



Musterlösung: IntToGerman

```
▶ class IntToGerman
{
▶ static final String[] digit =
{
    "", "ein", "zwei", "drei", "vier", "fünf", "sechs",
    "sieben", "acht", "neun", "zehn", "elf", "zwölf"
};
▶ static final String[] ten =
{
    "", "", "zwanzig", "dreißig", "vierzig", "fünfzig",
    "sechzig", "siebzig", "achtzig", "neunzig"
};
```

```
▶ static void main(String[] args)
{
    int num = Integer.parseInt(args[0]);
▶ if(num == 0)
    System.out.println("null");
▶ else
{
    millions(num);
▶ if(num % 100 == 1)
    System.out.println("s");
else
    System.out.println();
}
}
```



Musterlösung: IntToGerman

```
▶ static void tens(int num)
  {
▶   if(num < 13)
     System.out.print(digit[num]);
▶   else if(num < 20)
     System.out.print(digit[num % 10] + "zehn");
▶   else if(num % 10 == 0)
     System.out.print(ten[num / 10]);
▶   else
     System.out.print(digit[num % 10] +
       "und" + ten[num / 10]);
  }
}
```



Musterlösung: IntToGerman

```
▶ static void hundreds(int num)
{
▶   if(num > 99)
      System.out.print(digit[num / 100] +
        "hundert");
▶   tens(num % 100);
}
▶ static void thousands(int num)
{
▶   if(num > 999)
      {
        hundreds(num / 1000);
        System.out.print("tausend");
      }
▶   hundreds(num % 1000);
}
```

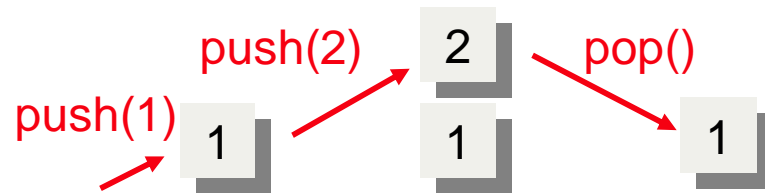
```
▶ static void millions(int num)
{
▶   if(num > 999999)
      {
        hundreds(num / 1000000);
▶   if(num / 1000000 == 1)
        System.out.print("emillion");
      else
        System.out.print("millionen");
      }
▶   thousands(num % 1000000);
}
```



Beispiel: IntToGerman

```
> javac IntToGerman.java  
> java IntToGerman 637239823  
sechshundertsiebenunddreißigmillionenzweihundertneununddreißigtausendachthundert  
dreiundzwanzig  
>
```

