

# Ich sortiere das. Ich, als Sortierer.

Ordnung 10

Florian Sihler ◦ KW 4



# Präsenzaufgabe

1

Bitte anstellen...

# Präsenzaufgabe

1

Bitte anstellen...

In dieser Aufgabe sollen Sie ihre eigene dynamische Datenstruktur implementieren. Hierbei solle es sich um eine Queue handeln, welche nach dem *FIFO*-Prinzip („first in first out“) arbeitet und Integer Werte speichert.

# Präsenzaufgabe

1

Bitte anstellen...

In dieser Aufgabe sollen Sie ihre eigene dynamische Datenstruktur implementieren. Hierbei solle es sich um eine Queue handeln, welche nach dem *FIFO*-Prinzip („first in first out“) arbeitet und Integer Werte speichert. Intern benötigen Sie hier zusätzlich zur Klasse `Queue` eine innere Klasse um die Elemente repräsentieren zu können.

# Präsenzaufgabe

1

Bitte anstellen...

In dieser Aufgabe sollen Sie ihre eigene dynamische Datenstruktur implementieren. Hierbei solle es sich um eine Queue handeln, welche nach dem *FIFO*-Prinzip („first in first out“) arbeitet und Integer Werte speichert. Intern benötigen Sie hier zusätzlich zur Klasse `Queue` eine innere Klasse um die Elemente repräsentieren zu können.

Implementieren Sie einen Konstruktor für die Klasse `Queue`, welcher die internen Variablen entsprechend initialisiert und zusätzlich die folgenden Methoden:

1. `public void enqueue(int value)` – soll ein neues Element mit dem Wert `value` am Ende der Queue einreihen.
2. `public boolean dequeue()` – soll das vorderste Element aus der `Queue` entfernen und zurückliefern ob das Entfernen möglich war.
3. `public int getLength()` – soll die Anzahl der eingereihten Elemente zurück geben

# Präsenzaufgabe

1

Bitte anstellen...

In dieser Aufgabe sollen Sie ihre eigene dynamische Datenstruktur implementieren. Hierbei solle es sich um eine Queue handeln, welche nach dem *FIFO*-Prinzip („first in first out“) arbeitet und Integer Werte speichert. Intern benötigen Sie hier zusätzlich zur Klasse `Queue` eine innere Klasse um die Elemente repräsentieren zu können.

Implementieren Sie einen Konstruktor für die Klasse `Queue`, welcher die internen Variablen entsprechend initialisiert und zusätzlich die folgenden Methoden:

**Bonusfrage:** Angenommen, Sie wollten die Methode `public boolean dequeue()` implementieren, die den Wert des ersten Elements zurückgibt. Welchen Rückgabetyt würden Sie wählen? Warum?

1. `public void enqueue(int value)` – füge am Ende an.
2. `public boolean dequeue()` – entferne vorderstes Element.
3. `public int getLength()` – Anzahl eingereihter Elemente.
4. `public String toString()` – Formatiert als String.

# Präsenzaufgabe

1

Bitte anstellen...

Implementieren Sie eine (Linked-)Queue Klasse, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Schreiben Sie dafür eine interne Klasse um die Elemente zu repräsentieren. Implementieren Sie weiter:

# Präsenzaufgabe

1

Bitte anstellen...

Implementieren Sie eine (Linked-)Queue Klasse, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Schreiben Sie dafür eine interne Klasse um die Elemente zu repräsentieren. Implementieren Sie weiter:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.



# Präsenzaufgabe

1

Bitte anstellen...

Implementieren Sie eine (Linked-)Queue Klasse, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Schreiben Sie dafür eine interne Klasse um die Elemente zu repräsentieren. Implementieren Sie weiter:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.

*Bonus:* Welchen Rückgabetypp würden Sie für Die Methode `public <?> head()` wählen, die den Wert des ersten Elements zurückgibt. Warum? **Hilfsmethoden sind erlaubt** 😊.

# Präsenzaufgabe

1

Diffamieren

Implementieren Sie eine (Linked-)Queue Klasse, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Schreiben Sie dafür eine interne Klasse um die Elemente zu repräsentieren. Implementieren Sie weiter:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.

*Bonus:* Welchen Rückgabebetyp würden Sie für Die Methode `public (?) head()` wählen, die den Wert des ersten Elements zurückgibt. Warum?      Hilfsmethoden sind erlaubt ☺.

# Präsenzaufgabe

1

Bitte anstellen

Implementieren Sie eine (Linked-)Queue, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Schreiben Sie eine interne Klasse um die Elemente zu repräsentieren. Implementieren Sie

- Den Konstruktor `public Queue()` – initialisiert zuweist.
- `public void enqueue(int value)` – fügt ein Element an.
- `public boolean dequeue()` – entfernt ein Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte der Queue formatiert.

*Bonus:* Welchen Rückgabebetyp würden Sie für Die Methode `public (?) head()` wählen, die den Wert des ersten Elements zurückgibt. Warum? Hilfsmethoden sind erlaubt ☺.



# Präsenzaufgabe

2

Bitte anstellen... die Zweite

Implementieren Sie eine (Linked-)Queue Klasse, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Implementieren Sie dafür:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.

*Bonus:* Welchen Rückgabetypp würden Sie für Die Methode `public <?> head()` wählen, die den Wert des ersten Elements zurückgibt. Warum? **Hilfsmethoden sind erlaubt** 😊.

# Präsenzaufgabe

2

Bitte anstellen... die Zweite

Implementieren Sie eine (Linked-)Queue Klasse, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Implementieren Sie dafür:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.

*Bonus:* Welchen Rückgabetyt würden Sie für Die Methode `public <?> head()` die den Wert des ersten Elements zurückgibt. Warum?

Hilfsmethoden sind



Implementieren Sie eine (Linked-)Queue Klasse, die nach dem FIFO-Prinzip („first in first out“) Integer-Werte verwaltet. Implementieren Sie dafür:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.

Bonus: Welchen Rückgabetyt würden Sie für Die Methode `public <?> head()` wählen, die den Wert des ersten Elements zurückgibt. Warum? **Hilfsmethoden sind erlaubt 😊.**



Implementieren Sie eine (Linked-)Queue Klasse, die nach dem FIFO-Prinzip („first in first out“) Integer-Werte verwaltet. Implementieren Sie dafür:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.

Bonus: Welchen Rückgabetyt würden Sie für Die Methode `public <?> head()` wählen, die den Wert des ersten Elements zurückgibt. Warum? **Hilfsmethoden sind erlaubt 😊.**

Implementieren Sie eine (Linked-)Queue Klasse, die nach dem FIFO-Prinzip („first in first out“) Integer-Werte verwaltet. Implementieren Sie dafür:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` – füge `value` am Ende an.
- `public boolean dequeue()` – entferne das vorderste Element (*false* wenn leer).
- `public int getLength()` – Anzahl der Elemente in der Queue.
- `public String toString()` – die Inhalte als String formatiert.

Bonus: Welchen Rückgabebetyp würden Sie für Die Methode `public <?> head()` wählen, die den Wert des ersten Elements zurückgibt. Warum? **Hilfsmethoden sind erlaubt 😊.**

```
class Queue {
    class Element {
        public final int value;
        public Element next = null;
        public Element(int value) {
            this.value = value;
        }
    }
    // ...
}
```



Implementieren Sie eine (Linked-)Queue Klasse, die nach dem FIFO-Prinzip („first in first out“) Integer-Werte verwaltet. Implementieren Sie dafür:

- Den Konstruktor **public** Queue(), der alles *explizit* zuweist.
- **public void** enqueue(**int** value) – füge value am Ende an.
- **public boolean** dequeue() – entferne das vorderste Element (*false* wenn leer).
- **public int** getLength() – Anzahl der Elemente in der Queue.
- **public String** toString() – die Inhalte als String formatiert.

Bonus: Welchen Rückgabetypp würden Sie für Die Methode **public** <?> head() wählen, die den Wert des ersten Elements zurückgibt. Warum?

Hilfsmethoden sind erlaubt 😊.

