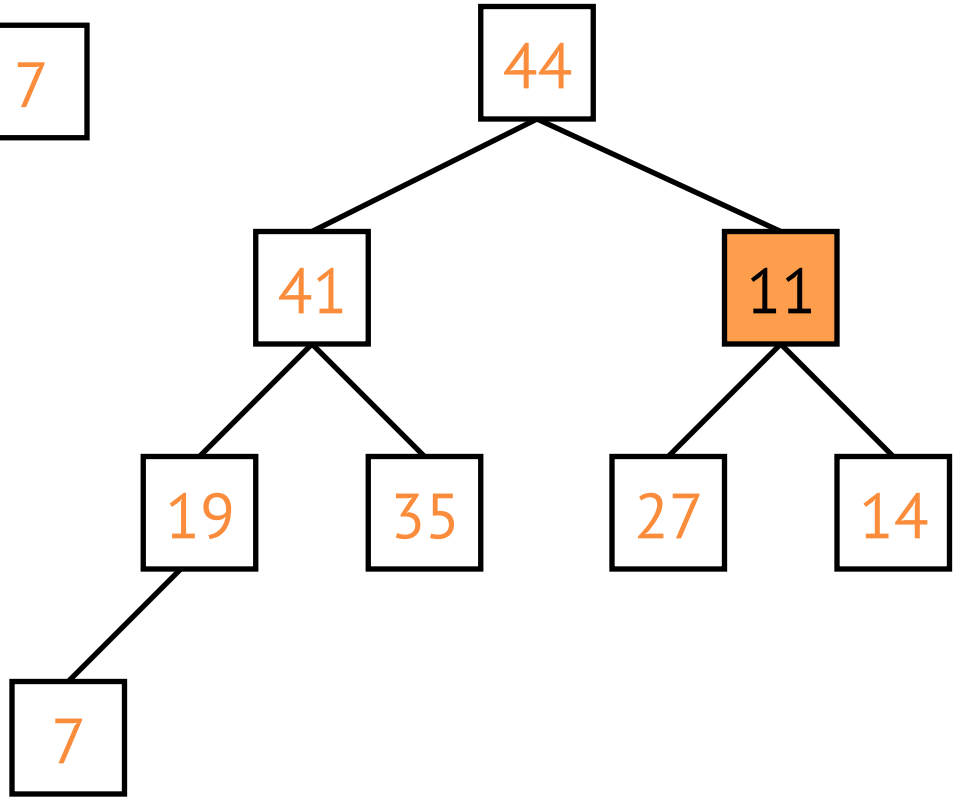




Heap-Sort: Beispiel

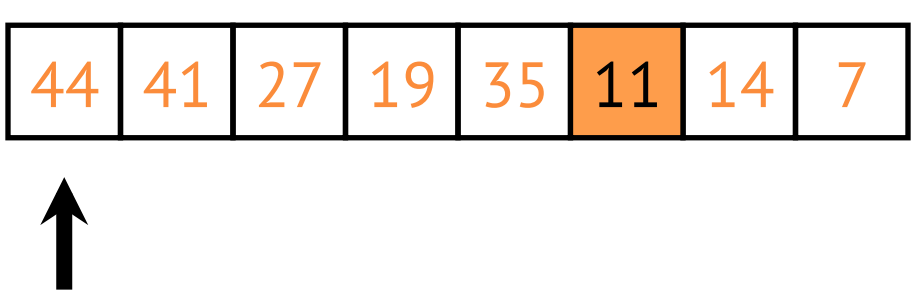
Phase I: Aufbau des Heaps

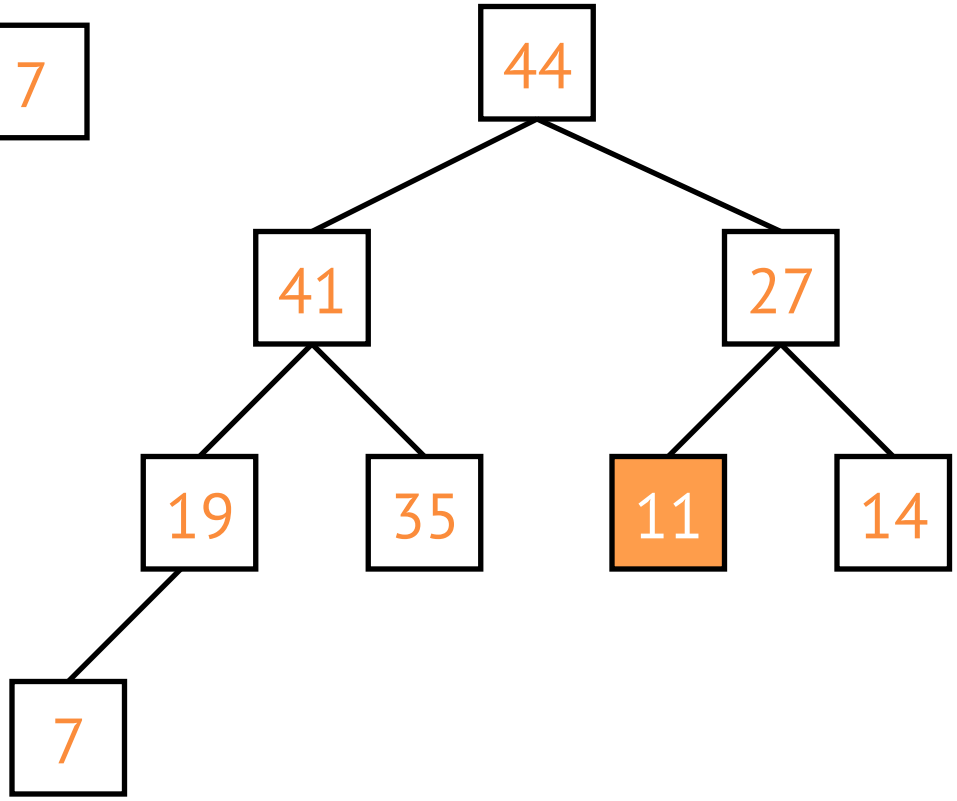




Heap-Sort: Beispiel

Phase I: Aufbau des Heaps

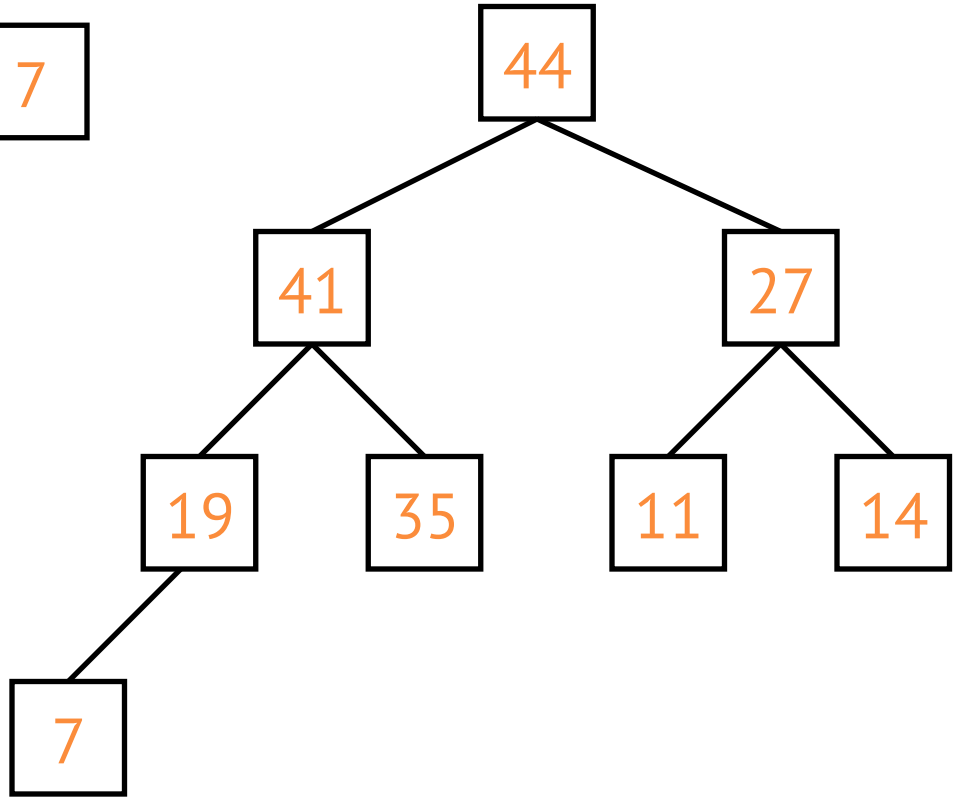




Heap-Sort: Beispiel

Phase I: Aufbau des Heaps

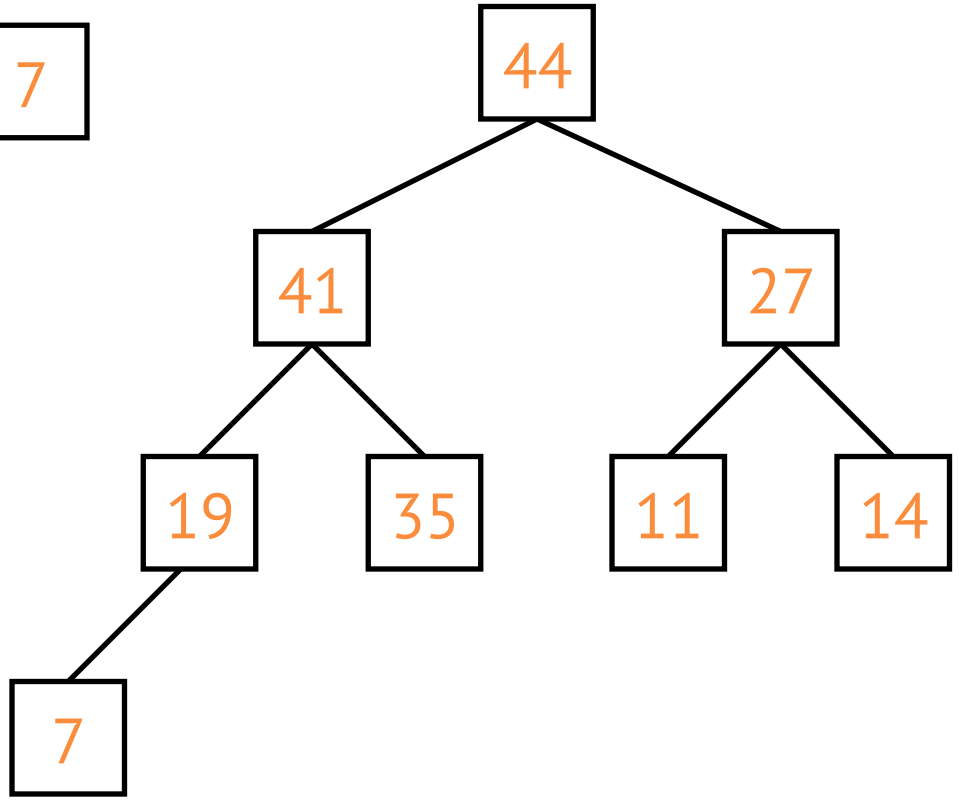
44	41	27	19	35	11	14	7



Heap-Sort: Beispiel