Dynamischer Stack



```
▶ class Stack
{
    double[] stack;

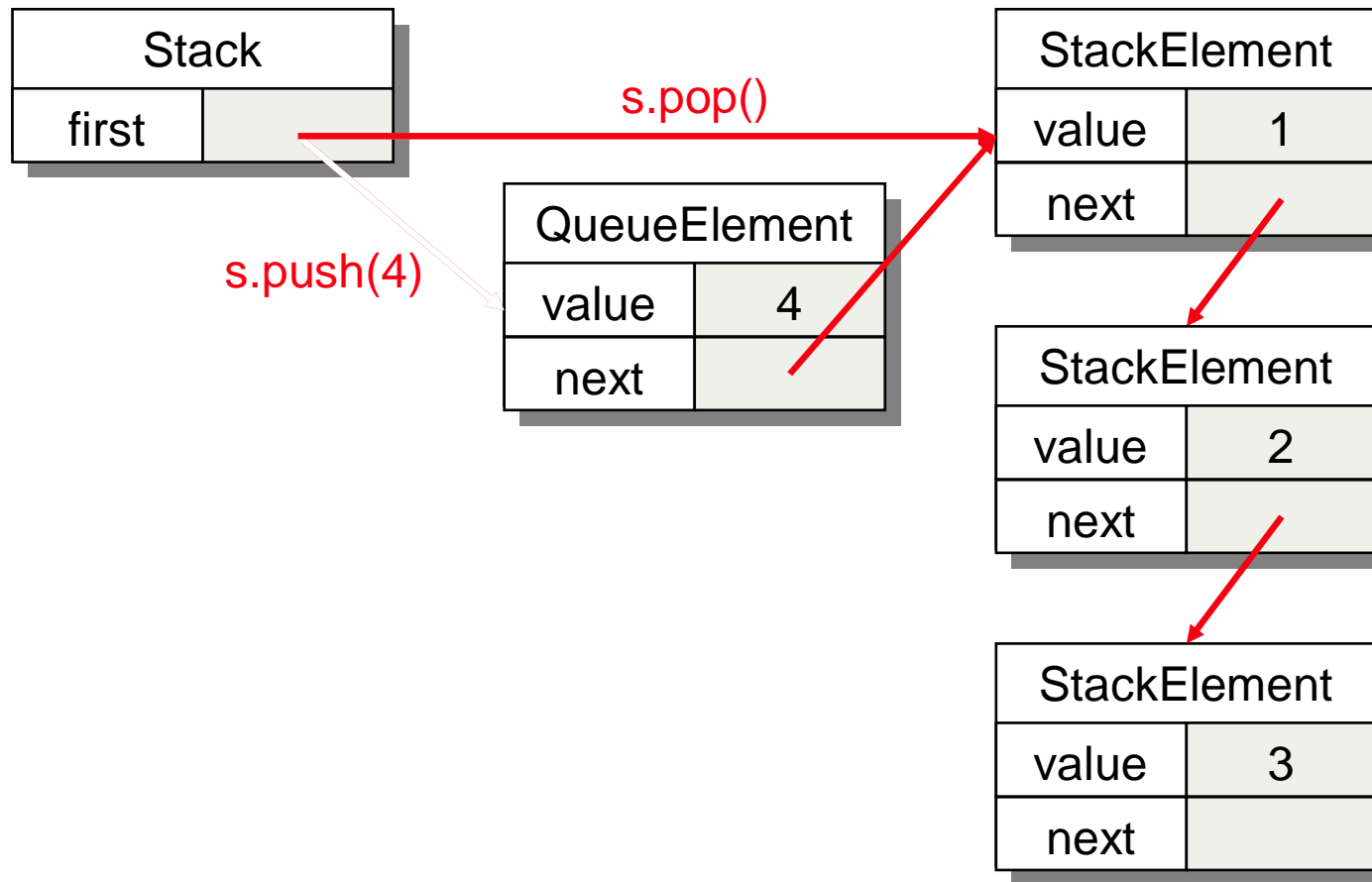
    Stack()
    {
        stack = new double[0];
    }
}
```

```
▶ void push(double d)
{
    double[] old = stack;
    stack = new double[old.length+1];
    for(int i = 0; i < old.length; ++i)
        stack[i] = old[i];
    stack[stack.length-1] = d;
}

▶ double pop()
{
    double[] old = stack;
    stack = new double[old.length-1];
    for(int i = 0; i < stack.length; ++i)
        stack[i] = old[i];
    return old[old.length-1];
}
}
```



Stack als einfach verkettete Liste





Stack als einfach verkettete Liste

```
▶ class StackElement
{
    StackElement next;
    int value;
}

▶ class Stack
{
    StackElement first;

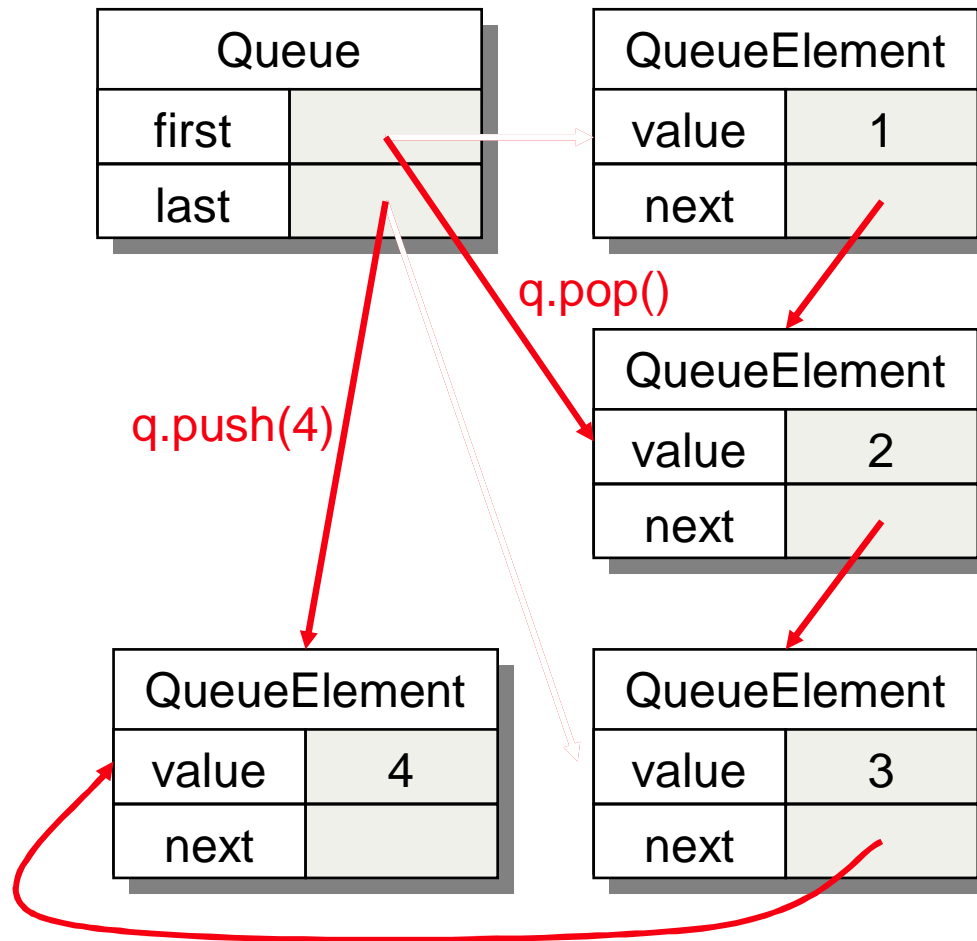
    Stack()
    {
        first = null;
    }

    boolean empty()
    {
        return first == null;
    }
}
```

```
▶ void push(int i)
{
    StackElement s = new StackElement();
    s.value = i;
    s.next = first;
    first = s;
}

▶ int pop()
{
    int i = first.value;
    first = first.next;
    return i;
}
```