Verwenden Sie für ihre Implementierung keine Arrays oder vorgefertigten dynamischen Datenstrukturen, sondern implementieren Sie die Queue selbst als (einfach) verkettete Liste.

```
class Queue {
    class Element {
        public final int value;
        public Element next = null;
        public Element(int value) {
            this.value = value;
        }
    }
    // ...
}
```

# Präsenzaufgabe - Lösung

*Or how to make  $\sim 100$  Lines Musterlösung verkraftbar*

- Wir beginnen mit der Grundstruktur: `Queue.java`

# Präsenzaufgabe - Lösung

*Or how to make ~100 Lines Musterlösung verkraftbar*

- Wir beginnen mit der Grundstruktur: `Queue.java`

```
public class Queue {
```

```
}
```



- Wir beginnen mit der Grundstruktur: `Queue.java`

```
public class Queue {  
    class Element {  
        private final int value;  
  
    }  
  
}
```

- Wir beginnen mit der Grundstruktur: `Queue.java`

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
  
    }  
  
}
```

- Wir beginnen mit der Grundstruktur: `Queue.java`

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        // ...  
    }  
  
}
```



- Wir beginnen mit der Grundstruktur: `Queue.java`

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        // ...  
    }  
  
    private Element first;  
    private Element last;  
    private int length;  
  
}
```

- Wir beginnen mit der Grundstruktur: `Queue.java`

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        // ...  
    }  
  
    private Element first;  
    private Element last;  
    private int length;  
    // ...  
}
```



```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
  
    }  
    // ...  
}
```

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        public Element(int value) {  
  
        }  
  
    }  
  
    }  
    // ...  
}
```

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        public Element(int value) {  
            this.value = value;  
            this.next = null;  
        }  
    }  
    // ...  
}
```

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        public Element(int value) {  
            this.value = value;  
            this.next = null;  
        }  
  
        public void setNext(Element next) { this.next = next; }  
  
    }  
    // ...  
}
```

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        public Element(int value) {  
            this.value = value;  
            this.next = null;  
        }  
  
        public void setNext(Element next) { this.next = next; }  
        public Element getNextElement() { return this.next; }  
    }  
    // ...  
}
```



```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        public Element(int value) {  
            this.value = value;  
            this.next = null;  
        }  
  
        public void setNext(Element next) { this.next = next; }  
        public Element getNextElement() { return this.next; }  
        public int getValue() { return this.value; }  
    }  
    // ...  
}
```

```
public class Queue {  
    class Element {  
        private final int value;  
        private Element next;  
        public Element(int value) {  
            this.value = value;  
            this.next = null;  
        }  
  
        public void setNext(Element next) { this.next = next; }  
        public Element getNextElement() { return this.next; }  
        public int getValue() { return this.value; }  
    }  
    // ...  
}
```

Oder kompakter:

```
class Element {  
    public final int value;  
    public Element next = null;  
    public Element(int value) {  
        this.value = value;  
    }  
}
```





```
public class Queue {  
    class Element { ... }  
    private Element first, last;  
    private int length;  
  
}
```

```
public class Queue {  
    class Element { ... }  
    private Element first, last;  
    private int length;  
  
    public Queue() {  
  
    }  
}
```

```
public class Queue {  
    class Element { ... }  
    private Element first, last;  
    private int length;  
  
    public Queue() {  
        this.first = null;  
        this.last = null;  
        this.length = 0;  
    }  
}
```



```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
  
}
```



```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public void enqueue(int value) {  
  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public void enqueue(int value) {  
        Element newElement = new Element(value);  
  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public void enqueue(int value) {  
        Element newElement = new Element(value);  
        if(length == 0) { // Sonderfall: Erstes  
  
        } else {  
  
        }  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public void enqueue(int value) {  
        Element newElement = new Element(value);  
        if(length == 0) { // Sonderfall: Erstes  
            this.first = this.last = newElement;  
        } else {  
  
        }  
    }  
}
```

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public void enqueue(int value) {
        Element newElement = new Element(value);
        if(length == 0) { // Sonderfall: Erstes
            this.first = this.last = newElement;
        } else {
            this.last.next = newElement;
            this.last = newElement; // Verschieben
        }
    }
}
```

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public void enqueue(int value) {
        Element newElement = new Element(value);
        if(length == 0) { // Sonderfall: Erstes
            this.first = this.last = newElement;
        } else {
            this.last.next = newElement;
            this.last = newElement; // Verschieben
        }
        length++;
    }
}
```



```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
  
}
```



```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public boolean dequeue() {  
  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public boolean dequeue() {  
        if(length > 0) {  
  
        }  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public boolean dequeue() {  
        if(length > 0) {  
            this.first = this.first.next; // Verschieben  
        }  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public boolean dequeue() {  
        if(length > 0) {  
            this.first = this.first.next; // Verschieben  
            length--;  
        }  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public boolean dequeue() {  
        if(length > 0) {  
            this.first = this.first.next; // Verschieben  
            length--;  
            if(length == 0)  
                this.last = null;  
        }  
    }  
}
```

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public boolean dequeue() {
        if(length > 0) {
            this.first = this.first.next; // Verschieben
            length--;
            if(length == 0)
                this.last = null;
            return true;
        }
    }
}
```

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public boolean dequeue() {
        if(length > 0) {
            this.first = this.first.next; // Verschieben
            length--;
            if(length == 0)
                this.last = null;
            return true;
        }
        return false;
    }
}
```





```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public int getLength() { return this.length; }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public int getLength() { return this.length; }  
    public String toString() {  
  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public int getLength() { return this.length; }  
    public String toString() {  
        if(length <= 0) return "[]";  
    }  
}
```

```
public class Queue {  
    class Element {  
        public final int value; public Element next = null;  
        public Element(int value) { this.value = value; }  
    }  
    private Element first, last; private int length;  
    public int getLength() { return this.length; }  
    public String toString() {  
        if(length <= 0) return "[]";  
  
        String s = "[" + this.first.value;  
  
        }  
}
```

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public int getLength() { return this.length; }
    public String toString() {
        if(length <= 0) return "[]";

        String s = "[" + this.first.value;
        for (Element cur = this.first.next; cur != null; cur = cur.next) {

        }
    }
}
```

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public int getLength() { return this.length; }
    public String toString() {
        if(length <= 0) return "[]";

        String s = "[" + this.first.value;
        for (Element cur = this.first.next; cur != null; cur = cur.next) {
            s += ", " + cur.value;
        }

    }
}
```

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public int getLength() { return this.length; }
    public String toString() {
        if(length <= 0) return "[]";

        String s = "[" + this.first.value;
        for (Element cur = this.first.next; cur != null; cur = cur.next) {
            s += ", " + cur.value;
        }
        return s + "]";
    }
}
```





- Doch was machen wir mit `public` `<?>` `head()`?

- Doch was machen wir mit `public` `<?>` `head()`?
- Idee: `int`.

- Doch was machen wir mit `public` `<?>` `head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?

- Doch was machen wir mit `public` `<?>` `head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!

- Doch was machen wir mit `public` `<?>` `head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!
- Idee: `Integer`.

- Doch was machen wir mit `public` `<?>` `head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!
- Idee: `Integer`.
  - Warum? Es erlaubt `null` als Rückgabe.

- Doch was machen wir mit `public` `<?>` `head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!
- Idee: `Integer`.
  - Warum? Es erlaubt `null` als Rückgabe.
  - Problem: Wir sollten kein `null` zurückgeben.



- Doch was machen wir mit `public` `<?>` `head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!
- Idee: `Integer`.
  - Warum? Es erlaubt `null` als Rückgabe.
  - Problem: Wir sollten kein `null` zurückgeben.
  - Abhilfe: Für solche Fälle gibt es `Optional<T>`.

- Doch was machen wir mit `public <?> head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!
- Idee: `Integer`.
  - Warum? Es erlaubt `null` als Rückgabe.
  - Problem: Wir sollten kein `null` zurückgeben.
  - Abhilfe: Für solche Fälle gibt es `Optional<T>`.
- Idee: Ein eigener ternärer/optionaler Wert (`MayQueueValue.java`).

```
public class MayQueueValue {  
    private final boolean stored; private final int value;  
    // ...  
}
```

- Doch was machen wir mit `public <?> head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!
- Idee: `Integer`.
  - Warum? Es erlaubt `null` als Rückgabe.
  - Problem: Wir sollten kein `null` zurückgeben.
  - Abhilfe: Für solche Fälle gibt es `Optional<T>`.
- Idee: Ein eigener ternärer/optionaler Wert (`MayQueueValue.java`).

