Kapitel 7
Sortieren

- Grundbegriffe
- Klassifikation von Sortierverfahren
- Quicksort
- Bucketsort
- Heapsort

Sortieren
Grundbegriffe

Sortieren: Umordnen einer Folge von Objekten $a_1, ..., a_n$ in eine Folge $a_{i_1}, ..., a_{i_n}$, so daß bezgl. einer Ordnung $\leq$ gilt:

$$a_{i_1} \leq ... \leq a_{i_h}$$

Sortieren
Grundbegriffe

- **Stabilität**: relative Reihenfolge gleichrangiger Objekte wird nicht verändert, d. h.

$$i_j < i_k, \text{ falls } a_{i_j} = a_{i_k} \text{ und } j < k.$$

- Wichtig für vorsortierte Datenmengen!
  - Beispiel: erst Sortierung nach Nachnamen, dann Sortierung nach Vornamen → Sortierung nach Nachnamen soll erhalten bleiben.

Sortieren
Grundbegriffe

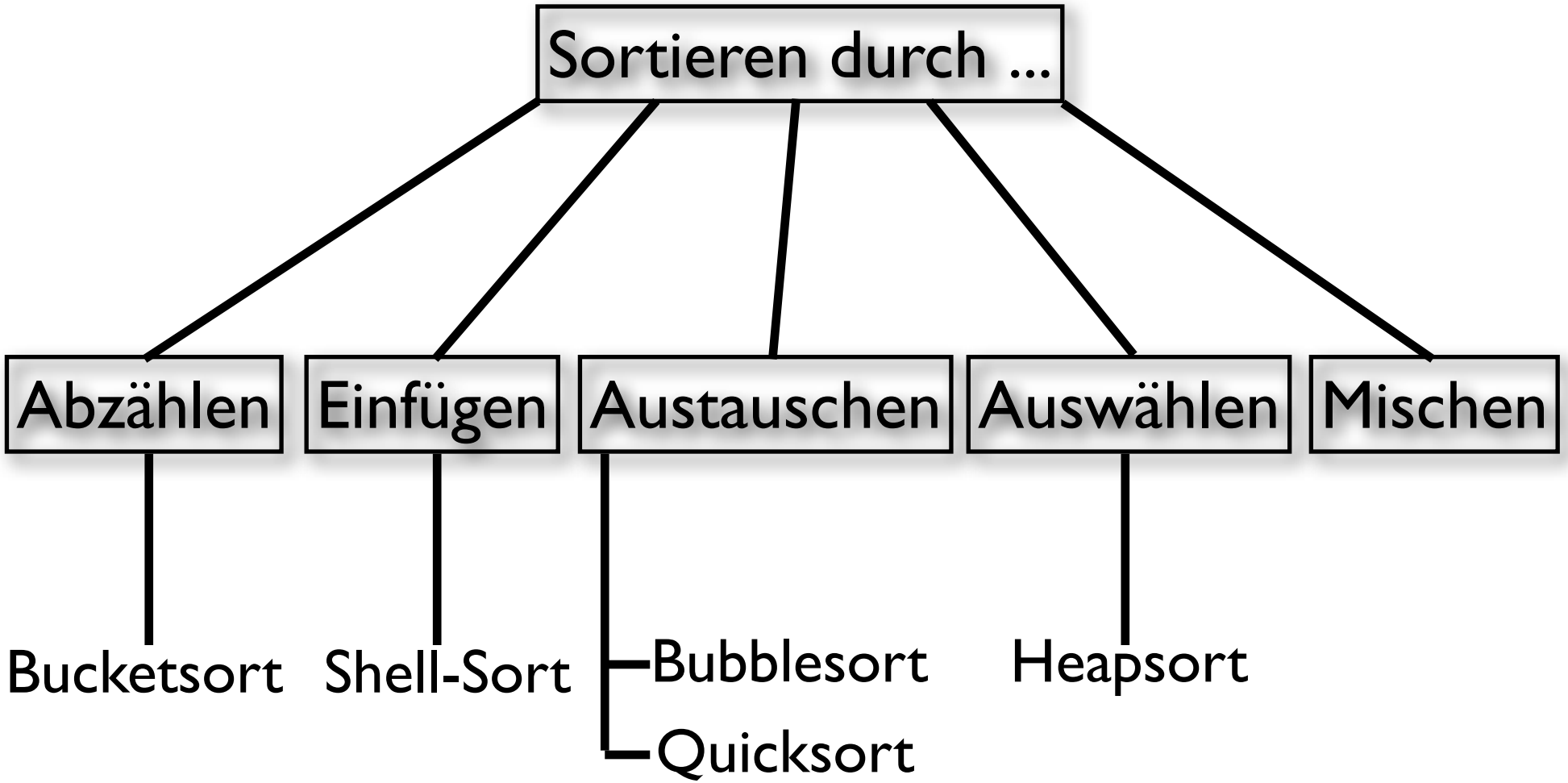
- **Internes Sortieren**: Datenmenge während des Sortierens vollständig im Hauptspeicher.
- **Externes** Sortieren: Datenmenge überwiegend auf externen Speichern
- Beispiel:
  - Bubblesort: intern
  - Sortieren durch Mischen: extern

Sortieren
Grundbegriffe

- **Zeitbedarf**:
  - Bewiesen: worst-case mindestens O(n log n) Vergleiche
  - Forderung: Nur Sortierverfahren betrachten, die diese Schranke erreichen.
- **Platzbedarf**: Wenig zusätzlicher Speicher (Sortieren **in-situ**)

Sortieren

Klassifikation v. Sortierverfahren



Sortieren
Quicksort - Idee

- Gebräuchlichstes internes Sortierverfahren (C.A.R. Hoare, 1962)
- Idee (rekursives Divide-and-Conquer-Verfahren): Gegeben lineares Array A mit n Elementen:
  - Transportiere die n/2 kleinsten Elemente nach A[1], ..., A[n/2] und die n/2 größten Elemente nach A[n/2+1], ..., A[n]
  - Wende Verfahren rekursiv auf die erste Hälfte an.
  - Wende Verfahren rekursiv auf die zweite Hälfte an.

Sortieren
Quicksort - Idee

- Gebräuchlichstes internes Sortierverfahren (C.A.R. Hoare, 1962)
- Idee (rekursives Divide-and-C[illegible]hren): Gegeben lineares Array A m[illegible]

Divide
Vorbereitung

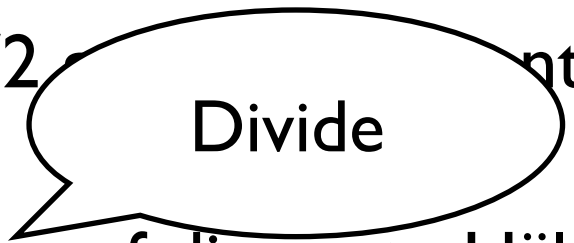
  - Transportiere die n/2 kleinsten Elemente nach A[1], ..., A[n/2] und die n/2 größten Elemente nach A[n/2+1], ..., A[n]
  - Wende Verfahren rekursiv auf die erste Hälfte an.
  - Wende Verfahren rekursiv auf die zweite Hälfte an.

Sortieren
Quicksort - Idee

- Gebräuchlichstes internes Sortierverfahren (C.A.R. Hoare, 1962)
- Idee (rekursives Divide-and-Conquer-Verfahren): Gegeben lineares Array A mit n Elementen:
  - Transportiere die n/2 kleinsten Elemente nach A[1], ..., A[n/2] und die n/2 größten Elemente nach A[n/2+1], ..., A[n]
  - Wende Verfahren rekursiv auf die erste Hälfte an.
  - Wende Verfahren rekursiv auf die zweite Hälfte an.

Sortieren
Quicksort - Idee

- Gebräuchlichstes internes Sortierverfahren (C.A.R. Hoare, 1962)
- Idee (rekursives Divide-and-Conquer-Verfahren): Gegeben lineares Array A mit n Elementen:
  - Transportiere die n/2 kleinsten Elemente nach A[1], ..., A[n/2] und die n/2 [illegible]nte nach A[n/2+1], ..., A[n]
  - Wende Verfahren rekursiv auf die erste Hälfte an.
  - Wende Verfahren rekursiv auf die zweite Hälfte an.

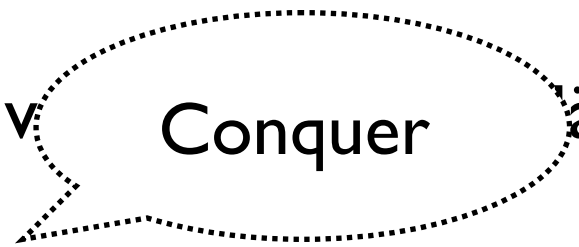


Sortieren
Quicksort - Idee

- Gebräuchlichstes internes Sortierverfahren (C.A.R. Hoare, 1962)
- Idee (rekursives Divide-and-Conquer-Verfahren): Gegeben lineares Array A mit n Elementen:
  - Transportiere die n/2 kleinsten Elemente nach A[1], ..., A[n/2] und die n/2 größten Elemente nach A[n/2+1], ..., A[n]
  - Wende Verfahren rekursiv auf die erste Hälfte an.
  - Wende Verfahren rekursiv auf die zweite Hälfte an.

Sortieren
Quicksort - Idee

- Gebräuchlichstes internes Sortierverfahren (C.A.R. Hoare, 1962)
- Idee (rekursives Divide-and-Conquer-Verfahren): Gegeben lineares Array A mit n Elementen:
  - Transportiere die n/2 kleinsten Elemente nach A[1], ..., A[n/2] und die n/2 größten Elemente nach A[n/2+1], ..., A[n]
  - Wende Verfahren rekursiv auf die erste Hälfte an.
  - Wende Verfahren rekursiv auf die zweite Hälfte an.



Sortieren
Quicksort - Median

- Wie findet man die Mitte?

Definition

Sei $M=\{m_1, ..., m_n\}$ eine Menge mit einer Ordnung $\leq$. Das Objekt $x \in M$ mit der Eigenschaft

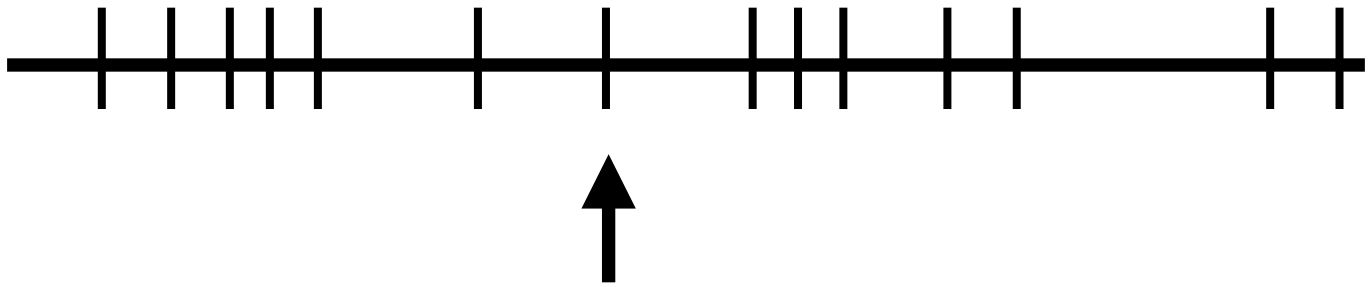
$$|\{m_i \mid m_i \leq x\}| \approx |\{m_i \mid x \leq m_i\}|$$

heißt **Median**.

("links vom Median liegen soviele Elemente wie rechts")