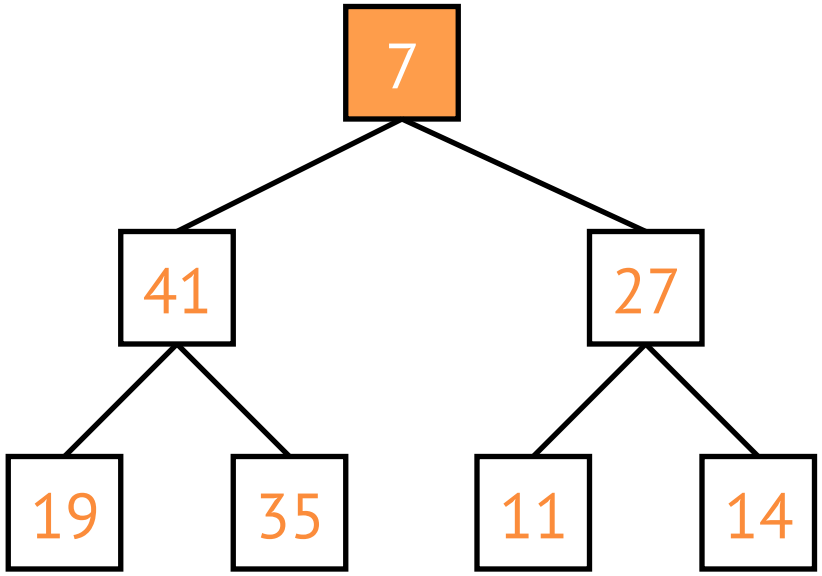
Phase II: Selection-Sort

44	41	27	19	35	11	14	7



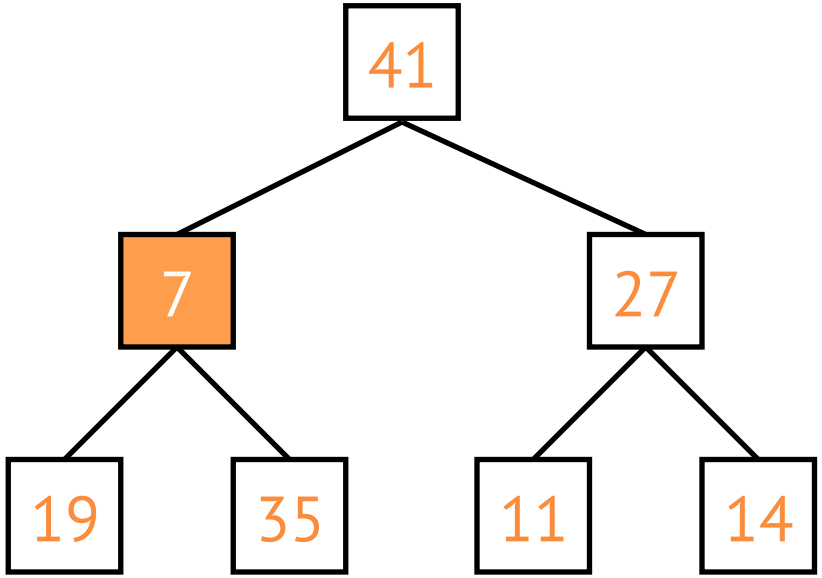
Heap-Sort: Beispiel

Phase II: Selection-Sort



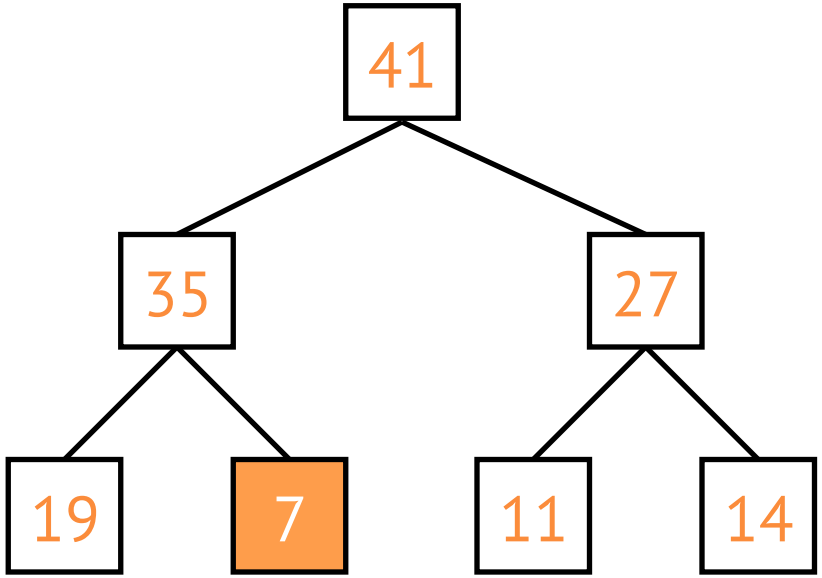
Heap-Sort: Beispiel

Phase II: Selection-Sort



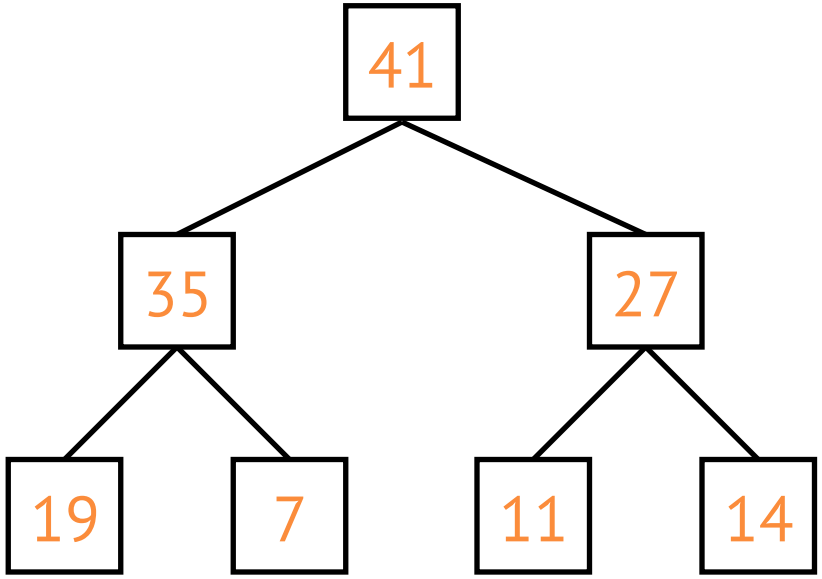
Heap-Sort: Beispiel

Phase II: Selection-Sort



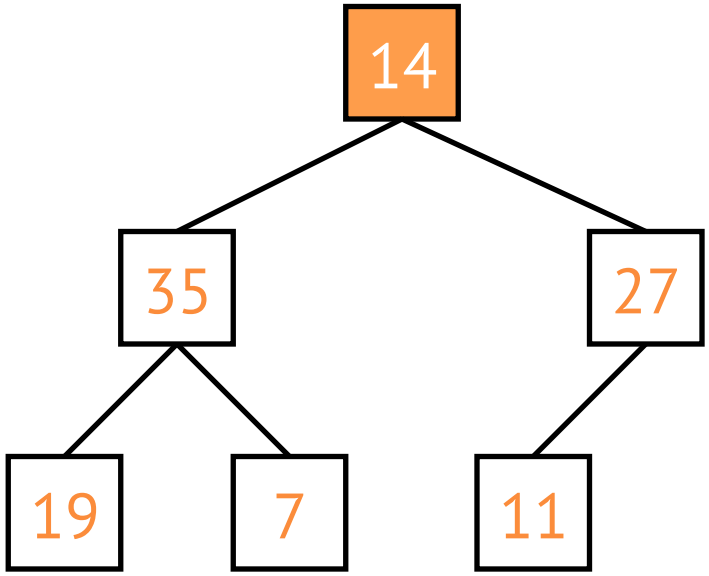
Heap-Sort: Beispiel