```
public class MayQueueValue {  
    private final boolean stored; private final int value;  
    // ...  
}
```

- Problem: Wirkt ein wenig Overkill für etwas, das eigentlich illegal ist! —> Exception!

# Übungsblatt 10 - Aufgabe 1

# Übungsblatt 10 - Aufgabe 1

Sortieren Sie die Zahlenfolge 8, 6, 3, 5, 1, 2, 7, 4 grafisch mit den folgenden Sortieralgorithmen:

# Übungsblatt 10 - Aufgabe 1

Sortieren Sie die Zahlenfolge 8, 6, 3, 5, 1, 2, 7, 4 grafisch mit den folgenden Sortieralgorithmen:

- Mergesort

# Übungsblatt 10 - Aufgabe 1

Sortieren Sie die Zahlenfolge 8, 6, 3, 5, 1, 2, 7, 4 grafisch mit den folgenden Sortieralgorithmen:

- Mergesort
- Quicksort — Nutzen Sie jeweils das erste Element als Pivotelement.

# Übungsblatt 10 - Aufgabe 1

Sortieren Sie die Zahlenfolge 8, 6, 3, 5, 1, 2, 7, 4 grafisch mit den folgenden Sortieralgorithmen:

- Mergesort
- Quicksort — Nutzen Sie jeweils das erste Element als Pivotelement.
- Insertionsort



# Übungsblatt 10 - Aufgabe 1

Sortieren Sie die Zahlenfolge 8, 6, 3, 5, 1, 2, 7, 4 grafisch mit den folgenden Sortieralgorithmen:

- Mergesort
- Quicksort — Nutzen Sie jeweils das erste Element als Pivotelement.
- Insertionsort



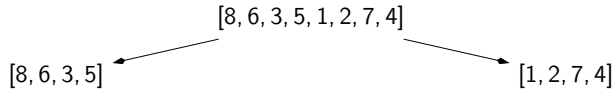
Ich nehmen für die Folien an, dass wir immer *aufsteigend* sortieren. Alle andere Sortierweisen (wie *absteigend*) gehen analog.

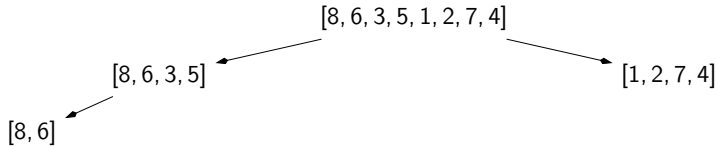
[8, 6, 3, 5, 1, 2, 7, 4]

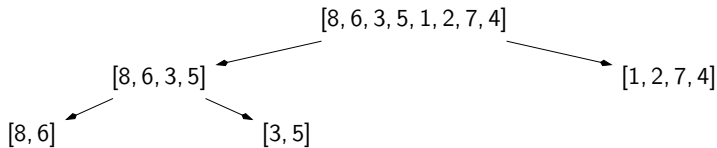
[8, 6, 3, 5, 1, 2, 7, 4]

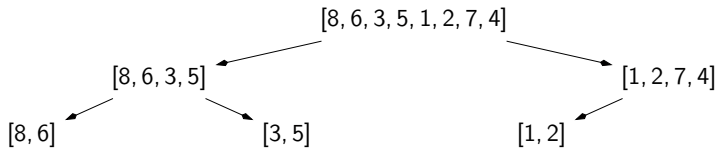
↙

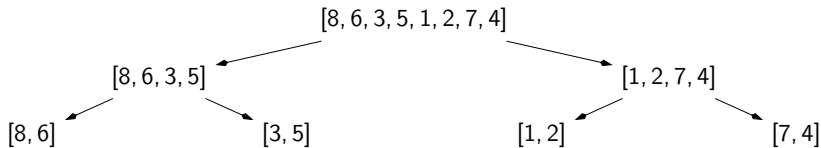
[8, 6, 3, 5]



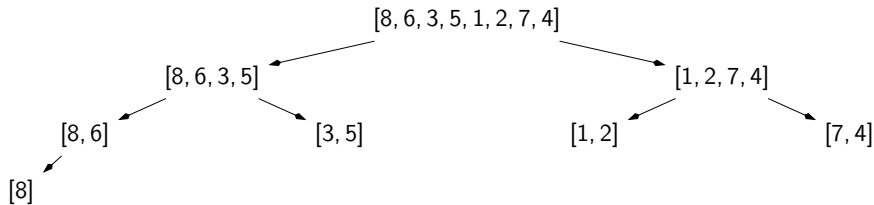


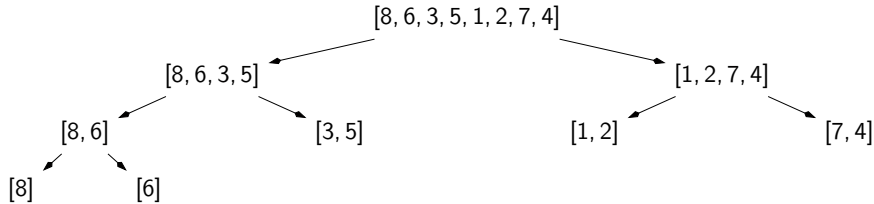


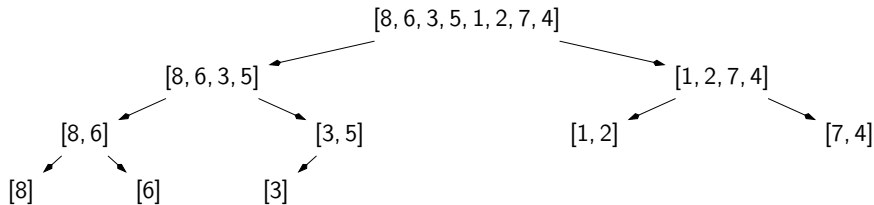


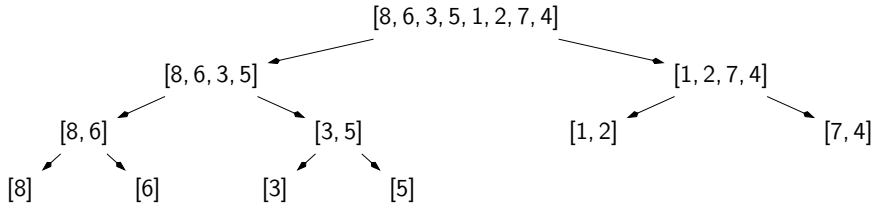


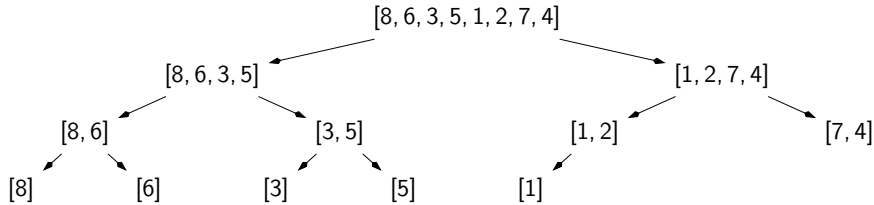


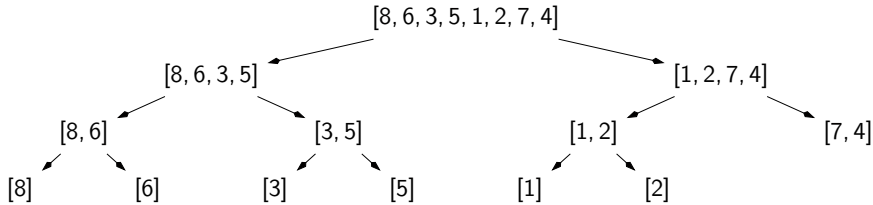


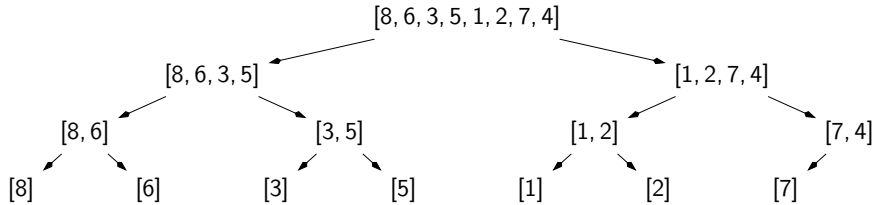


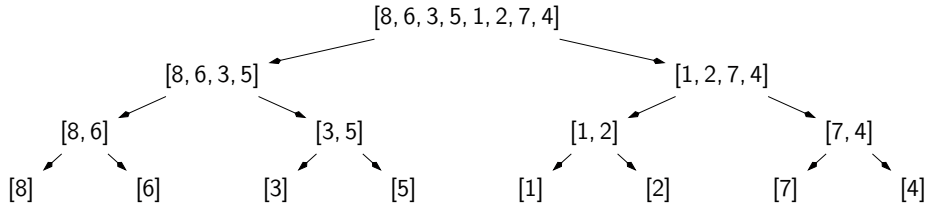




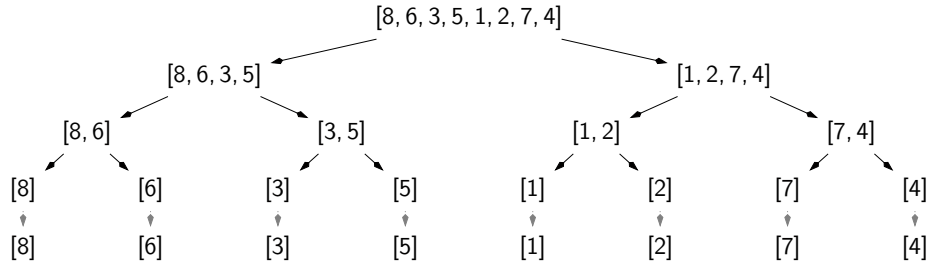


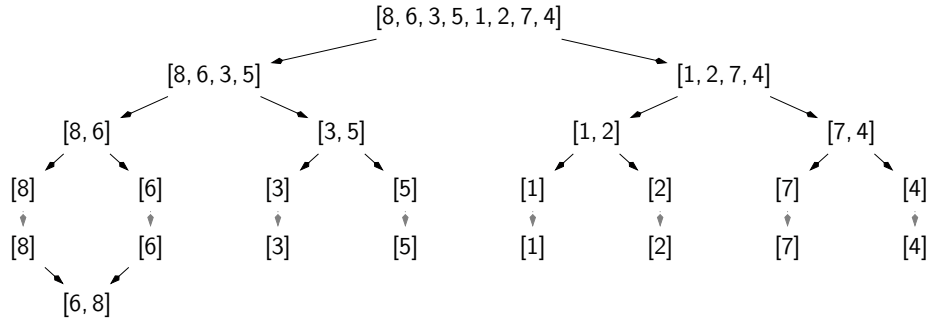


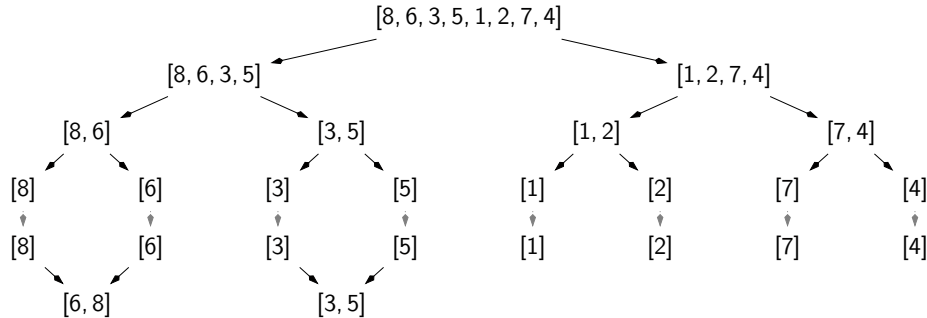


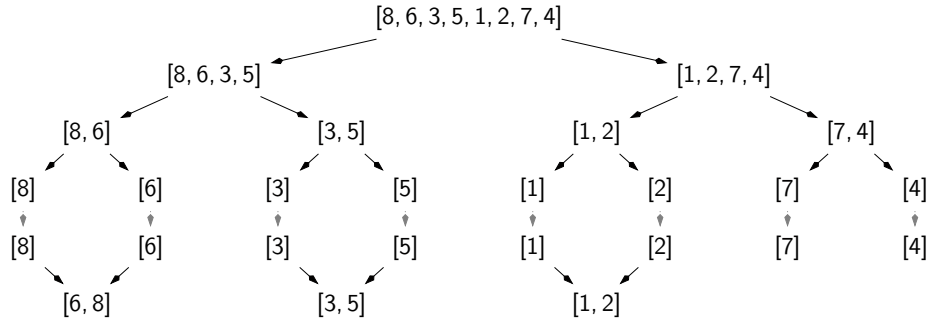


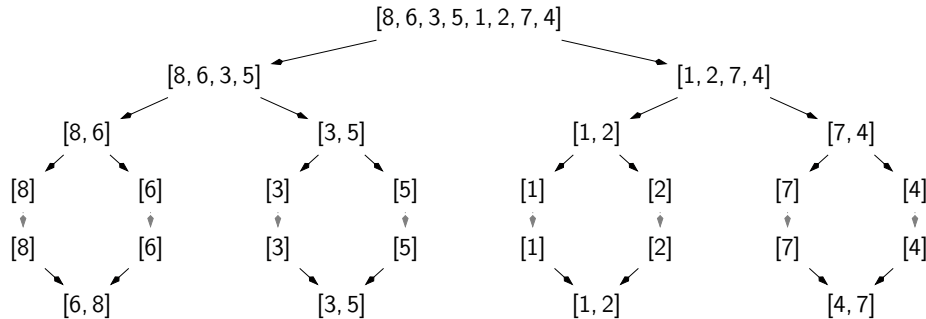


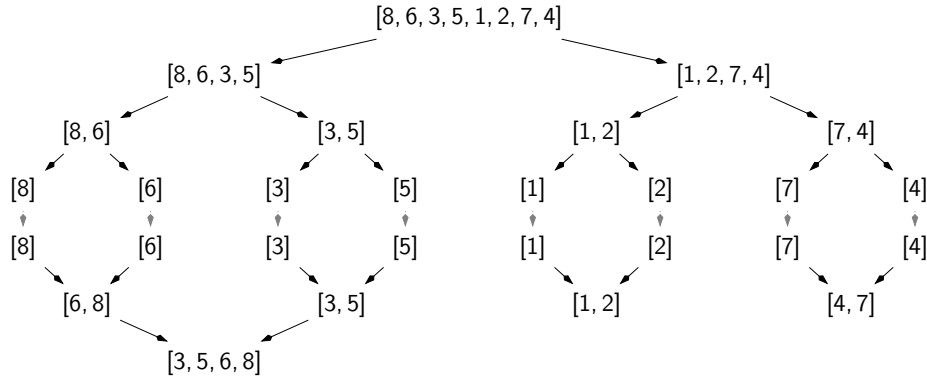


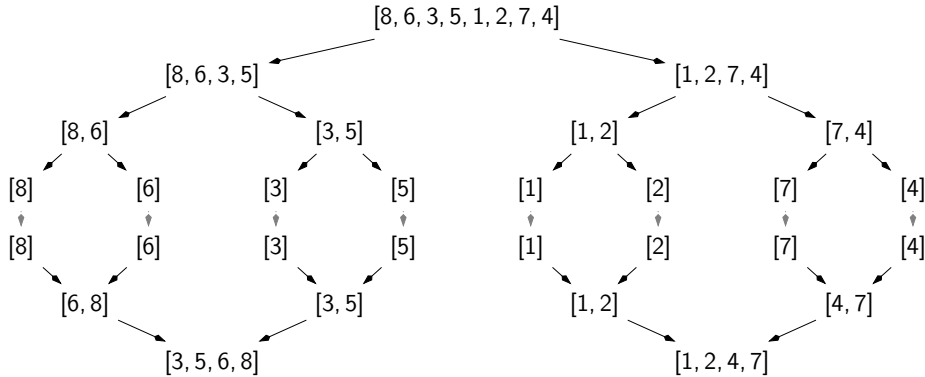


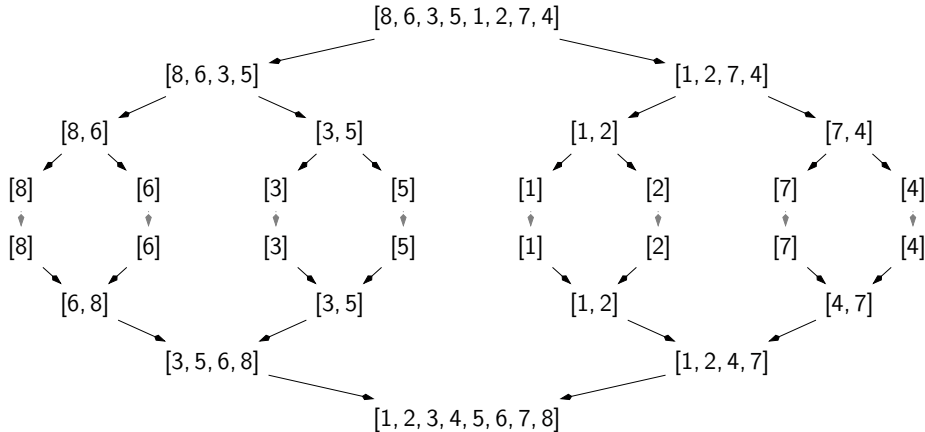




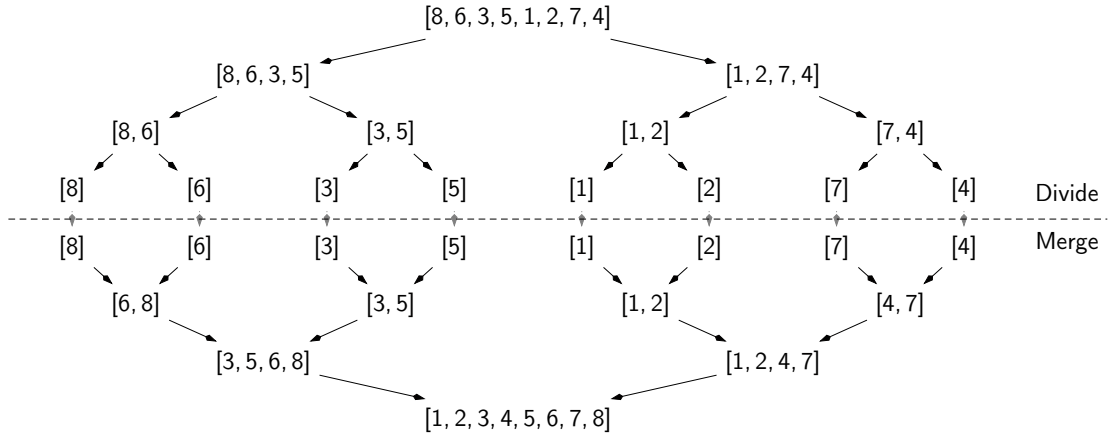














- Quicksort kann unterschiedlich angewendet werden.

- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:

- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:
  - Eine einfache mit Listen (die direkte Umsetzung, genügt nicht).

- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:
  - Eine einfache mit Listen (die direkte Umsetzung, genügt nicht).
  - Die Version der Vorlesung (Pivot ans Ende, Marker)

- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:
  - Eine einfache mit Listen (die direkte Umsetzung, genügt nicht).
  - Die Version der Vorlesung (Pivot ans Ende, Marker)