Schlange als einfach verkettete Liste





Schlange als einfach verkettete Liste

```
▶ class QueueElement
{
▶   QueueElement next;
  int value;

▶   QueueElement(int i)
  {
    value = i;
    next = null;
  }
}

▶ class Queue
{
▶   QueueElement first,last;

▶   Queue()
  {first = last = null;}

▶   boolean empty()
  {return first == null;}
```

```
▶   void push(int i)
  {
▶     if(empty())
      first = last = new QueueElement(i);
    else
      {
        last.next = new QueueElement(i);
        last = last.next;
      }
  }

▶   int pop()
  {
    int i = first.value;
    first = first.next;
    return i;
  }

▶   int peek()
  {return first.value;}
}
```


Einfach verkettete Liste mit Index

```
▶ class ListElement
{
▶ int value;
  ListElement next;


▶ ListElement(int x)
  {
    value = x;
    next = null;
  }
▶ boolean isLast()
  {
    return next == null;
  }
}
```

```
▶ ListElement getLast()
  {
    if(isLast())
      return this;
    else
      return next.getLast();
  }
▶ void println()
  {
    if(isLast())
      System.out.println(value);
    else
      {
        System.out.print(value + " ");
        next.println();
      }
  }
}
```

ListElement	
value	1
next	



ListElement	
value	2
next	



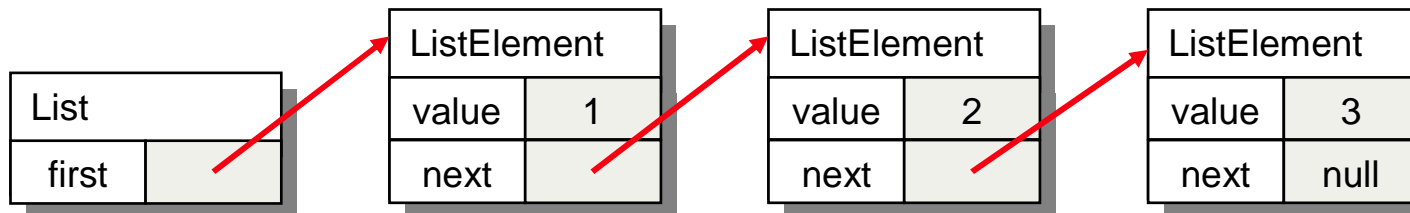
ListElement	
value	3
next	null



Einfach verkettete Liste mit Index

```
▶ class List
{
▶   ListElement first;
▶   List() {first = null;}
▶   boolean isEmpty() {return first == null;}
▶   ListElement getElemAt(int pos)
  {
    ListElement current = first;
    while(pos > 0 && current != null)
    {
      current = current.next;
      --pos;
    }
    return current;
  }
}
```

```
▶ void insertBefore(int pos,int value)
{
▶   ListElement newElem =
    new ListElement(value);
▶   if(pos == 0)
    {
      newElem.next = first;
      first = newElem;
    }
▶   else
    {
      ListElement current = getElemAt(pos - 1);
      newElem.next = current.next;
      current.next = newElem;
    }
}
```