Sortieren

Quicksort - Median - Beispiel



7 Elemente $\leq$, 8 Elemente $\geq$

Beachte: meist Median $\neq$ Mittelwert

Beispiel: 1, 10, 100

Median: 10

Mittelwert: 55,5

Sortieren

Quicksort - Median - Beispiel

Sortieren

Quicksort - Median - Beispiel

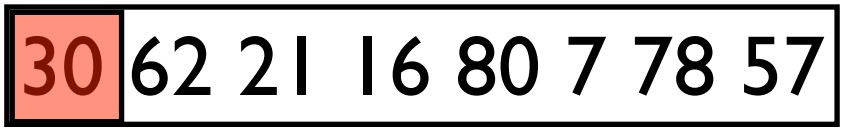
30 62 21 16 80 7 78 57

Sortieren

Quicksort - Median - Beispiel

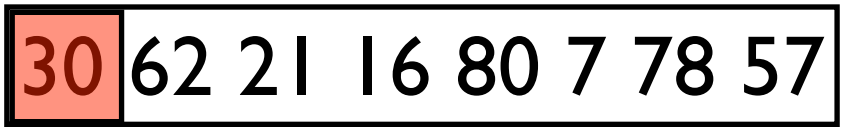
Sortieren

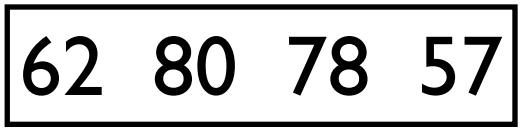
Quicksort - Median - Beispiel



Sortieren

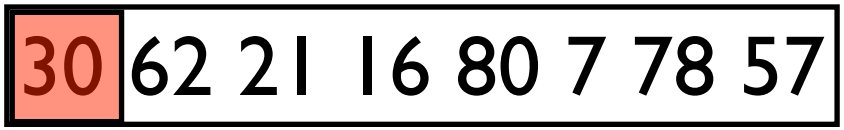
Quicksort - Median - Beispiel

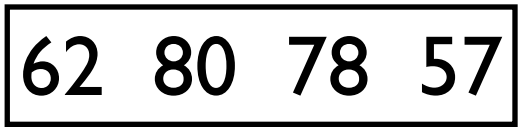




Sortieren

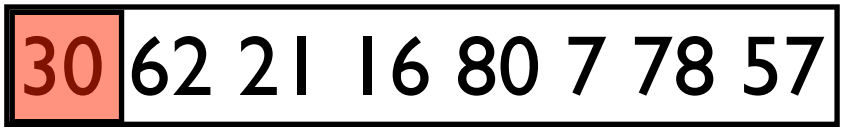
Quicksort - Median - Beispiel

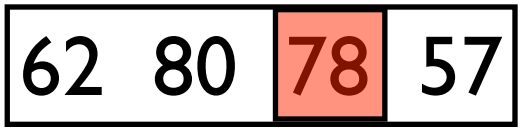




Sortieren

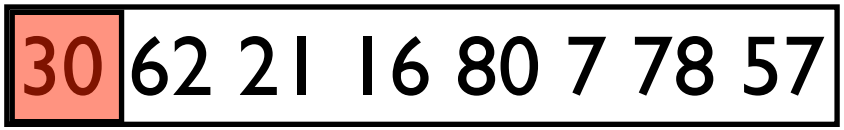
Quicksort - Median - Beispiel

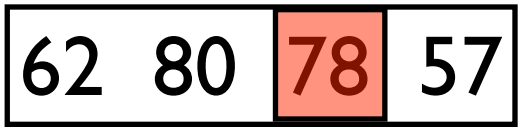


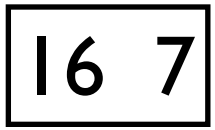


Sortieren

Quicksort - Median - Beispiel

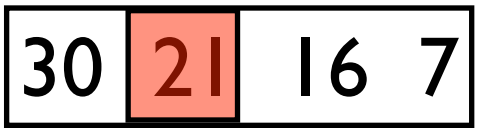


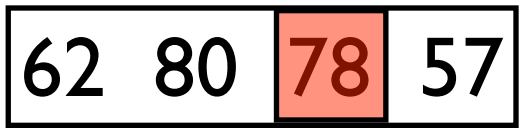


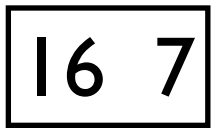


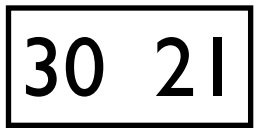
Sortieren

Quicksort - Median - Beispiel



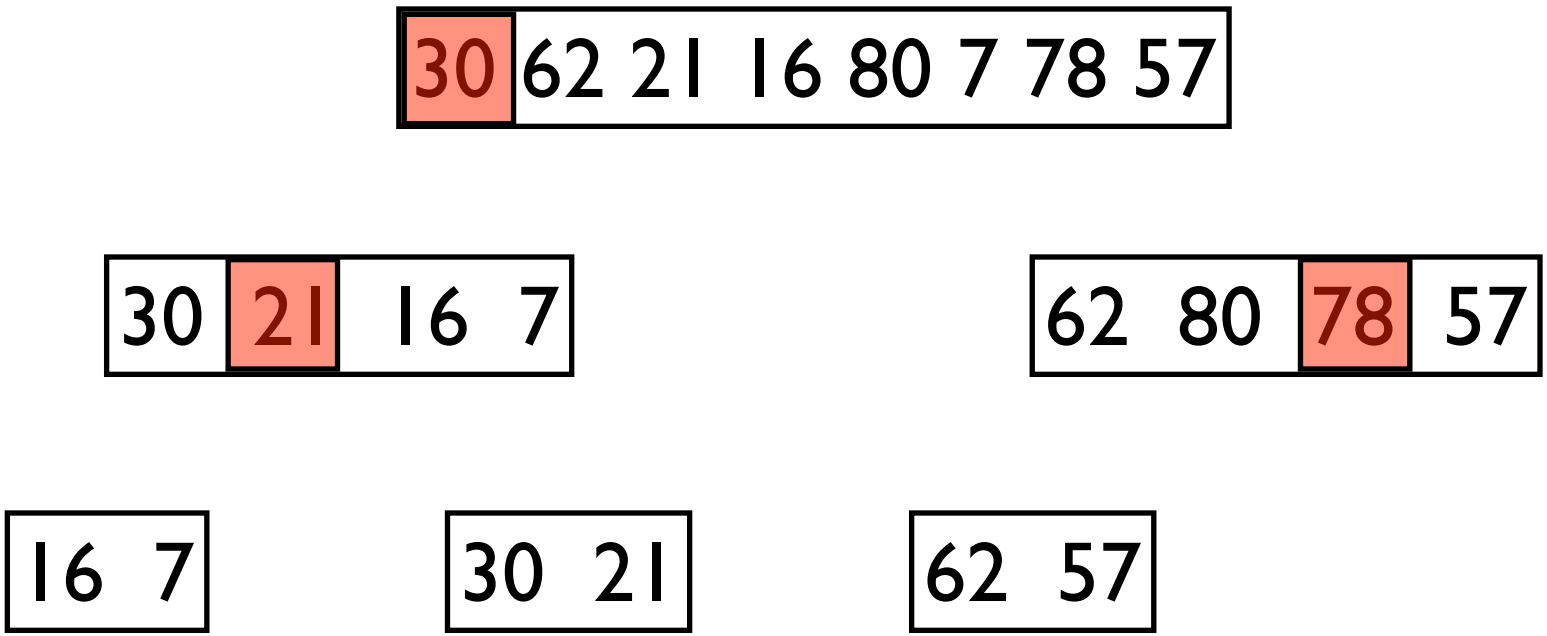






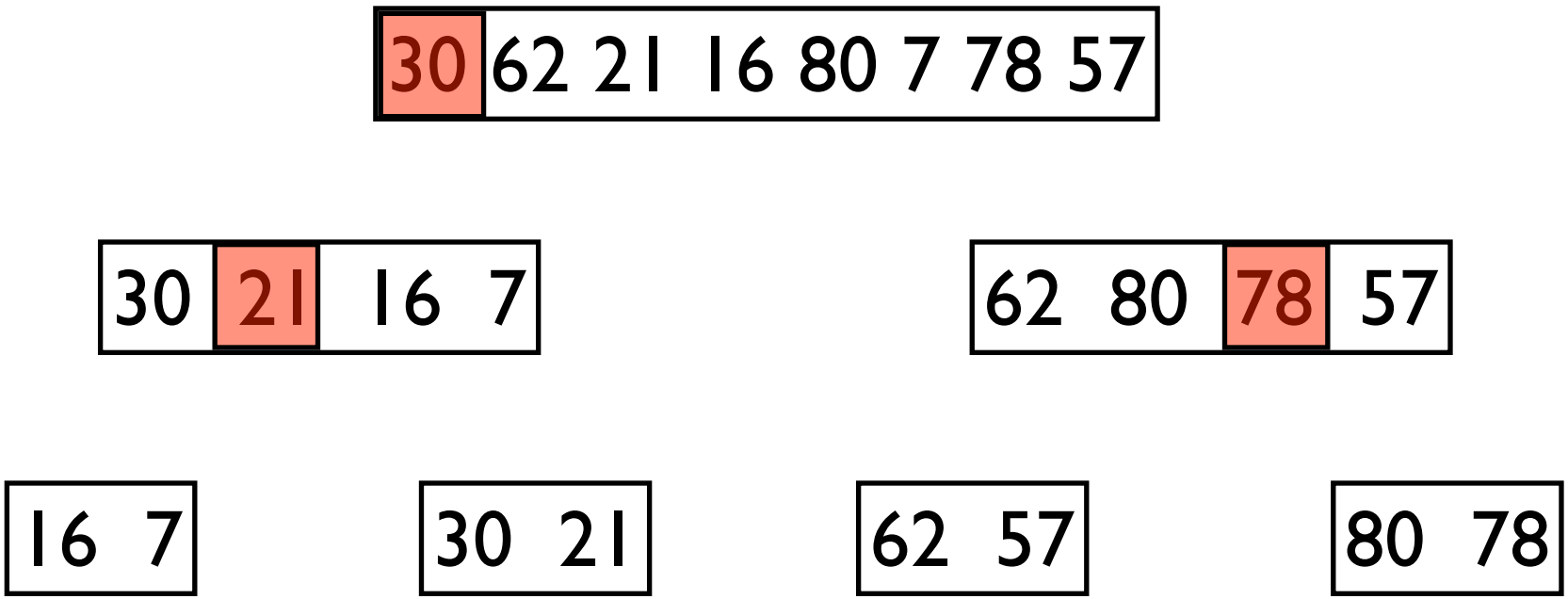
Sortieren

Quicksort - Median - Beispiel



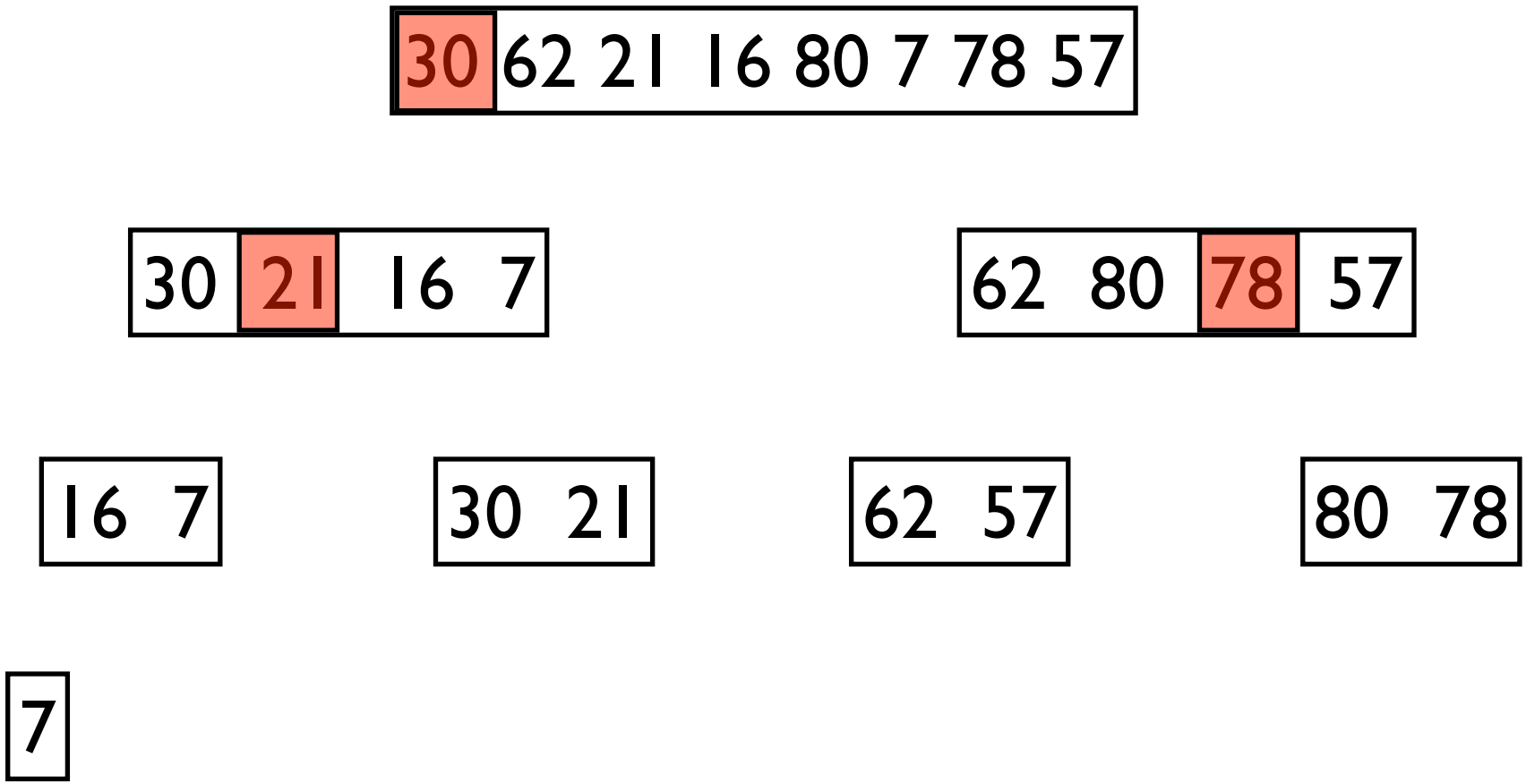
Sortieren

Quicksort - Median - Beispiel



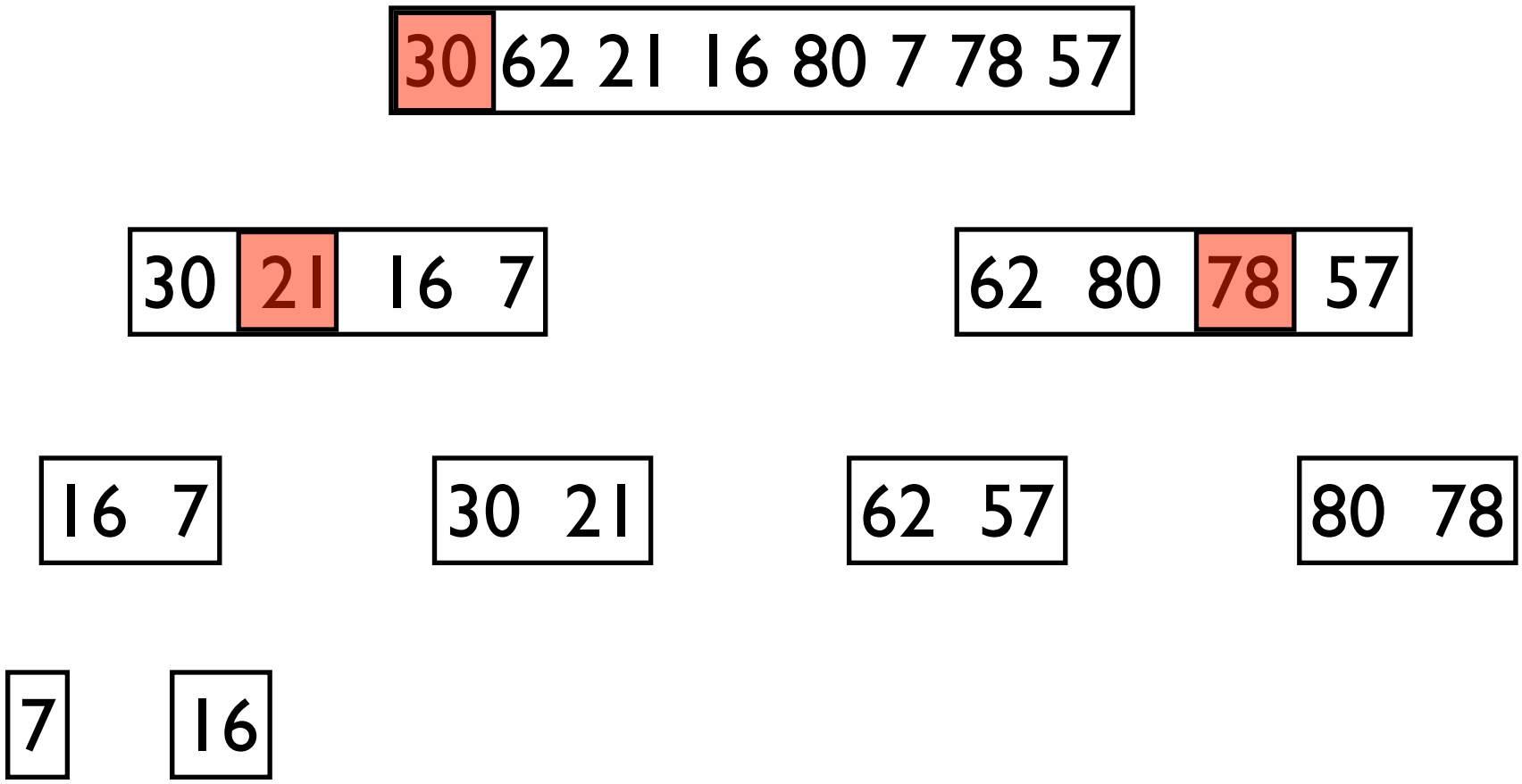
Sortieren

Quicksort - Median - Beispiel



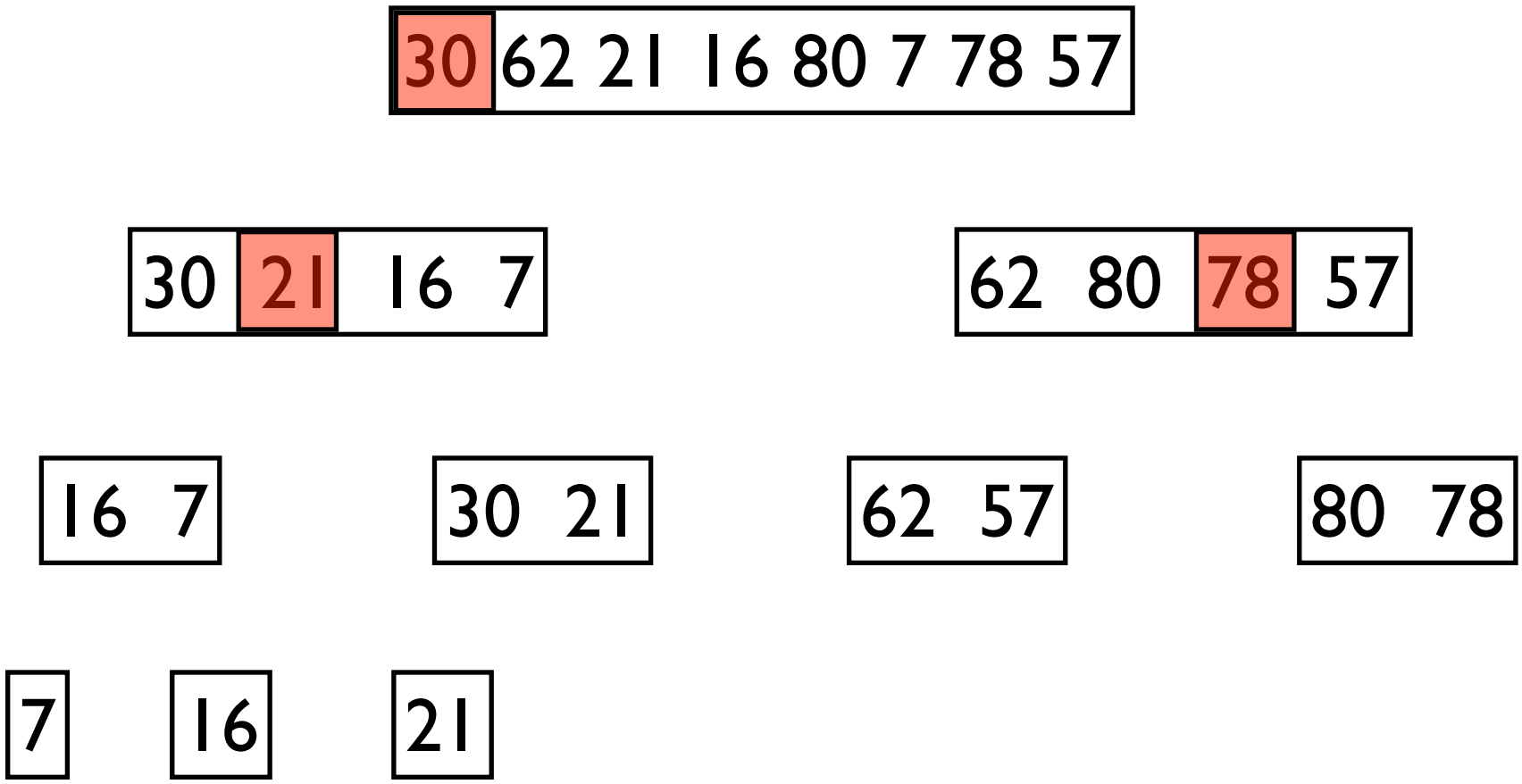
Sortieren

Quicksort - Median - Beispiel



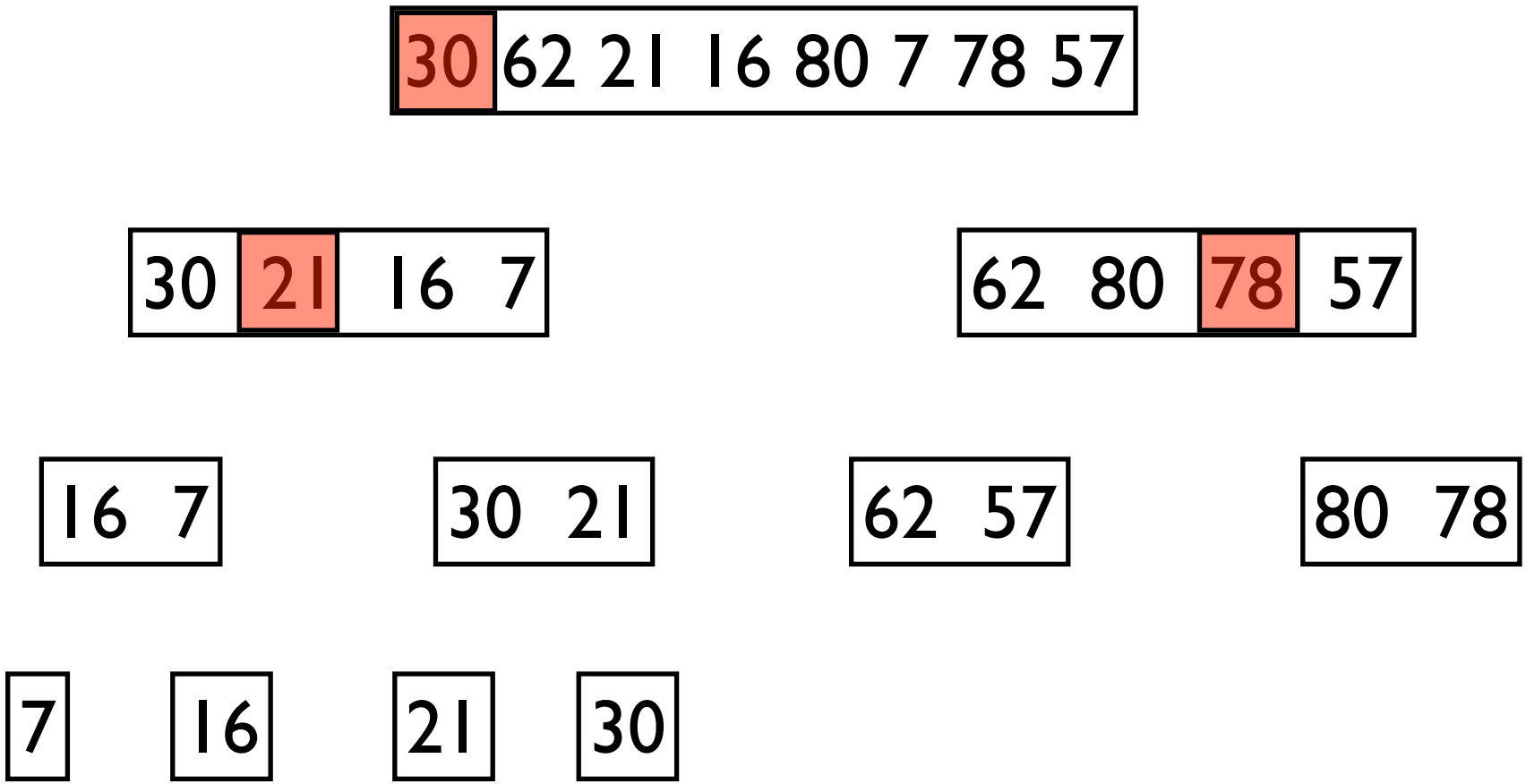
Sortieren

Quicksort - Median - Beispiel



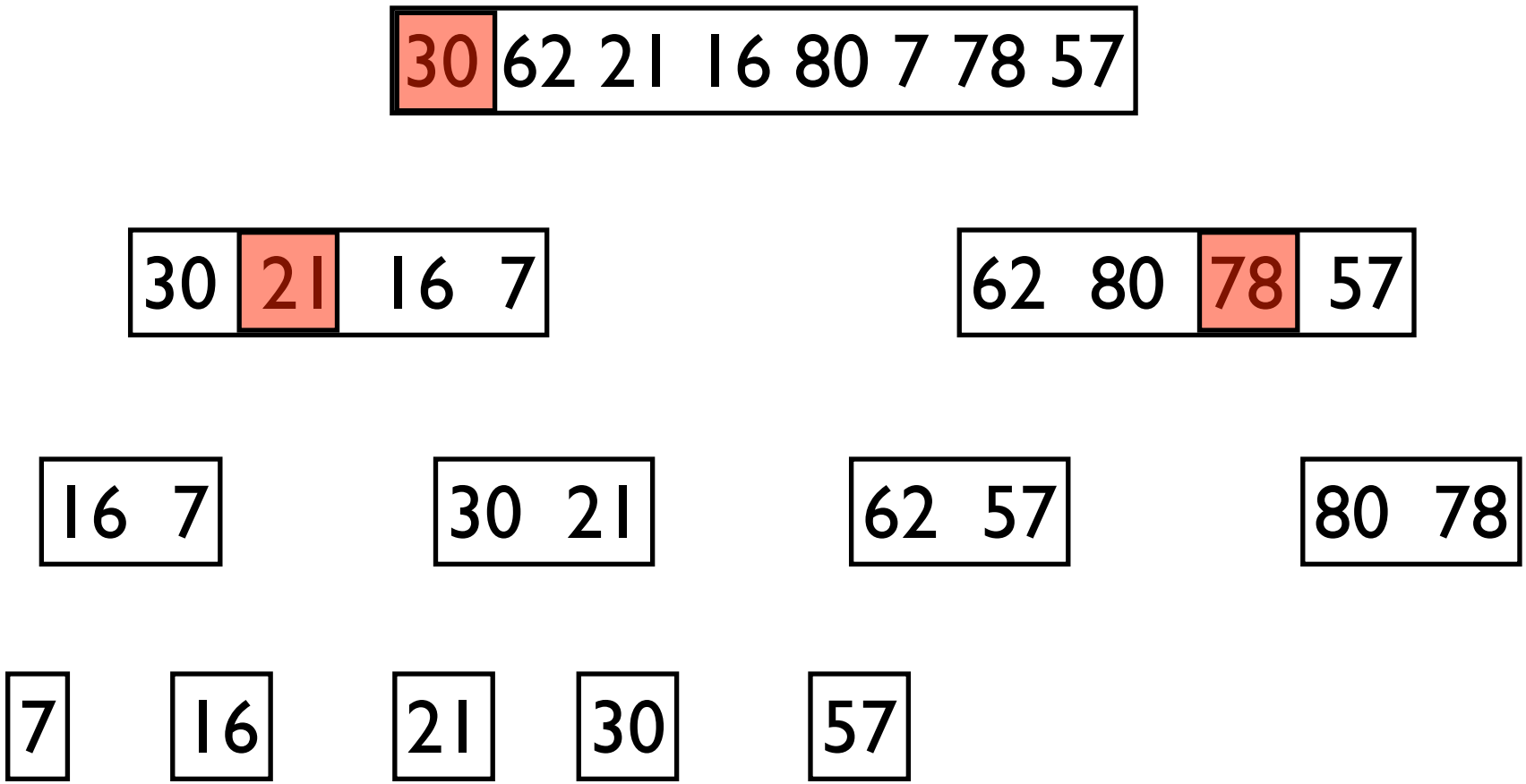
Sortieren

Quicksort - Median - Beispiel



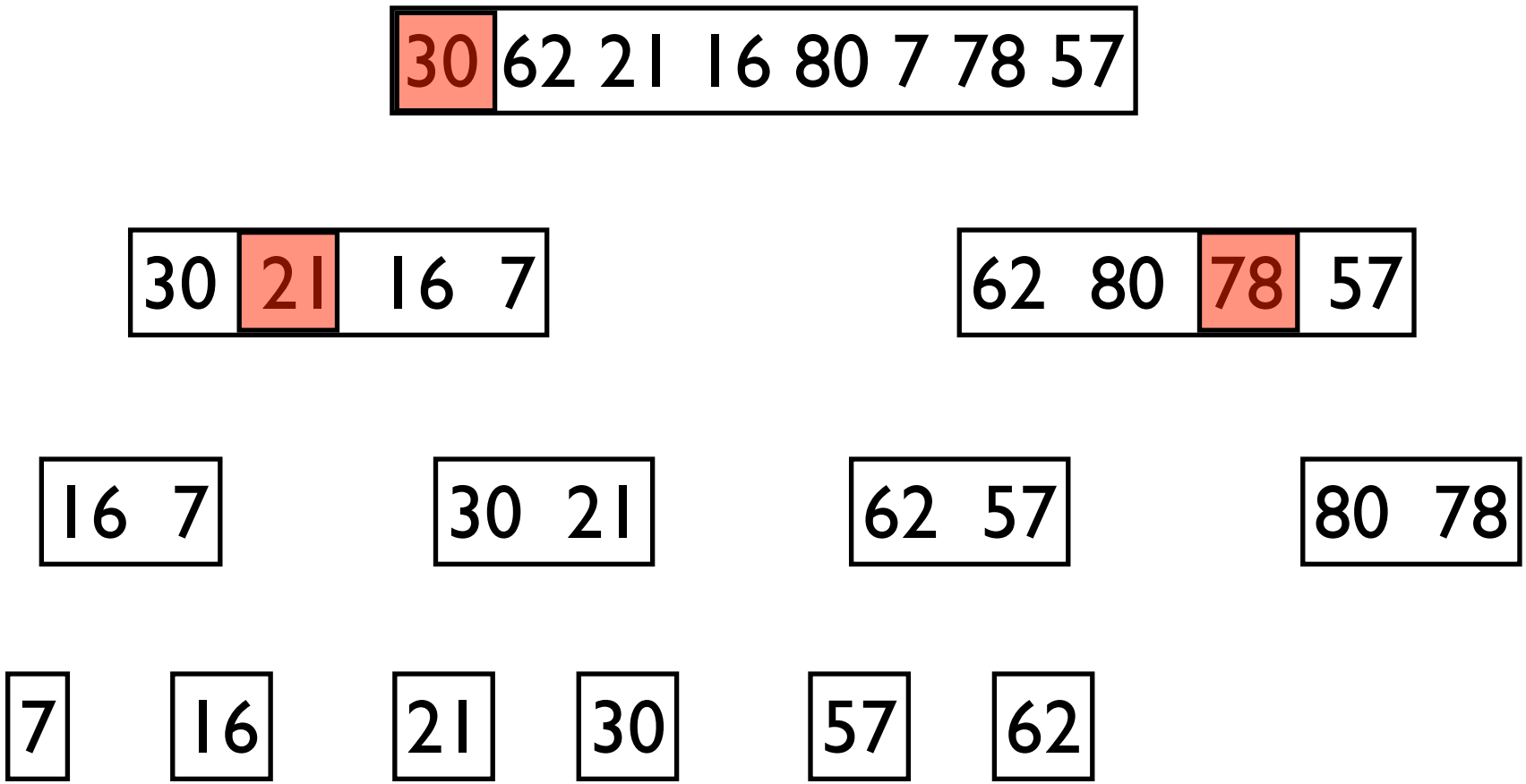
Sortieren

Quicksort - Median - Beispiel



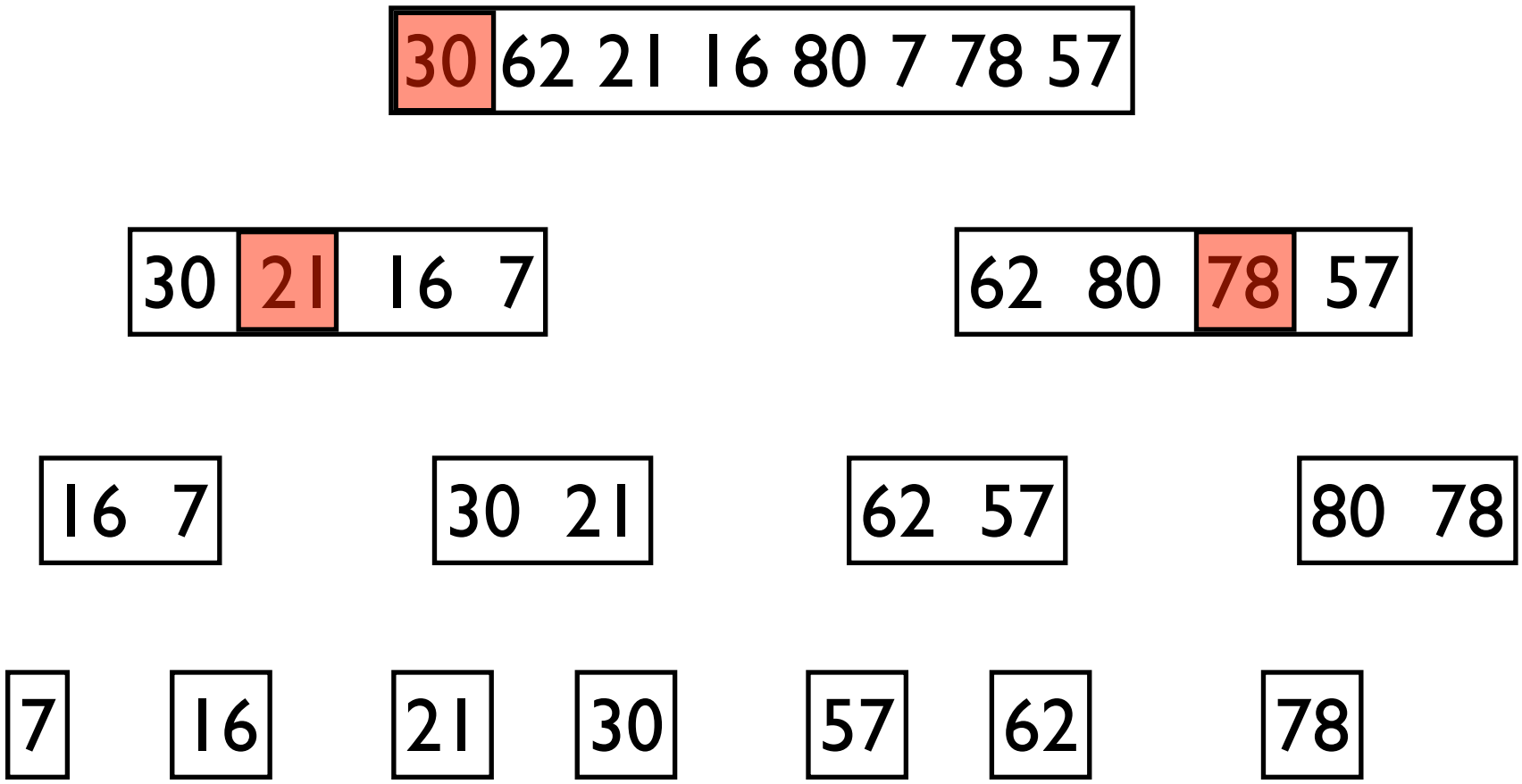
Sortieren

Quicksort - Median - Beispiel



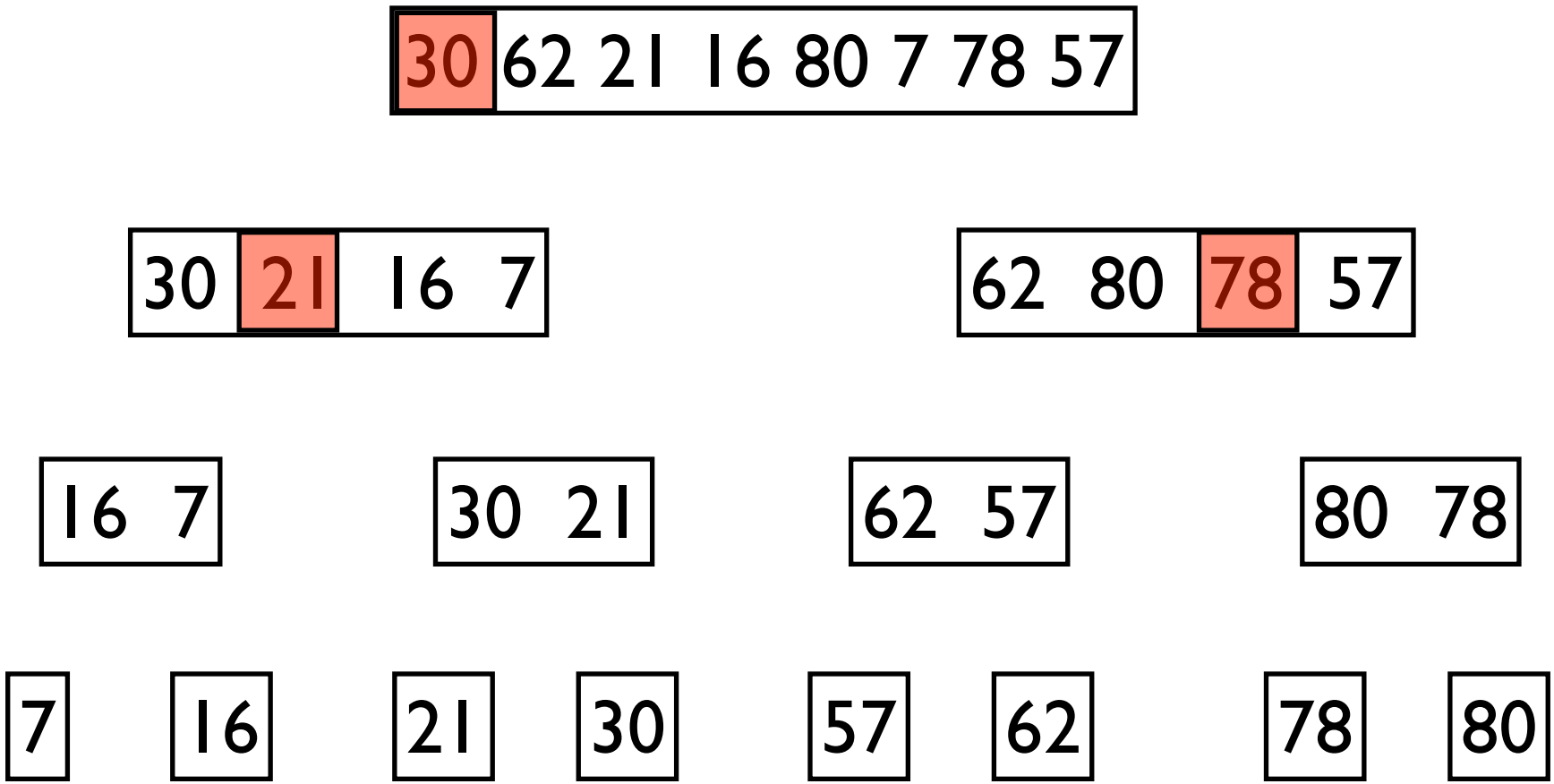
Sortieren

Quicksort - Median - Beispiel



Sortieren

Quicksort - Median - Beispiel



Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ... A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Algorithmus

```
procedure quick (l, r: 1...n);
vor i, j: 1...n; x, w: Objekt;
begin  i:=l; j:=r;
    "Bestimme Median x in A[l] ...A[r]";
    repeat
        while A[i] < x do i:= i+1;
        while A[j] > x do j:= j-1;
        if i ≤ j then
        begin
            "tausche A[i] und A[j]"; i:= i+1; j:= j-1
        end
    until i > j;
    if l < j then quick (l,j);
    if i < r then quick (i,r);
end
```

Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

30	62	21	16	80	7	78	57

Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

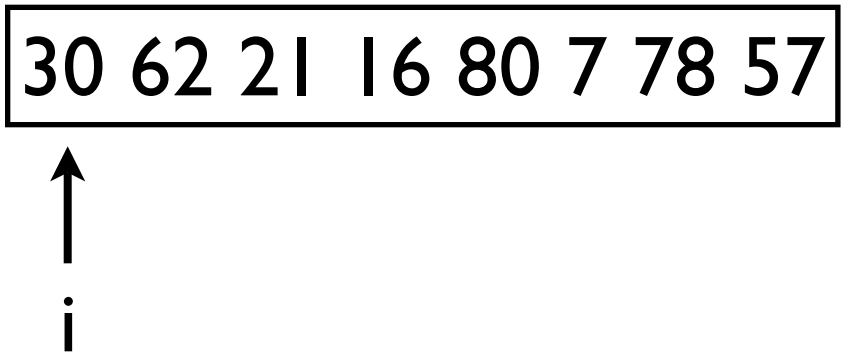
Median: 30

30	62	21	16	80	7	78	57

Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

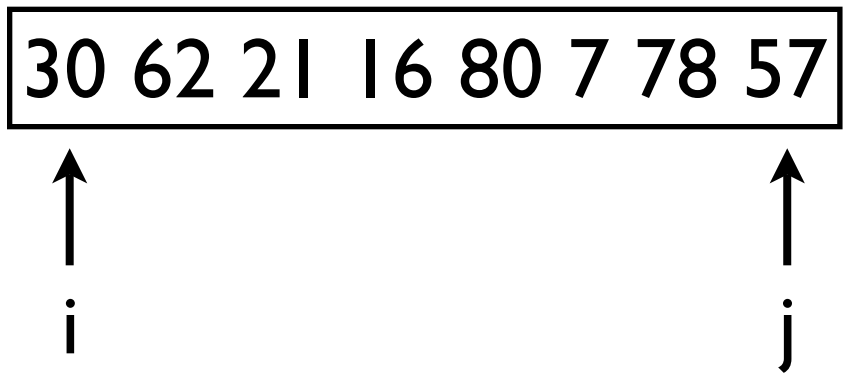
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

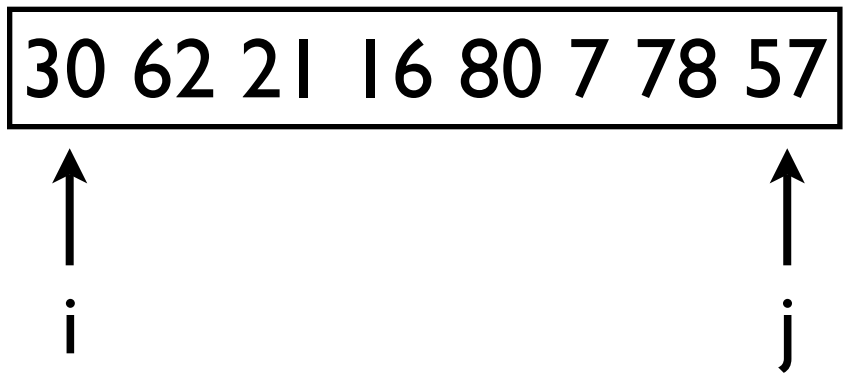
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

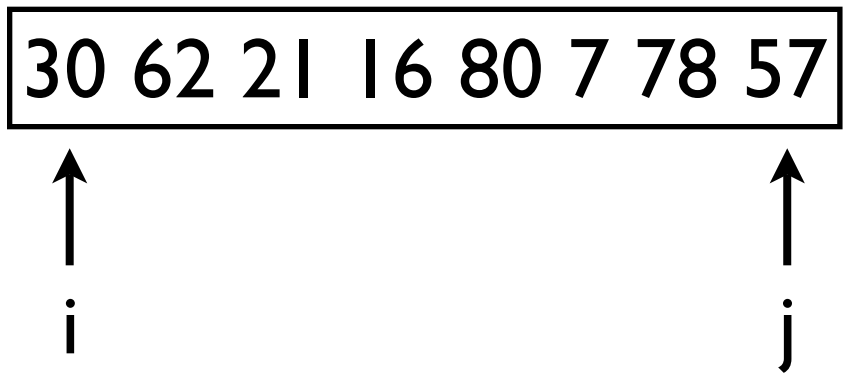
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

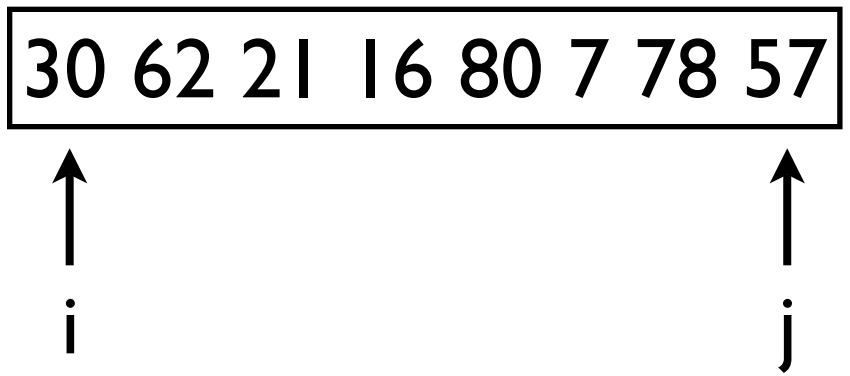
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

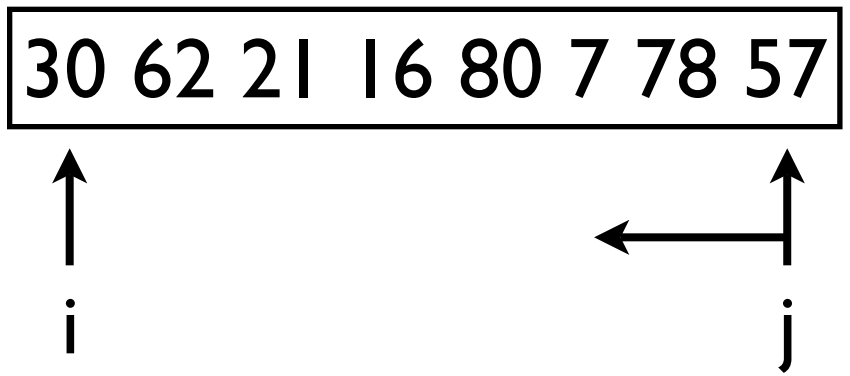
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

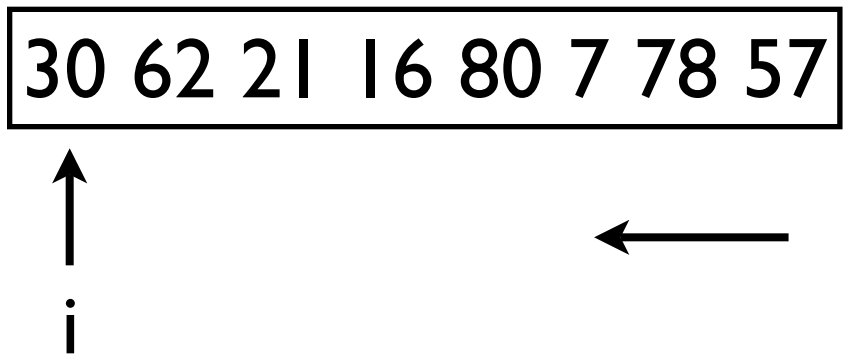
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

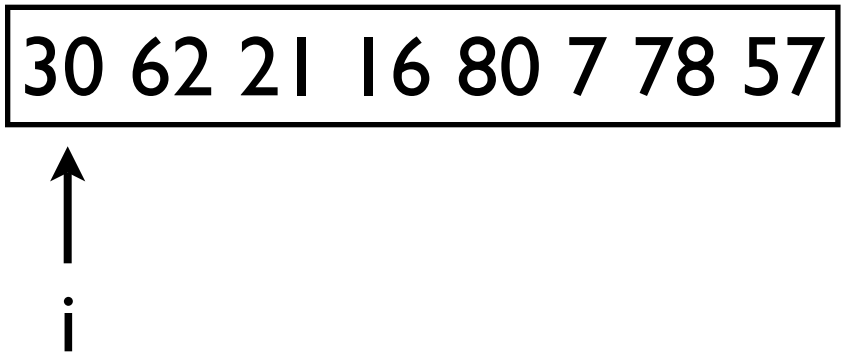
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

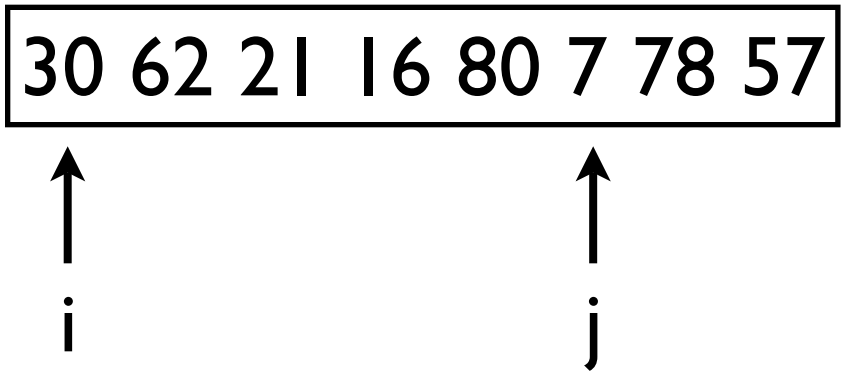
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

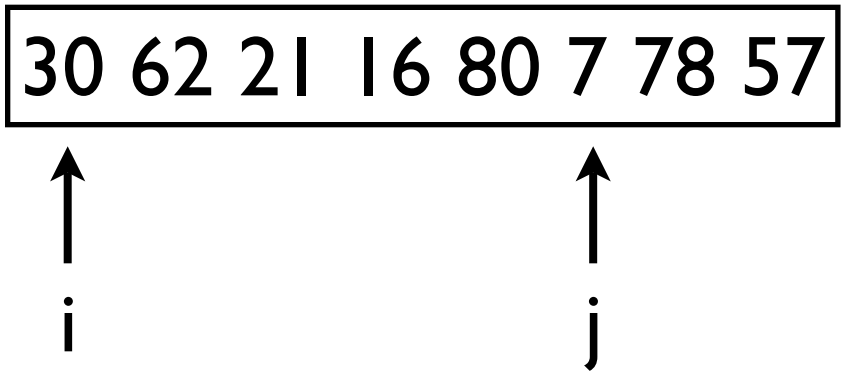
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

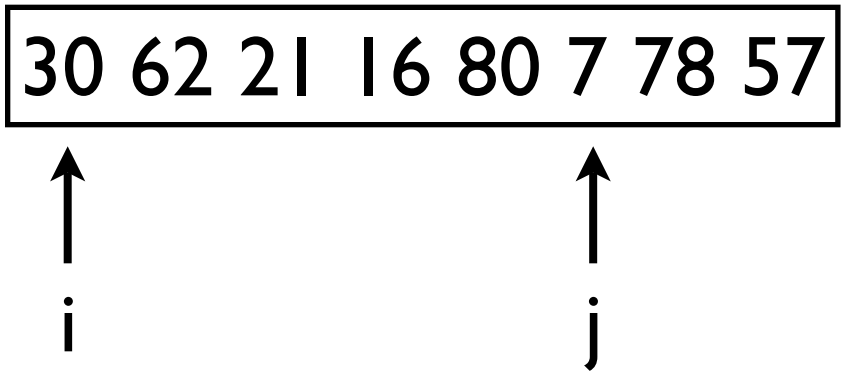
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

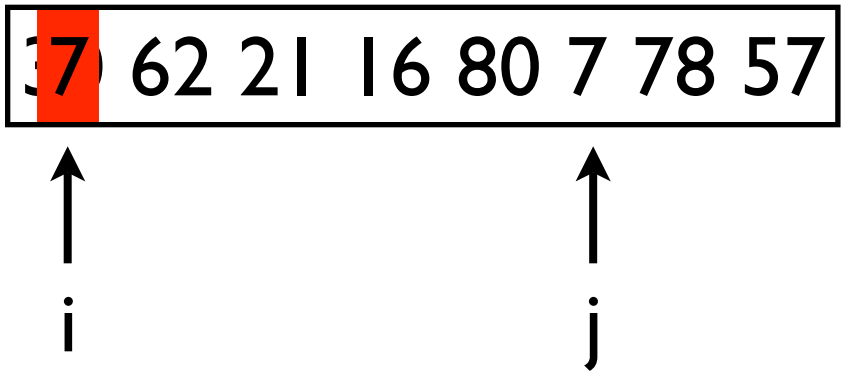
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