  - Eine etwas schnellere Version (nur mit Markern).

- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:
  - Eine einfache mit Listen (die direkte Umsetzung, genügt nicht).
  - Die Version der Vorlesung (Pivot ans Ende, Marker)
  - Eine etwas schnellere Version (nur mit Markern).
- Alle folgen dem dem bekannten Schema:



- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:
  - Eine einfache mit Listen (die direkte Umsetzung, genügt nicht).
  - Die Version der Vorlesung (Pivot ans Ende, Marker)
  - Eine etwas schnellere Version (nur mit Markern).
- Alle folgen dem dem bekannten Schema:
  - Wenn Liste einelementig oder leer: fertig.

- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:
  - Eine einfache mit Listen (die direkte Umsetzung, genügt nicht).
  - Die Version der Vorlesung (Pivot ans Ende, Marker)
  - Eine etwas schnellere Version (nur mit Markern).
- Alle folgen dem dem bekannten Schema:
  - Wenn Liste einelementig oder leer: fertig.
  - Wähle Pivotelement und teile Liste in „kleiner“ und „größergleich“.

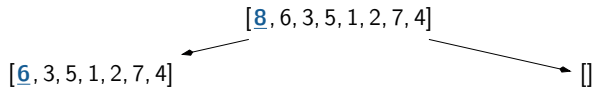
- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten drei verschiedene Varianten:
  - Eine einfache mit Listen (die direkte Umsetzung, genügt nicht).
  - Die Version der Vorlesung (Pivot ans Ende, Marker)
  - Eine etwas schnellere Version (nur mit Markern).
- Alle folgen dem dem bekannten Schema:
  - Wenn Liste einelementig oder leer: fertig.
  - Wähle Pivotelement und teile Liste in „kleiner“ und „größergleich“.
  - Wiederhole rekursiv für beide Teile.

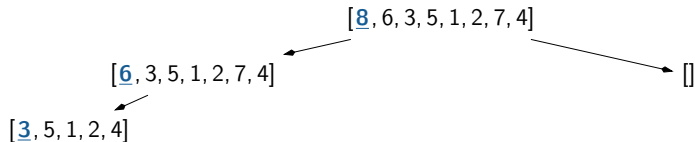
[8, 6, 3, 5, 1, 2, 7, 4]

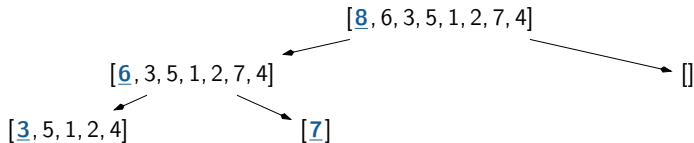
[8, 6, 3, 5, 1, 2, 7, 4]

↙

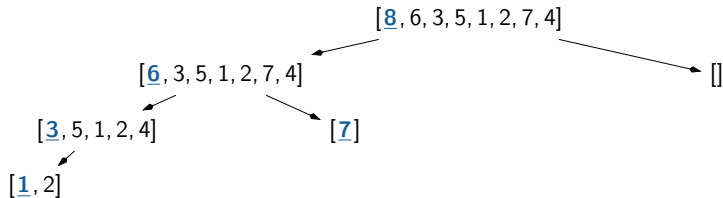
[6, 3, 5, 1, 2, 7, 4]

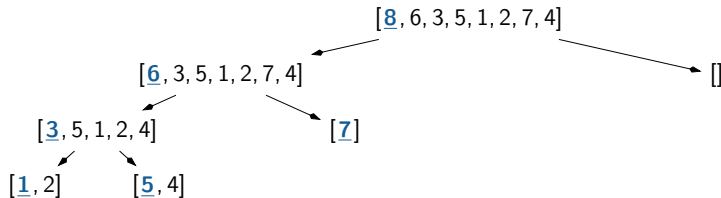


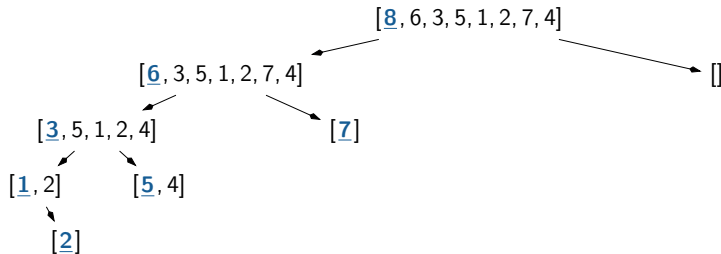


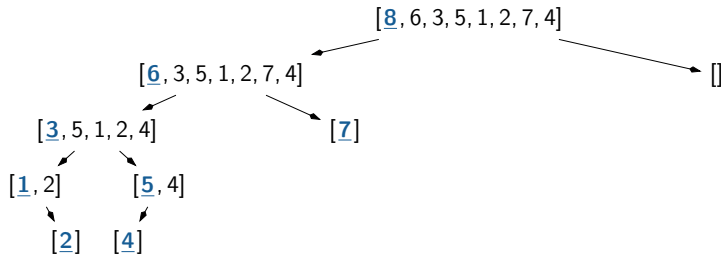


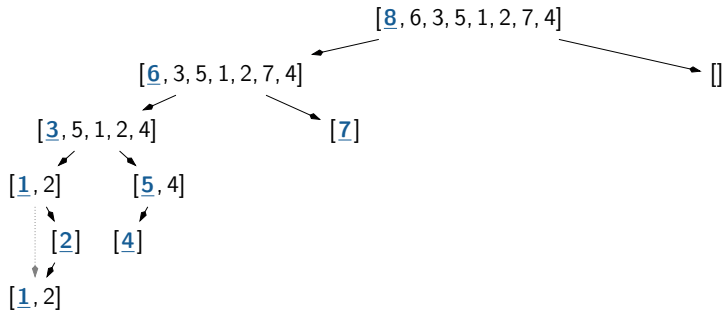


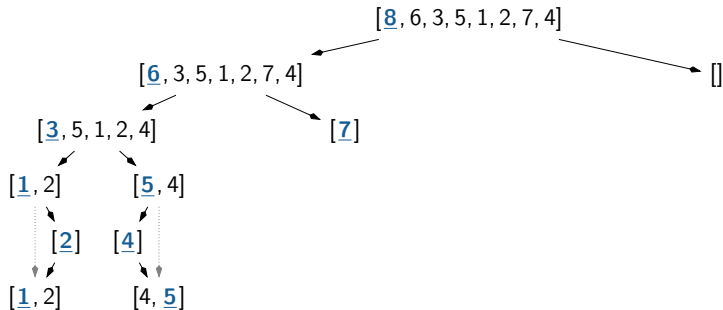


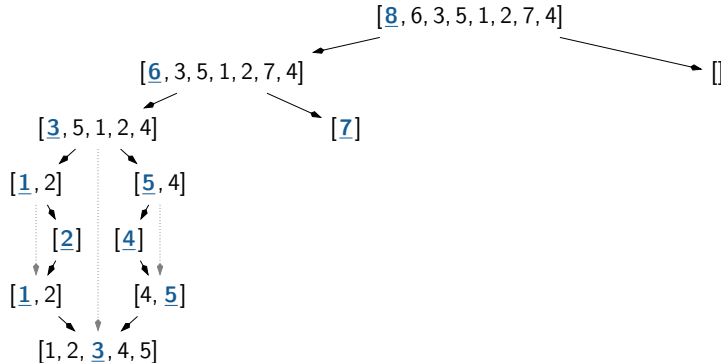


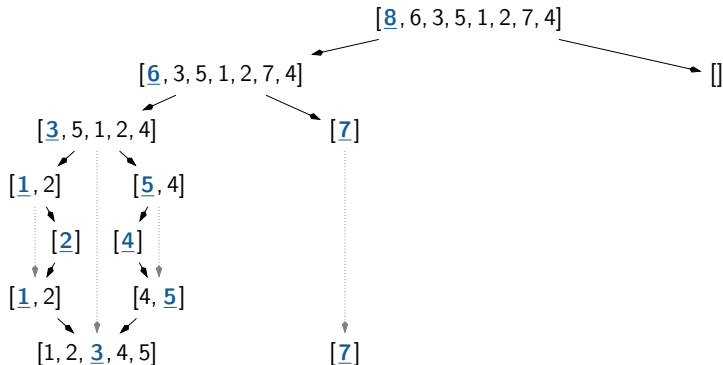




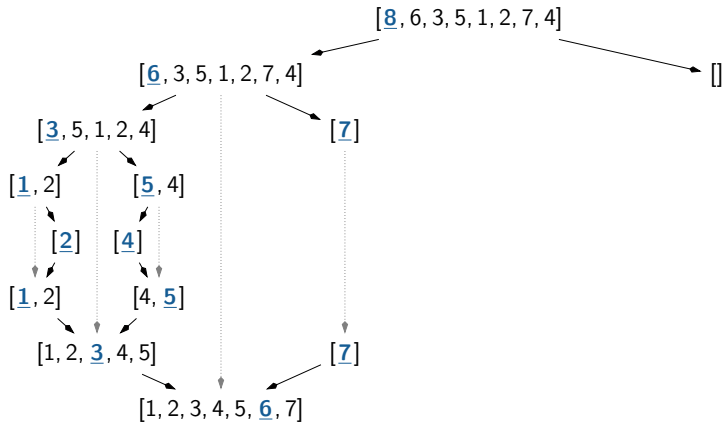


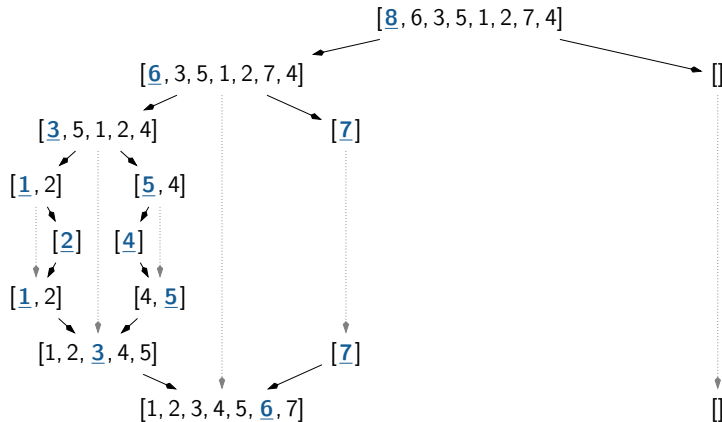


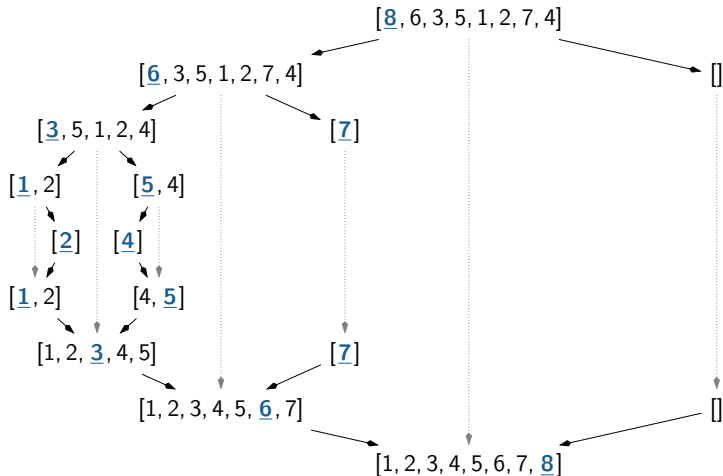














Ich köööönnte es wieder mit Klammern machen.



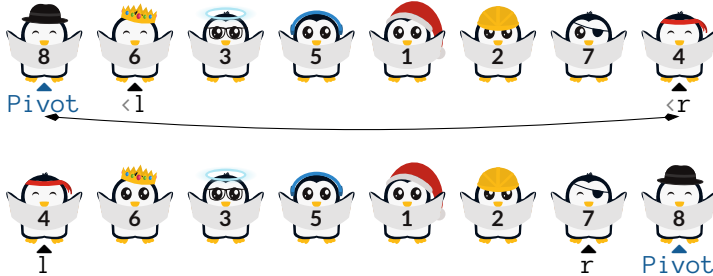


Nach der Vorlesung muss das Pivotelement ans Ende!

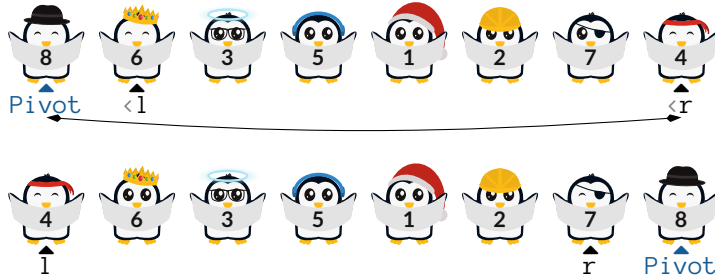


Nach der Vorlesung muss das Pivot-element ans Ende!



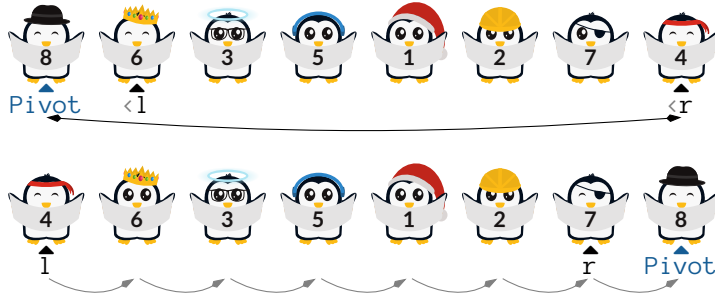


Nach der Vorlesung muss das Pivotelement ans Ende!



Nach der Vorlesung muss das Pivotelement ans Ende!

1 geht nach rechts, solange es auf ein Element zeigt, das kleiner als das Pivotelement ist.



Nach der Vorlesung muss das Pivotelement ans Ende!

l geht nach rechts, solange es auf ein Element zeigt, das kleiner als das Pivotelement ist.



Nach der Vorlesung muss das Pivotelement ans Ende!



1 geht nach rechts, solange es auf ein Element zeigt, das kleiner als das Pivotelement ist.