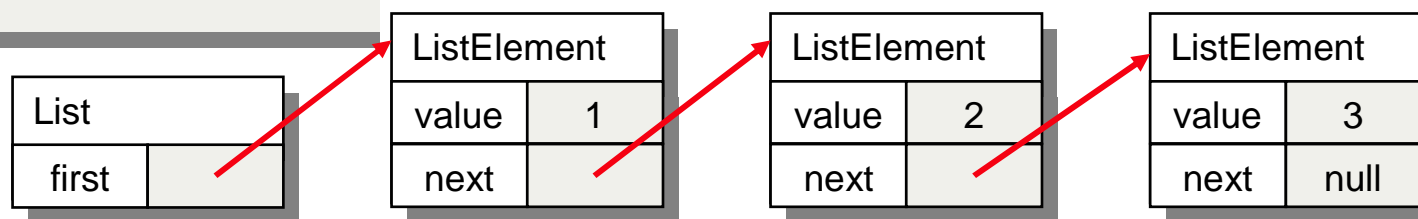




Einfach verkettete Liste mit Index

```
▶ void deleteAt(int pos)
{
▶   if(pos == 0)
      first = first.next; // Exception?
▶   else
      {
        ListElement current = getElemAt(pos - 1);
        current.next = current.next.next; // Excp?
      }
}
▶ void println()
{
  if(isEmpty())
    System.out.println("Liste ist leer!");
  else
    first.println();
}
```

```
▶ static void main(String args[])
{
  List l = new List();
  l.println();
▶   l.insertBefore(0,3);
  l.insertBefore(1,4);
  l.insertBefore(2,5);
  l.insertBefore(0,2);
  l.println();
▶   l.deleteAt(3); l.deleteAt(0);
  l.println();
▶   l.deleteAt(3);
}
```





Beispiel: Einfach verkettete Liste

```
>javac List.java
>java List
Liste ist leer!
2 3 4 5
3 4
Exception in thread "main" java.lang.NullPointerException:
    at List.deleteAt(List.java:72)
    at List.main(List.java:90)
>_
```

Doppelt verkettete Listen

```
▶ class DLListElement
{
  int value;
  DLListElement next,previous;
▶ void insertBehind(int x)
{
  DLListElement newElem = new DLListElement();
  newElem.value = x;
  newElem.next = next;      // newElem.next = this.next
  newElem.previous = this; // newElem.previous = this
  if(next != null)
    next.previous = newElem; // this.next.previous = newElem
  next = newElem;          // this.next = newElem
}
```

DLListElement	
value	1
previous	null
next	

DLListElement	
value	1
previous	
next	

DLListElement	
value	1
previous	
next	null

Doppelt verkettete Listen

```
▶ void delete()
{
    if(previous != null)
        previous.next = next; // this.previous.next = this.next
    if(next != null)
        next.previous = previous; // this.next.previous = this.previous
}
```