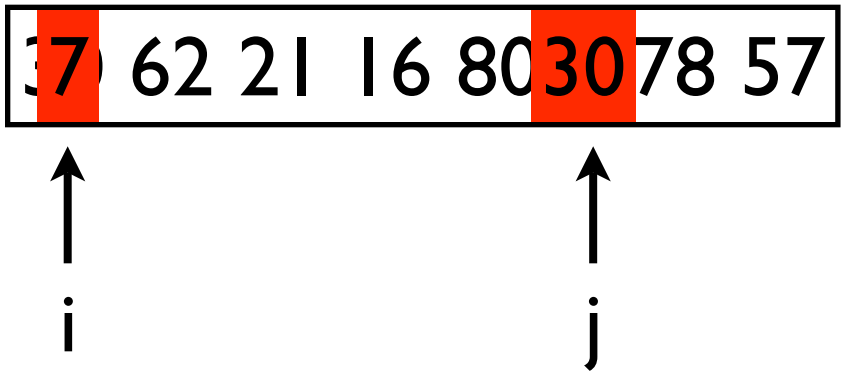
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

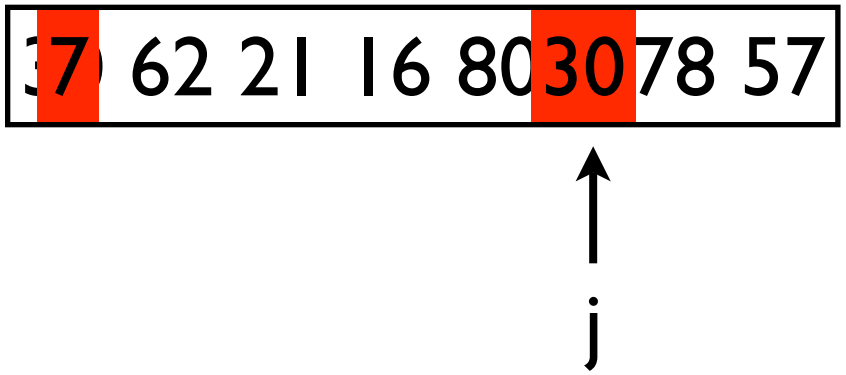
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

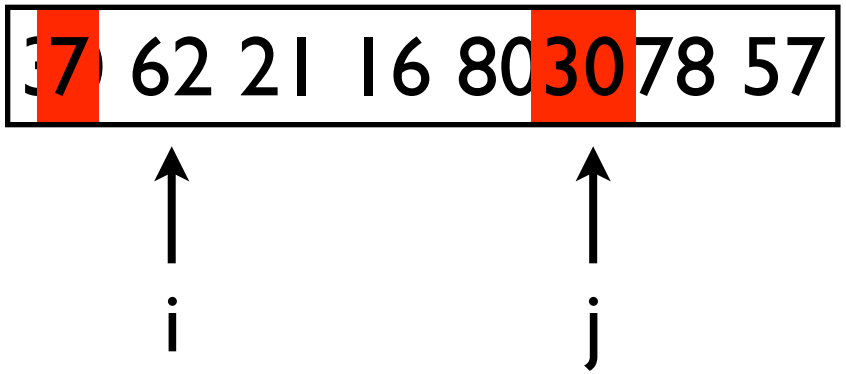
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

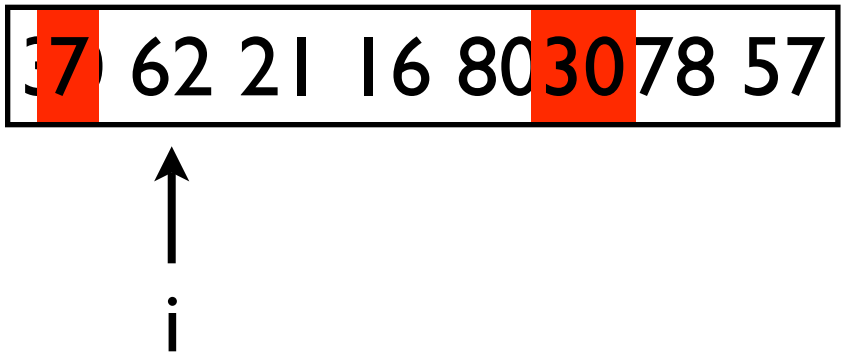
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

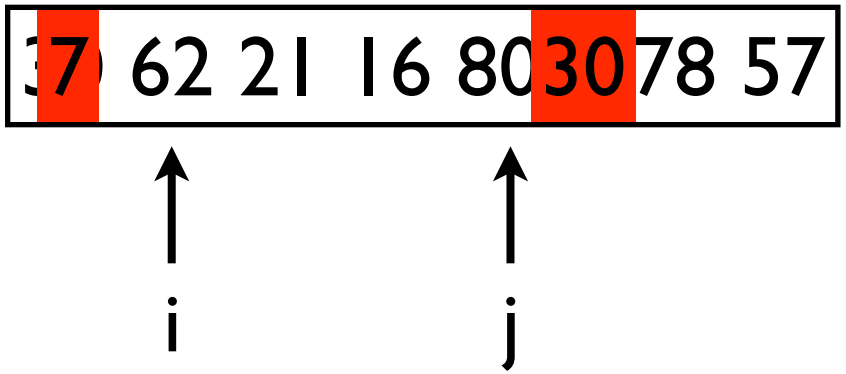
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

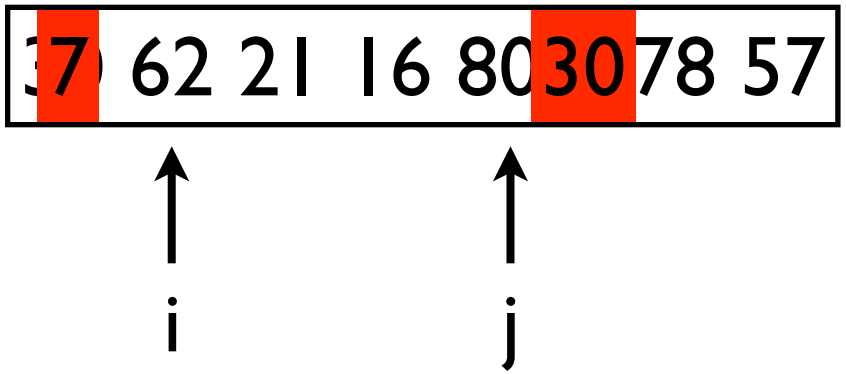
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

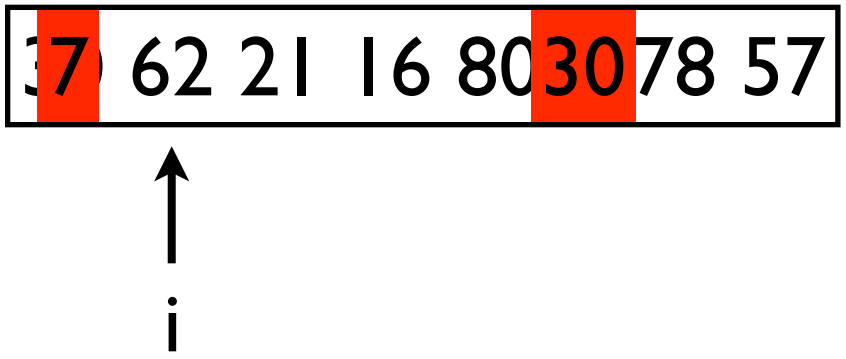
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

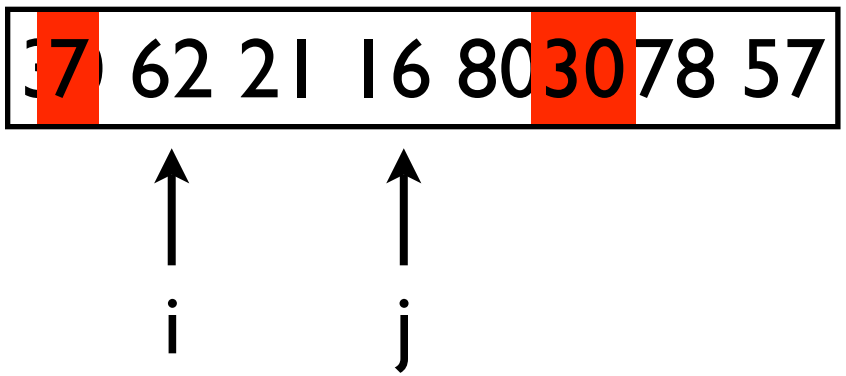
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

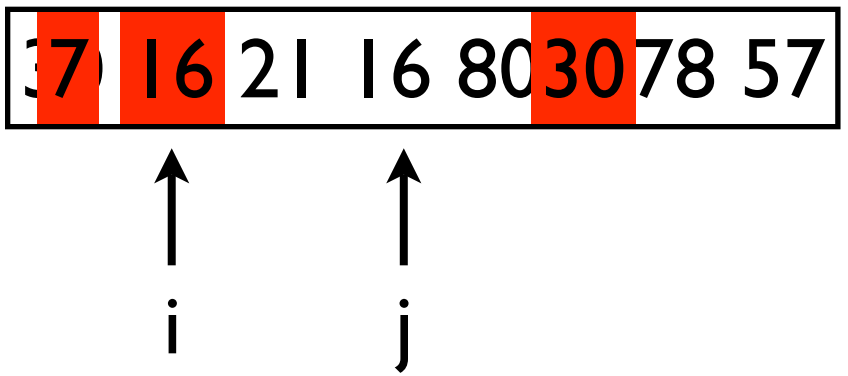
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

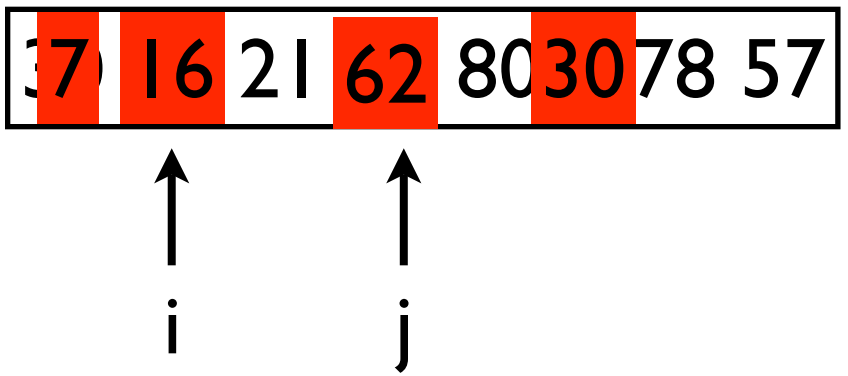
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

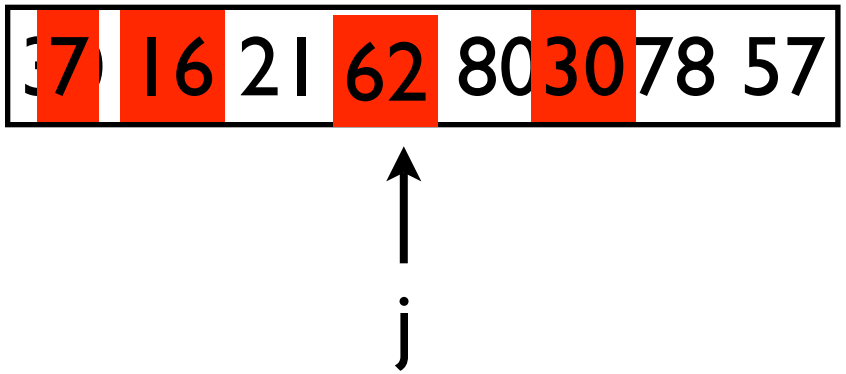
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

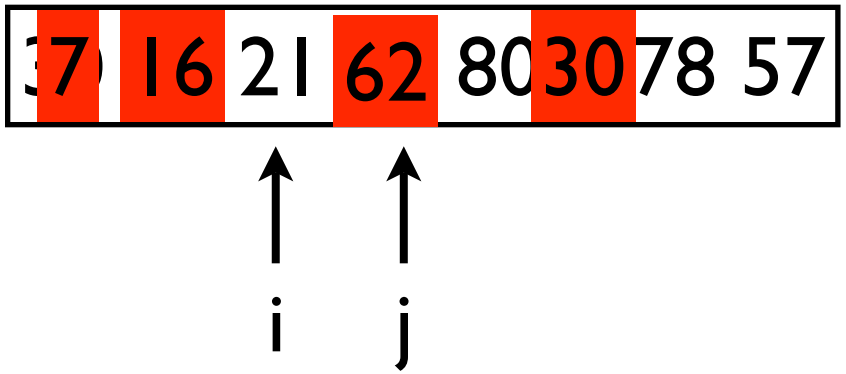
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

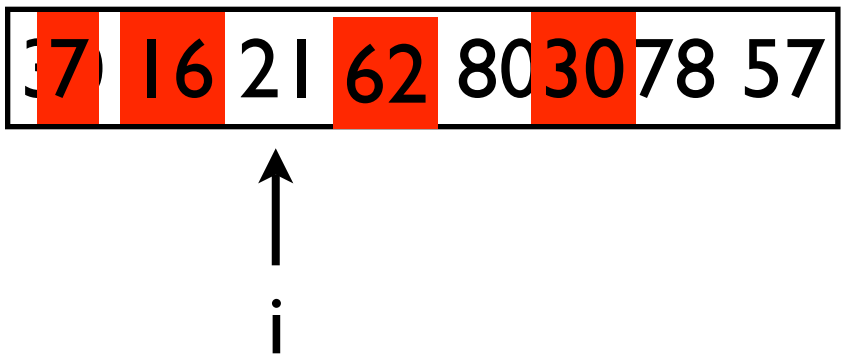
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

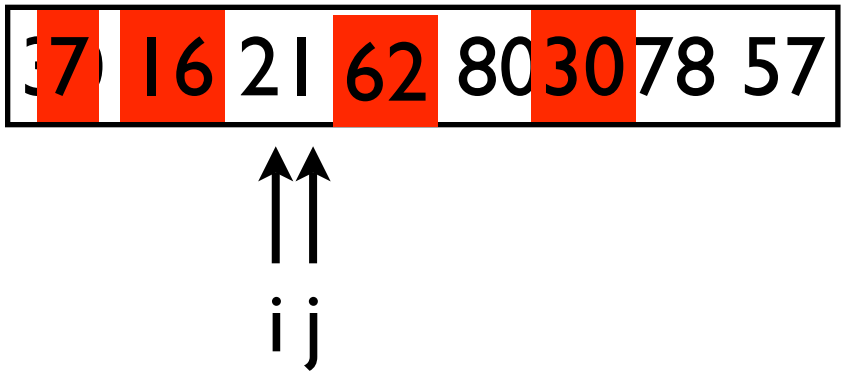
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

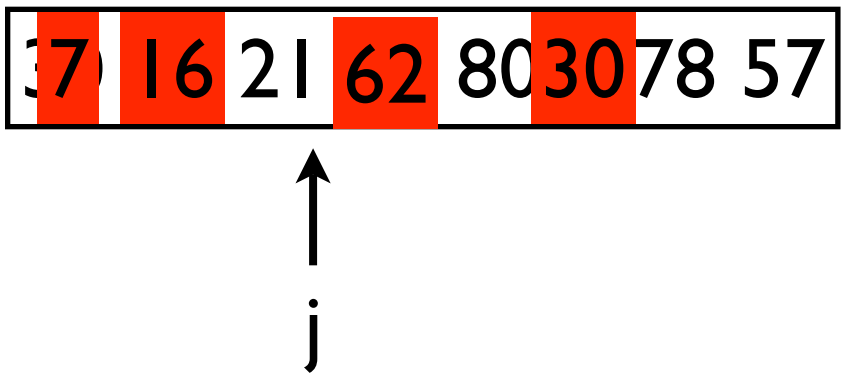
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

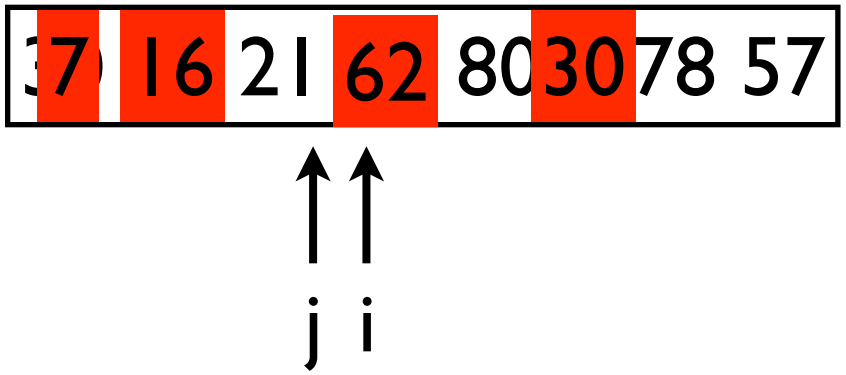
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

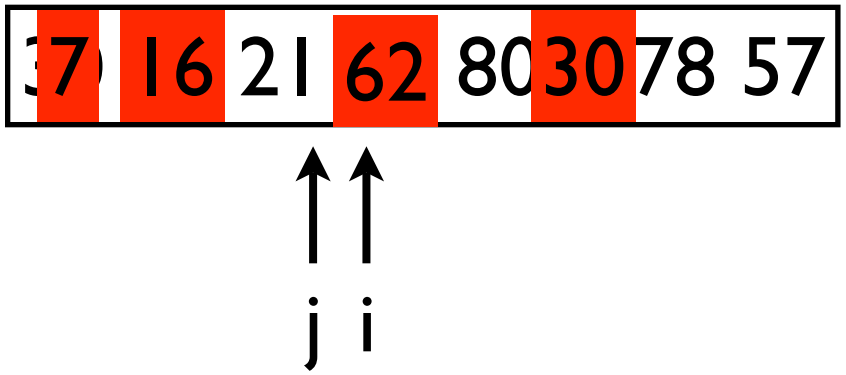
Median: 30



Sortieren
Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

Median: 30



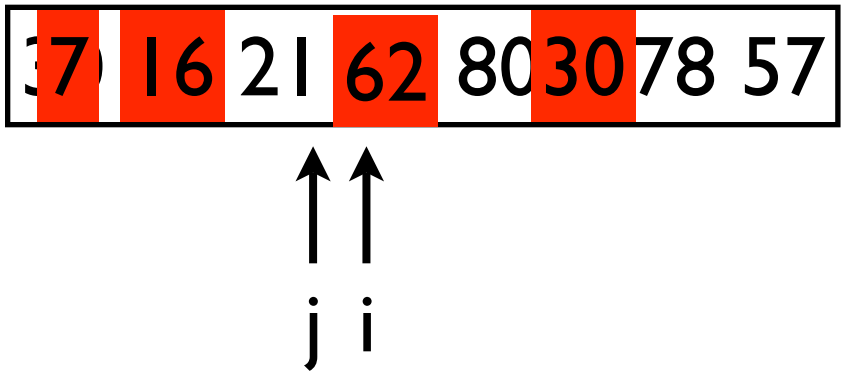
Abbruch: i>j

Sortieren

Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

Median: 30



Abbruch: i>j

7 16 21

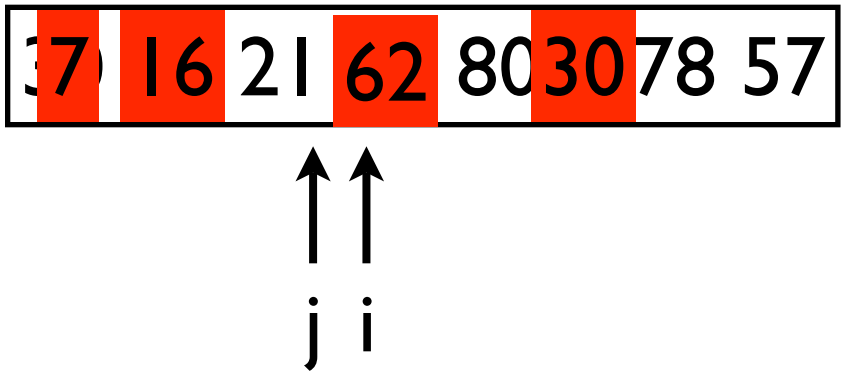
Quicksort

Sortieren

Quicksort - Beispiel

```
repeat
    while A[i] < x do i:= i+1;
    while A[j] > x do j:= j-1;
    if i ≤ j then
    begin
        "tausche A[i] und A[j]";
        i:= i+1; j:= j-1
    end
until i > j;
```

Median: 30



Abbruch: i>j

7 16 21

Quicksort

62 80 30 78 57

Quicksort

Sortieren
Quicksort - Laufzeit

- Median teilt A jeweils in zwei gleichgroße Hälften.
- Nach $O(\log_2 n)$ Teilungsvorgängen Arrays nur noch einelementig und Verfahren bricht ab.
- Der übrige Rumpf von quick benötigt

$$O((r-l)+M(r-l)) = O(n+M(n)) \text{ Zeit,}$$

wobei M(n) Zeit für Bestimmen des Medians im Feld der Länge n ist.

- Ergebnis: $T_{quick}(n)=2T_{quick}(n/2)+O(n+M(n))$

Sortieren
Quicksort - Laufzeit

- Median teilt A jeweils in zwei gleichgroße Hälften.
- Nach $O(\log_2 n)$ Teilungsvorgängen Arrays nur noch einelementig und Verfahren bricht ab.
- Der übrige Rumpf von quick benötigt

$$O((r-l)+M(r-l)) = O(n+M(n)) \text{ Zeit,}$$

wobei M(n) Zeit für Bestimmen des Medians im Feld der Länge n ist.

- Ergebnis: $T_{quick}(n)=2T_{quick}(n/2)+O(n+M(n))$

Sortieren
Quicksort - Medianbestimmung

- Medianbestimmung geht in linearer Zeit (s. Buch von Ottmann/Widmayer)
- Folglich:

$$T_{quick}(n)=2T_{quick}(n/2)+O(n)=O(n \log n)$$

- Problem: Medianbestimmung sehr aufwendig

Sortieren
Quicksort - Pivotelement

- Abschwächung: Wähle nicht in jedem Rekursionsschritt den Median, sondern irgendein Element x
- Hoffnung: x teilt Feld in zwei etwa gleichgroße Teile
- x heißt **Pivotelement**

Sortieren
Quicksort - Pivotelement

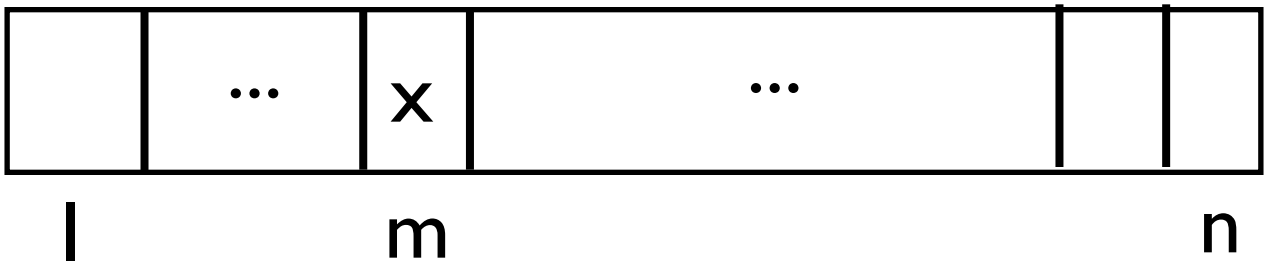
- Abschwächung: Rekursionsstopp, wenn Feld nur noch irgendein Element enthält
- Hoffnung: x teilt Feld in zwei etwa gleichgroße Teile
- x heißt **Pivotelement**

analoge Idee beim AVL-Baum: "weniger ausgeglichener Baum ist genauso gut wie exakt ausgeglichener."

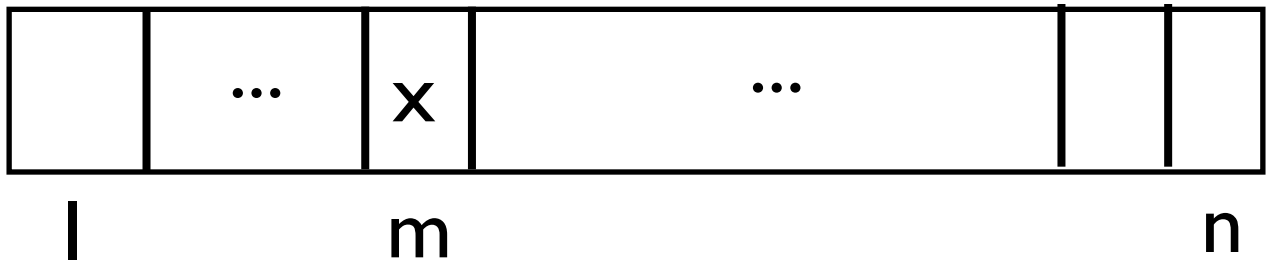
Sortieren
Quicksort - Pivotelement

- Laufzeit mit Pivotelement x:

$$T(n) = O(n) + T(m) + T(n-m)$$



Sortieren
Quicksort - Laufzeit



- Sonderfälle:
  - Pivotelement x ist jeweils der Median:

    $$T(n) = O(n)+2T(n/2) = O(n \log n)$$

  - Pivotelement x ist jeweils das kleinste (oder größte) Element:

    $$T(n) = O(n)+T(1)+T(n-1) = O(n^2)$$

    Worst case: Hier wird Quicksort zum **Slowsort** mit unbrauchbarer Laufzeit $O(n^2)$.
  - Fall tritt bei schon sortiertem Feld auf.

Sortieren
Quicksort - Analyse

- Wie oft tritt der worst case auf, wenn x in jedem rekursiven Aufruf zufällig gewählt wird?
- Annahmen:
  1. Zu sortieren sind die Zahlen 1, ..., n.
  2. Alle n! Anordnungen der Zahlen im Ausgangsfeld sind gleich wahrscheinlich
  3. Als Pivotelement wählen wir immer a_n.

Sortieren
Quicksort - Analyse

- **Folgerung**: Wird quick für $a_1, ..., a_n$ aufgerufen, so folgt aus den Annahmen, daß jedes k mit $1 \leq k \leq n$ mit gleicher Wahrscheinlichkeit 1/n an Position a_n auftritt und daher als Pivotelement gewählt wird.
- Anschließend werden zwei Folgen der Längen k-1 und n-k erzeugt
- Zeige nun: Teilfolgen sind wieder zufällig, so daß im *Mittelwert* eine Laufzeit von O(n log n) gehalten wird.

Sortieren
Quicksort - mittlere Laufzeit

Satz

Quicksort benötigt im Mittel eine Laufzeit von

$$O(n \log n)$$

wenn auf jeder Stufe der Rekursion das Pivotelement zufällig gewählt wird.

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

Mittelwert des
Aufwands für alle
Aufrufe je nach Teilung

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

Aufwand für Teilung

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

$$T(0)=0, T(1)=c$$

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

$$T(0)=0, T(1)=c$$

$$n \geq 2: T(n) \leq (2/n) \sum_{k=1}^{n-1} T(k) + bn$$

Sortieren

Quicksort - mittlere Laufzeit

Beweis

Berechne den Mittelwert der Laufzeit für Quicksort:

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(k-1)+T(n-k)) + bn$$

$$T(0)=0, T(1)=c$$

$$n \geq 2: T(n) \leq (2/n) \sum_{k=1}^{n-1} T(k) + bn$$

Zeige nun: $T(n) \leq cn \log n$ für hinreichend großes c.

Sortieren
Quicksort - mittlere Laufzeit

Beweis durch Induktion:
Induktionsanfang: $n=1,2$ ist klar.
Induktionsvoraussetzung: Sei die Aussage wahr für alle $i<n$, also $T(i) = ci \log i$ für $i<n$.

Dann gilt:

$$T(n) \leq (2/n) \sum_{k=1}^{n-1} T(k) + bn$$
$$= (2c/n) \sum_{k=1}^{n-1} (k \log k) + bn$$
$$= (2c/n) \sum_{k=1}^{n/2} (k \log k) + \sum_{k=1}^{n/2-1} ((n/2)+k) \log ((n/2)+k))) + bn$$

Sortieren

Quicksort - mittlere Laufzeit

Beweis durch Induktion:
Induktionsanfang: $n=1,2$ ist klar.
Induktionsvoraussetzung: Sei die Aussage wahr für alle $i<n$, also $T(i) = ci \log i$ für $i<n$.

Dann gilt:

$$T(n) \leq (2/n) \sum_{k=1}^{n-1} T(k) + bn$$
$$= (2c/n) \sum_{k=1}^{n-1} (k \log k) + bn$$
$$= (2c/n) \sum_{k=1}^{n/2} (k \log k) + \sum_{k=1}^{n/2-1} ((n/2)+k) \log ((n/2)+k))) + bn$$

$\leq \log n-1$

Sortieren
Quicksort - mittlere Laufzeit

Beweis durch Induktion:
Induktionsanfang: $n=1,2$ ist klar.
Induktionsvoraussetzung: Sei die Aussage wahr für alle $i<n$, also $T(i) = ci \log i$ für $i<n$.

Dann gilt:

$$T(n) \leq (2/n) \sum_{k=1}^{n-1} T(k) + bn$$
$$= (2c/n) \sum_{k=1}^{n-1} (k \log k) + bn$$
$$= (2c/n) \sum_{k=1}^{n/2} (k \log k) + \sum_{k=1}^{n/2-1} ((n/2)+k) \log ((n/2)+k))) + bn$$

Sortieren

Quicksort - mittlere Laufzeit

Beweis durch Induktion:
Induktionsanfang: n=1,2 ist klar.
Induktionsvoraussetzung: Sei die Aussage wahr für alle $i<n$, also $T(i) = ci \log i$ für $i<n$.

Dann gilt:

$$T(n) \leq (2/n) \sum_{k=1}^{n-1} T(k) + bn$$

$$= (2c/n) \sum_{k=1}^{n-1} (k \log k) + bn$$

$$= (2c/n) \sum_{k=1}^{n/2} (k \log k) \qquad \leq \log (n-1)$$

$$+ \sum_{k=1}^{n/2-1} ((n/2)+k) \log ((n/2)+k))) + bn$$

Sortieren
Quicksort - mittlere Laufzeit

Beweis durch Induktion:
Induktionsanfang: $n=1,2$ ist klar.
Induktionsvoraussetzung: Sei die Aussage wahr für alle $i<n$, also $T(i) = ci \log i$ für $i<n$.

Dann gilt:

$$T(n) \leq (2/n) \sum_{k=1}^{n-1} T(k) + bn$$
$$= (2c/n) \sum_{k=1}^{n-1} (k \log k) + bn$$
$$= (2c/n) \sum_{k=1}^{n/2} (k \log k) + \sum_{k=1}^{n/2-1} ((n/2)+k) \log ((n/2)+k))) + bn$$

Sortieren

Quicksort - mittlere Laufzeit

$$= (2c/n)\sum_{k=1}^{n/2}(k \log k) + \sum_{k=1}^{n/2-1}((n/2)+k) \log ((n/2)+k)))+bn$$

$$\leq (2c/n)(\log n-1) \sum_{k=1}^{n/2} k + (\log (n-1))\sum_{k=1}^{n/2-1}((n/2)+k)+bn$$

$$\leq \frac{2c}{n}\left[\frac{n}{4}\left(\frac{n}{2}+1\right)\log n - \frac{n^2}{8} - \frac{n}{4} + \left(\frac{3n^2}{8} - \frac{3n}{4}\right) \log n\right]+bn$$

$$= \frac{2c}{n}\left[\left(\frac{n^2}{2} - \frac{n}{2}\right)\log n - \frac{n^2}{8} - \frac{n}{4}\right]+bn$$

$$= cn \log n - c \log n - \frac{cn}{4} - \frac{c}{2} + bn$$

Sortieren

Quicksort - mittlere Laufzeit

$$= (2c/n)\sum_{k=1}^{n/2}(k \log k) + \sum_{k=1}^{n/2-1}((n/2)+k) \log ((n/2)+k)))+bn$$

$$\leq (2c/n)(\log n-1) \sum_{k=1}^{n/2} k + (\log (n-1))\sum_{k=1}^{n/2-1}((n/2)+k)+bn$$

$$\leq \frac{2c}{n}\left[\frac{n}{4}\left(\frac{n}{2}+1\right)\log n - \frac{n^2}{8} - \frac{n}{4} + \left(\frac{n^2}{8} - \frac{n}{4}\right) \log n\right]+bn$$

$$= \frac{2c}{n}\left[\left(\frac{n^2}{2} - \frac{n}{2}\right)\log n - \frac{n^2}{8} - \frac{n}{4}\right]+bn$$

$$= cn \log n - c \log n - \frac{cn}{4} - \frac{c}{2} + bn \qquad \geq 0$$

Sortieren

Quicksort - mittlere Laufzeit

$$= cn \log n - c \log n - \frac{cn}{4} - \frac{c}{2} + bn$$

$$\leq cn \log n - \frac{cn}{4} - \frac{c}{2} + bn$$

Für $c \geq 4b$ gilt dann: $T(n) \leq cn \log n$.