Nach der Vorlesung muss das Pivotelement ans Ende!



1 geht nach rechts, solange es auf ein Element zeigt, das kleiner als das Pivotelement ist.



Es ist  $l \geq r$ . Tausche 1 mit dem Pivotelement (hier die selben).



Nach der Vorlesung muss das Pivotelement ans Ende!



1 geht nach rechts, solange es auf ein Element zeigt, das kleiner als das Pivotelement ist.



Es ist  $l \geq r$ . Tausche 1 mit dem Pivotelement (hier die selben).



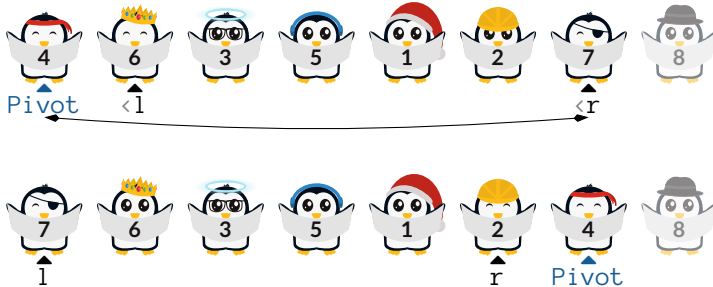


Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...

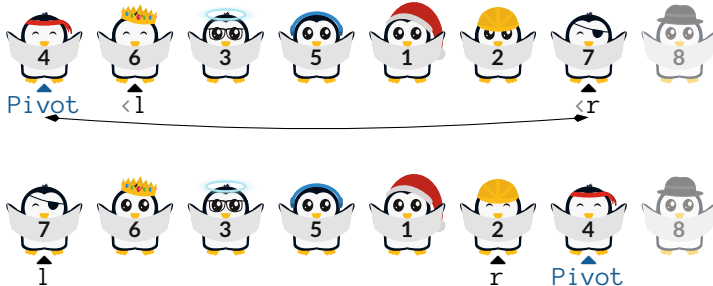




Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...

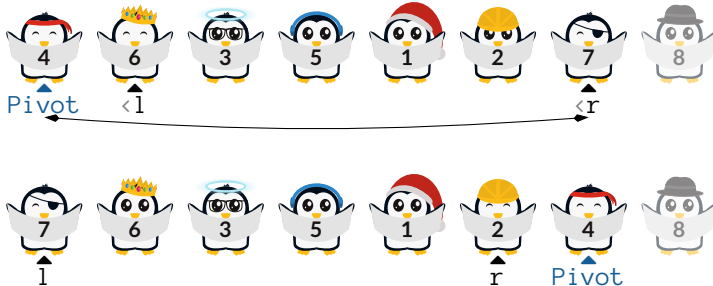


Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...



Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...

Wir tauschen sofort.  $r$  zeigt auf ein Element kleiner und  $l$  auf eines größer gleich dem Pivotelement.



Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...

Wir tauschen sofort.  $r$  zeigt auf ein Element kleiner und  $l$  auf eines größer gleich dem Pivotelement.



Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...



Wir tauschen sofort. r zeigt auf ein Element kleiner und l auf eines größer gleich dem Pivotelement.





Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...



Wir tauschen sofort. r zeigt auf ein Element kleiner und l auf eines größer gleich dem Pivotelement.



Aaaand we continue with the (little) steppies. And the swappies.



Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...



Wir tauschen sofort.  $r$  zeigt auf ein Element kleiner und  $l$  auf eines größer gleich dem Pivotelement.



Aaaand we continue with the (little) steppies. And the swappies.



Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...



Wir tauschen sofort.  $r$  zeigt auf ein Element kleiner und  $l$  auf eines größer gleich dem Pivotelement.



Aaaand we continue with the (little) steppies. And the swappies.





Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke...



Wir tauschen sofort.  $r$  zeigt auf ein Element kleiner und  $l$  auf eines größer gleich dem Pivotelement.



Aaaand we continue with the (little) steppies. And the swappies.





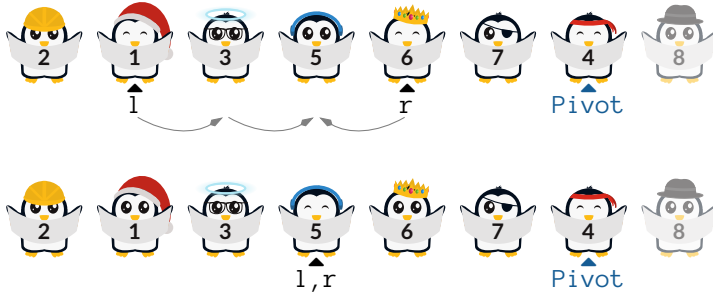
Und sie kommen ganz nah zusammen....



Und sie kommen ganz nah zusammen....



Und sie kommen ganz nah zusammen....



Und sie kommen ganz nah zusammen....



Und sie kommen ganz nah zusammen....



Es ist  $l = r$ , wir vertauschen also  $l$  und das Pivotelement.



Und sie kommen ganz nah zusammen....



Es ist  $l = r$ , wir vertauschen also  $l$  und das Pivotelement.





Und sie kommen ganz nah zusammen....



Es ist  $l = r$ , wir vertauschen also  $l$  und das Pivotelement.





Und sie kommen ganz nah zusammen....



Es ist  $l = r$ , wir vertauschen also  $l$  und das Pivotelement.



Meet me at my happy place. We go to the left.