Phase II: Selection-Sort



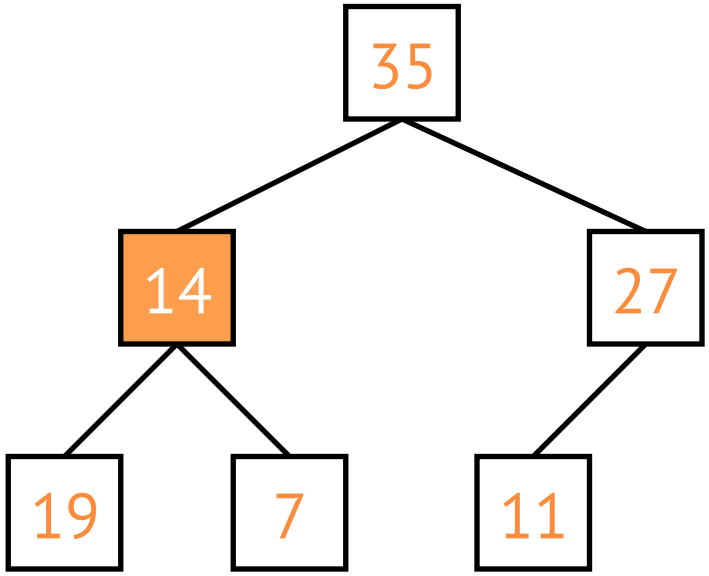
Heap-Sort: Beispiel

Phase II: Selection-Sort



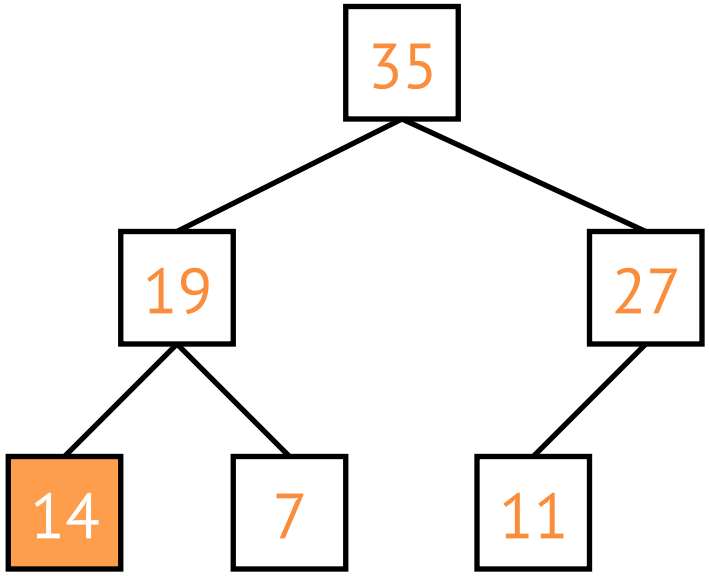
Heap-Sort: Beispiel

Phase II: Selection-Sort



Heap-Sort: Beispiel

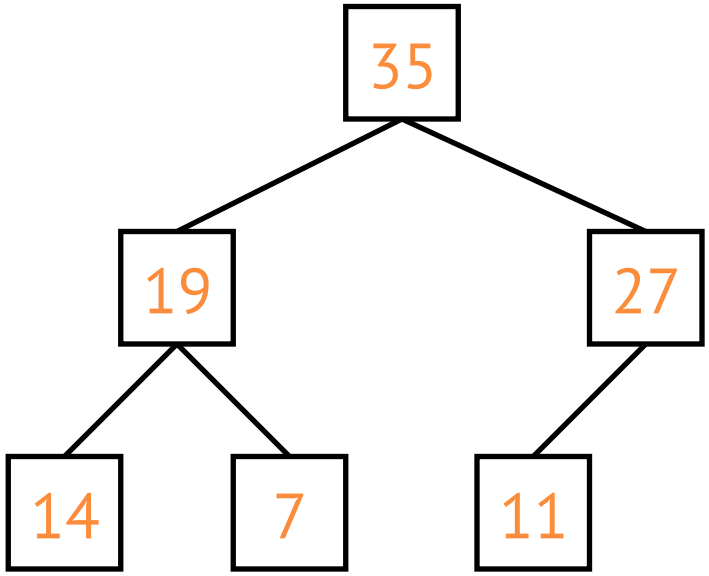
Phase II: Selection-Sort



Heap-Sort: Beispiel

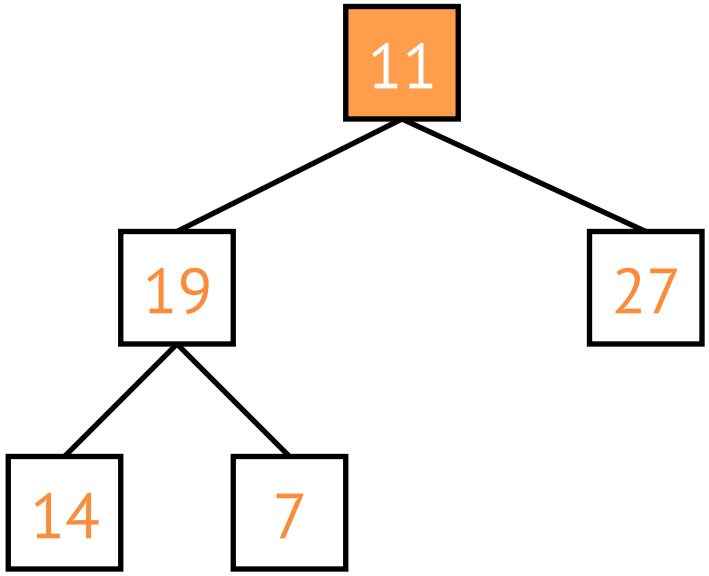
Phase II: Selection-Sort

35	19	27	14	7	11	41	44