DLListElement	
value	1
previous	null
next	

DLListElement	
value	1
previous	
next	

DLListElement	
value	1
previous	
next	null



Sortieren durch Auswählen

▶ Sortierung S von T

- ▶ $S = \text{perm}(T)$, $s_0 \leq s_1 \leq \dots \leq s_{n-1}$

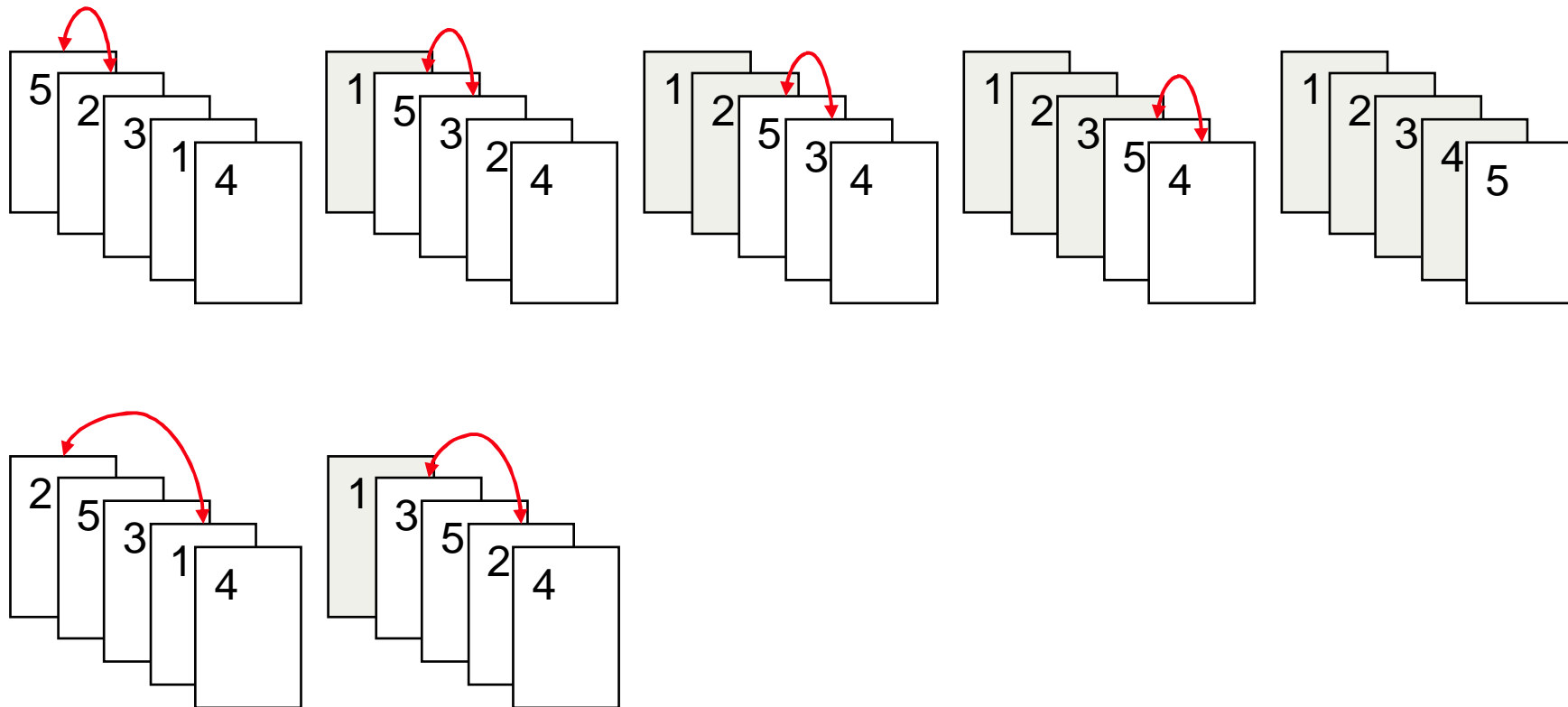
▶ Algorithmus von SelectSort

- ▶ Falls nur noch ein Element übrig ist, ist das Array sortiert.
- ▶ Ansonsten
 - ▶ *Suche nach dem kleinsten Element und stelle es an den Anfang,*
 - ▶ *sortiere den Rest.*

▶ Aufwand

- ▶ In jedem Durchgang wird ein Element ausgewählt
- ▶ Um das Element auszuwählen, müssen im Mittel $n/2$ Elemente durchsucht werden
- ▶ $\Rightarrow O(n/2 * n) = O(n^2/2) = O(n^2)$

Sortieren durch Auswählen





Sortieren durch Auswählen

```
▶ class SelectSort
{
▶ static void sort(int[] a)
{
    for(int i = 0; i < a.length-1; ++i)
        for(int j = i+1; j < a.length; ++j)
            if(a[j] < a[i])
                swap(a,i,j);
}

▶ static void swap(int[] a,int i, int j)
{
    int t = a[i];
    a[i] = a[j];
    a[j] = t;
}
```

```
▶ static void main(String[] args)
{
▶ int[] a = new int[args.length];
▶ for(int i = 0; i < a.length; ++i)
    a[i] = Integer.parseInt(args[i]);
▶ sort(a);
▶ for(int i = 0; i < a.length; ++i)
    System.out.print(a[i] + " ");
}
}
```



Beispiel: Sortieren durch Auswählen

```
> javac SelectSort.java  
> java SelectSort 23 435 657 8778 2332 65645 3332 6767 2121321 87787  
23 435 657 2332 3332 6767 8778 65645 87787 2121321  
> _
```



Sortieren durch Zerlegen (QuickSort)