Ergebnis: Im Mittel benötigt Quicksort zum Sortieren von n Zahlen $O(n \log n)$ Vergleiche.

Sortieren
Quicksort - Praxis der Pivotbestimmung

- Praxis der Pivotbestimmung: Wähle Pivotelement
  - aufgrund einer Stichprobe
  - zufällig
  - als Median der Elemente a_l, $a_{(l+r)/2}$, a_r

Sortieren
Quicksort - Speicherbedarf

- Speicherbedarf wird vor allem durch die Rekursionstiefe bestimmt:
  - im schlimmsten Fall: O(n)
  - im mittleren Fall: O(log n)
- Durch geschickte Implementierung läßt sich der Stack auch im worst-case auf O(log n) begrenzen.

Sortieren
Bucketsort

- Sortierverfahren, das **ohne** Vergleiche auskommt und besondere Voraussetzungen an die zu sortierenden Objekte stellt.
- Geht das überhaupt???? - Vgl. Beweis zur unteren Schranke für Sortieren in Kapitel 3.

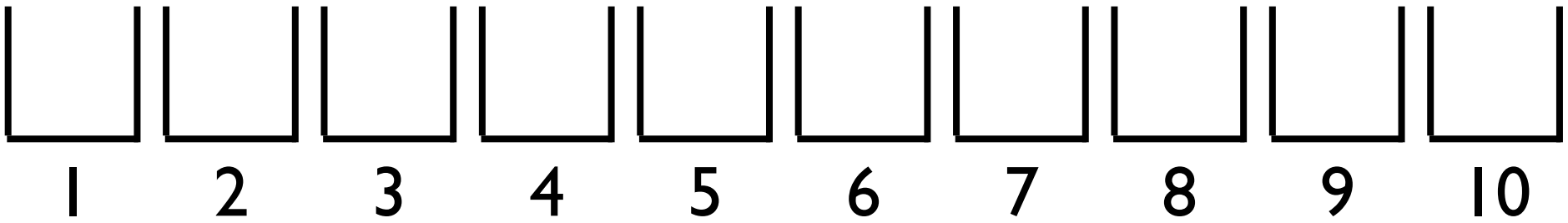
Sortieren
Bucketsort - Grundprinzip

- Idee:
  - Lege für jedes zu sortierende Objekt ein Bucket (Behälter, Eimer, Fach) an
  - Lege jedes Objekt in das zugehörige Bucket (gleiche Objekte in denselben Bucket)
- Gib die Buckets der Ordnung nach aus
- Voraussetzung: Grundtyp der zu sortierenden Elemente endlich (nicht zu groß)

Sortieren

Bucketsort - Beispiel

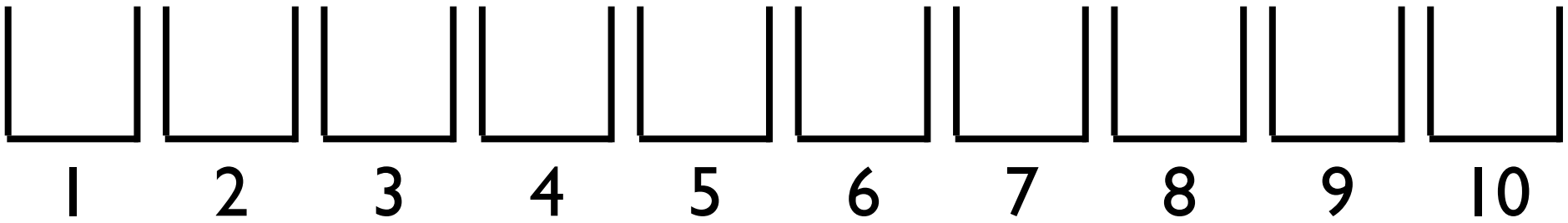
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

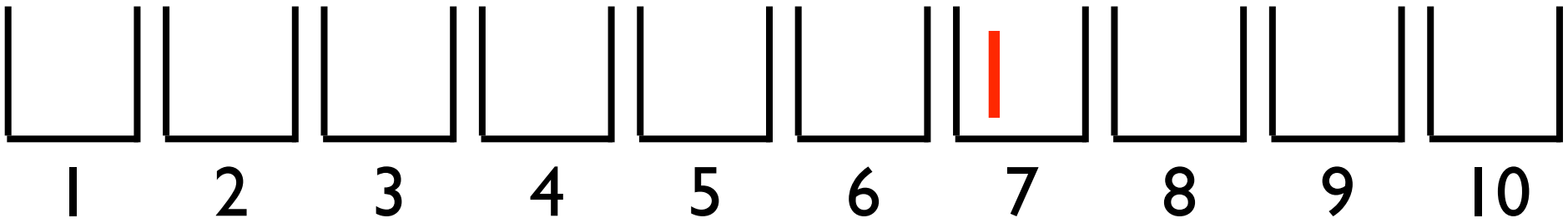
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

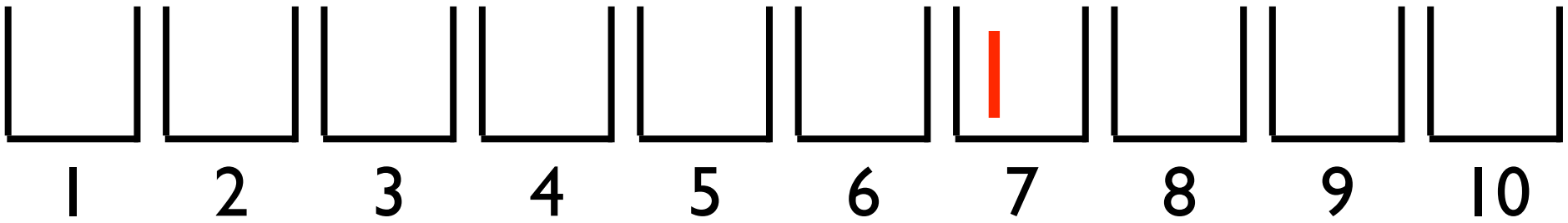
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

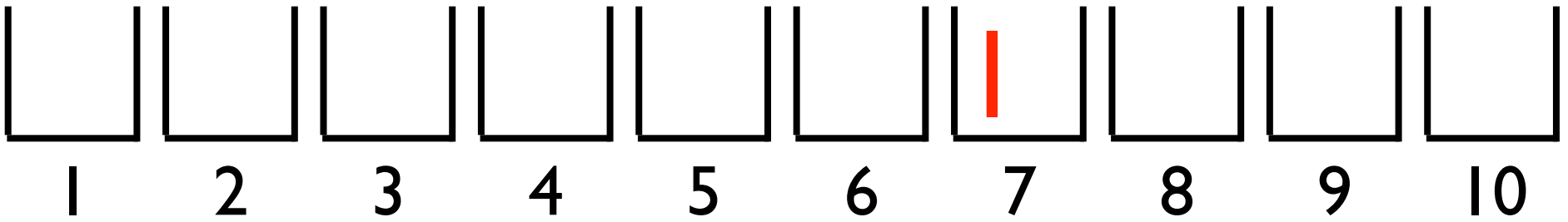
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

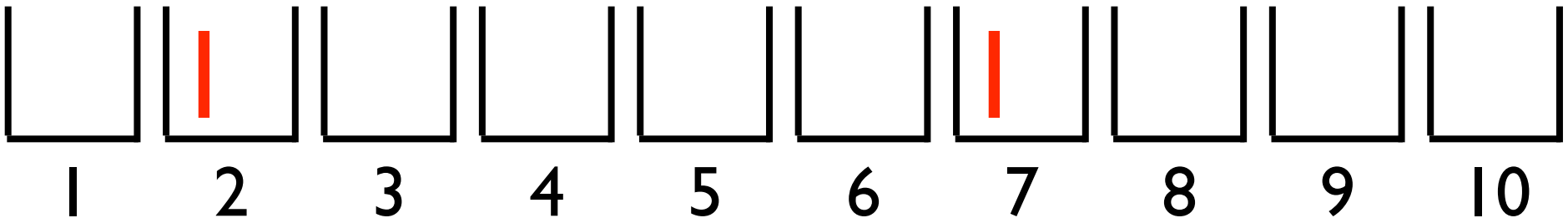
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

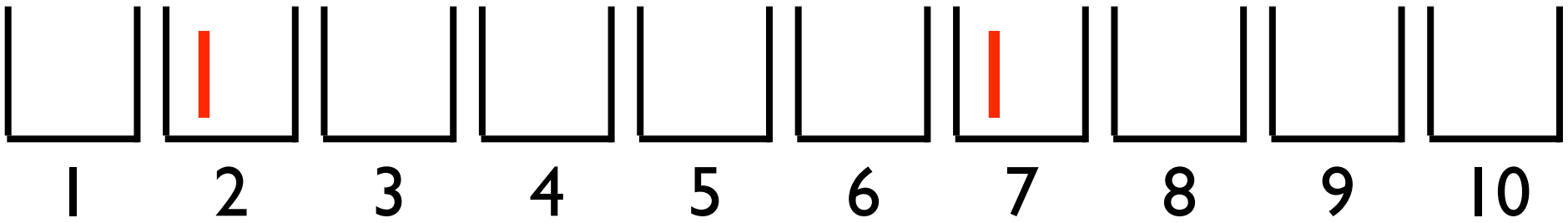
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

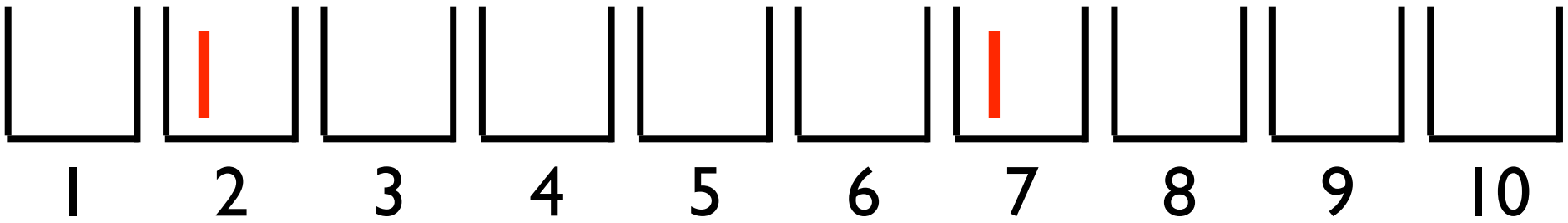
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

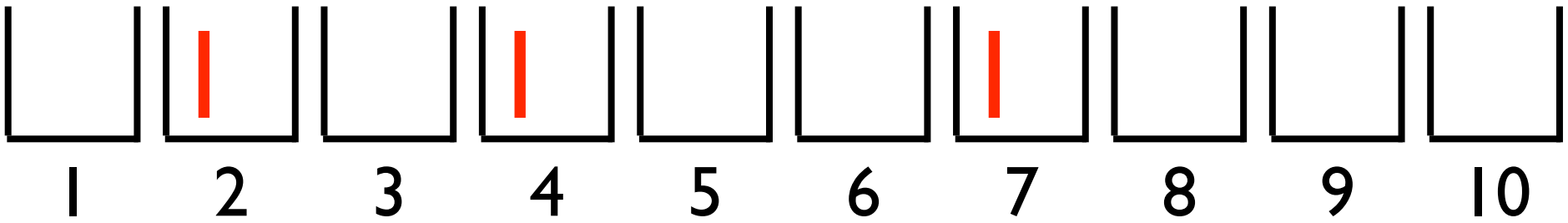
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

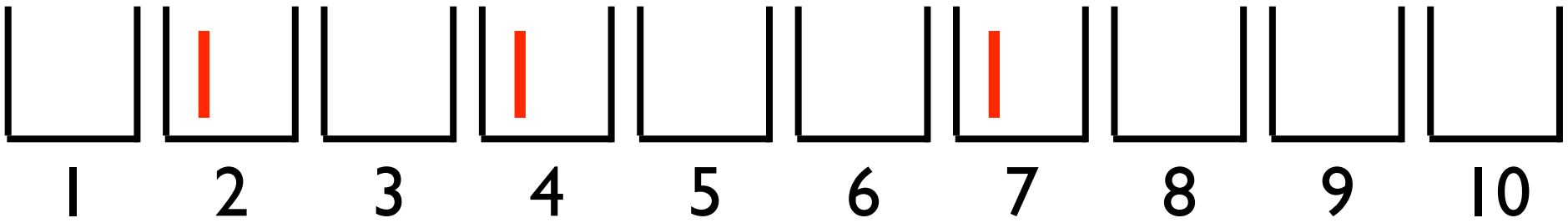
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

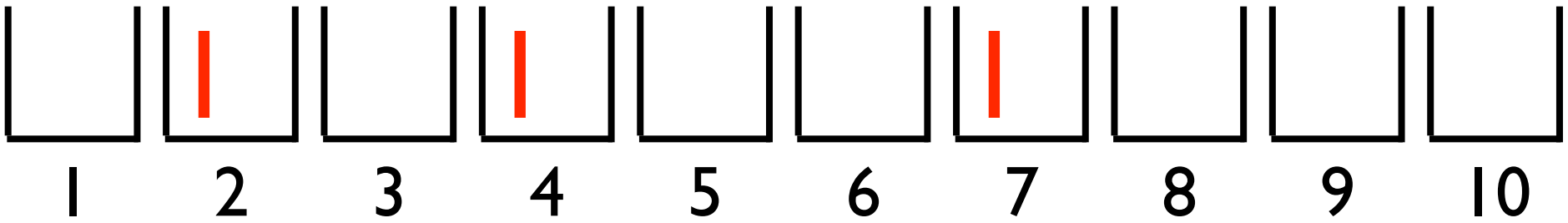
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

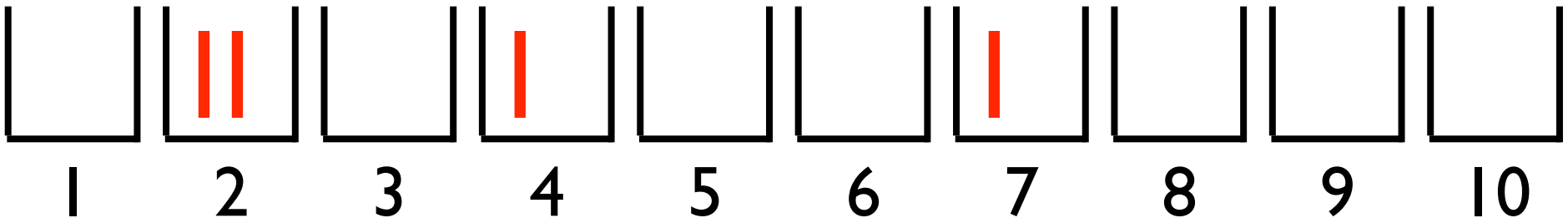
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

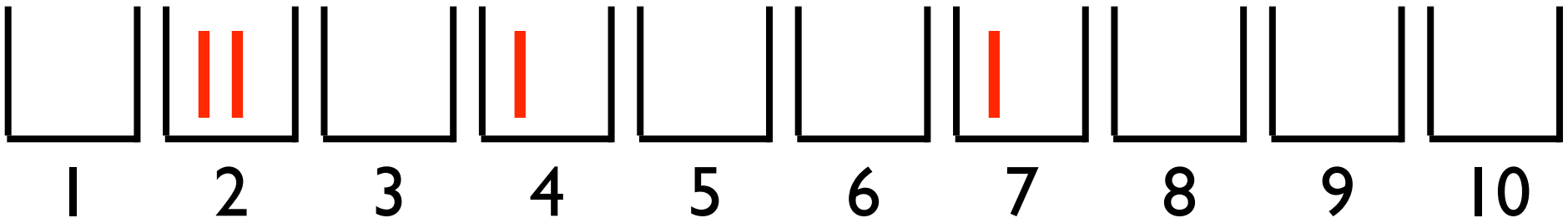
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

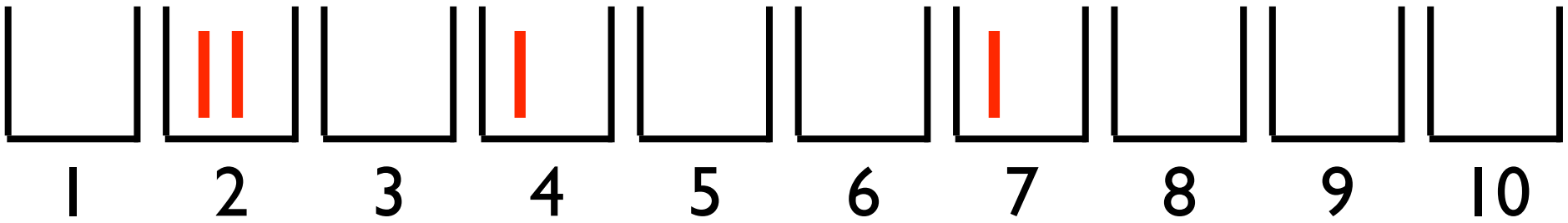
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

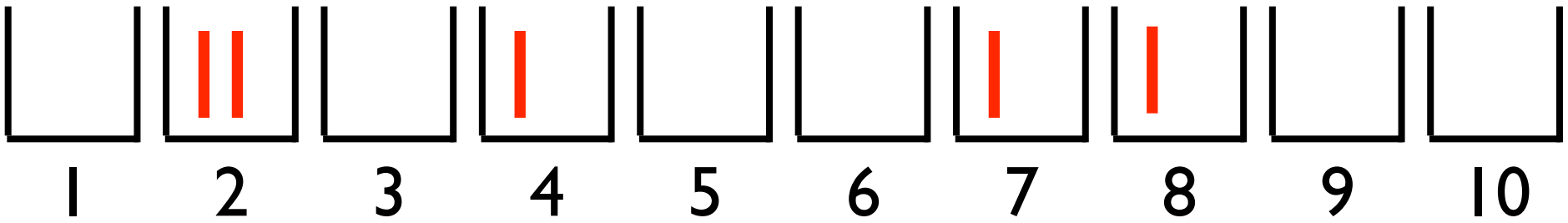
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

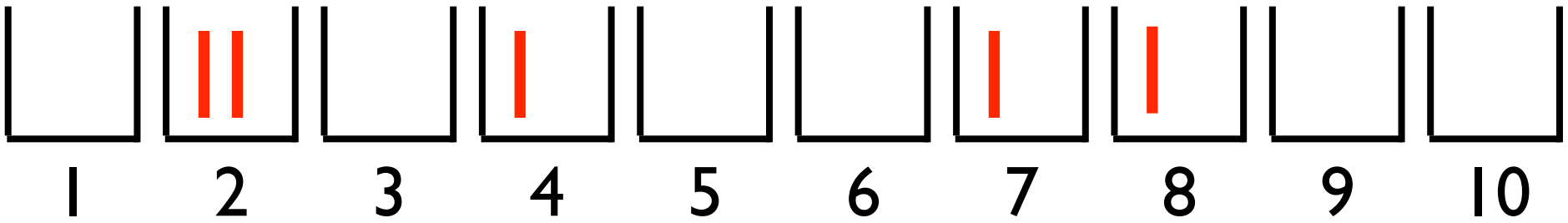
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

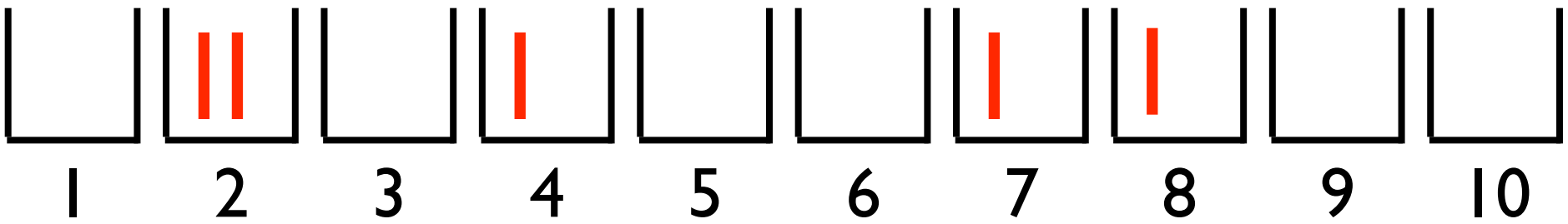
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

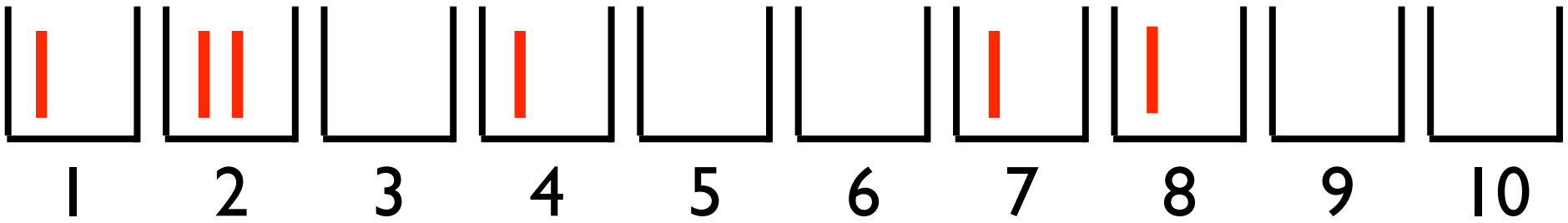
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

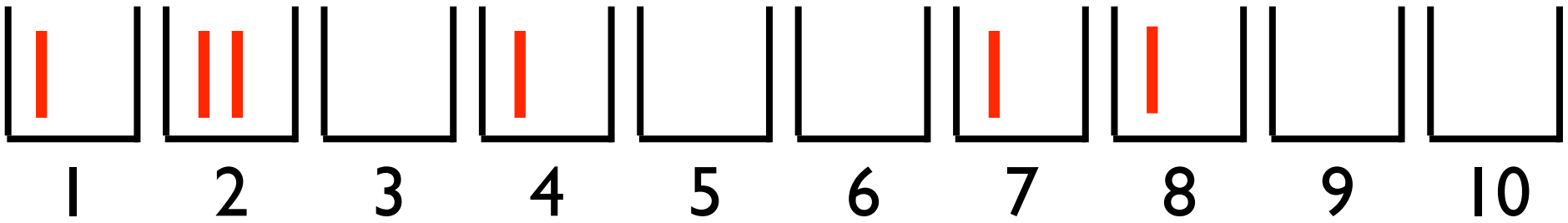
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

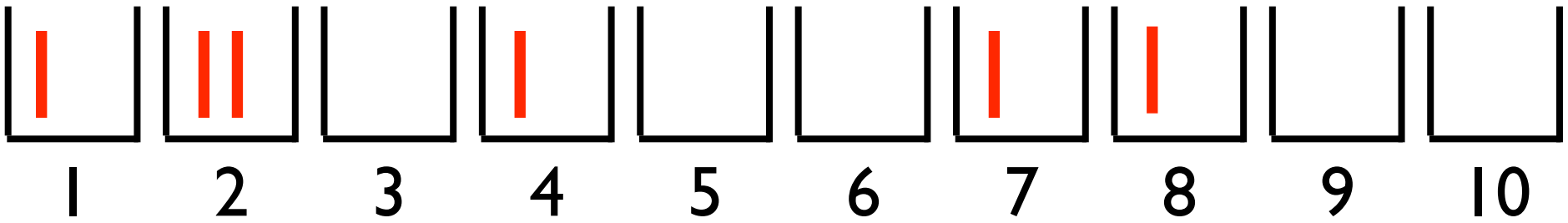
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

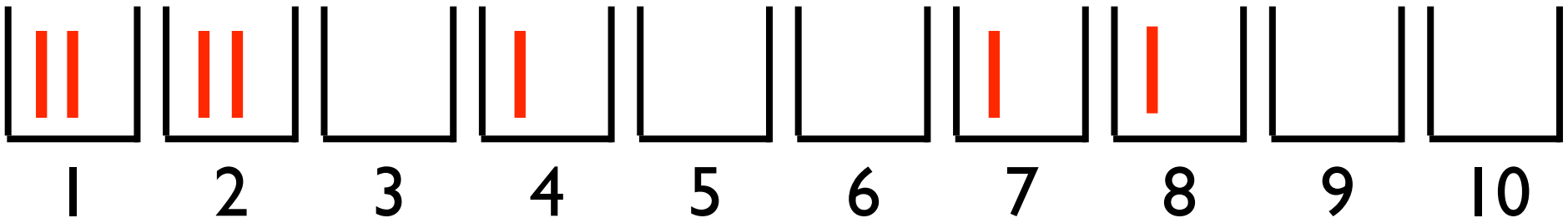
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

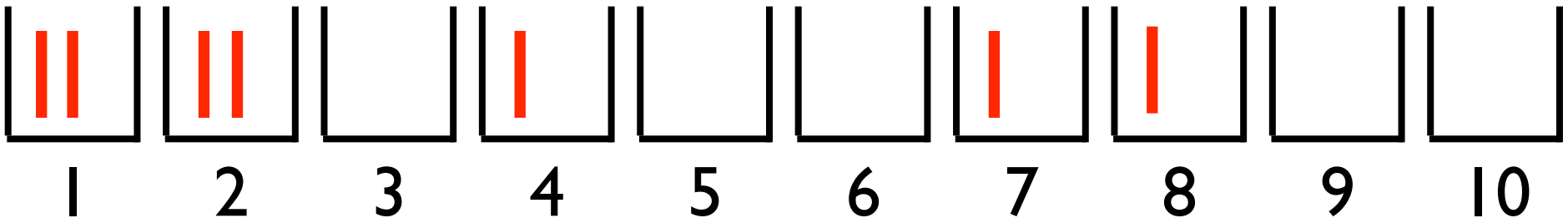
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

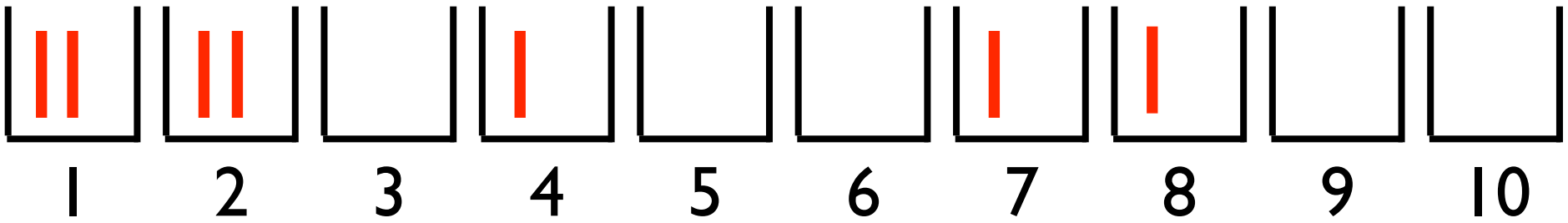
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

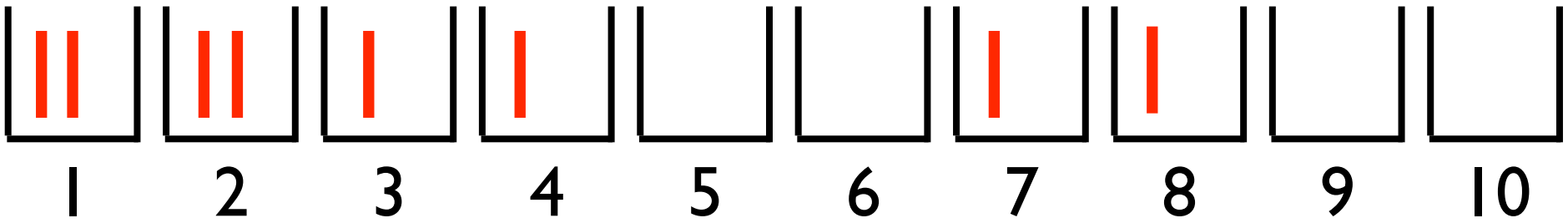
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

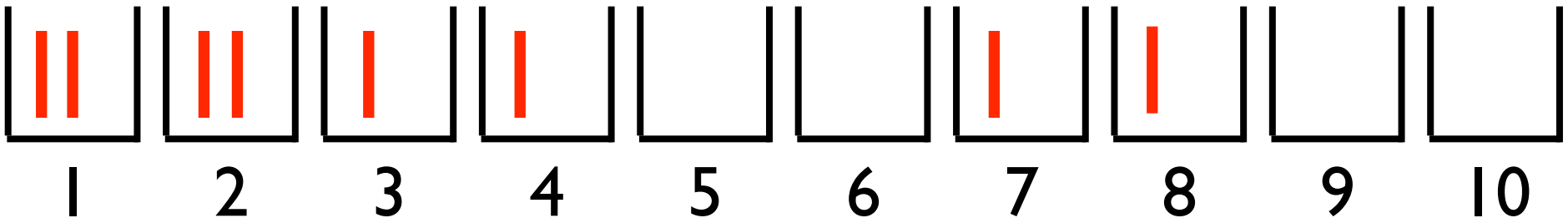
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

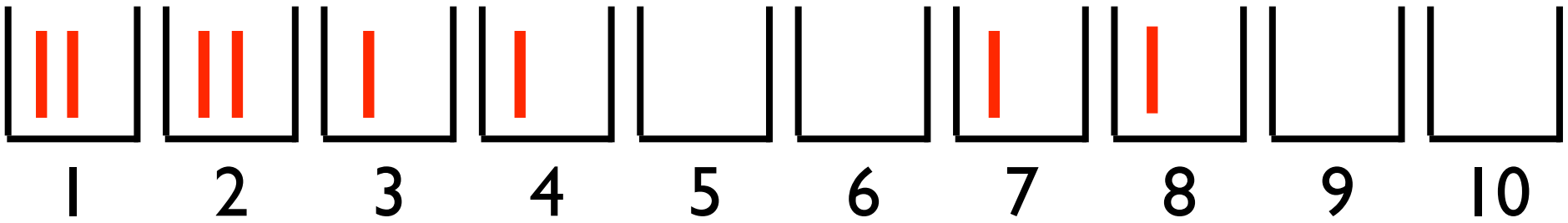
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

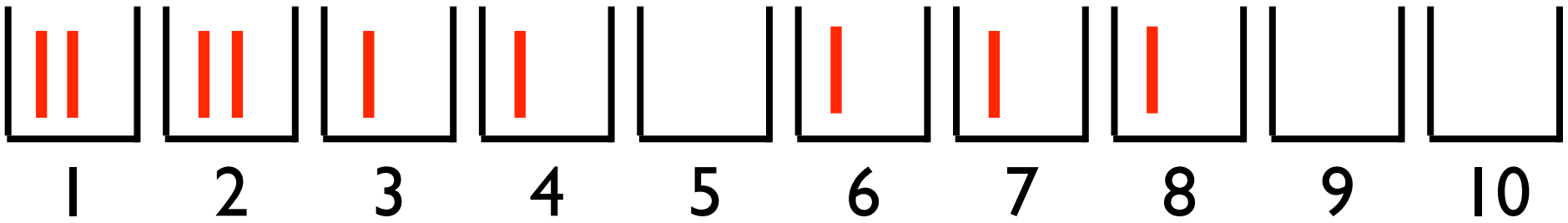
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

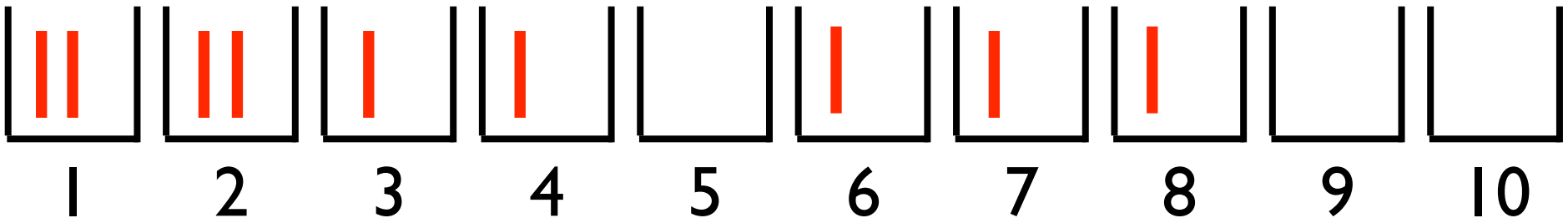
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