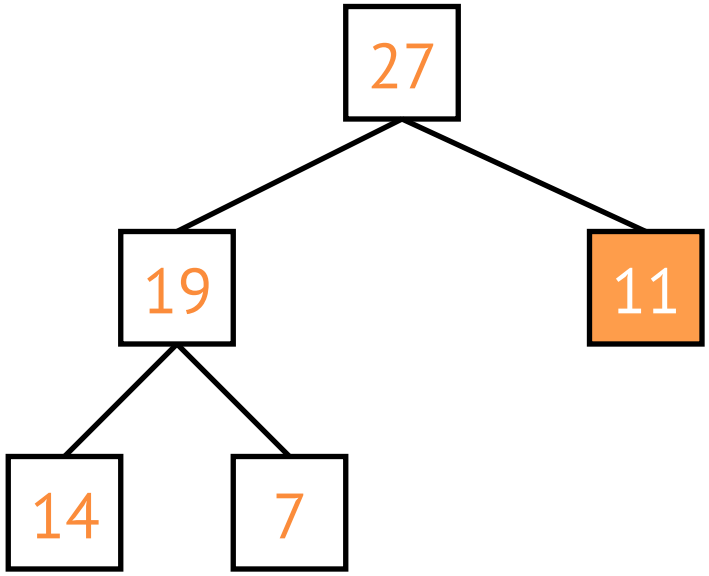
Heap-Sort: Beispiel

Phase II: Selection-Sort



Heap-Sort: Beispiel

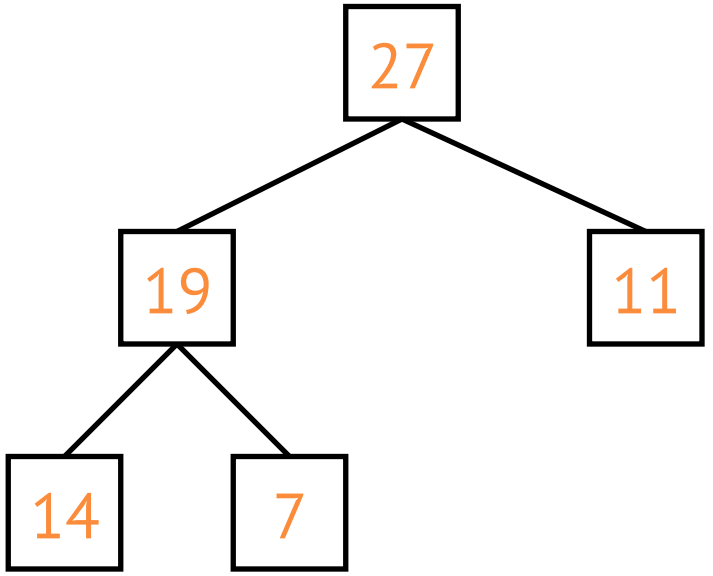
Phase II: Selection-Sort



Heap-Sort: Beispiel

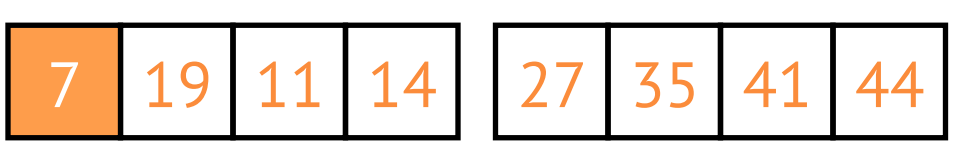
Phase II: Selection-Sort

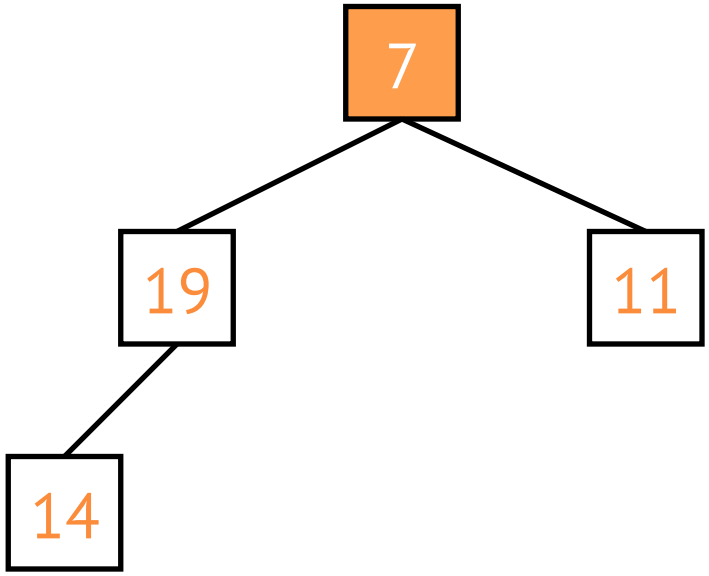
27	19	11	14	7	35	41	44



Heap-Sort: Beispiel

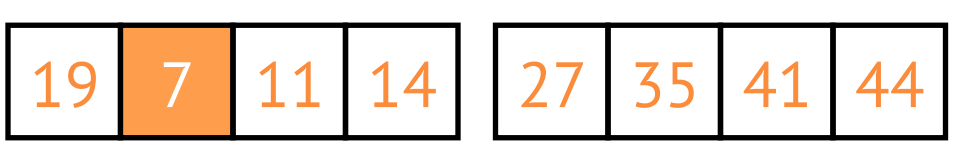
Phase II: Selection-Sort