▶ Algorithmus

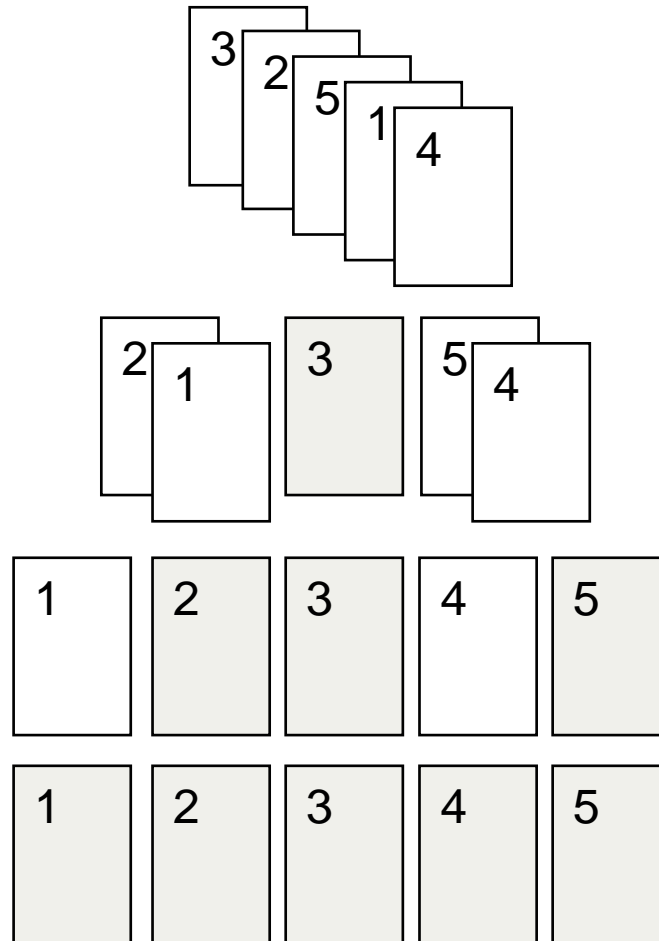
- ▶ Ist die Liste leer, ist sie sortiert
- ▶ Ansonsten
 - ▶ *entnimm ein Element,*
 - ▶ *bilde eine neue Liste aus allen kleineren Elementen und lasse diese sortieren,*
 - ▶ *bilde eine neue Liste aus allen größeren Elementen und lasse diese sortieren,*
 - ▶ *Hänge die sortierte Liste der kleineren Elemente, das gewählte Element und die sortierte Liste der größeren Elemente aneinander und liefere diese zurück.*

▶ Aufwand

- ▶ Im optimalen Fall sind beide Teillisten gleich lang $\Rightarrow O(n \log n)$
- ▶ Im schlechtesten Fall ist immer eine der beiden Teillisten leer $\Rightarrow O(n^2)$