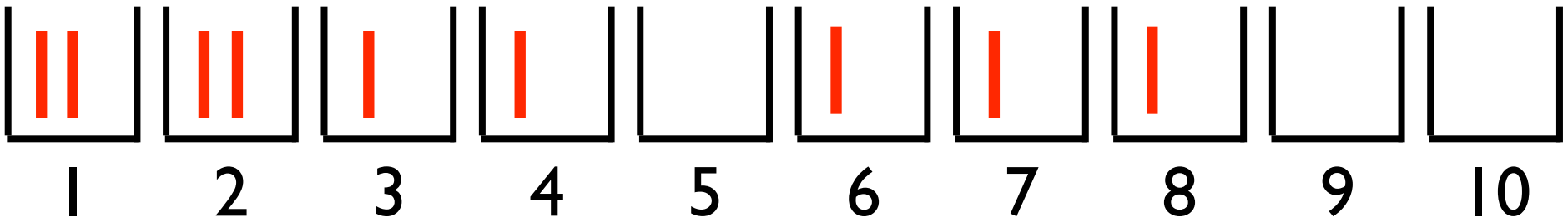
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

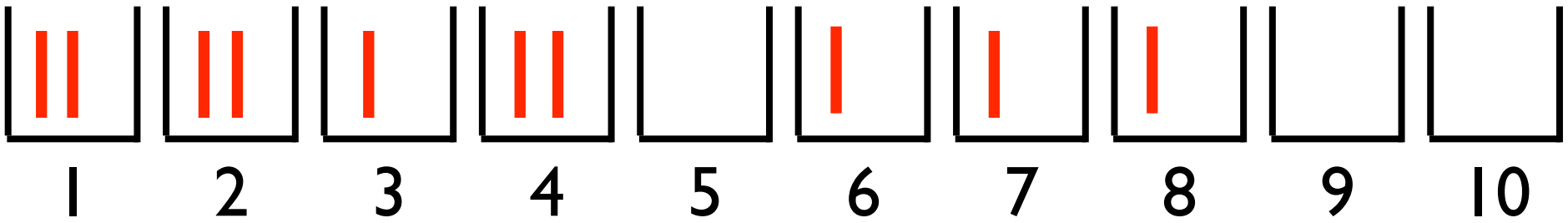
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

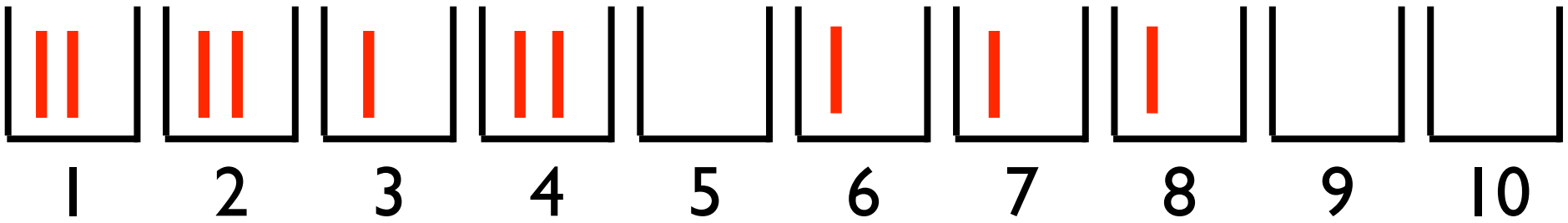
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

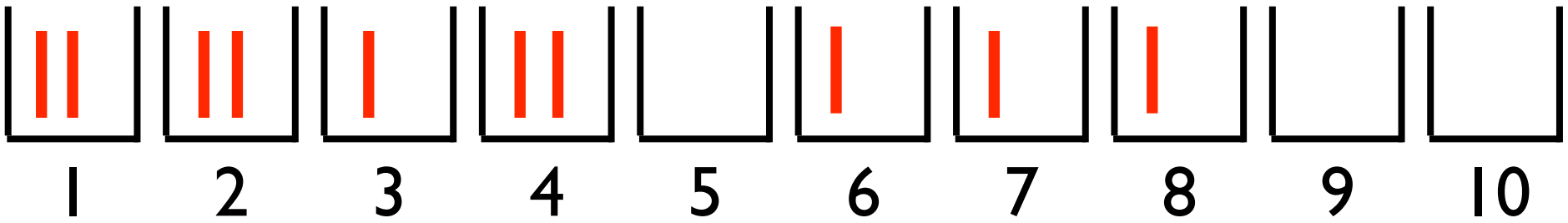
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

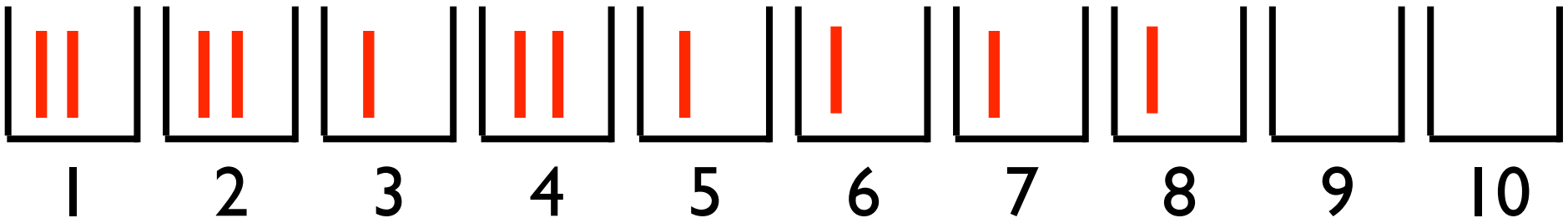
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

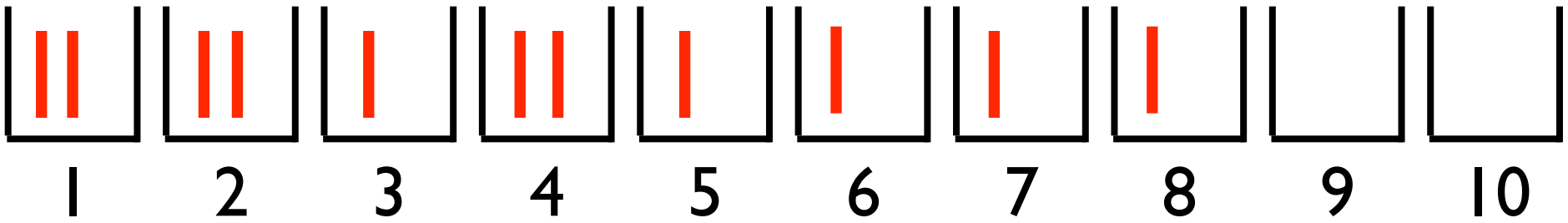
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

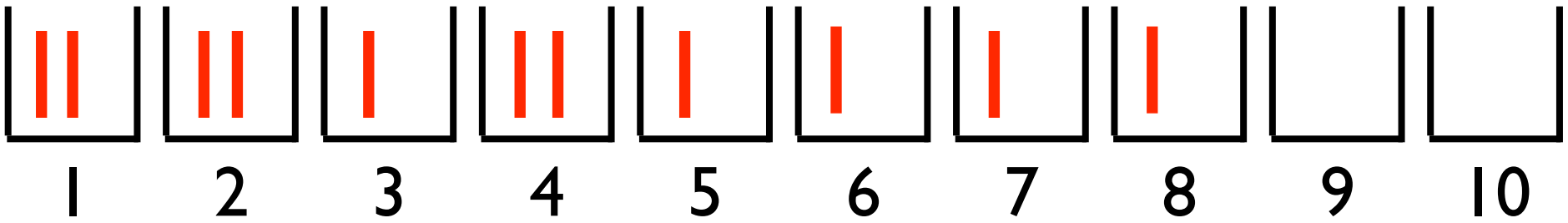
- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

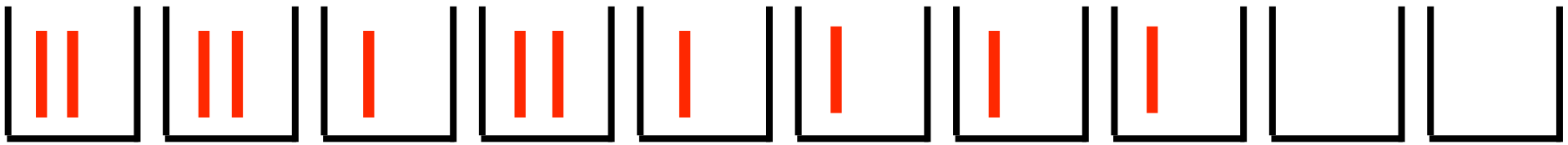


1

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

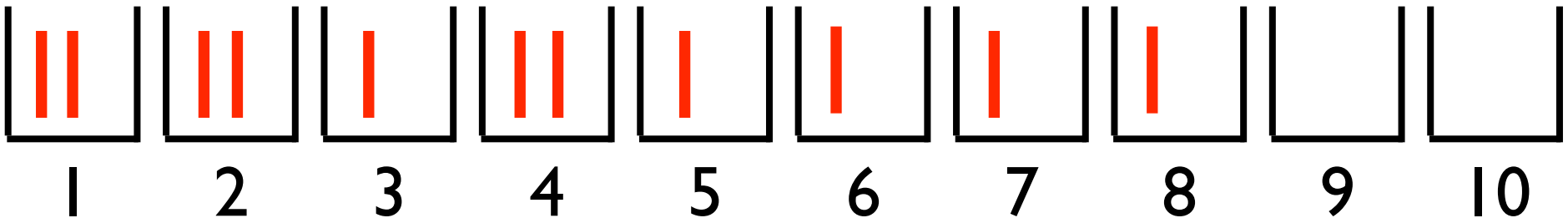


1 1

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

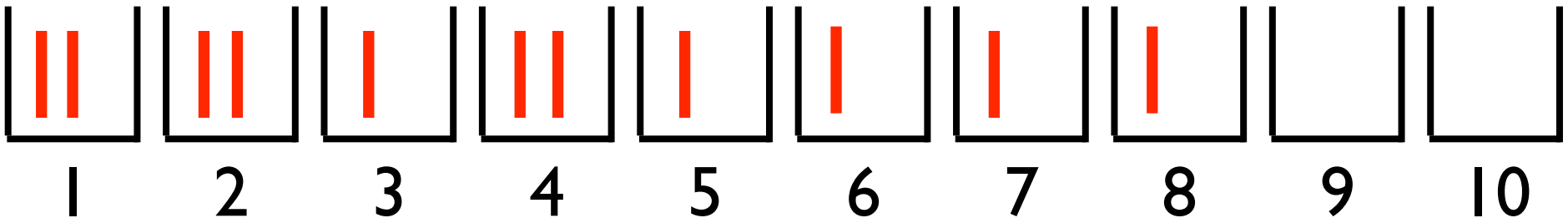


1 1 2

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

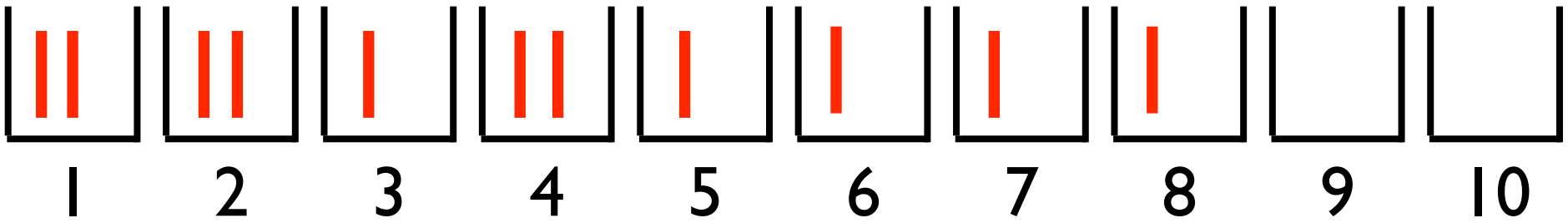


1 1 2 2

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

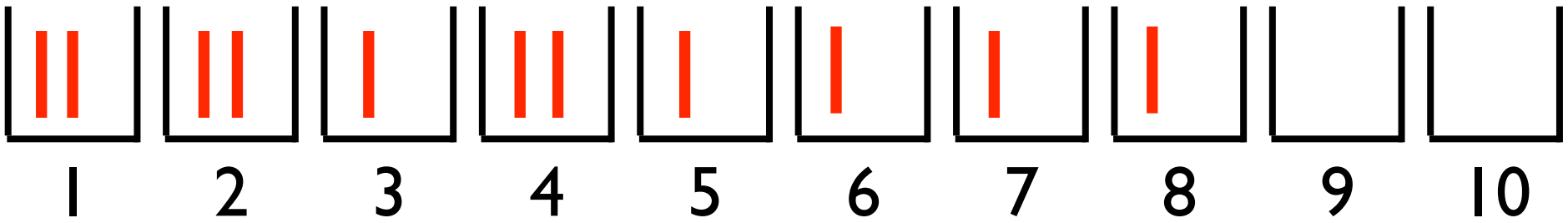


1 1 2 2 3

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

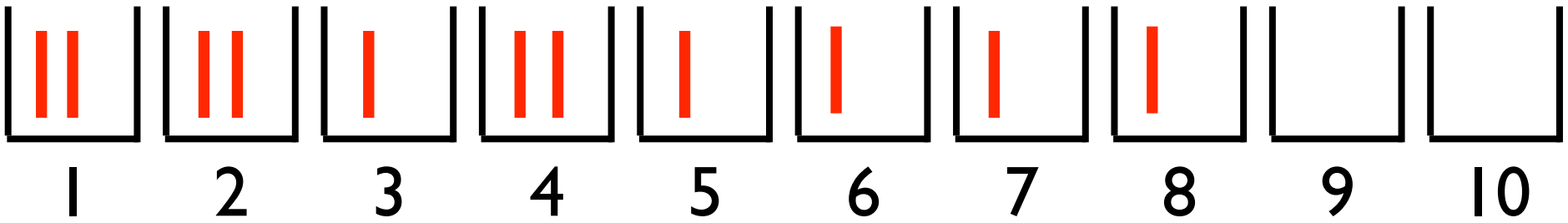


1 1 2 2 3 4

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

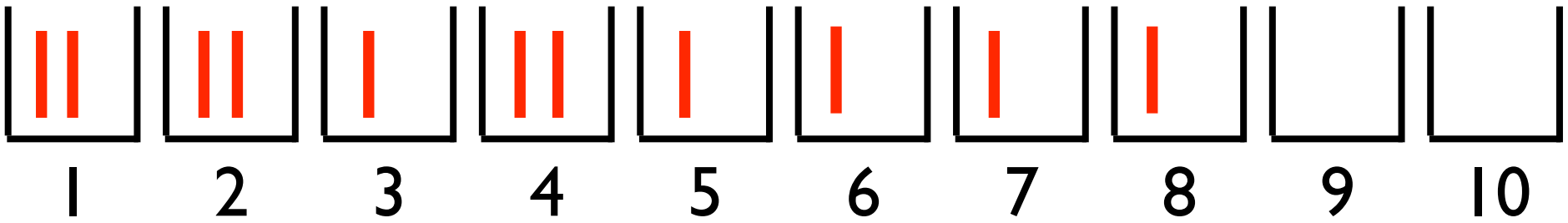


1 1 2 2 3 4 4

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

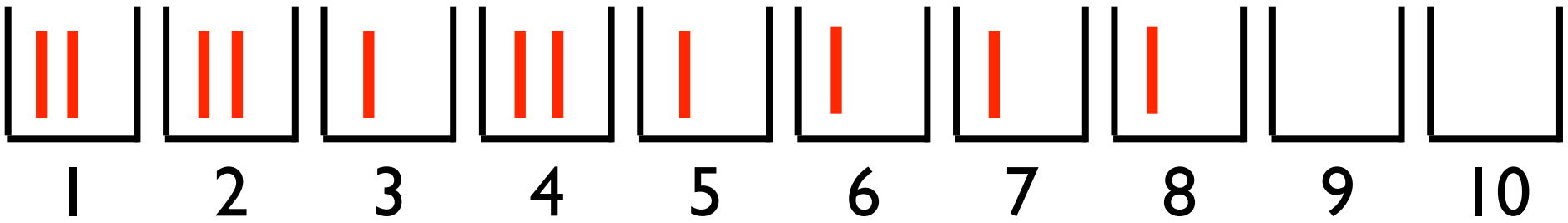


1 1 2 2 3 4 4 5

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

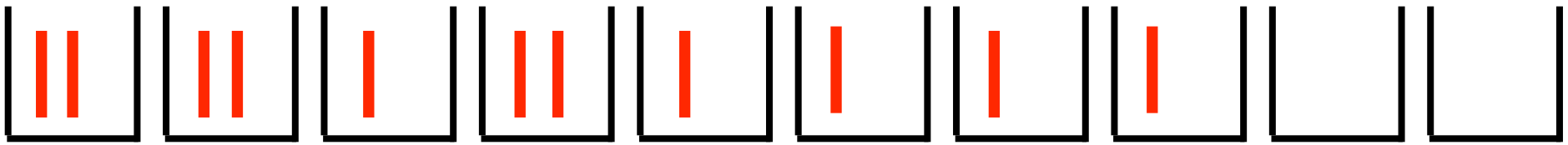


1 1 2 2 3 4 4 5 6

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets

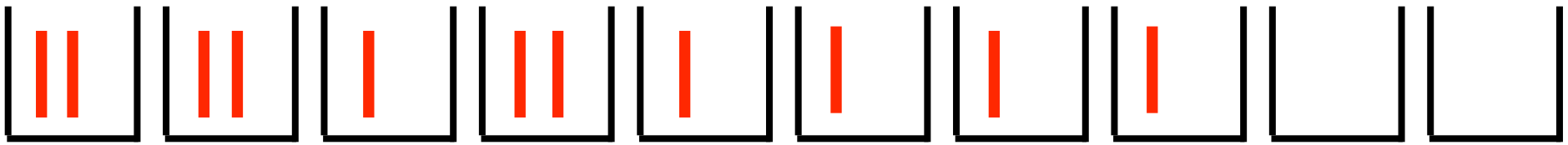


1 1 2 2 3 4 4 5 6 7

Sortieren

Bucketsort - Beispiel

- Grundtyp 1..10
- Eingabe: 7 2 4 2 8 1 1 3 6 4 5
- Benötigt: 10 Buckets



1 1 2 2 3 4 4 5 6 7 8

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Sortieren

Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Buckets löschen

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Buckets füllen

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Buckets ausleeren

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Sortieren

Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

O(m)

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Sortieren

Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

O(n)

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

O(n)

Sortieren
Bucketsort - Algorithmus

```
type T=(t1,...,tm);
var buckets: array [T] of integer;
for i:=t1 to tm do buckets[ti]:=0;
while not eof do
begin
   read(x); buckets[x]:=buckets[x]+1
end;
for i:=t1 to tm do
   while buckets [ti]>0 do
   begin
      write(ti);
      buckets[ti]:=buckets[ti]-1
   end
```

Sortieren
Bucketsort - Komplexität

- Laufzeit:
  - allgemein: O(m+n+m)=O(m+n)
  - m konstant: O(n)
- Speicher:
  - allgemein: O(m)
  - m konstant: O(1)

Sortieren
Bucketsort - Anwendung Radixsort

- Sortieren von Zeichenfolgen in lexikographischer (“wie im Lexikon”)Reihenfolge, z.B.:

  unsortiert: acd, bad, dca, dba, bcd

  sortiert: acd, bad, bcd, dba, dca

Sortieren

Bucketsort - Einschub: lexikographisch

Definition: Sei $A=\{a_1, a_2, ..., a_m\}$ ein Alphabet mit der Ordnungsrelation $a_1<a_2<...<a_m$ und A^* die Menge aller endlichen Wörter. Es seien $u=u_1u_2...u_r$ und $v=v_1v_2... v_t$ zwei Wörter, d.h., $u,v\in A^*$ und $u_i,v_j\in A$ für $i=1, ..., s$ und $j=1, ..., t$.

Das Wort u ist genau dann **lexikographisch** kleiner als v, wenn entweder

u echtes Anfangswort von v ist (d.h., es gibt ein nicht-leeres Wort w mit v=uw)

oder

es einen Index i gibt mit $u_1=v_1, u_2=v_2, ..., u_{i-1}=v_{i-1}$ und $u_i<v_i$.

Sortieren
Radixsort

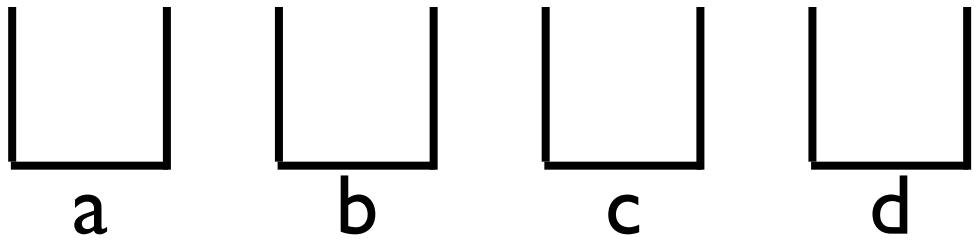
- Vorgehen:
  - Anlegen von 26 Buckets für die Buchstaben (+1 Bucket für Leerzeichen)
  - Bucketsort beginnend mit dem *letzten* Zeichen so oft aufrufen, wie das längste Wort lang ist
  - Beobachtung: Bucketsort ist *stabiles* Sortierverfahren

Sortieren

Radixsort - Beispiel

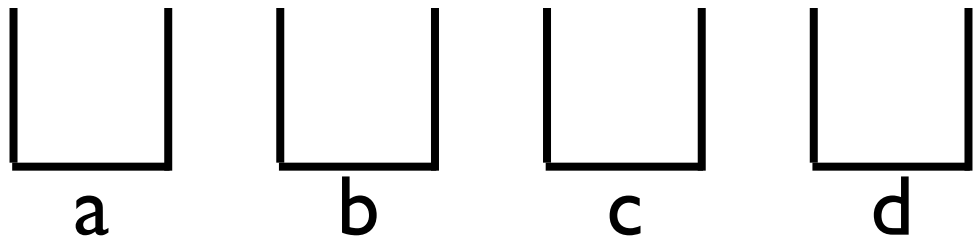
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

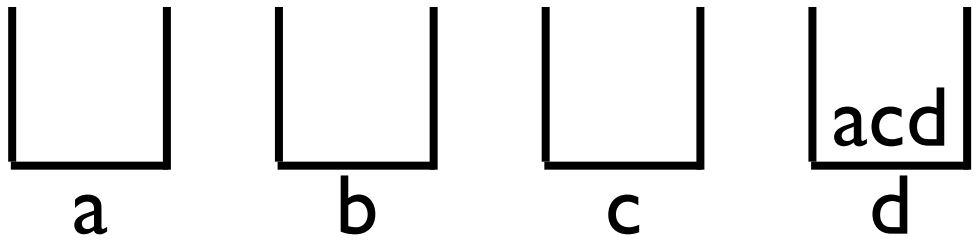


Sortieren

Radixsort - Beispiel

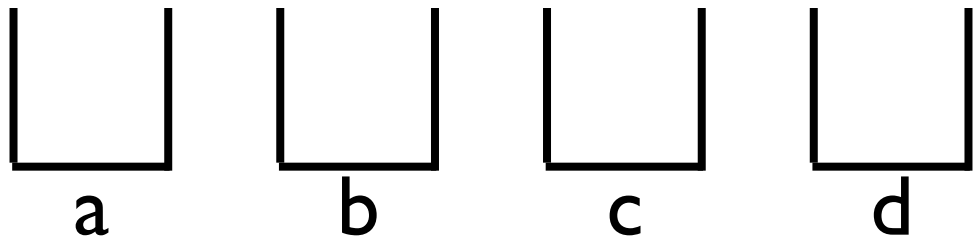
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

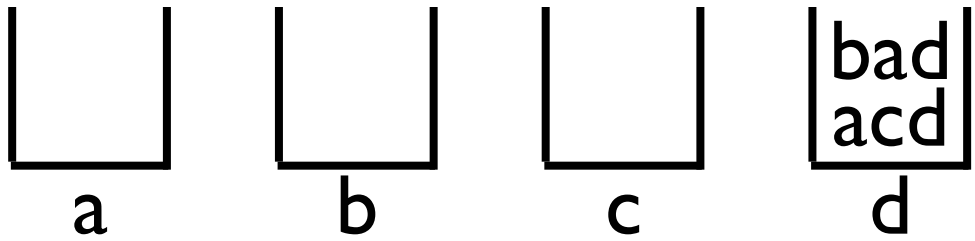


Sortieren

Radixsort - Beispiel

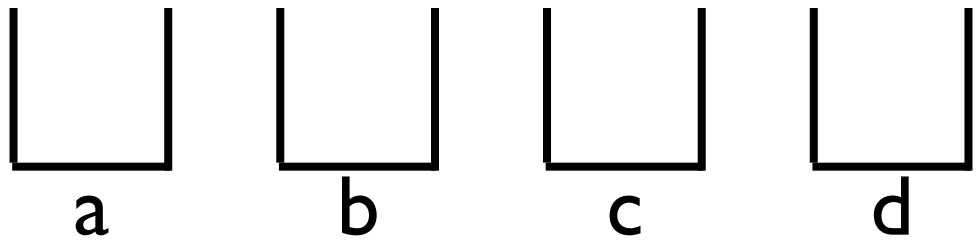
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

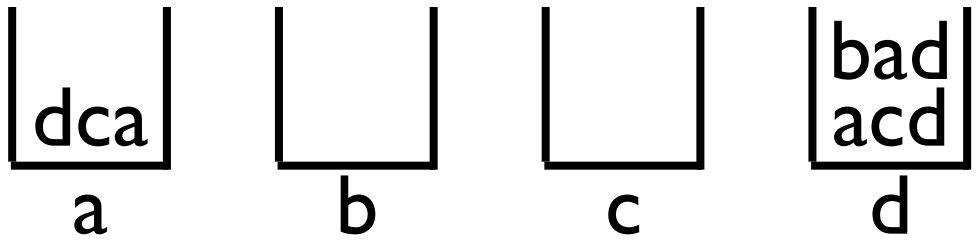


Sortieren

Radixsort - Beispiel

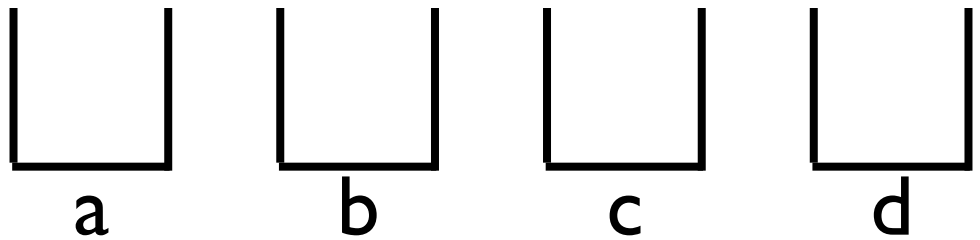
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

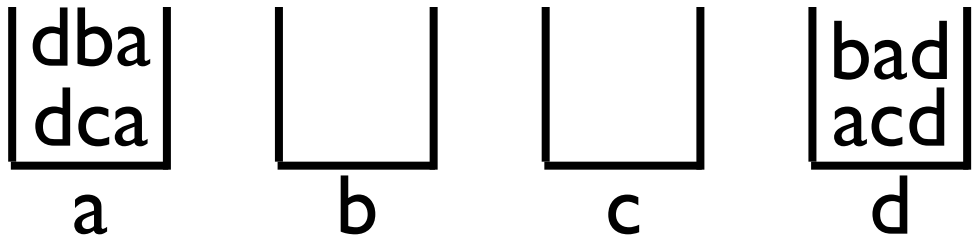


Sortieren

Radixsort - Beispiel

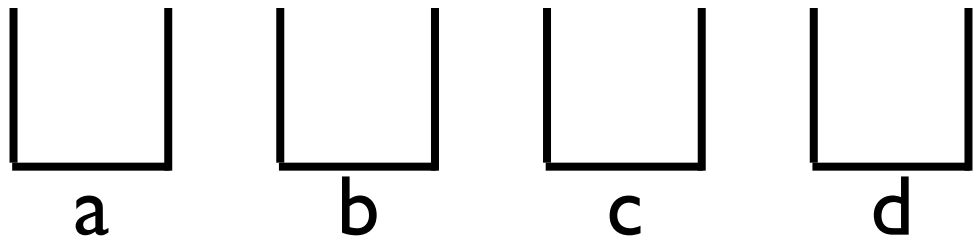
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

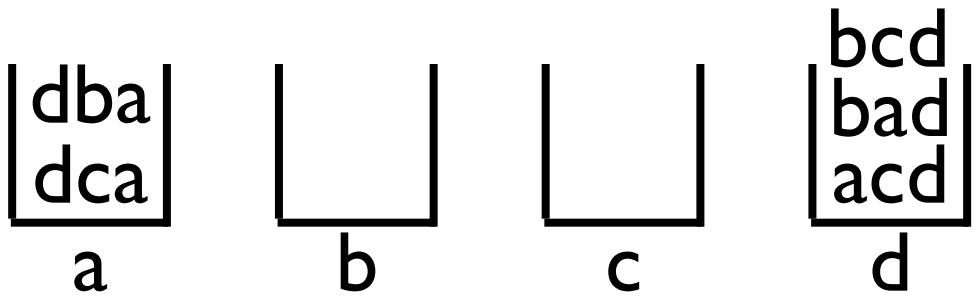


Sortieren

Radixsort - Beispiel

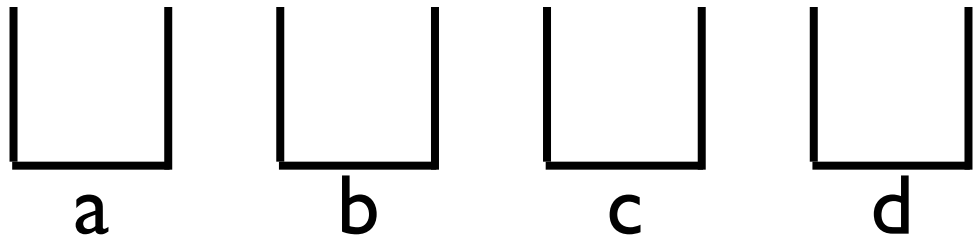
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

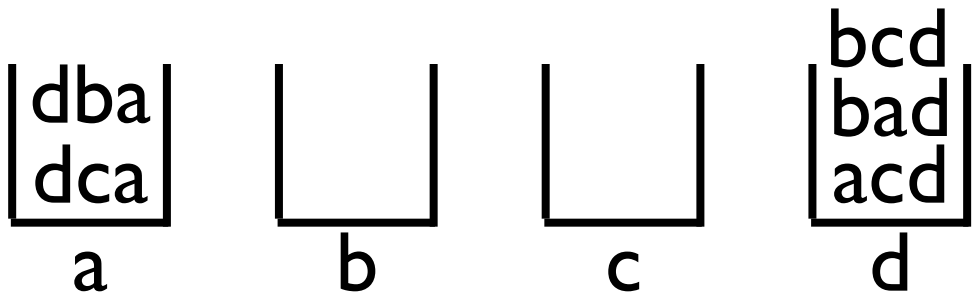


Sortieren

Radixsort - Beispiel

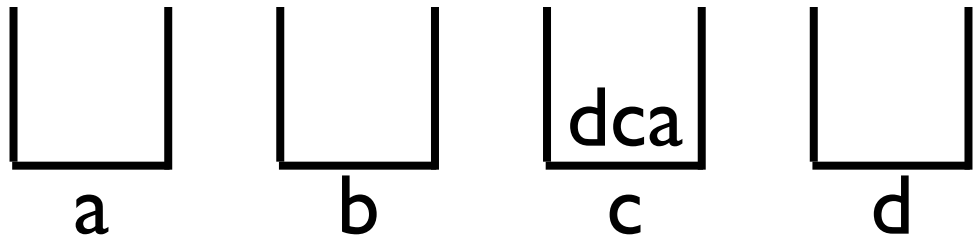
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