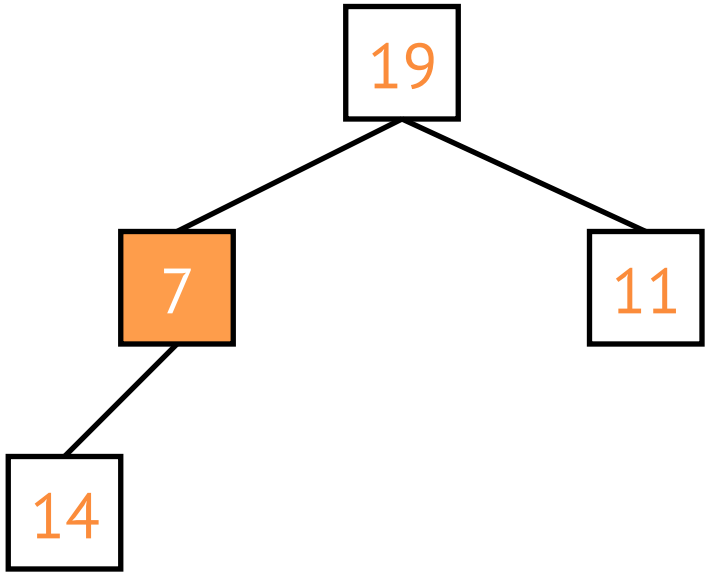




Heap-Sort: Beispiel

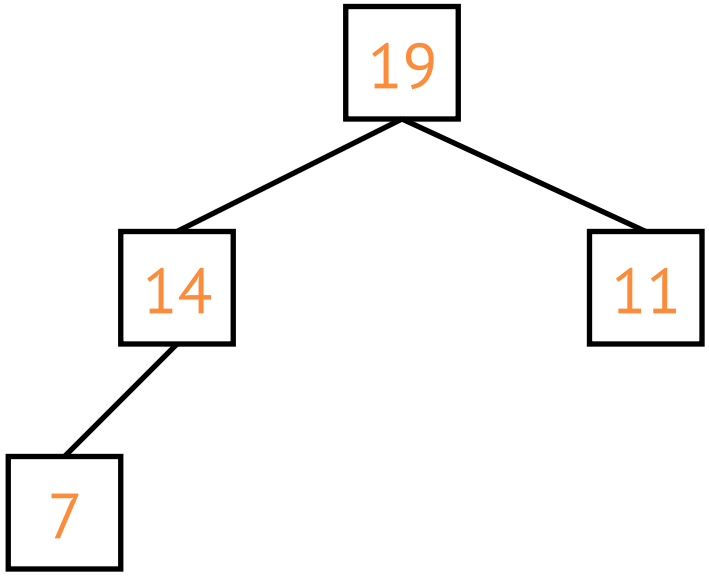
Phase II: Selection-Sort



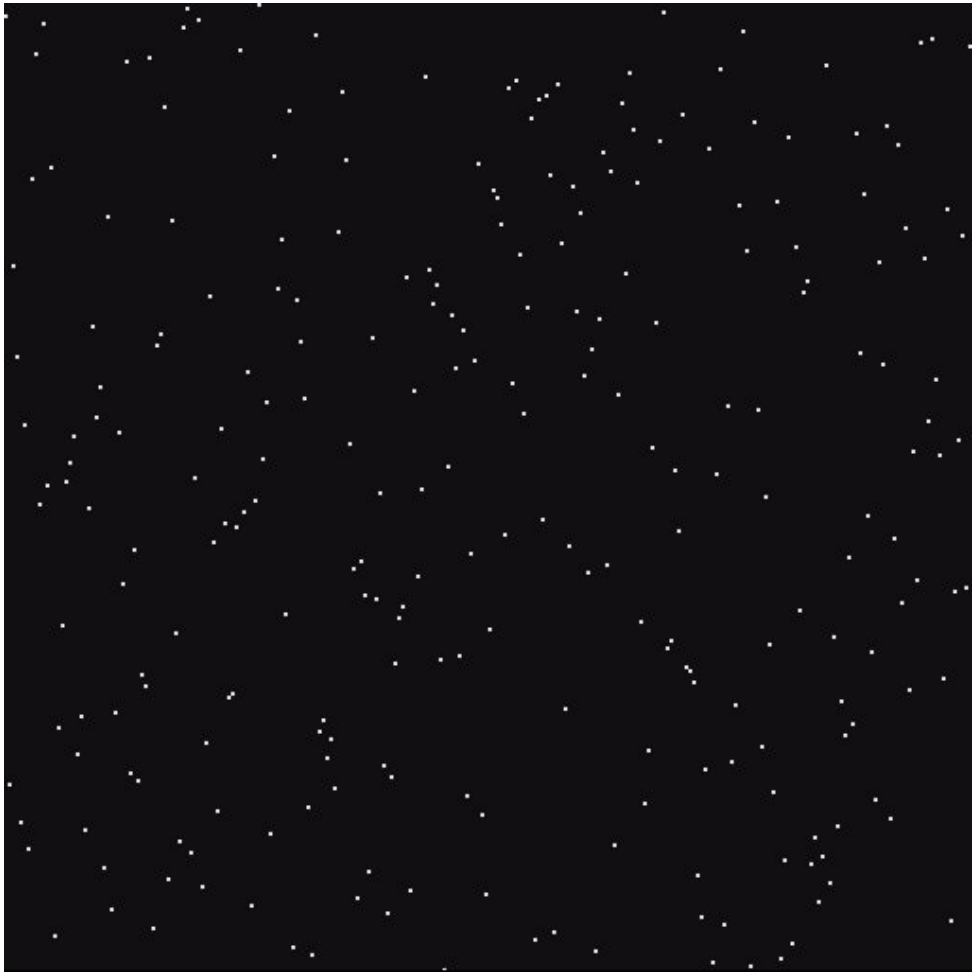


Heap-Sort: Beispiel

Phase II: Selection-Sort



Heap-Sort: Beispiel



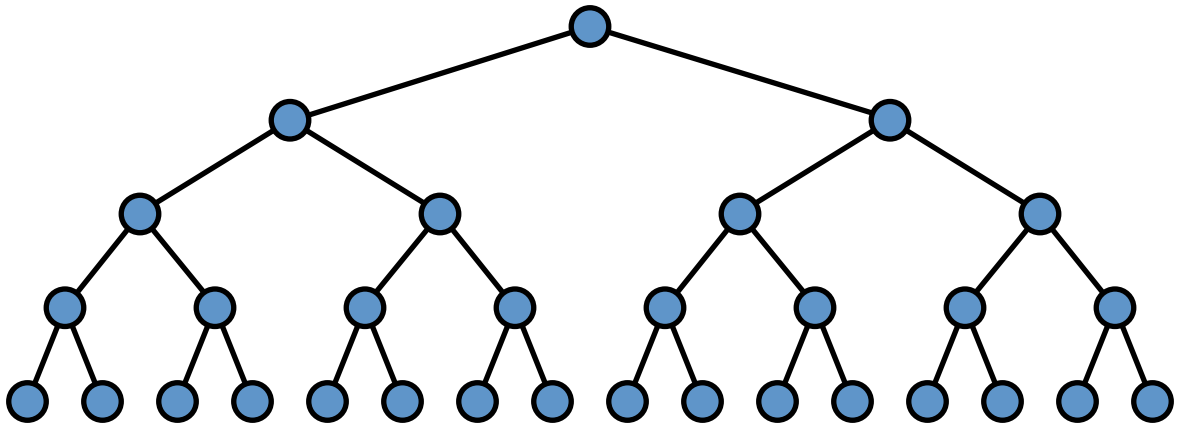
Heap-Sort: Aufwand

- Aufwand
  - Aufwand zum Aufbau des Heaps
  - Aufwand Selection-Sort

Aufwand: ConvertToHeap()