Für die linke Hälfte machen wir ein swappsie.



Für die linke Hälfte machen wir ein swappsie.



Für die linke Hälfte machen wir ein swappsie.





Für die linke Hälfte machen wir ein swappsie.



Weiter geht l eins nach rechts, da es nun auf ein kleineres Element zeigt. Nun hat es r getroffen, wir tauschen also l und Pivot.



Für die linke Hälfte machen wir ein swappsie.



Weiter geht  $l$  eins nach rechts, da es nun auf ein kleineres Element zeigt. Nun hat es  $r$  getroffen, wir tauschen also  $l$  und Pivot.





Für die linke Hälfte machen wir ein swappsie.



Weiter geht  $l$  eins nach rechts, da es nun auf ein kleineres Element zeigt. Nun hat es  $r$  getroffen, wir tauschen also  $l$  und Pivot.



Für die linke Hälfte machen wir ein swappie.



Weiter geht  $l$  eins nach rechts, da es nun auf ein kleineres Element zeigt. Nun hat es  $r$  getroffen, wir tauschen also  $l$  und Pivot.





Für die linke Hälfte machen wir ein swappie.



Weiter geht l eins nach rechts, da es nun auf ein kleineres Element zeigt. Nun hat es r getroffen, wir tauschen also l und Pivot.



Eigentlich gings rekursiv weiter. Einelementige oder sortierte Listen lassen sich durch Kommentare überspringen („passiert ja nichts“).

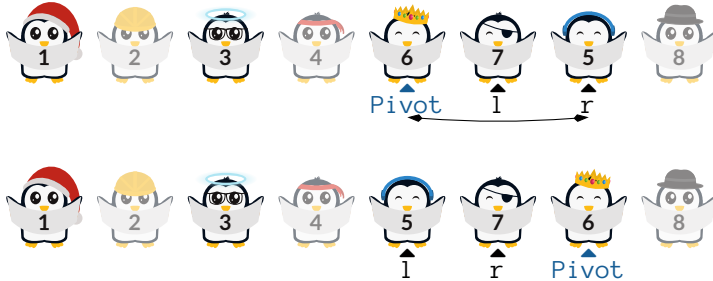




Es geht also wieder um ein Dreierpärchen



Es geht also wieder um ein Dreierpärchen



Es geht also wieder um ein Dreierpärchen



Es geht also wieder um ein Dreierpärchen



Es zeigt l auf ein kleineres Element, es geht also eins nach rechts, trifft r und tauscht mit Pivot.





Es geht also wieder um ein Dreierpärchen



Es zeigt  $l$  auf ein kleineres Element, es geht also eins nach rechts, trifft  $r$  und tauscht mit Pivot.



Es geht also wieder um ein Dreierpärchen



Es zeigt  $l$  auf ein kleineres Element, es geht also eins nach rechts, trifft  $r$  und tauscht mit Pivot.



Es geht also wieder um ein Dreierpärchen



Es zeigt  $l$  auf ein kleineres Element, es geht also eins nach rechts, trifft  $r$  und tauscht mit Pivot.





Es geht also wieder um ein Dreierpärchen



Es zeigt  $l$  auf ein kleineres Element, es geht also eins nach rechts, trifft  $r$  und tauscht mit Pivot.



Für alle weiteren Teillisten ist  $l = r = \text{Pivot}$ . Wir verkürzen das in dieser Variante



1. Betrachte die gegebene Teilliste (initial die Ganze).

1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.

1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze  $l$  ganz nach links und  $r$  ganz nach rechts.  
(Wahlweise auch  $r$  direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)



1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze  $l$  ganz nach links und  $r$  ganz nach rechts.  
(Wahlweise auch  $r$  direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)
4. Schiebe  $l$  nach rechts solange es auf Elemente kleiner als das Pivotelement zeigt (lessen).