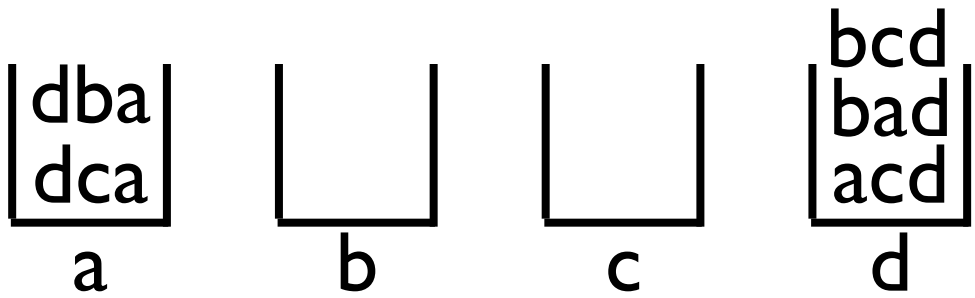


Sortieren

Radixsort - Beispiel

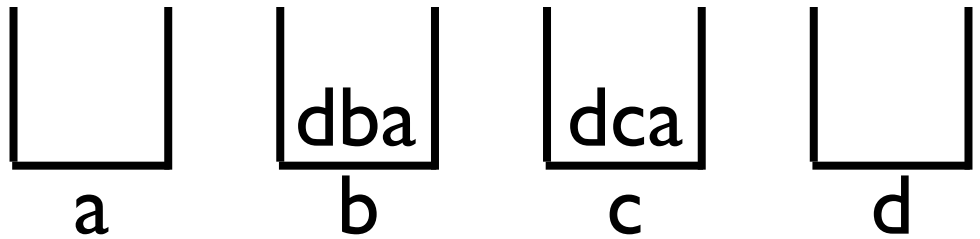
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

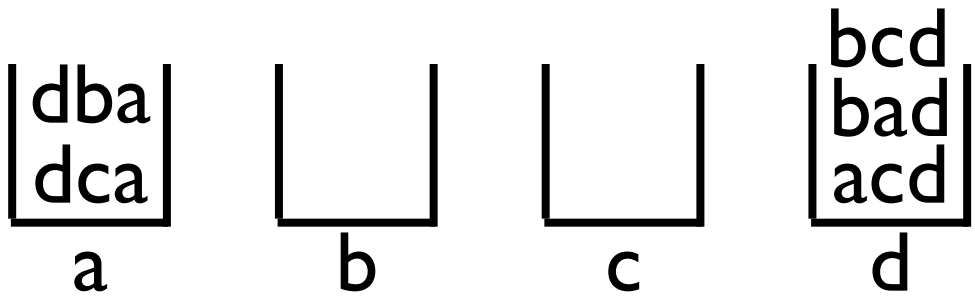


Sortieren

Radixsort - Beispiel

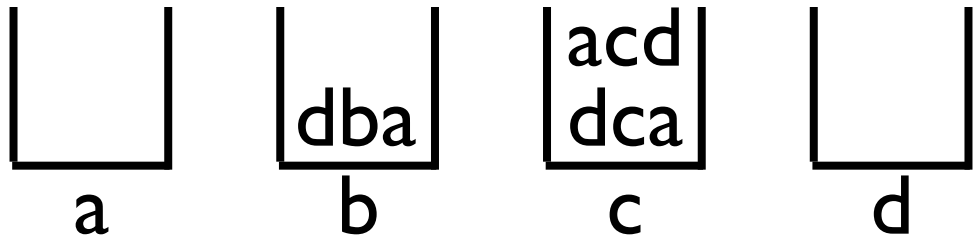
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

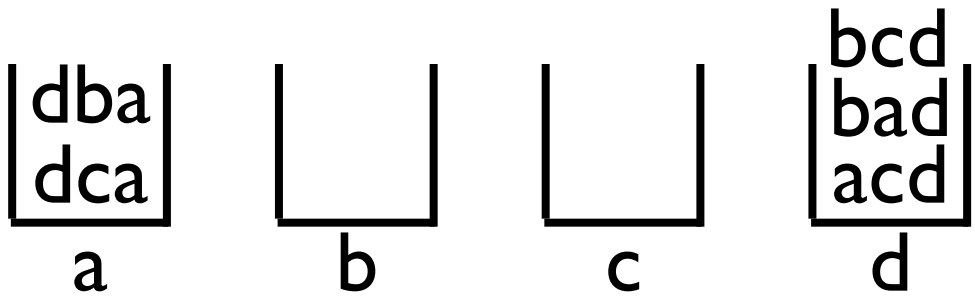


Sortieren

Radixsort - Beispiel

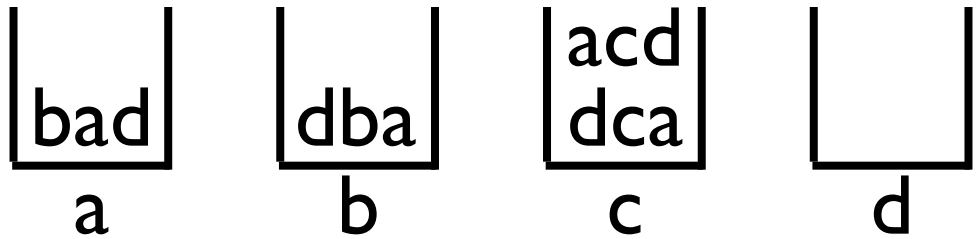
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

Phase 2: Sortieren nach vorletztem Buchstaben

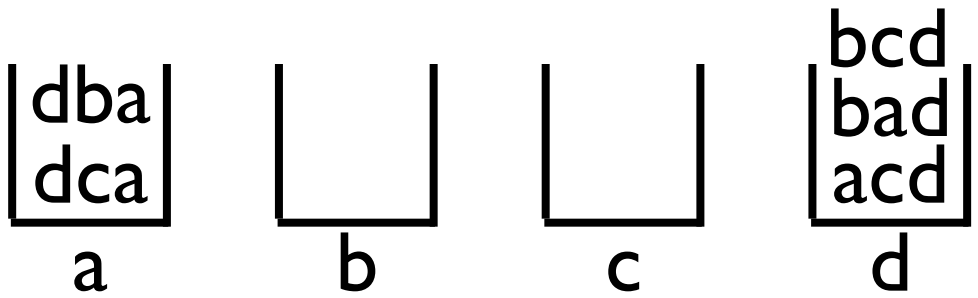


Sortieren

Radixsort - Beispiel

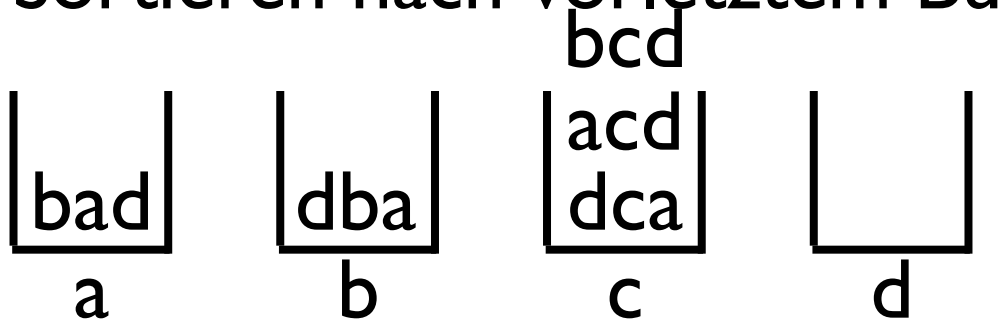
Eingabe: acd, bad, dca, dba, bcd

Phase 1: Sortieren nach letztem Buchstaben



Entleerung: dca, dba, acd, bad, bcd

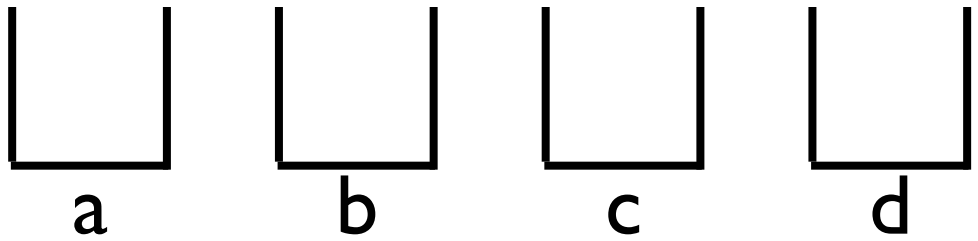
Phase 2: Sortieren nach vorletztem Buchstaben



Sortieren
Radixsort - Beispiel

Entleerung: bad, dba, dca, acd, bcd

Phase 3: Sortieren nach erstem Buchstaben



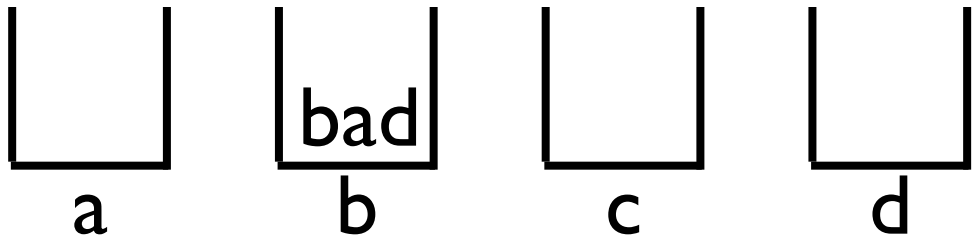
Entleerung: acd, bad, bcd, dba, dca

Fertig.

Sortieren
Radixsort - Beispiel

Entleerung: bad, dba, dca, acd, bcd

Phase 3: Sortieren nach erstem Buchstaben



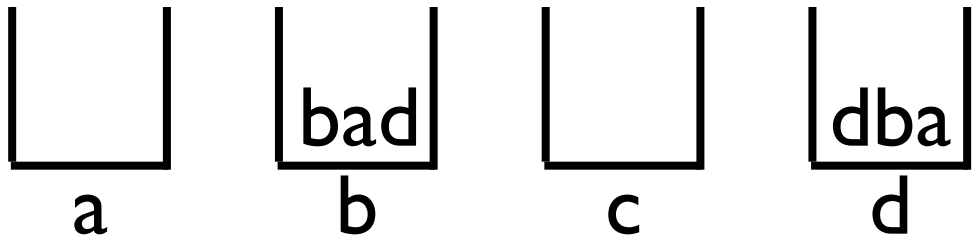
Entleerung: acd, bad, bcd, dba, dca

Fertig.

Sortieren
Radixsort - Beispiel

Entleerung: bad, dba, dca, acd, bcd

Phase 3: Sortieren nach erstem Buchstaben



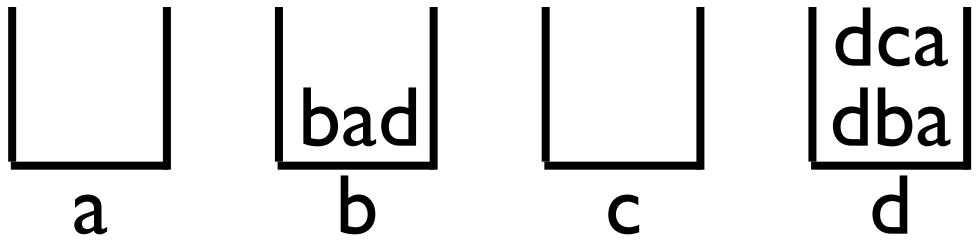
Entleerung: acd, bad, bcd, dba, dca

Fertig.

Sortieren
Radixsort - Beispiel

Entleerung: bad, dba, dca, acd, bcd

Phase 3: Sortieren nach erstem Buchstaben



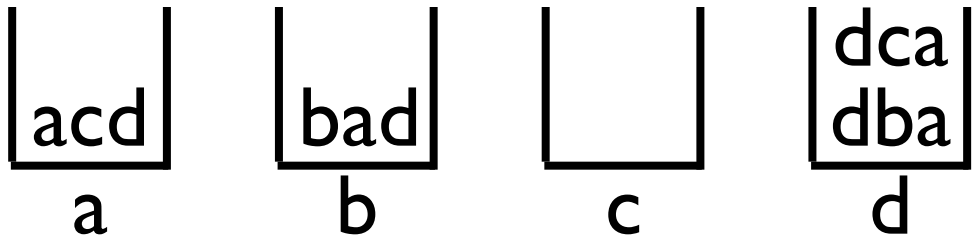
Entleerung: acd, bad, bcd, dba, dca

Fertig.

Sortieren
Radixsort - Beispiel

Entleerung: bad, dba, dca, acd, bcd

Phase 3: Sortieren nach erstem Buchstaben



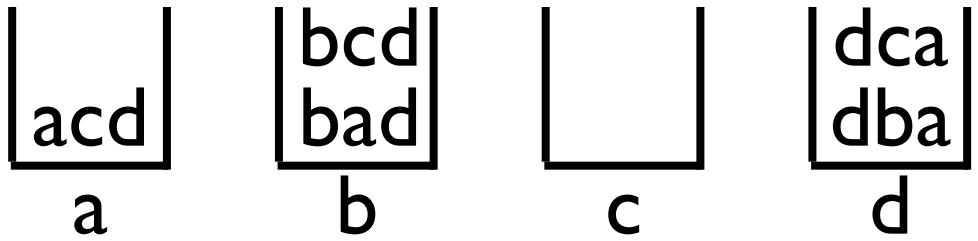
Entleerung: acd, bad, bcd, dba, dca

Fertig.

Sortieren
Radixsort - Beispiel

Entleerung: bad, dba, dca, acd, bcd

Phase 3: Sortieren nach erstem Buchstaben



Entleerung: acd, bad, bcd, dba, dca

Fertig.

Sortieren
Bucketsort - Laufzeit

- Früher bewiesen: Sortieren benötigt mindestens O(n log n) Vergleiche
- Bucketsort arbeitet ganz ohne Vergleiche
- Wie paßt das zusammen?

Sortieren
Bucketsort - Laufzeit

Klar:

- Bucketsort: kein allgemeines Sortierverfahren, das nur auf Vergleichen basiert.
- Es nutzt zusätzliche Eigenschaften (indirekte Adressierung) des zu sortierenden Raumes aus; nicht jede Menge besitzt diese Eigenschaften.
- Kein Widerspruch zwischen beiden Aussagen.
- **Wichtig**: Komplexitätsaussagen sind immer relativ zu den zur Verfügung stehenden elementaren Operationen zu interpretieren.

Sortieren
Heapsort

Merkmale:

- Sortieren durch Auswählen
- **garantiertes** O(n log n)-Verfahren
- im Mittel schlechter als Quicksort (es gibt allerdings auch bessere Implementierungen: bottom-up-Heapsort)

Sortieren
Heapsort - Idee

Idee:

- Wähle in jedem Auswahlschritt das Maximum des verbleibenden Feldes und gib es aus.
- Geschickte Auswahl → O(n log n)-Verfahren
- Zentrale Datenstruktur: binärer Baum als sog. **Heap** organisiert.

Sortieren
Heapsort - Definition Heap

Definition

Eine Folge von Objekten $a_1, ..., a_n$ heißt **Heap**, falls gilt:

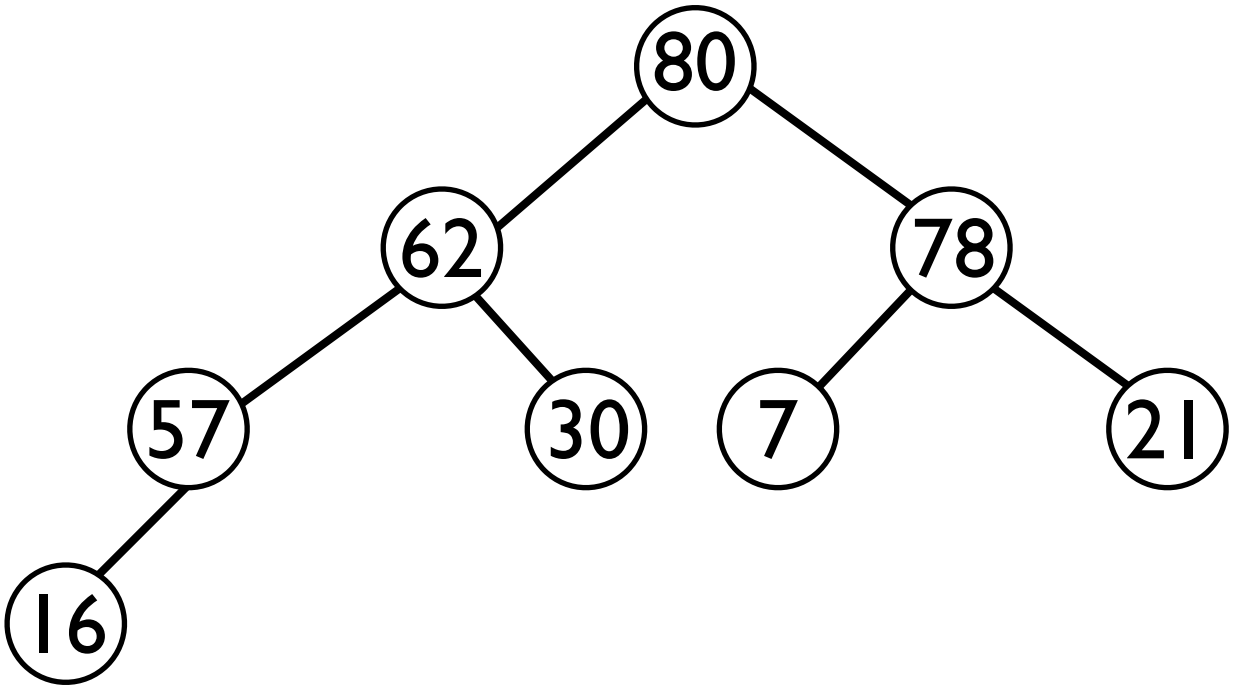
$$a_j \geq a_{2j},\ a_j \geq a_{2j+1},\ \text{für } 2j,\ 2j+1 \leq n$$

Folgerung: $a_1 = \max_{1 \leq j \leq n} a_j$

Sortieren

Heapsort - Beispiel Heap

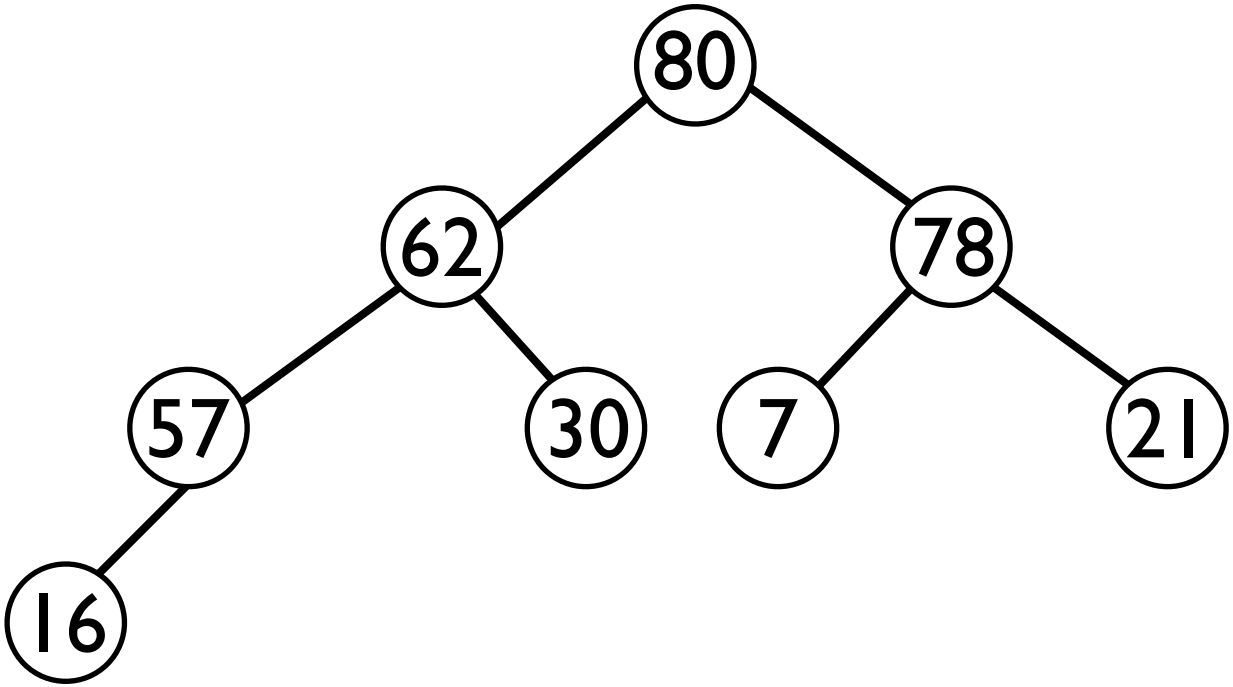
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

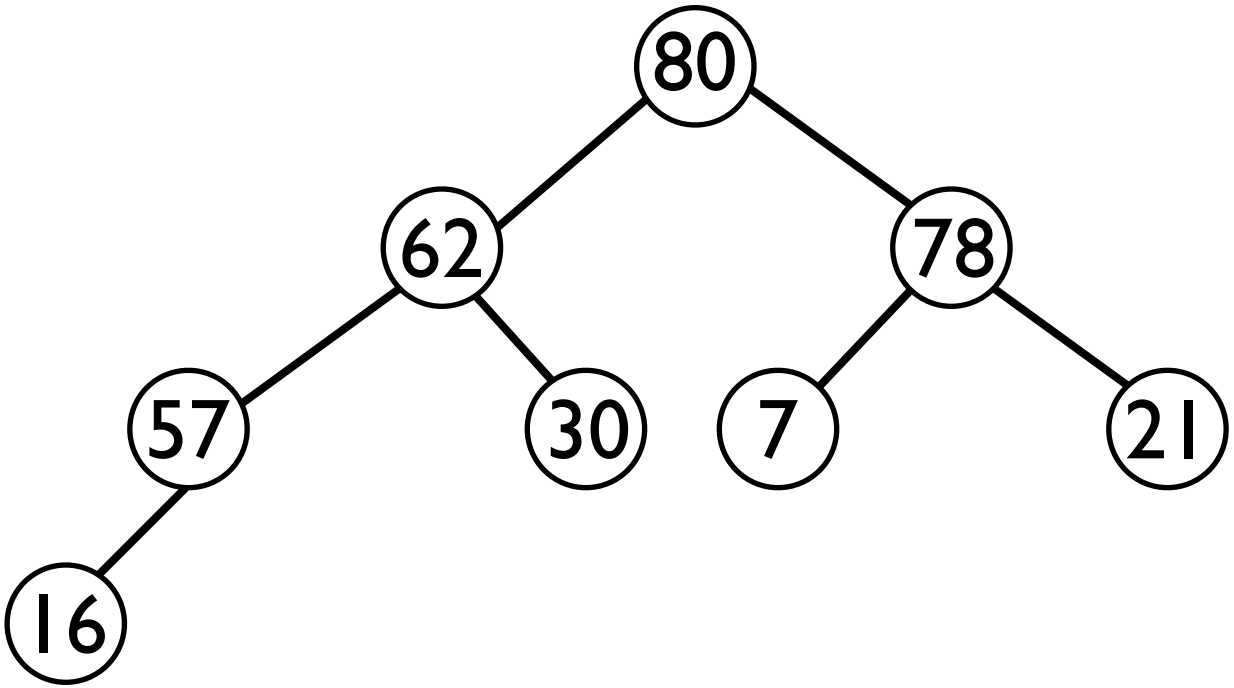
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

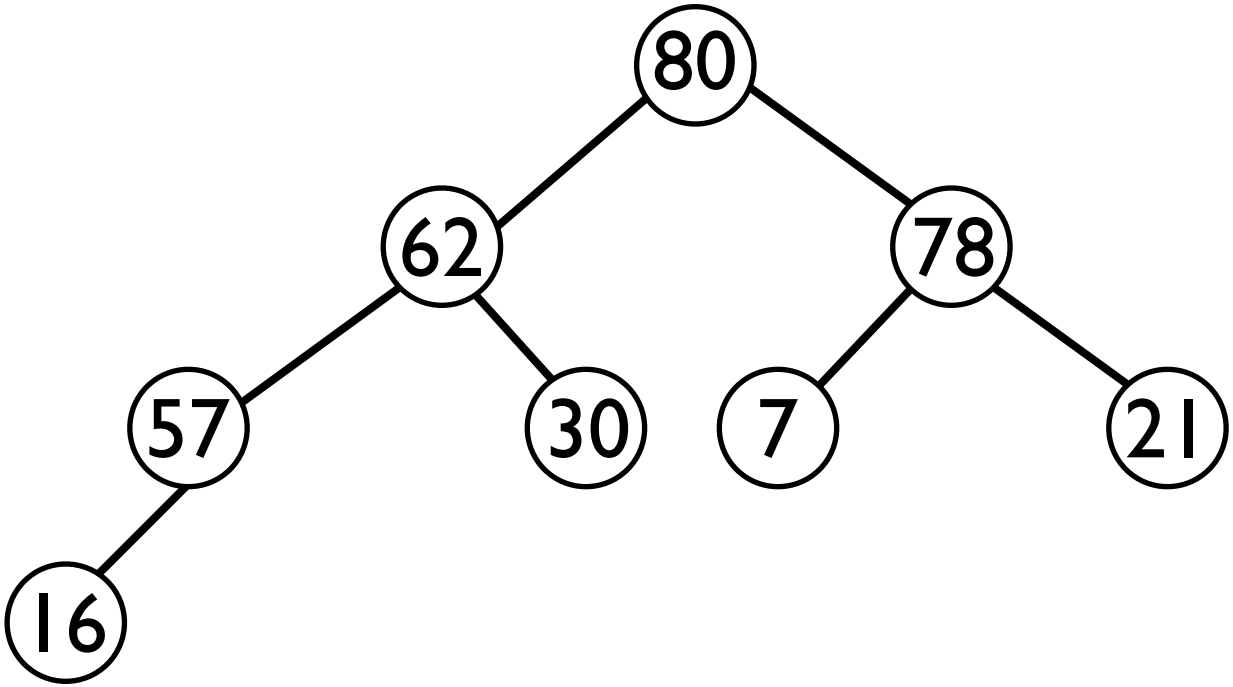
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

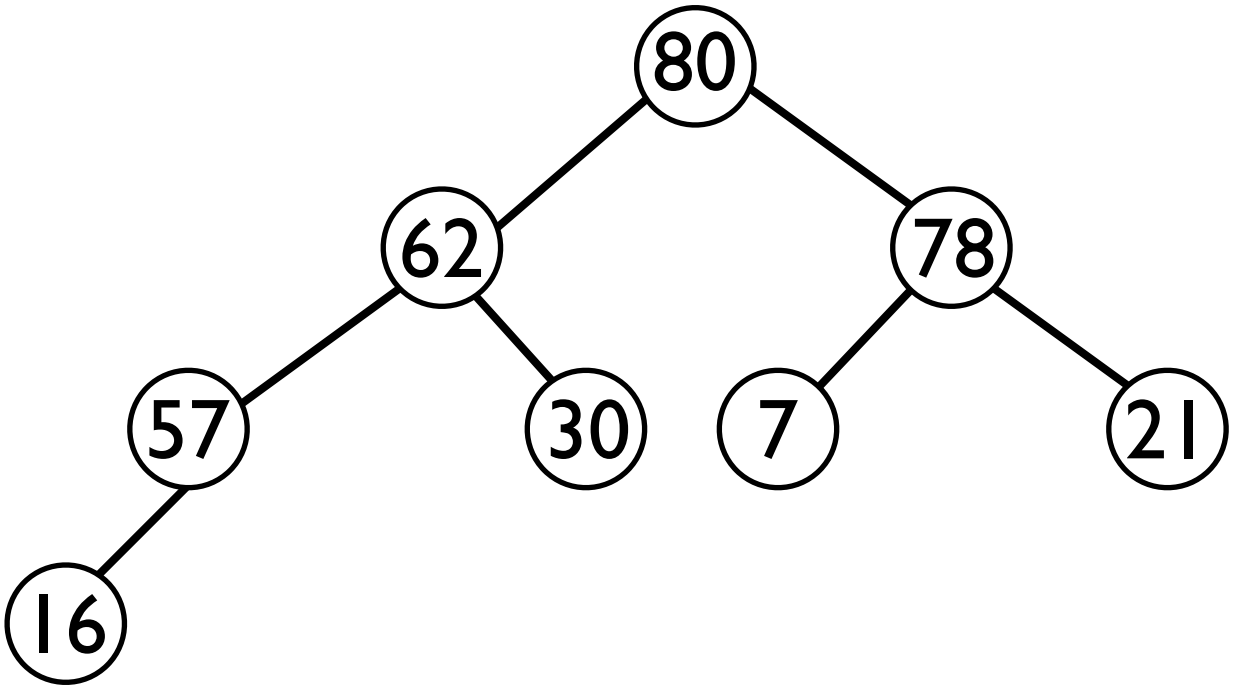
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

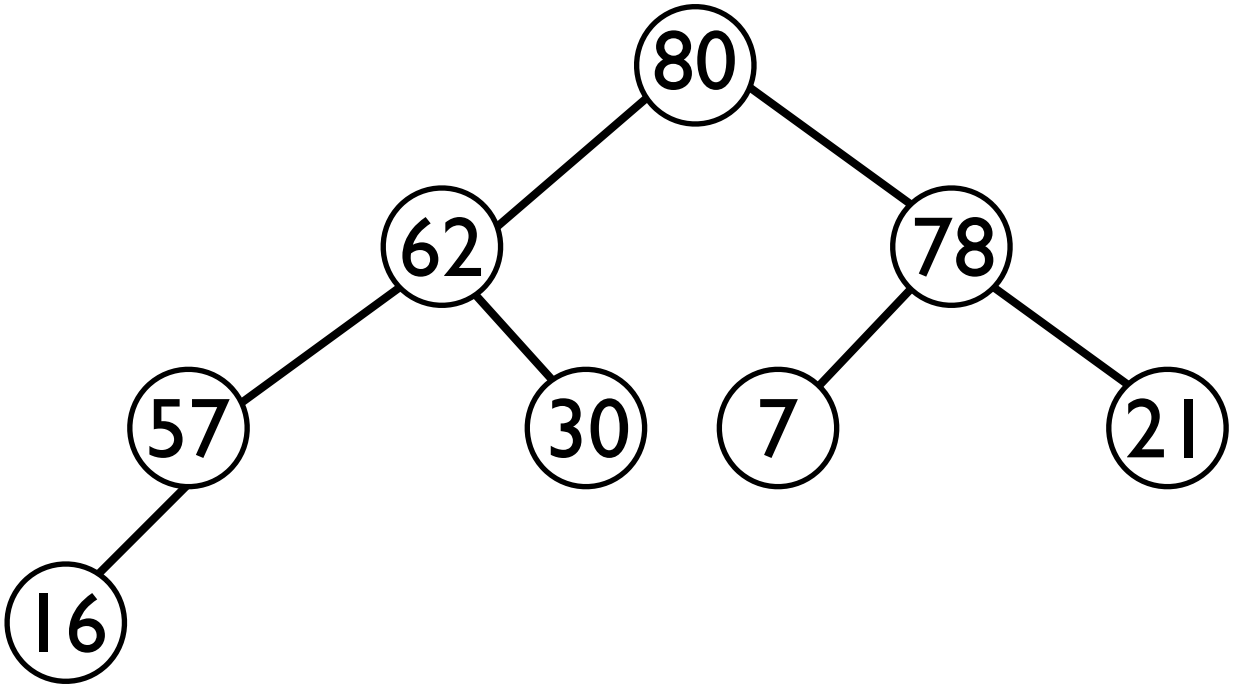
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

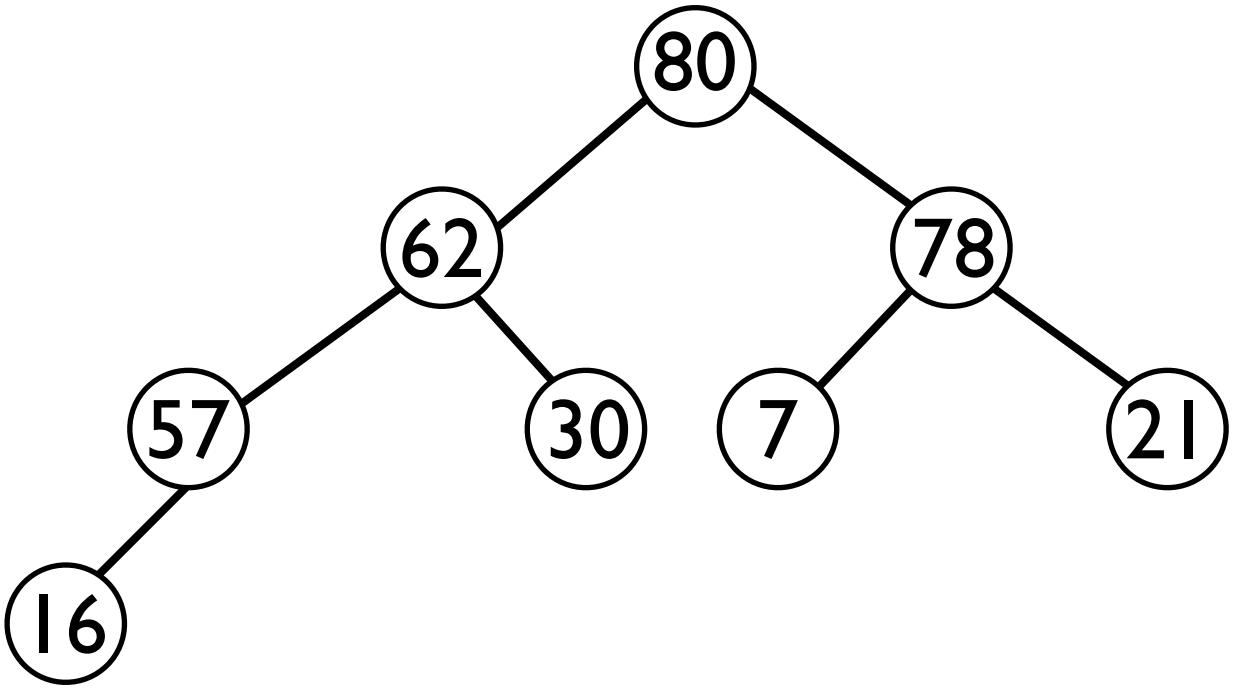
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