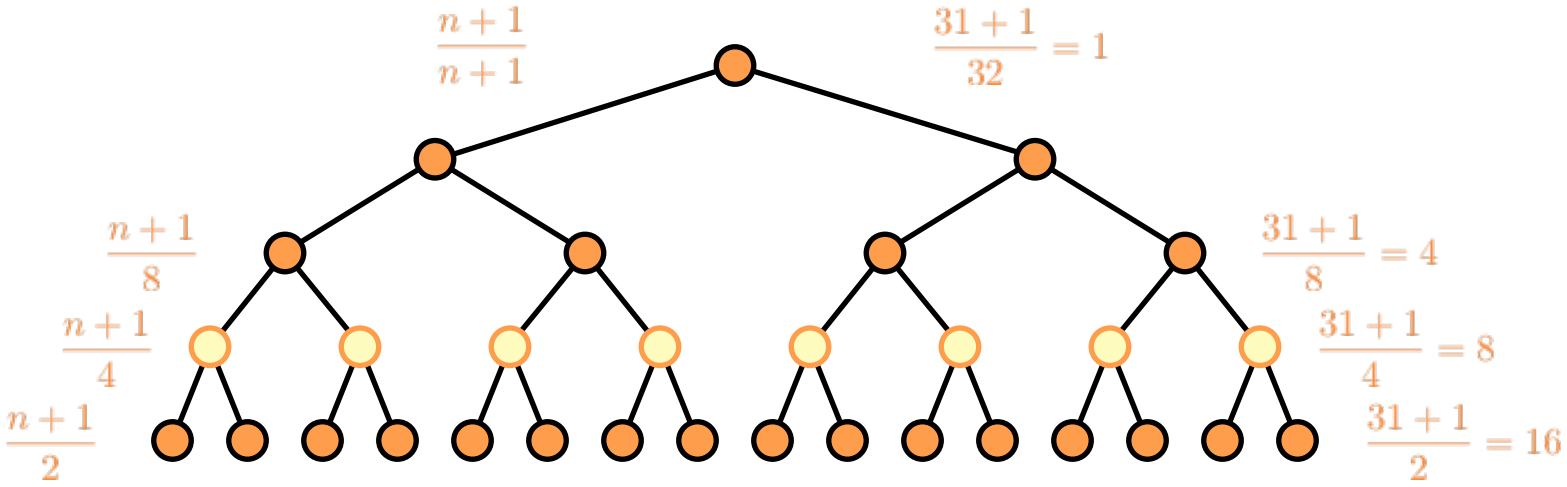
- ConvertToHeap()
  - Sink()-Operation = O(Höhe des Heap)
  - Sei $n = 2^k - 1$...

$n = 31, k = 5$



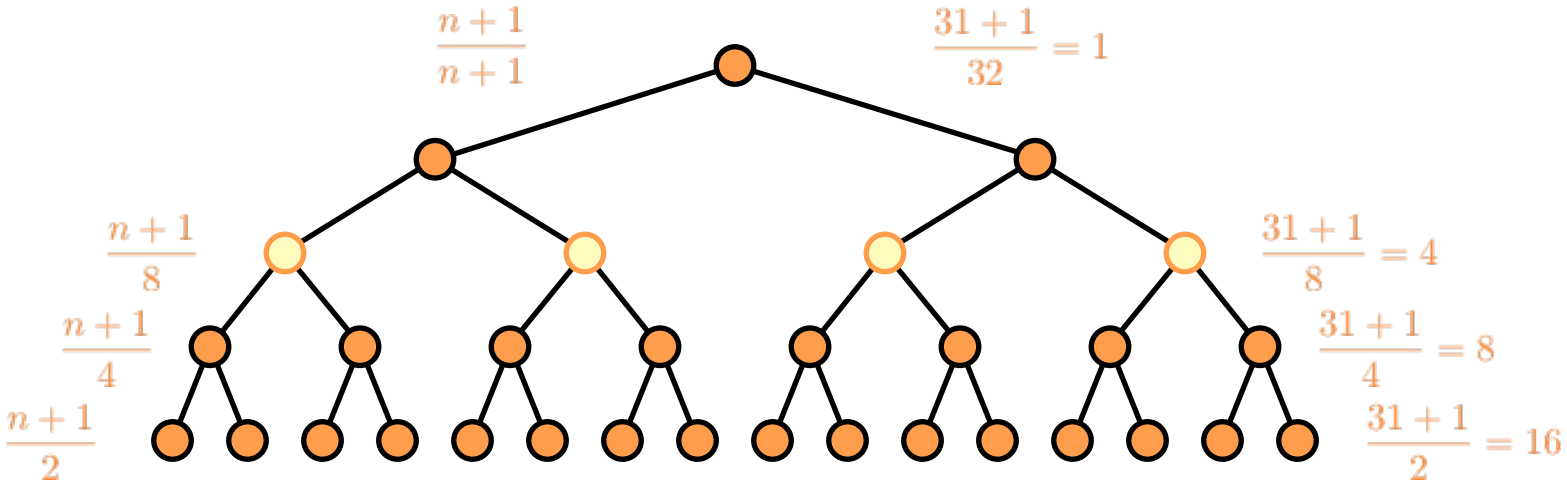
Aufwand: ConvertToHeap()

$$
\begin{aligned}
T_1(n) &= c\left(1\tfrac{n+1}{4} + 2\tfrac{n+1}{8} + \cdots + (\log(n+1)-1)\tfrac{n+1}{n+1}\right) \\
&= c\left(1\tfrac{n+1}{4} + \cdots + i\tfrac{n+1}{2^{i+1}} + \cdots + (k-1)\tfrac{n+1}{2^k}\right) \\
&= \tfrac{c}{2}(n+1)\textstyle\sum_{i=1}^{k-1} \tfrac{i}{2^i}
\end{aligned}
$$



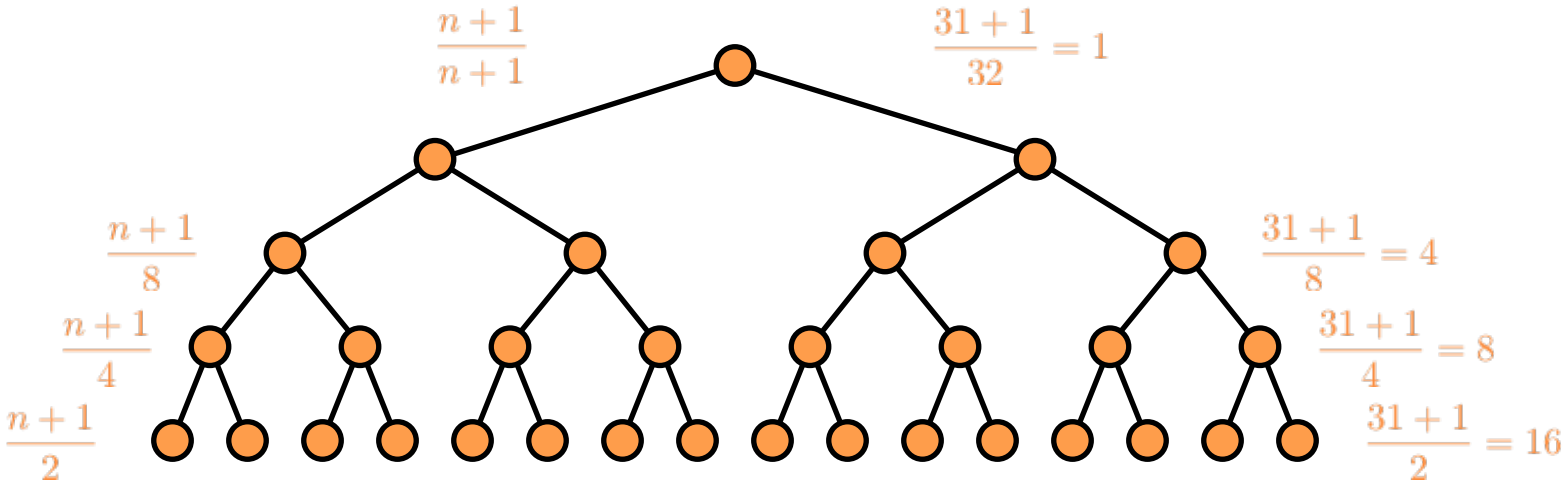
Aufwand: ConvertToHeap()