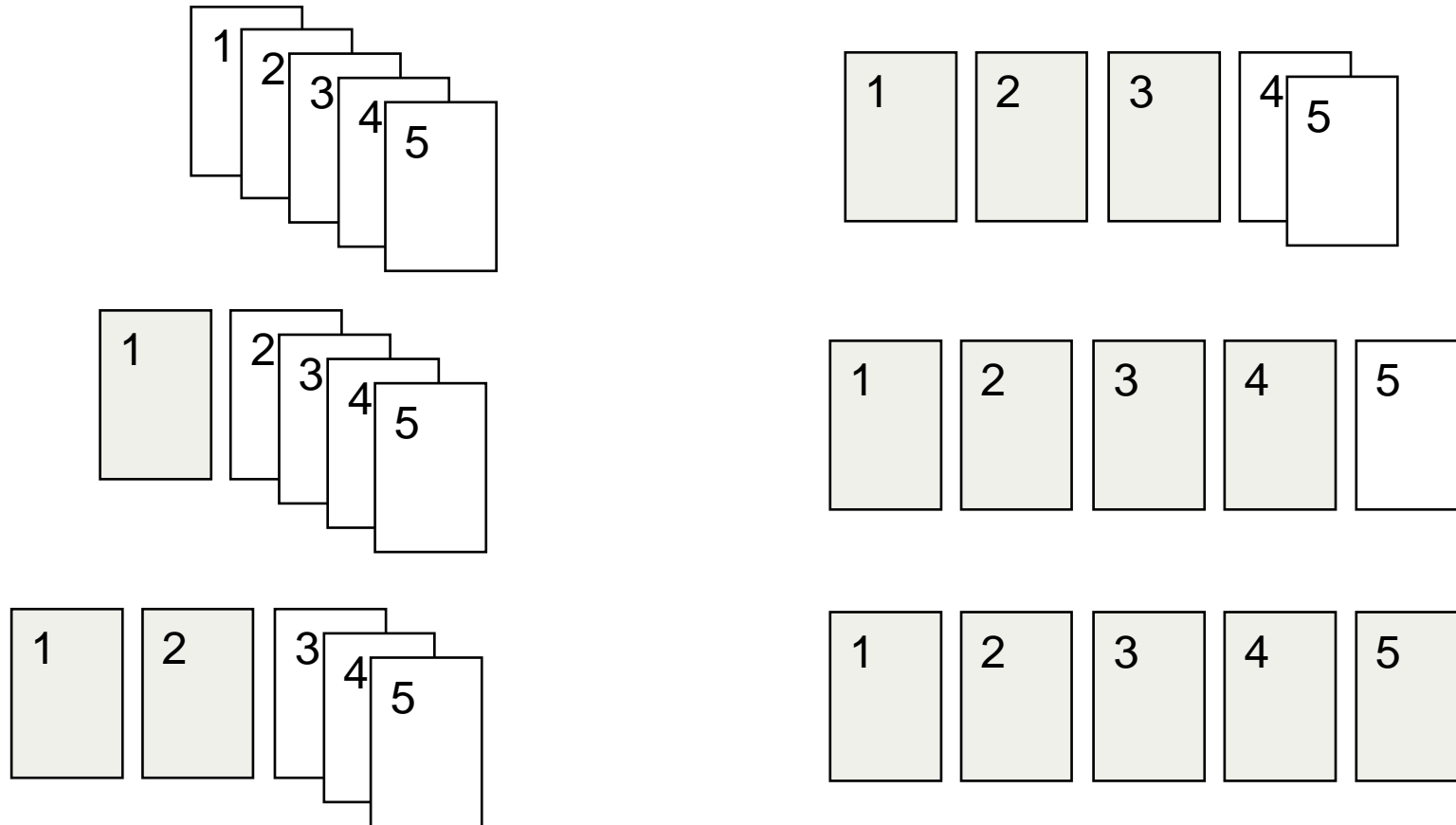


QuickSort: Günstige Vorsortierung





QuickSort: Ungünstige Vorsortierung





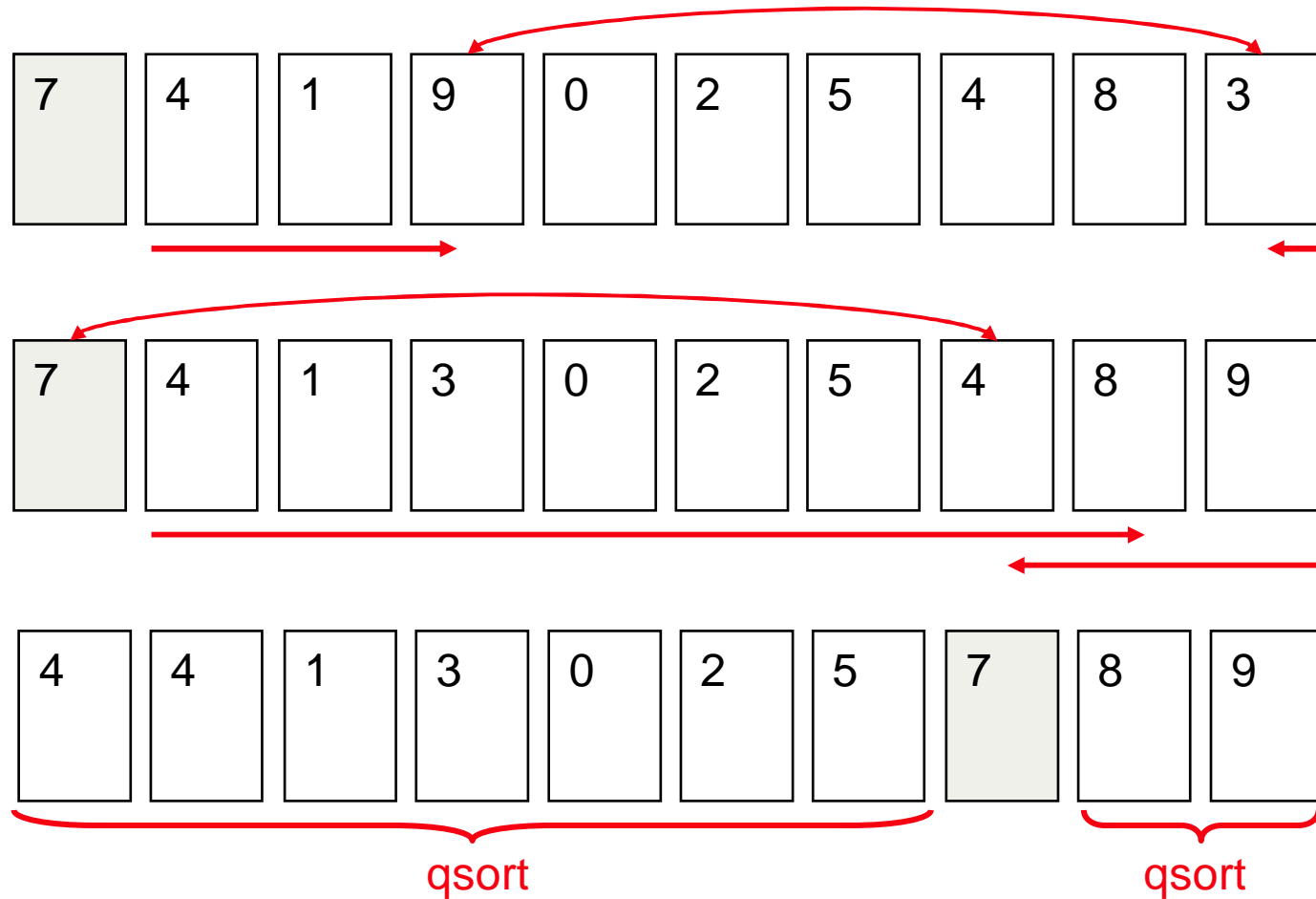
QuickSort mit Schlangen

```
▶ class QuickSort
{
▶ static void sort(Queue q) {
    if(!q.empty())
    {
        int x = q.pop();
        Queue q1 = new Queue(),
            q2 = new Queue();
        split(q,x,q1,q2);
▶ sort(q1);
        sort(q2);
▶ while(!q1.empty())
            q.push(q1.pop());
        q.push(x);
        while(!q2.empty())
            q.push(q2.pop());
    }
}
```

```
▶ static void split(Queue q,int x,Queue q1,Queue q2)
{
    while(!q.empty())
        if(q.peek() < x)
            q1.push(q.pop());
        else
            q2.push(q.pop());
}
▶ static void main(String[] args)
{
    Queue q = new Queue();
    for(int i = 0; i < args.length; ++i)
        q.push(Integer.parseInt(args[i]));
▶ sort(q);
▶ while(!q.empty())
    System.out.print(q.pop() + " ");
}}
```