1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze  $l$  ganz nach links und  $r$  ganz nach rechts.  
(Wahlweise auch  $r$  direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)
4. Schiebe  $l$  nach rechts solange es auf Elemente kleiner als das Pivotelement zeigt (**l**esser).
5. Nur wenn  $l$  noch weiter links als  $r$  ist, schiebe  $r$  nach links, solange es auf Elemente größer gleich dem Pivotelement zeigt (**g**reater).

1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze  $l$  ganz nach links und  $r$  ganz nach rechts.  
(Wahlweise auch  $r$  direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)
4. Schiebe  $l$  nach rechts solange es auf Elemente kleiner als das Pivotelement zeigt (lesser).
5. Nur wenn  $l$  noch weiter links als  $r$  ist, schiebe  $r$  nach links, solange es auf Elemente größer gleich dem Pivotelement zeigt (greater).
6. Ist  $l$  nun immer noch weiter links von  $r$ , dann vertausche die Elemente auf die sie zeigen und wiederhole bei 4. Andernfalls, vertausche  $l$  und Pivotelement.

1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze  $l$  ganz nach links und  $r$  ganz nach rechts.  
(Wahlweise auch  $r$  direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)
4. Schiebe  $l$  nach rechts solange es auf Elemente kleiner als das Pivotelement zeigt (lesser).
5. Nur wenn  $l$  noch weiter links als  $r$  ist, schiebe  $r$  nach links, solange es auf Elemente größer gleich dem Pivotelement zeigt (greater).
6. Ist  $l$  nun immer noch weiter links von  $r$ , dann vertausche die Elemente auf die sie zeigen und wiederhole bei 4. Andernfalls, vertausche  $l$  und Pivotelement.
7. Wiederhole ab 1 rekursiv für die Teilliste links und rechts vom Pivotelement.

1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze  $l$  ganz nach links und  $r$  ganz nach rechts.  
(Wahlweise auch  $r$  direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)
4. Schiebe  $l$  nach rechts solange es auf Elemente kleiner als das Pivotelement zeigt (**l**esser).
5. Nur wenn  $l$  noch weiter links als  $r$  ist, schiebe  $r$  nach links, solange es auf Elemente größer gleich dem Pivotelement zeigt (**g**reater).
6. Ist  $l$  nun immer noch weiter links von  $r$ , dann vertausche die Elemente auf die sie zeigen und wiederhole bei 4. Andernfalls, vertausche  $l$  und Pivotelement.
7. Wiederhole ab 1 rekursiv für die Teilliste links und rechts vom Pivotelement.

Lesser & Greater tauschen,  
bis  $l \geq r$ . Dann  $l$  & Pivot  
und rekursiv.



1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze  $l$  ganz nach links und  $r$  ganz nach rechts.  
(Wahlweise auch  $r$  direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)
4. Schiebe  $l$  nach rechts solange es auf Elemente kleiner als das Pivotelement zeigt (**l**esser).
5. Nur wenn  $l$  noch weiter links als  $r$  ist, schiebe  $r$  nach links, solange es auf Elemente größer gleich dem Pivotelement zeigt (**g**reater).