$$
\begin{aligned}
T_1(n) &= c\left(1\tfrac{n+1}{4} + 2\tfrac{n+1}{8} + \cdots + (\log(n+1)-1)\tfrac{n+1}{n+1}\right) \\
&= c\left(1\tfrac{n+1}{4} + \cdots + i\tfrac{n+1}{2^{i+1}} + \cdots + (k-1)\tfrac{n+1}{2^k}\right) \\
&= \tfrac{c}{2}(n+1)\textstyle\sum_{i=1}^{k-1} \tfrac{i}{2^i}
\end{aligned}
$$



Aufwand: ConvertToHeap()

$$
\begin{aligned}
T_1(n) &= c\left(1\tfrac{n+1}{4} + 2\tfrac{n+1}{8} + \cdots + (\log(n+1)-1)\tfrac{n+1}{n+1}\right) \\
&= c\left(1\tfrac{n+1}{4} + \cdots + i\tfrac{n+1}{2^{i+1}} + \cdots + (k-1)\tfrac{n+1}{2^k}\right) \\
&= \tfrac{c}{2}(n+1)\textstyle\sum_{i=1}^{k-1} \tfrac{i}{2^i}
\end{aligned}
$$



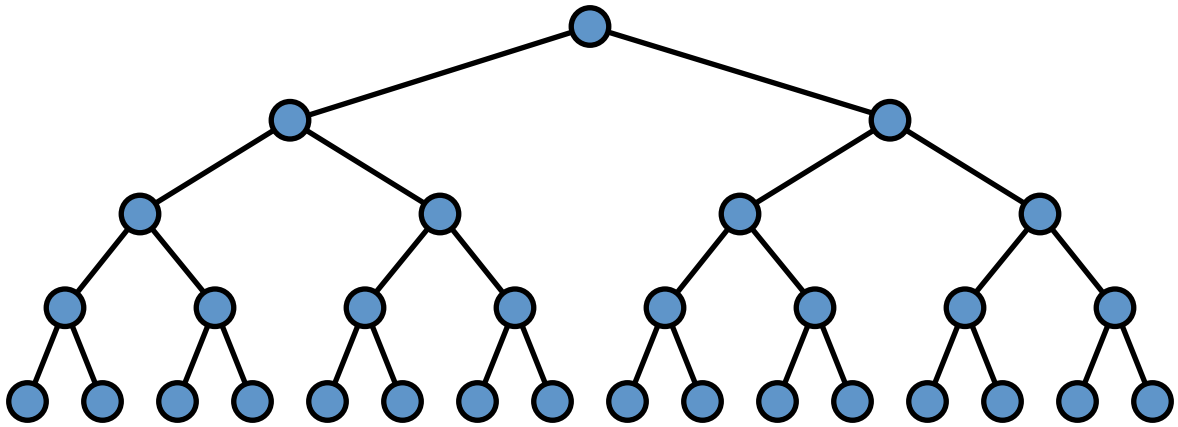
Aufwand: ConvertToHeap()

$$
\begin{aligned}
T_1(n) &\leq \frac{c}{2}(n+1)\sum_{i=1}^{k-1}\frac{i}{2^i} \\
&= \frac{c}{2}(n+1)\underbrace{\left(2-\frac{k+1}{2^{k-1}}\right)}_{<2} \\
&= O(n)
\end{aligned}
$$

Aufwand: Selection-Sort

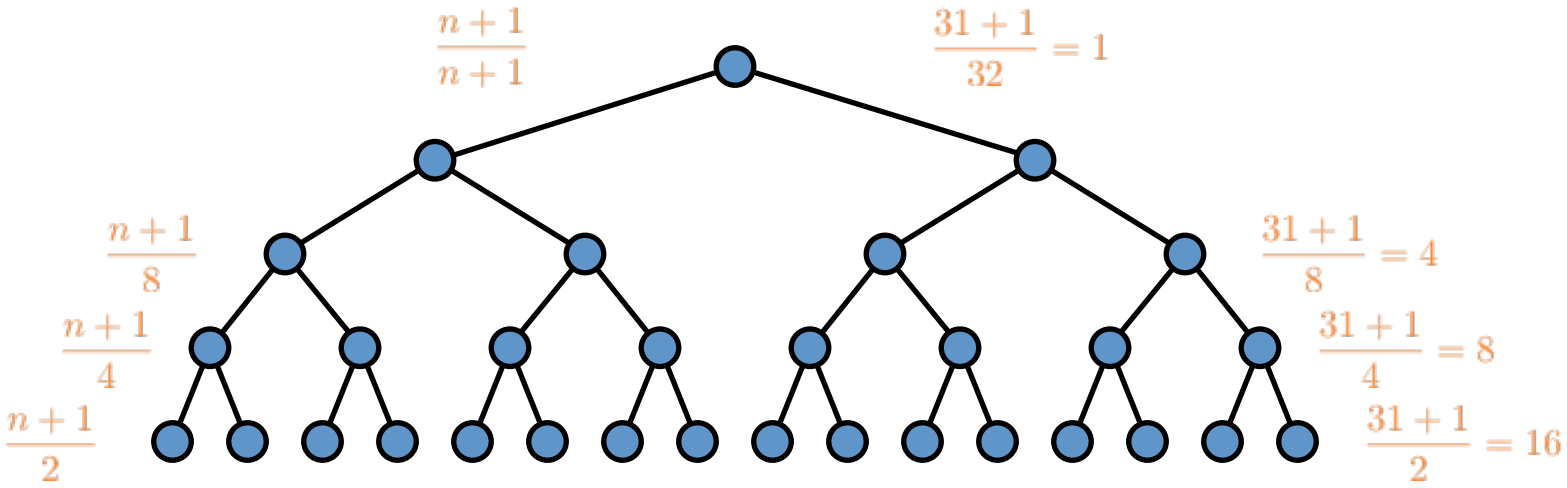
- Selection-Sort
  - Kosten für die Wiederherstellung der Heap-Eigenschaft = O(Höhe des restlichen Heaps)
  - Sei $n = 2^k - 1$...