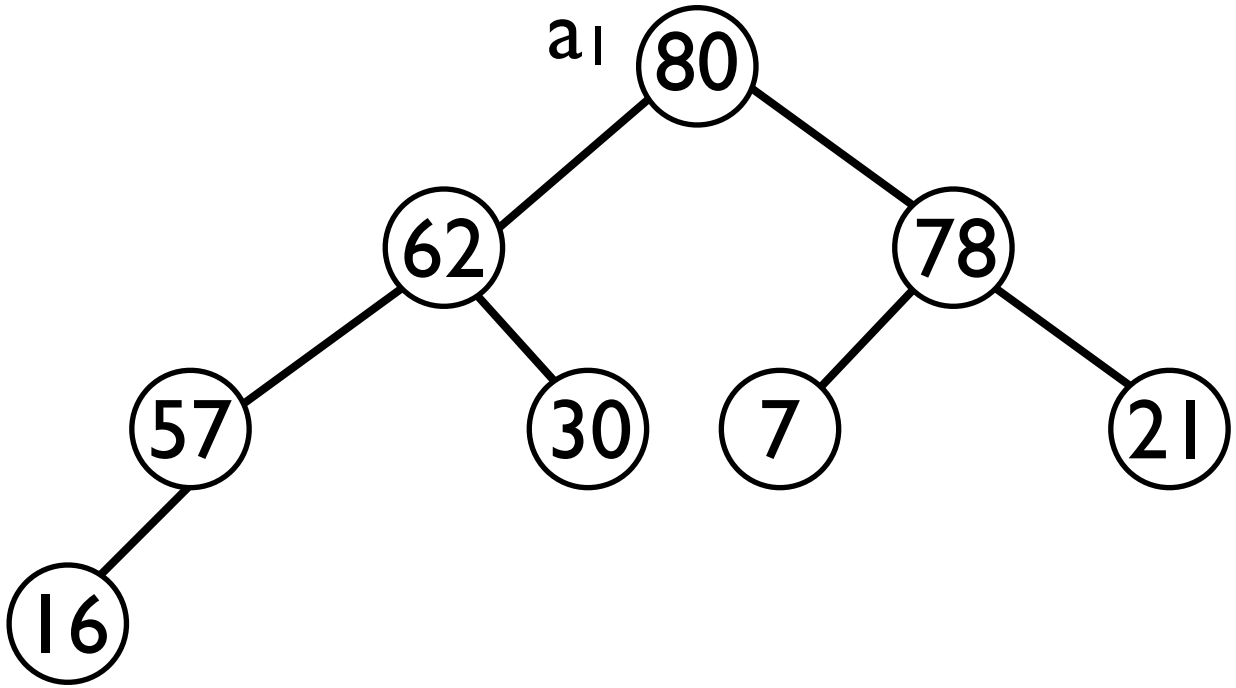
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

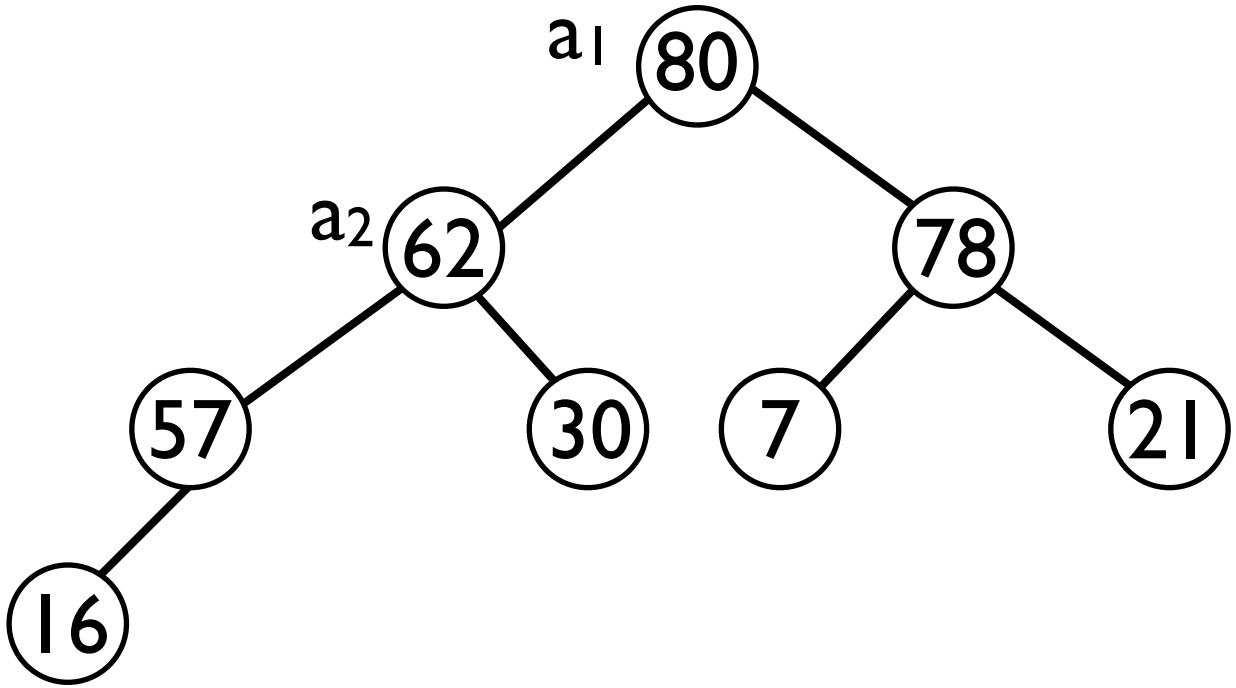
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

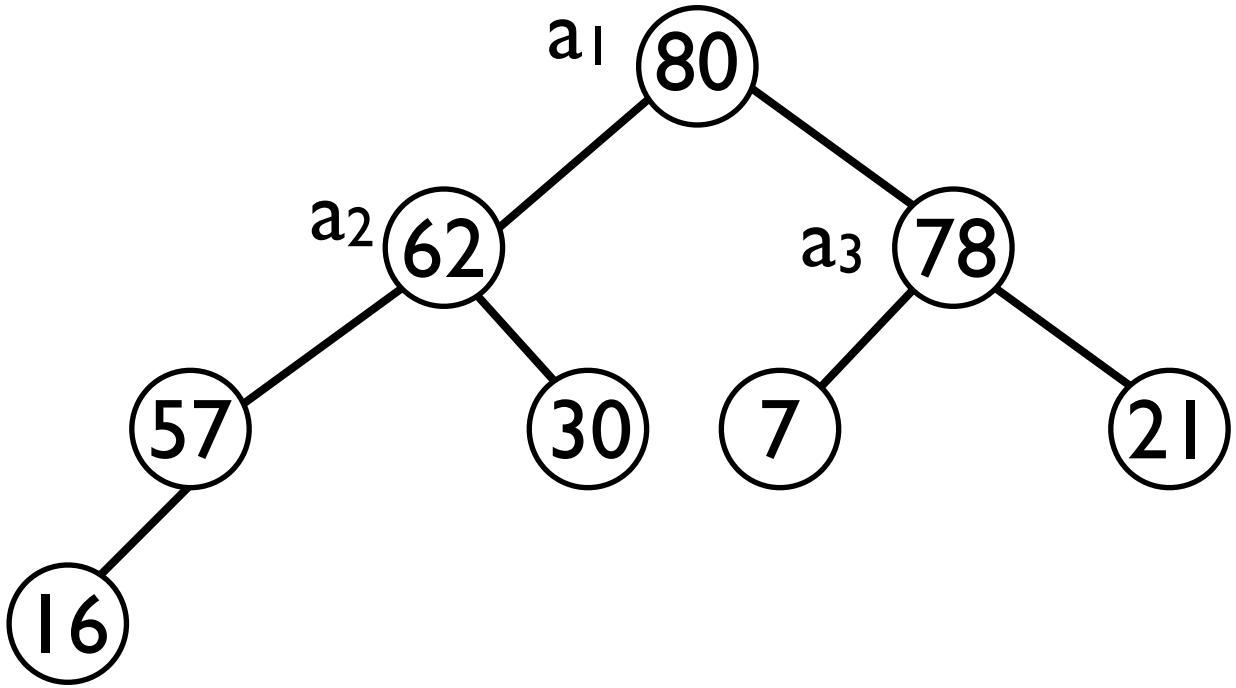
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

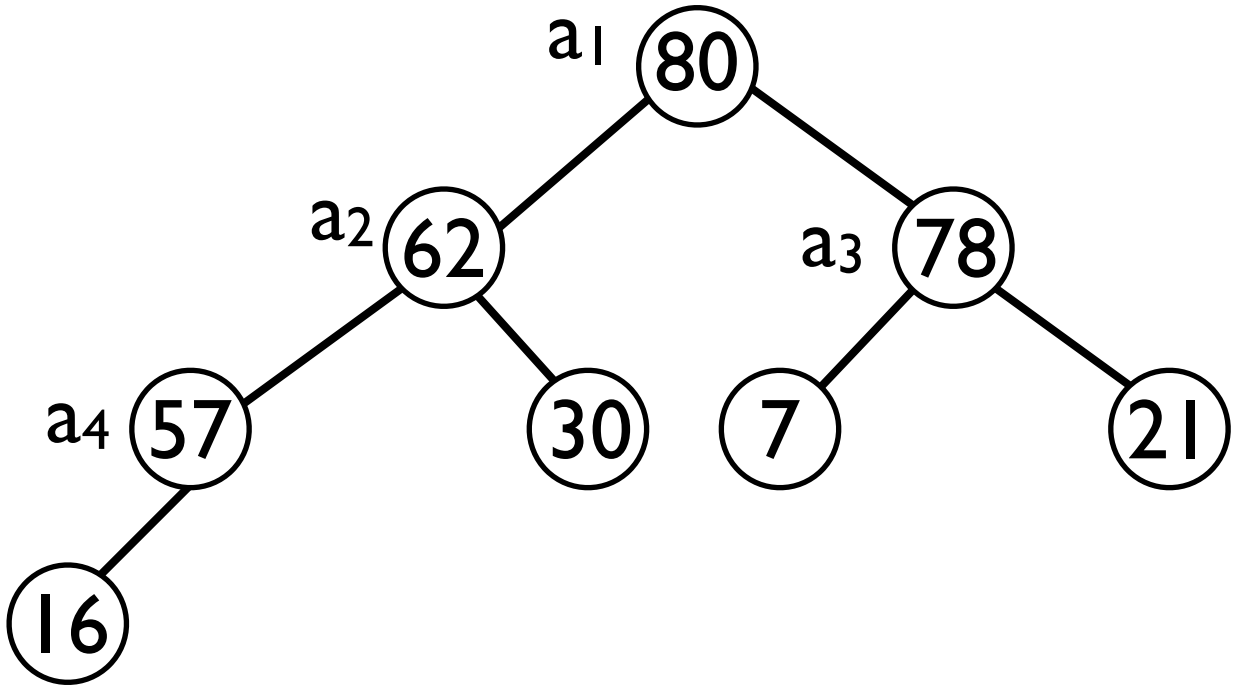
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

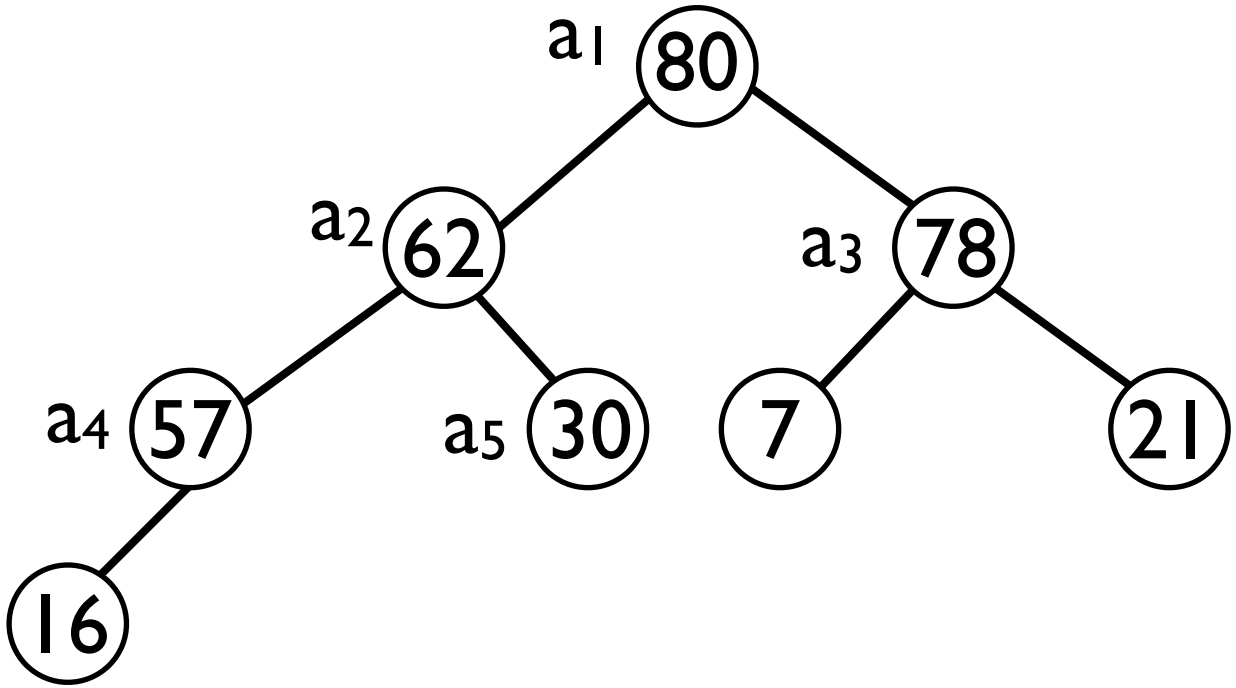
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

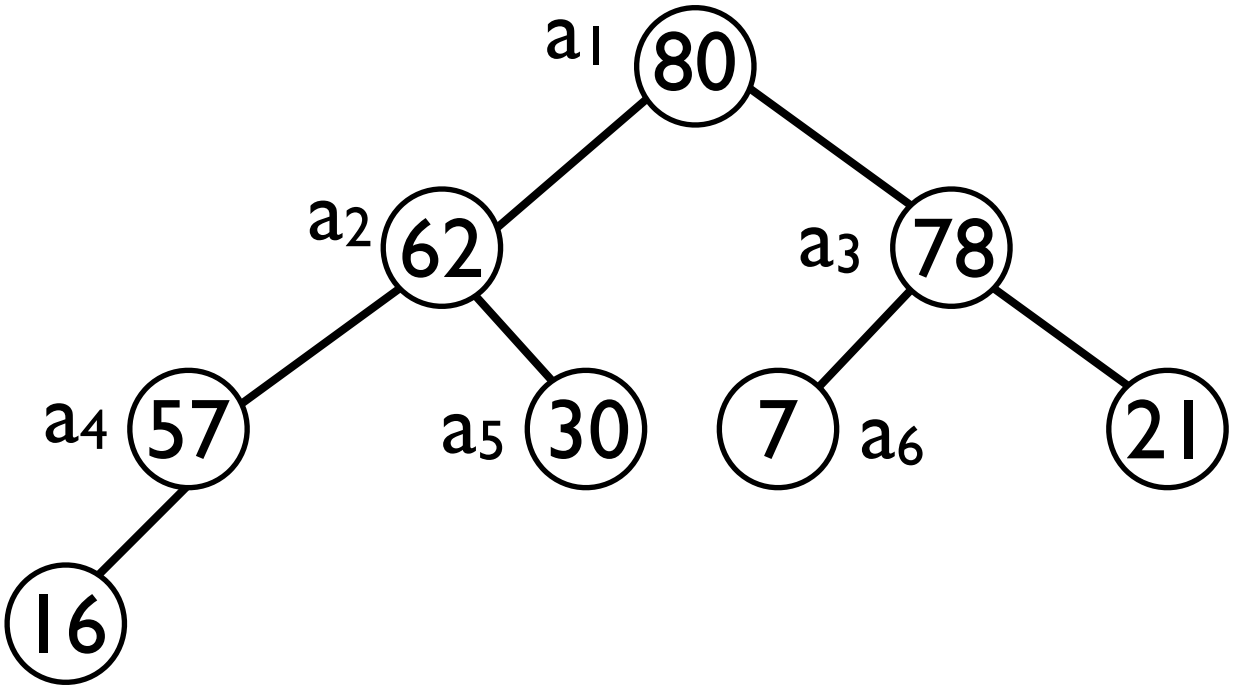
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

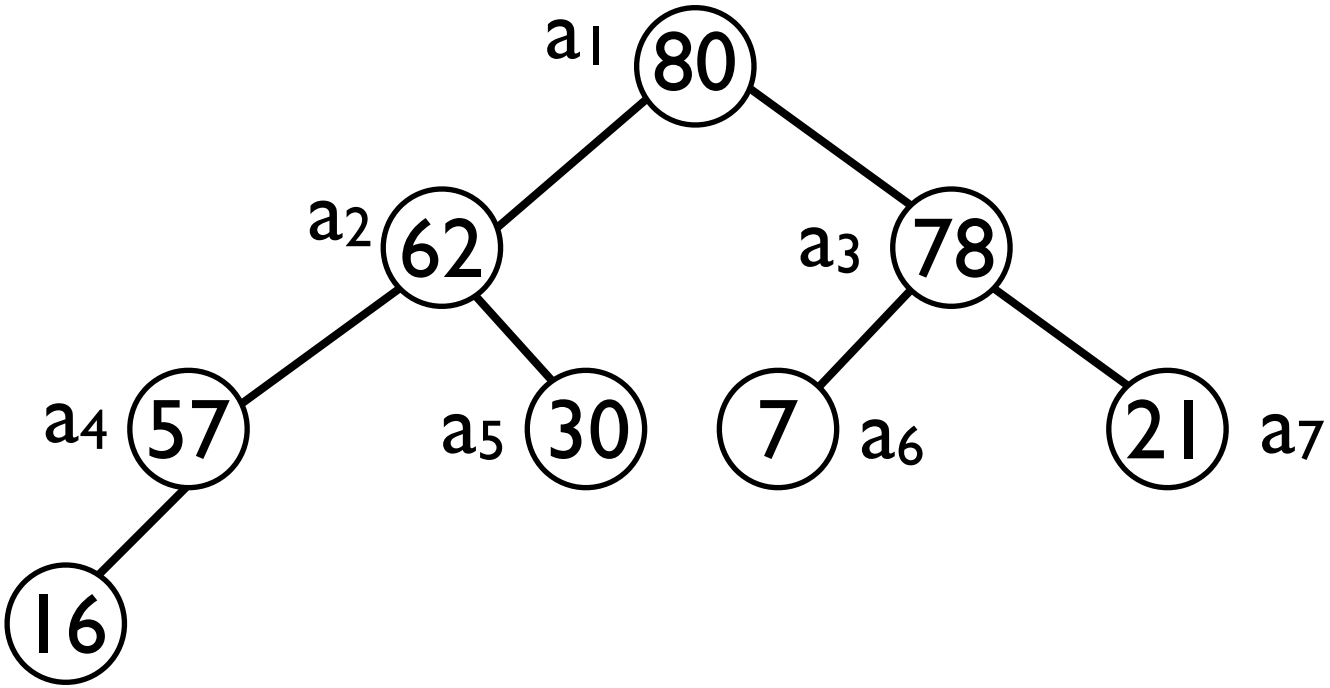
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

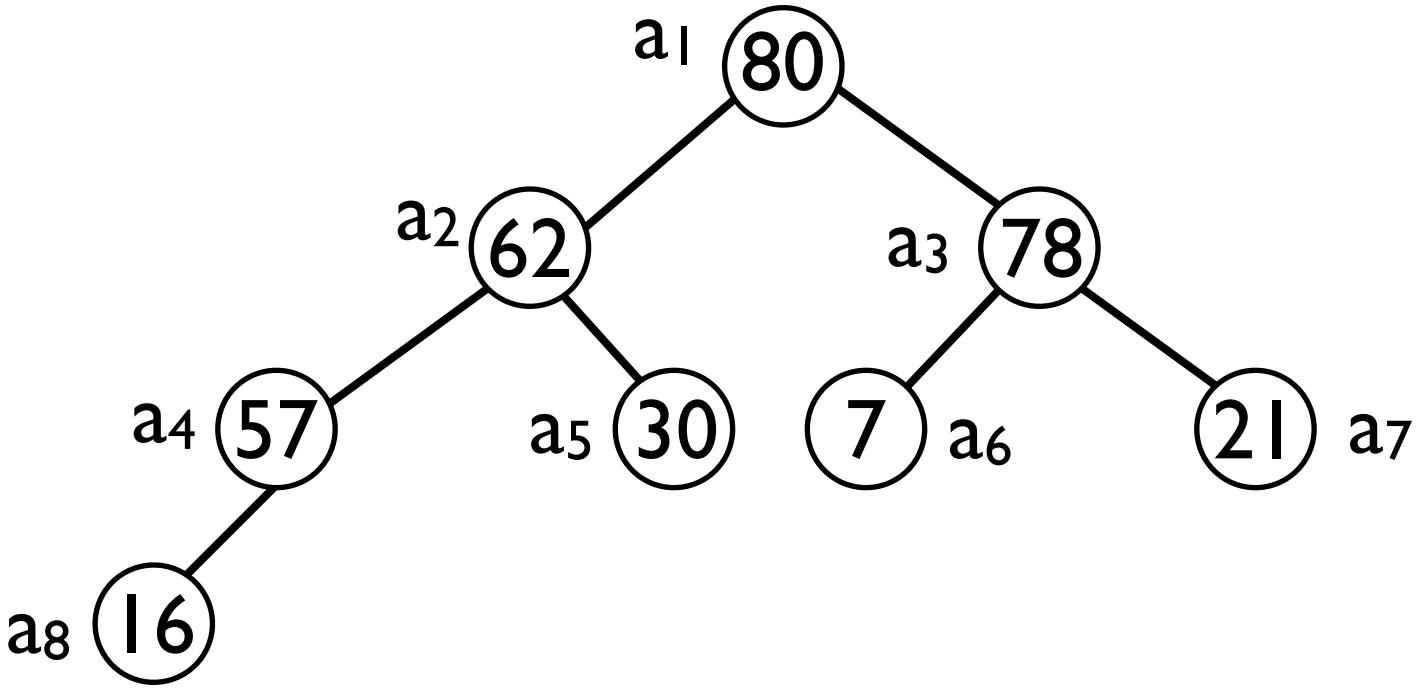
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

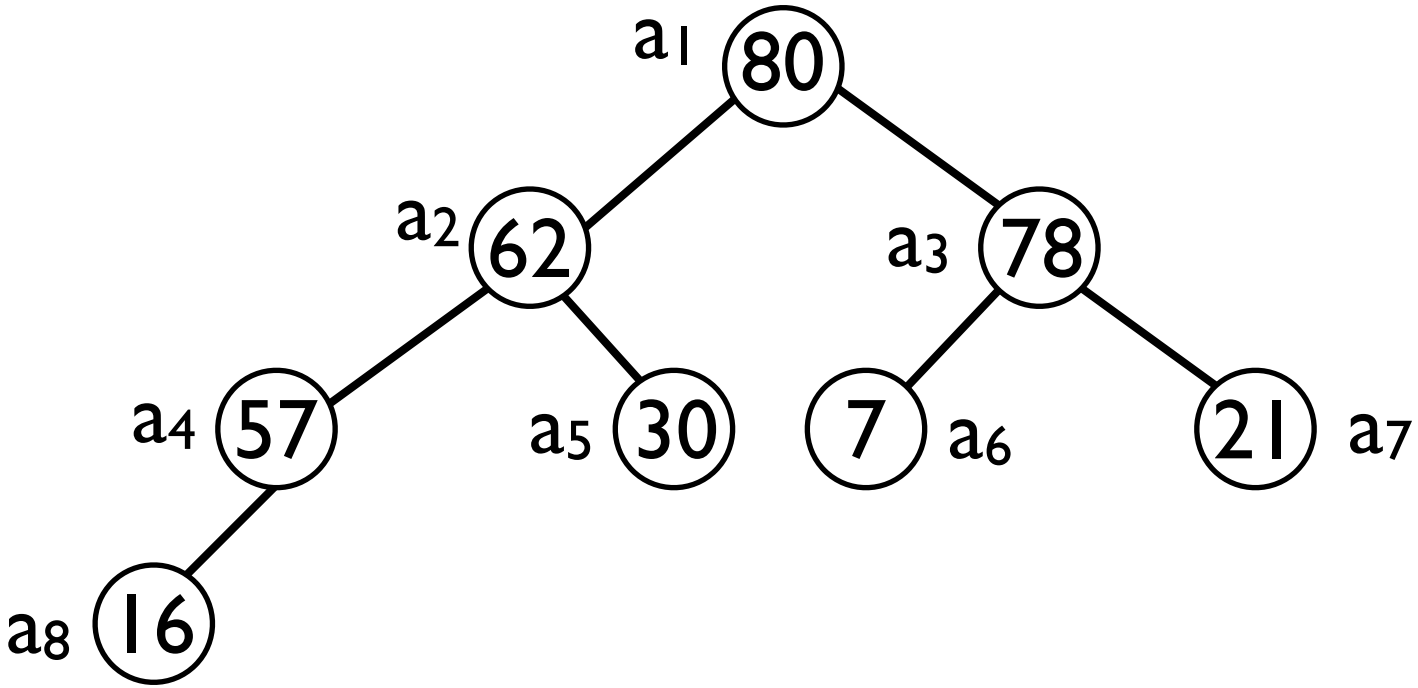
80 62 78 57 30 7 21 16



Sortieren

Heapsort - Beispiel Heap

80 62 78 57 30 7 21 16

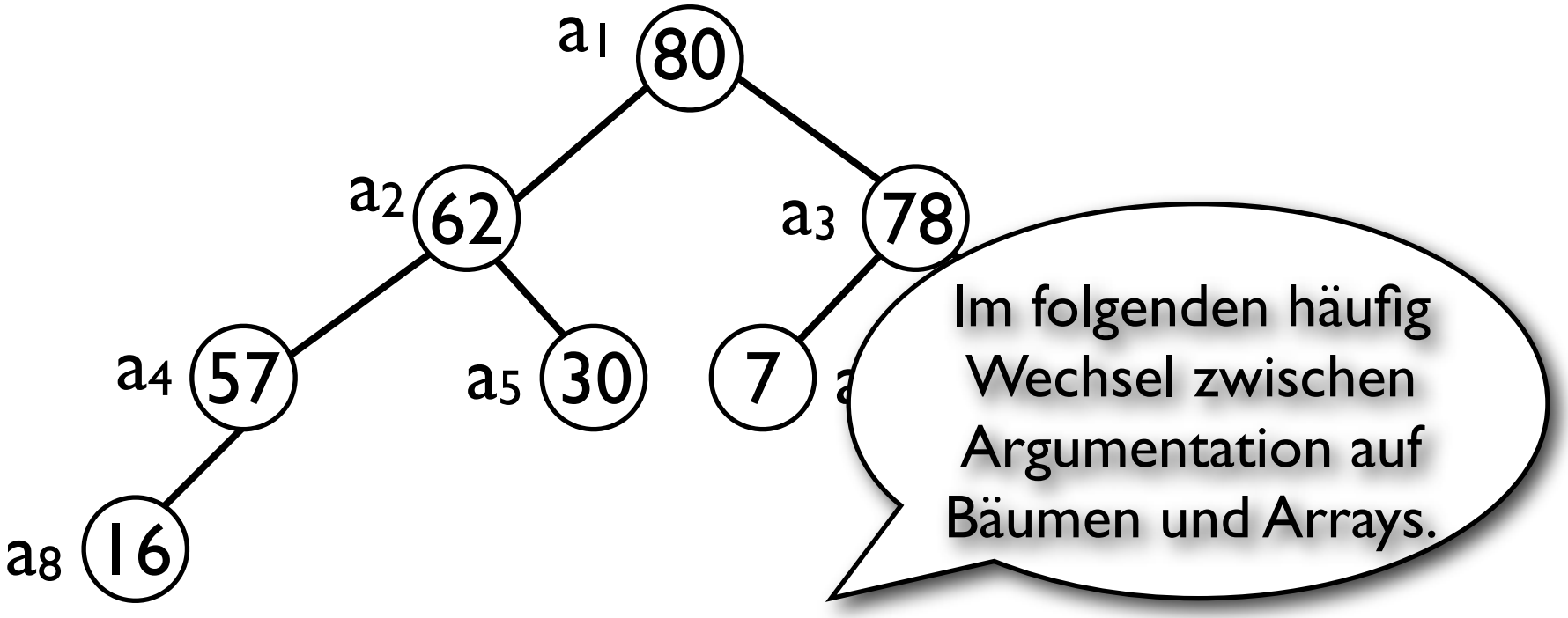


a1	a2	a3	a4	a5	a6	a7	a8

Sortieren

Heapsort - Beispiel Heap

80 62 78 57 30 7 21 16



a1	a2	a3	a4	a5	a6	a7	a8

Sortieren
Heapsort - Algorithmus

- Gegeben: Heap mit Elementen $a_1, ..., a_n$
- Methode:

```
while Heap nicht leer do
    Gib a1 aus;
    Entferne a1 aus dem Heap;
    Stelle Heap-Eigenschaft für Schlüssel
    a2, ..., an im Indexbereich 1, ..., n-1
    wieder her
od.
```

Sortieren
Heapsort - Algorithmus

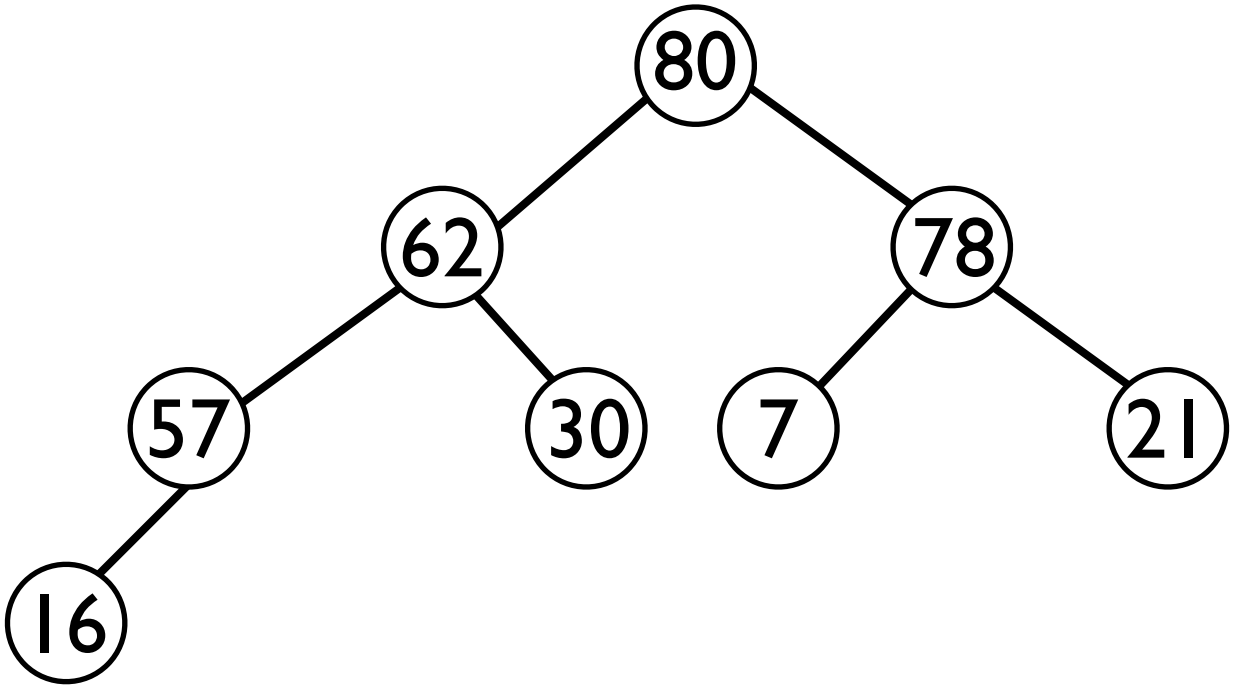
- Gegeben: Heap mit Elementen $a_1, ..., a_n$
- Methode:

while Heap nicht leer do
Gib a_1 aus;
Entferne a_1 aus dem Heap;
Stelle Heap-Eigenschaft für Schlüssel $a_2, ..., a_n$ im Indexbereich 1, ..., n-1 wieder her
od