QuickSort mit Arrays





QuickSort mit Arrays

```
▶ class QuickSort
{
▶ static void qsort(int[] a,int left,int right)
{
▶ if(right > left)
{
▶ int x = a[left], i = left+1, j = right;
▶ do
{
▶ while(i <= right && a[i] < x)
++i;
▶ while(j > left && a[j] >= x)
--j;
▶ if(i < j)
SelectSort.swap(a,i,j);
▶ }
while(i < j);
```

```
▶ SelectSort.swap(a,i-1,left);
▶ qsort(a,left,i-2);
qsort(a,i,right);
}
}
▶ static void sort(int[] a)
{
qsort(a,0,a.length-1);
}
▶ static void main(String[] args)
{
int[] a = new int[args.length];
for(int i = 0; i < a.length; ++i)
a[i] = Integer.parseInt(args[i]);
sort(a);
for(int i = 0; i < a.length; ++i)
System.out.print(a[i] + " ");
}}
```



Übungsaufgabe: Wortstatistik

▶ Aufgabe

- ▶ Erstellen einer Statistik, wie häufig einzelne Wörter in einem Text vorkommen

▶ Ansatz

- ▶ Einlesen der Datei, dabei in Wörter zerlegen
- ▶ Die Wörter werden in einer Liste gespeichert
- ▶ Diese wird sortiert
- ▶ Die sortierte Liste wird dann einmal durchlaufen, um die Anzahl des Vorkommens jedes Wortes zu zählen

▶ Tipps

- ▶ Strings vergleicht man mit

- ▶ $s1 < s2$:

$s1.compareTo(s2) < 0$

- ▶ $s1 = s2$:

$s1.compareTo(s2) == 0$ oder $s1.equals(s2)$

- ▶ $s1 > s2$:

$s1.compareTo(s2) > 0$

- ▶ Groß- und Kleinschreibung ignorieren: `compareToIgnoreCase` statt `compareTo`



Übungsaufgabe: Wortstatistik

```
1      terrific
4      that
17     the
1      then
1      they'll
2      thing
5      this
1      thought
11     to
1      trappings
1      turned
1      up
3      very
2      was
1      way
1      we'll
3      what
1      who
3      will
2      world
4      would
1      write
1      writers
2
```