$n = 31, k = 5$



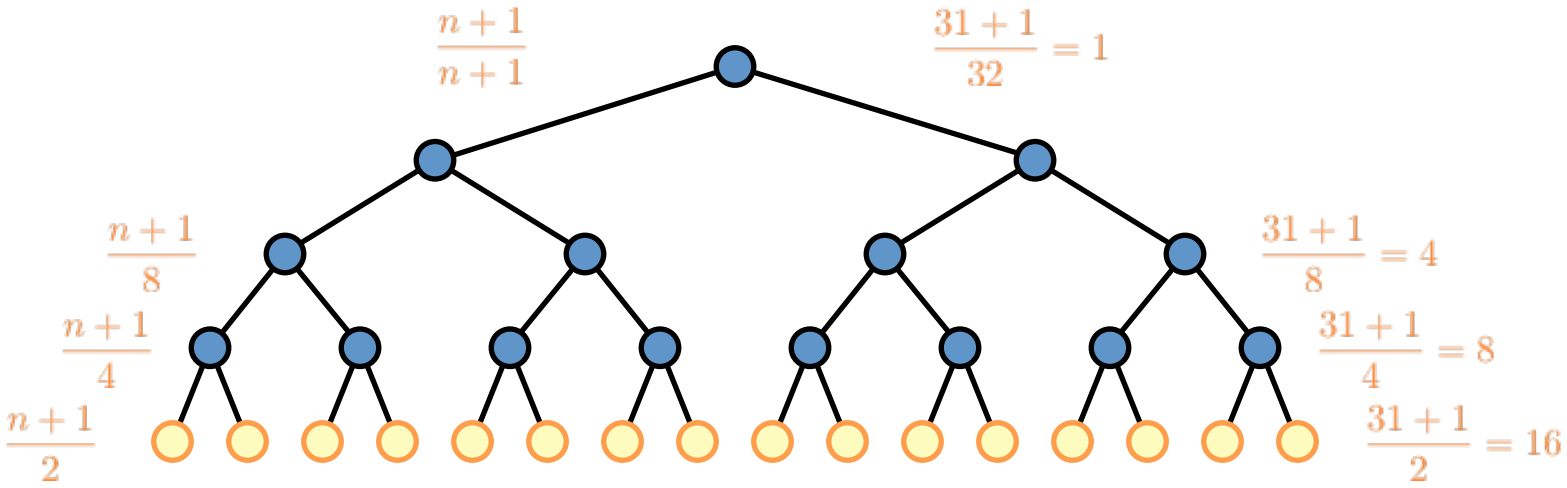
Aufwand: Selection-Sort

- Selection-Sort
  - Kosten für die Wiederherstellung der Heap-Eigenschaft = O(Höhe des restlichen Heaps)
  - Sei $n = 2^k - 1$...



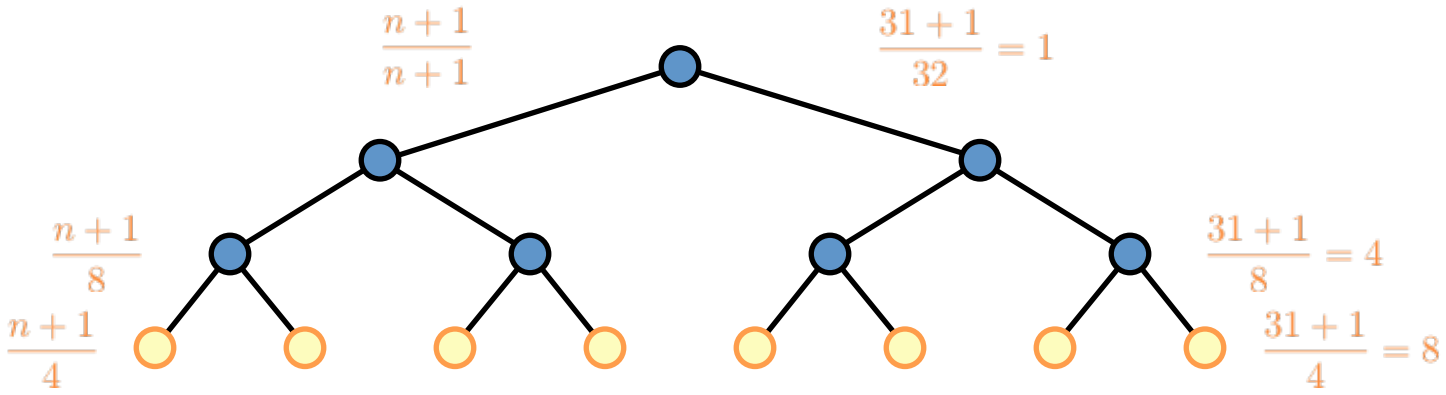
Aufwand: Selection-Sort

- Selection-Sort
  - Kosten für die Wiederherstellung der Heap-Eigenschaft = O(Höhe des restlichen Heaps)
  - Sei $n = 2^k - 1$...



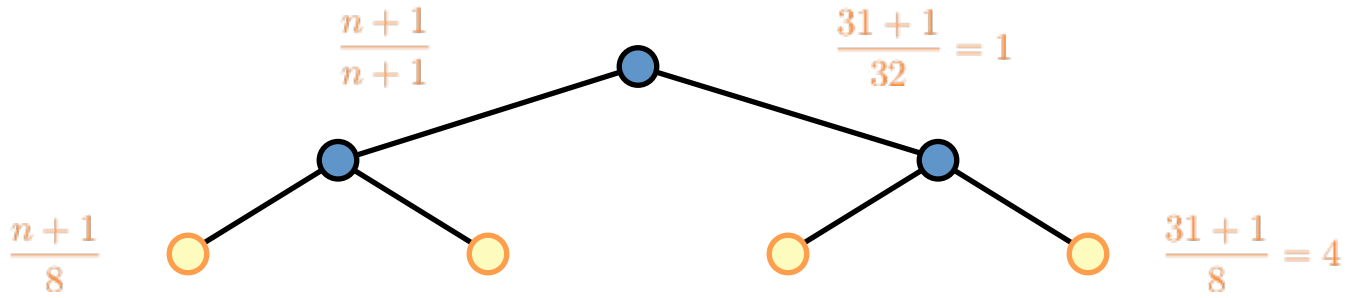
Aufwand: Selection-Sort

- Selection-Sort
  - Kosten für die Wiederherstellung der Heap-Eigenschaft = O(Höhe des restlichen Heaps)
  - Sei $n = 2^k - 1$...



Aufwand: Selection-Sort

- Selection-Sort
  - Kosten für die Wiederherstellung der Heap-Eigenschaft = O(Höhe des restlichen Heaps)
  - Sei $n = 2^k - 1$...



Aufwand: SelectionSort

$$
\begin{aligned}
T_2(n) &\leq c\left((k-1)\tfrac{n+1}{2} + (k-2)\tfrac{n+1}{4} + \cdots + 0\tfrac{n+1}{n+1}\right) \\
&= c\left((k-1)\tfrac{n+1}{2} + \cdots + (k-i)\tfrac{n+1}{2^i} + \cdots\right) \\
&= c(n+1)\textstyle\sum_{i=1}^{k-1}(k-i)2^{-i} \\
&= c(n+1)\textstyle\sum_{i=1}^{k-1} i2^{i-k} \\
&= c(n+1)\tfrac{1}{2^k}\textstyle\sum_{i=1}^{k-1} i2^{i} \\
&= c(n+1)\tfrac{1}{2^k}\left(2^k(k-2)+2\right) \\
&= c(n+1)(k-2+\tfrac{1}{2^{k-1}}) \\
&= O(n\log n)
\end{aligned}
$$

Aufwandsabschätzung

- Insgesamt … worst / average case

$$T(n) = T_1(n) + T_2(n)$$
$$= O(n) + O(n \times \log n)$$
$$= O(n \times \log n)$$

- Best case: $O(n)$ … alle Elemente gleich
- Vorsortierung wird nicht ausgenutzt

Vergleich

	Average	Worst
Quick-Sort	$O(n \times \log n)$	$O(n^2)$
Merge-Sort (doppelter Speicher)	$O(n \times \log n)$	$O(n \times \log n)$
Heap-Sort	$O(n \times \log n)$	$O(n \times \log n)$