← danach steht also $\max(a_2, ..., a_n)$ an Position 1 im Array bzw. an der Wurzel

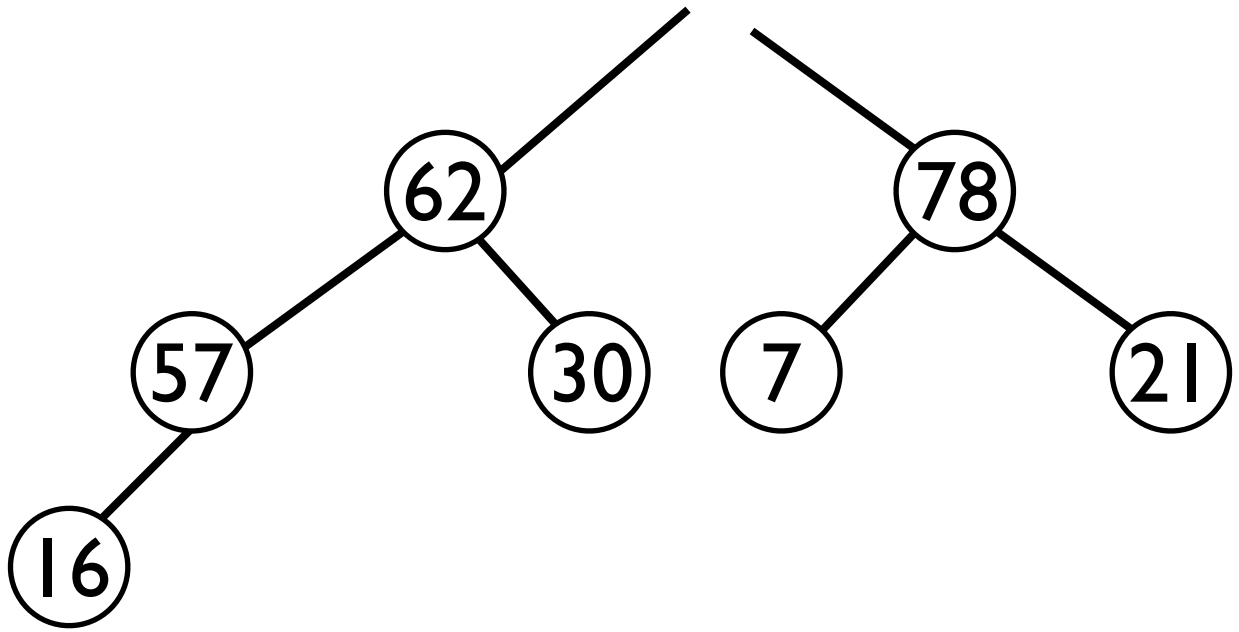
Sortieren

Heapsort - Algorithmus



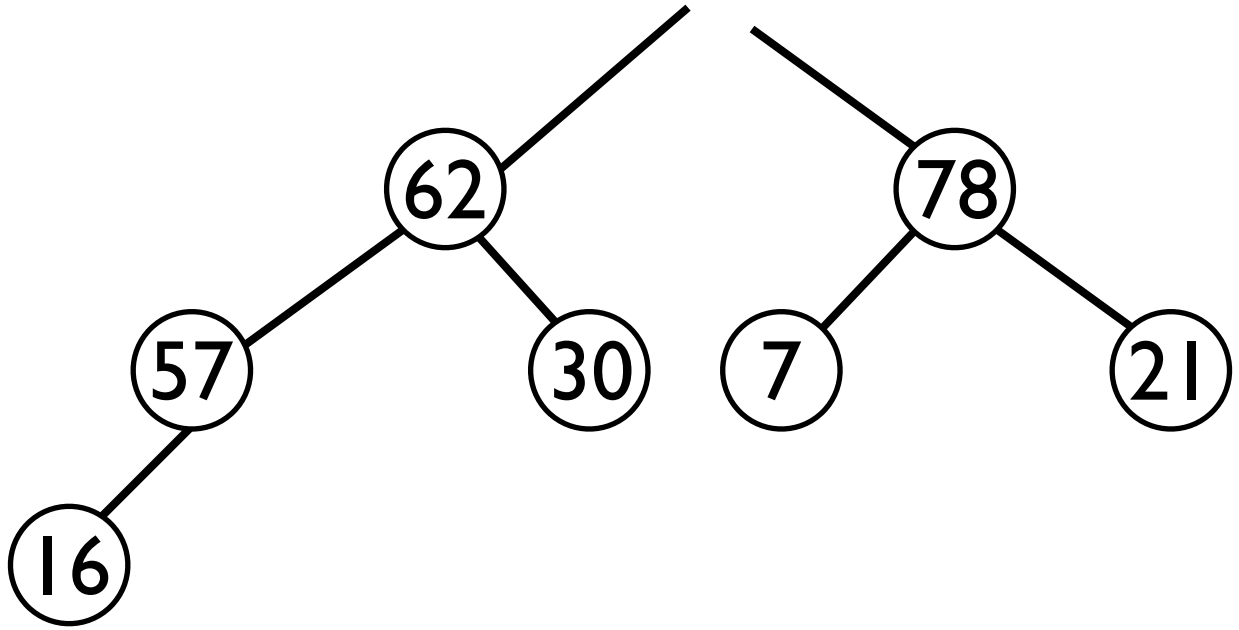
Sortieren

Heapsort - Algorithmus



Sortieren

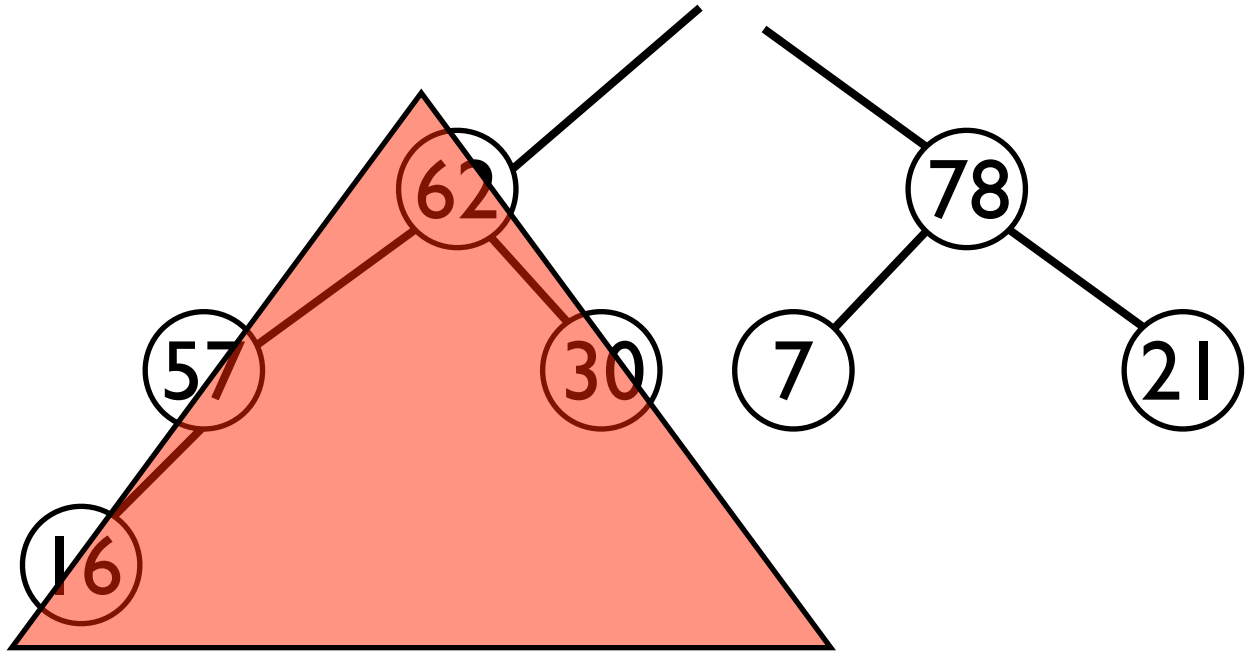
Heapsort - Algorithmus



Problem: Wiederherstellen der Heap-Bedingung

Sortieren

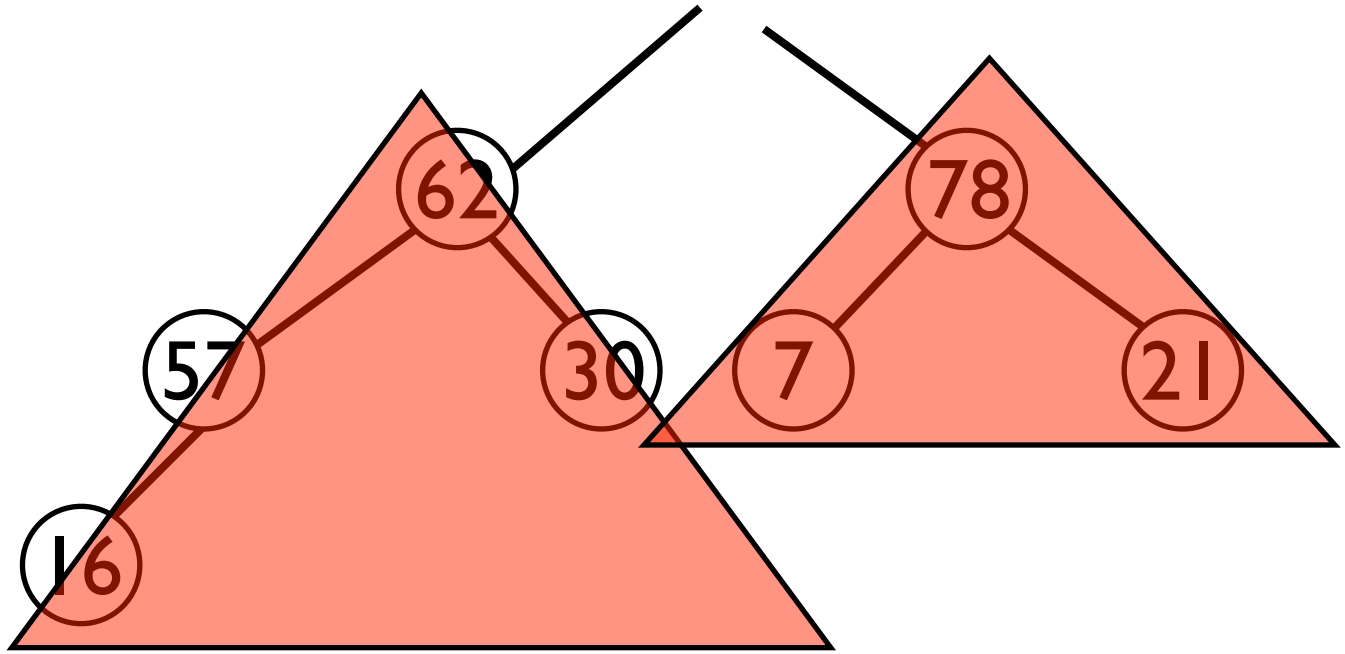
Heapsort - Algorithmus



Problem: Wiederherstellen der Heap-Bedingung

Sortieren

Heapsort - Algorithmus



Problem: Wiederherstellen der Heap-Bedingung

Sortieren
Heapsort - Heap wiederherstellen

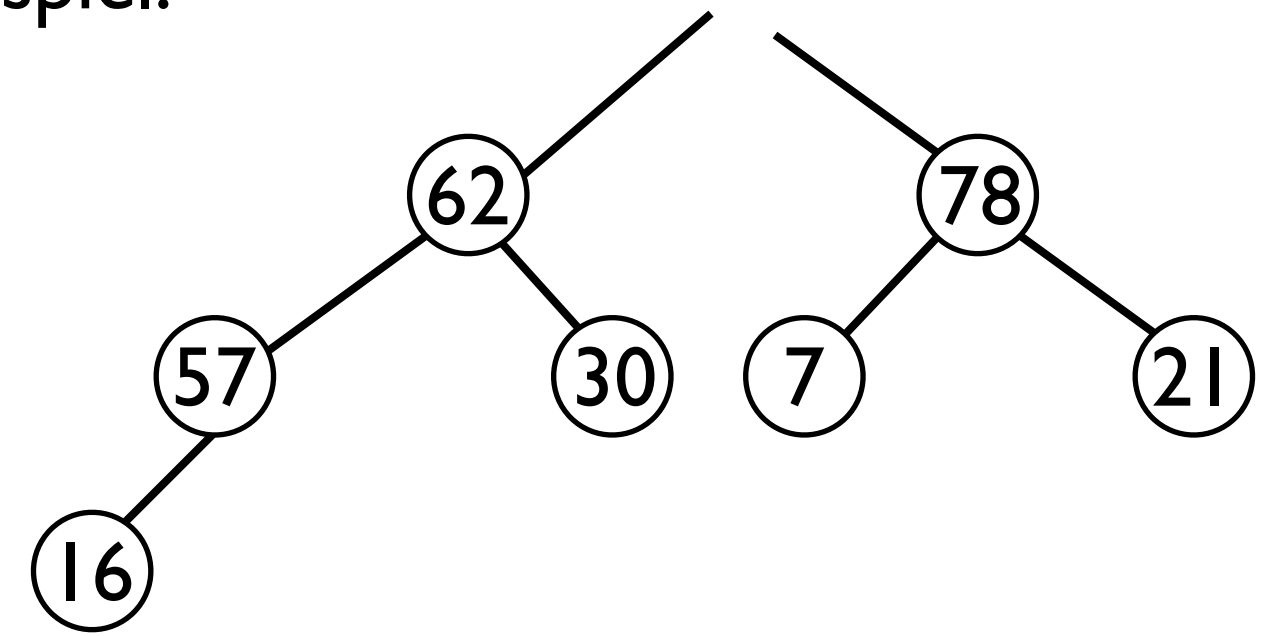
Vorgehen:

- Setze Element mit höchstem Index an Position 1 → meist Heap-Eigenschaft verletzt
- Lasse Element im Heap **absinken**, indem es solange immer wieder mit dem *größeren* der beiden Söhne vertauscht wird, bis beide Söhne kleiner sind oder das Element unten angekommen ist

Sortieren

Heapsort - Heap wiederherstellen

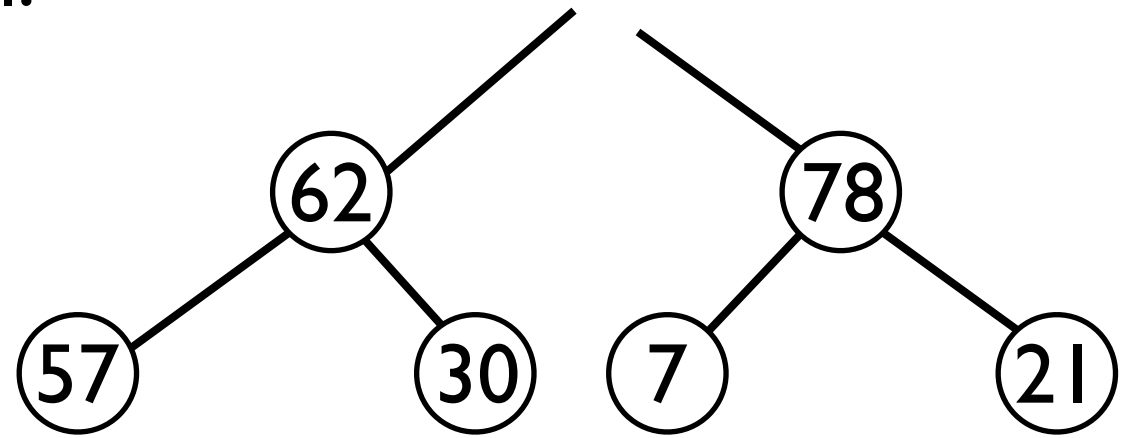
Beispiel:



Sortieren

Heapsort - Heap wiederherstellen

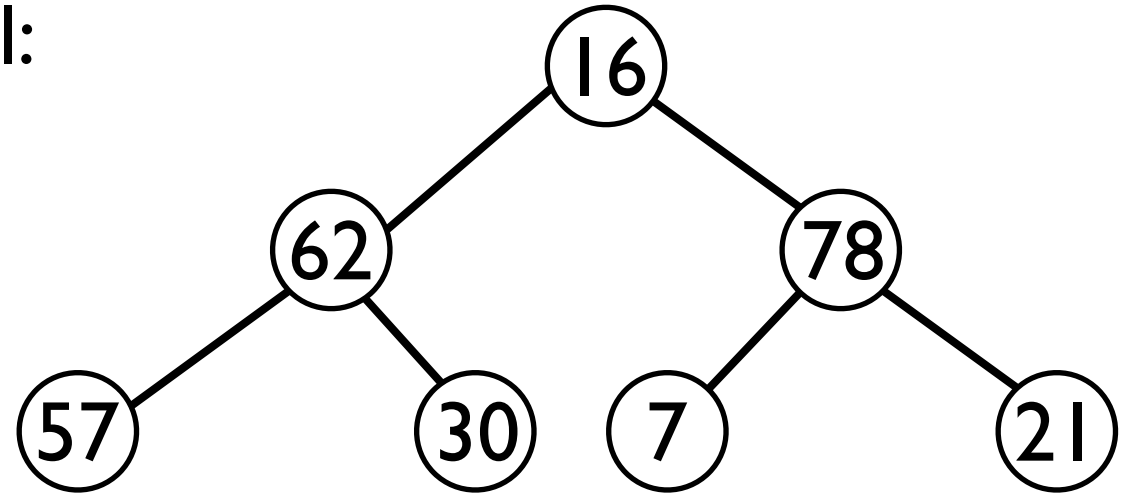
Beispiel:



Sortieren

Heapsort - Heap wiederherstellen

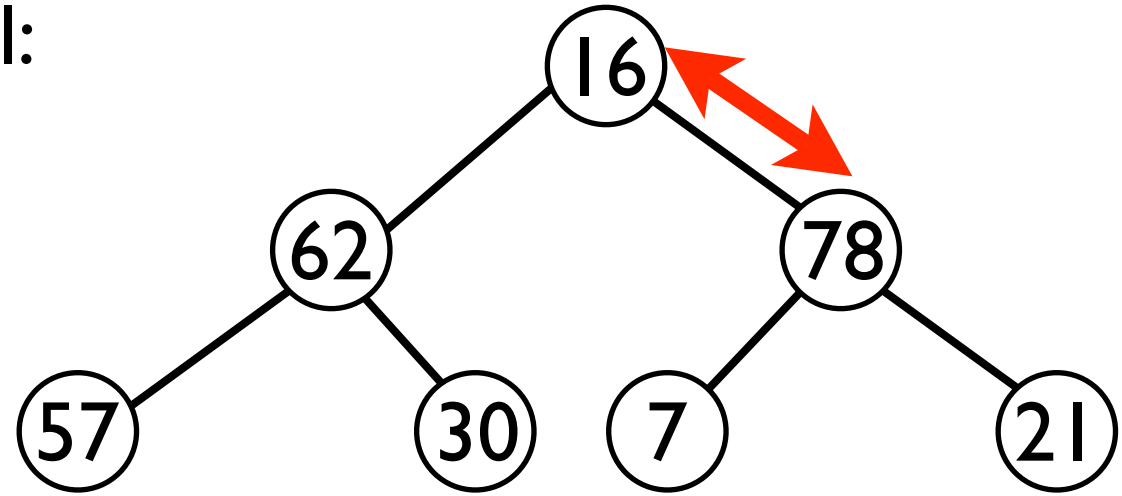
Beispiel:



Sortieren

Heapsort - Heap wiederherstellen

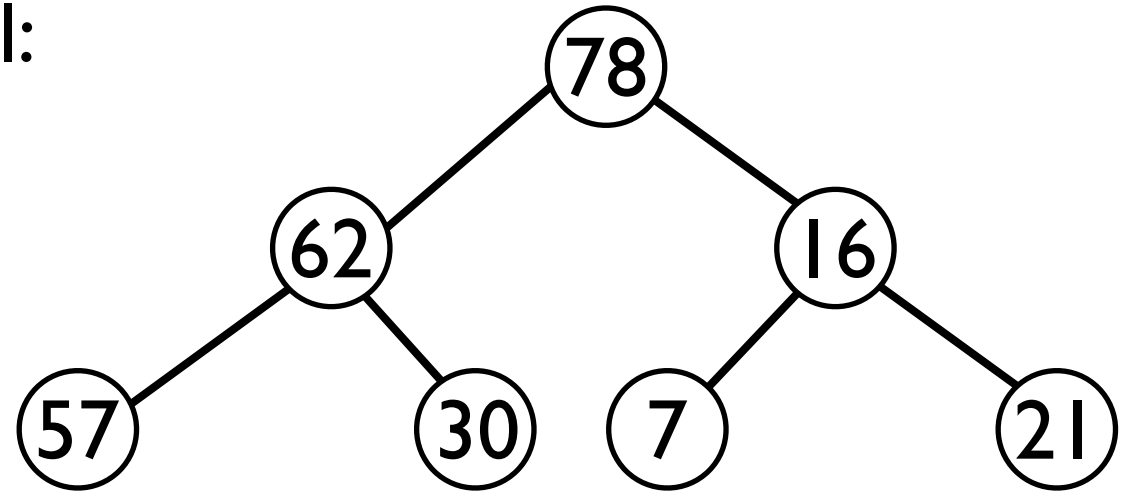
Beispiel:



Sortieren

Heapsort - Heap wiederherstellen

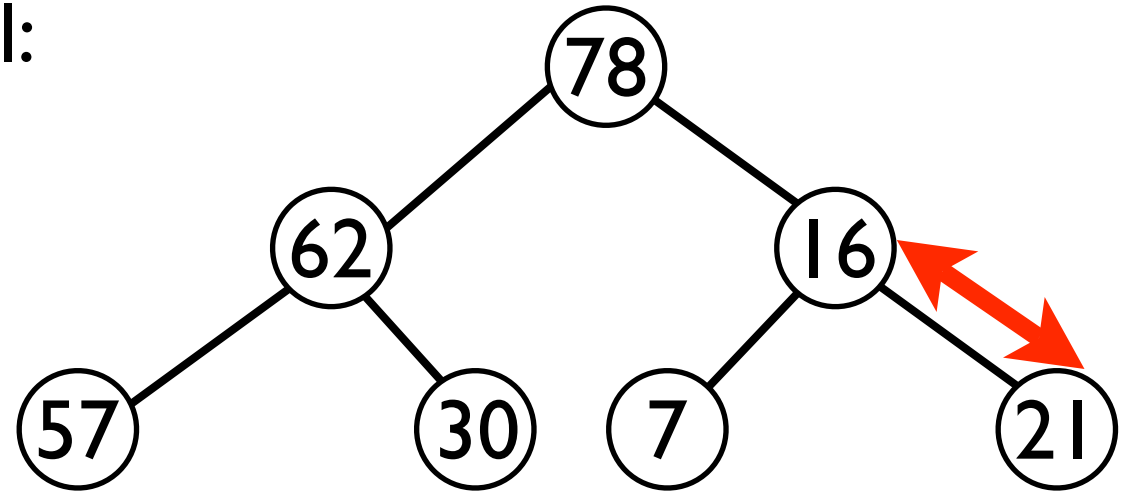
Beispiel:



Sortieren

Heapsort - Heap wiederherstellen

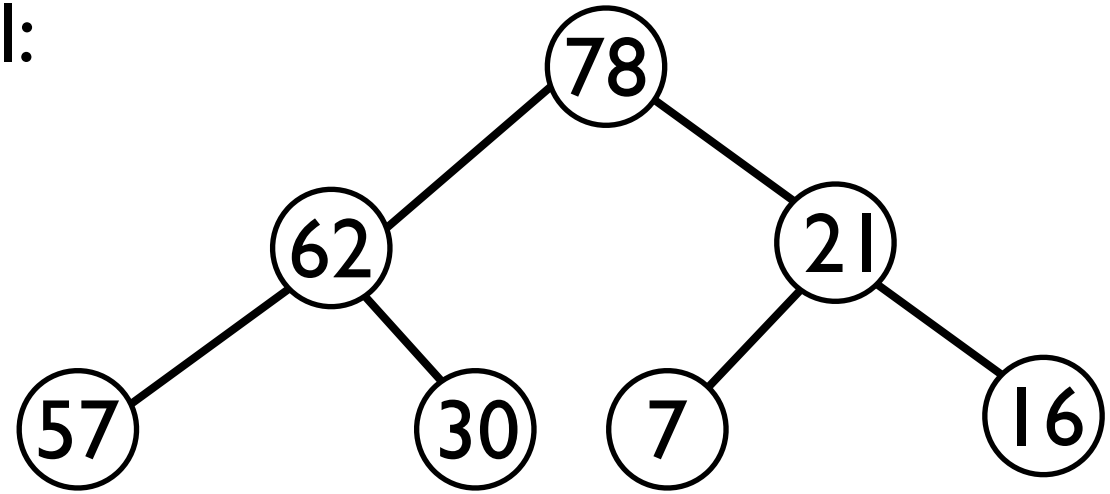
Beispiel:



Sortieren

Heapsort - Heap wiederherstellen

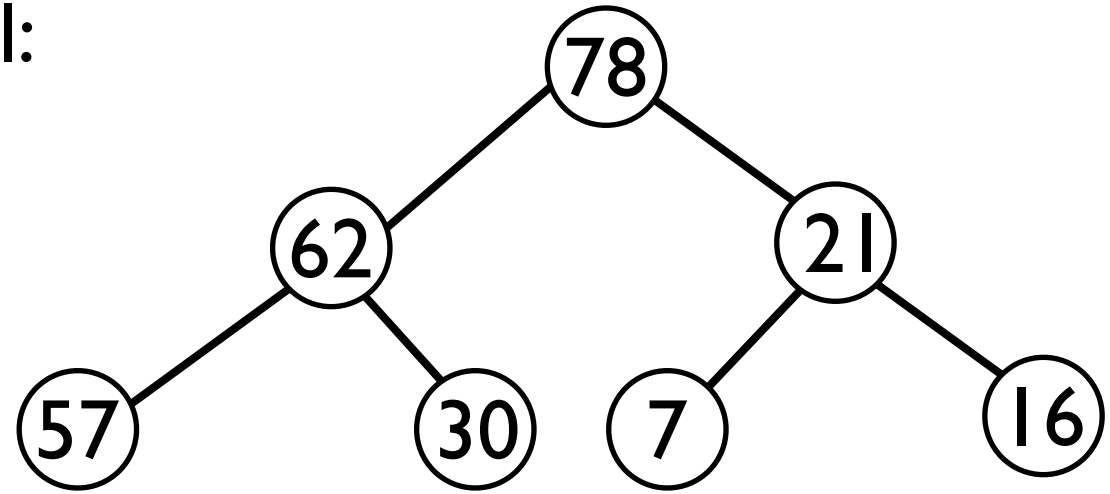
Beispiel:



Sortieren

Heapsort - Heap wiederherstellen

Beispiel:



Heap-Eigenschaft erneut hergestellt

Sortieren
Heapsort - Algorithmus Absinken

Wiederherstellen der Heap-Eigenschaft für $a_1,...,a_{m+1}$:

Setze a_{m+1} an die Stelle von a_1;
while a_i hat linken Sohn do
  if a_i hat rechten Sohn then
    a_j:=Sohn von a_i mit größerem Wert;
  if $a_i<a_j$ then
    vertausche a_i und a_j;
    i:=j
  else
    STOP
od.

Sortieren
Heapsort - Initialisierung

Initialisierung:

- Aufbau des ersten Heaps
- Vorgehen klar: Versenke die zu sortierenden Elemente nacheinander im Heap

Sortieren
Heapsort - Prozedur sink

```
procedure sink (i, t: integer);
var j, k: integer;
begin
    j:=2i; k:=j+1;
    if j≤t then
    begin
        if A[i]≥A[j] then j:=i;
            if k≤t then
                if A[k]>A[j] then j:=k;
            if i≠j then
            begin
                "vertausche A[i] und A[j]";
                sink (j, t)
            end
    end
end.
```

Sortieren
Heapsort - Prozedur sink

```
procedure sink (i, t: integer);
var j, k: integer;
begin
    j:=2i; k:=j+1;
    if j≤t then
    begin
        if A[i]≥A[j] then j:=i;
            if k≤t then
                if A[k]>A[j] then j:=k;
            if i≠j then
            begin
                "vertausche A[i] und A[j]";
                sink (j, t)
            end
    end
end.
```

Absinken des Elements i im Bereich 1..t

Sortieren
Heapsort - Prozedur sink

```
for i:=(n div 2) downto 1 do sink (i, n);
for m:=n downto 2 do
begin
   “vertausche A[1] und A[m]”;
   sink (1, m-1)
end
```

Sortieren
Heapsort - Prozedur sink

Aufbau des Heap

```
for i:=(n div 2) downto 1 do sink (i, n);
for m:=n downto 2 do
begin
   “vertausche A[1] und A[m]”;
   sink (1, m-1)
end
```

Sortieren
Heapsort - Prozedur sink

```
for i:=(n div 2) downto 1 do sink (i, n);
for m:=n downto 2 do
begin
   “vertausche A[1] und A[m]”;
   sink (1, m-1)
end
```

Sortieren
Heapsort - Prozedur sink

```
for i:=(n div 2) downto 1 do sink (i, n);
for m:=n downto 2 do
begin
   “vertausche A[1] und A[m]”;
   sink (1, m-1)
end
```

Sortierphase

Sortieren
Heapsort - Prozedur sink

```
for i:=(n div 2) downto 1 do sink (i, n);
for m:=n downto 2 do
begin
   “vertausche A[1] und A[m]”;
   sink (1, m-1)
end
```

Sortieren
Heapsort - Prozedur sink

```
for i:=(n div 2) downto 1 do sink (i, n);
for m:=n downto 2 do
begin
   “vertausche A[1] und A[m]”;
   sink (1, m-1)
end
```

Nutze, daß Elemente $a_{\lfloor n/2+1 \rfloor},\ldots,a_n$ bereits Heap-Eigenschaft besitzen

Sortieren

Heapsort - Analyse

Sortieren
Heapsort - Analyse

- Laufzeit von sink bestimmt Laufzeit von Heapsort

Sortieren
Heapsort - Analyse

- Laufzeit von sink bestimmt Laufzeit von Heapsort
- Versickern bis max. zu den Blättern durch fortlfd. Vertauschen

Sortieren
Heapsort - Analyse

- Laufzeit von sink bestimmt Laufzeit von Heapsort
- Versickern bis max. zu den Blättern durch fortlfd. Vertauschen
- Bei n Objekten Höhe des Baumes O(log n)

Sortieren
Heapsort - Analyse

- Laufzeit von sink bestimmt Laufzeit von Heapsort
- Versickern bis max. zu den Blättern durch fortlfd. Vertauschen
- Bei n Objekten Höhe des Baumes O(log n)
- Je Versickerungsprozeß O(log n) Schritte im worst-case

Sortieren
Heapsort - Analyse

- Laufzeit von sink bestimmt Laufzeit von Heapsort
- Versickern bis max. zu den Blättern durch fortlfd. Vertauschen
- Bei n Objekten Höhe des Baumes O(log n)
- Je Versickerungsprozeß O(log n) Schritte im worst-case
- Aufbau eines Heaps also O(n log n) [genauer nachrechnen: O(n)]

Sortieren
Heapsort - Analyse

- Laufzeit von sink bestimmt Laufzeit von Heapsort
- Versickern bis max. zu den Blättern durch fortlfd. Vertauschen
- Bei n Objekten Höhe des Baumes O(log n)
- Je Versickerungsprozeß O(log n) Schritte im worst-case
- Aufbau eines Heaps also O(n log n) [genauer nachrechnen: O(n)]
- Sortieren also O(n log n), da genau n Versickerungsschritte auszuführen sind.

Sortieren
Heapsort - Bemerkungen

- echtes in-situ-Verfahren – nur konstant viel Speicher zusätzlich (anders: gängige Varianten von Quicksort)
- Vorsortierung der Daten spielt keine Rolle(anders: gängige Versionen von Quicksort)
- Heapsort ist **nicht** stabil