6. Ist  $l$  nun immer noch weiter links von  $r$ , dann vertausche die Elemente auf die sie zeigen und wiederhole bei 4. Andernfalls, vertausche  $l$  und Pivotelement.
7. Wiederhole ab 1 rekursiv für die Teilliste links und rechts vom Pivotelement.

(Der „gleich“-Fall ist an sich beliebig. Man kann also auch  $l$  für kleiner gleich und  $r$  für größer als das Pivotelement verwenden. Man muss nur eine der Varianten wählen.)

Lesser & Greater tauschen,  
bis  $l \geq r$ . Dann  $l$  & Pivot  
und rekursiv.





- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.



- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen **P**:

8 6 3 5 1 2 7 4

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$     8     $\downarrow$  6   3   5   1   2   7    $\downarrow$  4

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$     8    6   3   5   1   2   7   4

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

		↓	→	→	→	→	↓
		l					r
<u>8</u>	6	3	5	1	2	7	4

							↓
							l,r
<u>8</u>	6	3	5	1	2	7	4

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$  8 6 3 5 1 2 7 4

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$  8 6 3 5 1 2 7 4

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8 6 3 5 1 2 7 4

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$

8 6 3 5 1 2 7 4

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8 6 3 5 1 2 7 4

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$

8 6 3 5 1 2 7 4

4 6 3 5 1 2 7 8



- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8 6 3 5 1 2 7 4

Diagram showing the initial array [8, 6, 3, 5, 1, 2, 7, 4] with pivot P=8. The left pointer l points to index 1 (value 6) and the right pointer r points to index 7 (value 4). Arrows above the array indicate the movement of pointers: l moves right from index 1 to 2, 3, 4, 5, 6; r moves left from index 7 to 6.

2) Es ist:  $l \geq r$ . Hier tauschen wir r und p

8 6 3 5 1 2 7 4

Diagram showing the array after swapping the pivot 8 with the element at index 7 (value 4). The pivot 8 is now at index 7. A curved arrow indicates the swap between the element at index 1 (value 6) and the pivot at index 7 (value 8). The left pointer l and right pointer r both point to index 7.

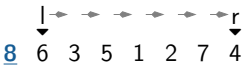
3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$

4 6 3 5 1 2 7 8

Diagram showing the array after the left partitioning step. The pivot 8 is at index 7. The left pointer l points to index 1 (value 6) and the right pointer r points to index 6 (value 7). The element at index 1 (value 6) has been swapped with the element at index 6 (value 4). The array is now [4, 6, 3, 5, 1, 2, 7, 8].

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$



2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$



3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$



- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8 6 3 5 1 2 7 4

Diagram showing the initial state of the array [8, 6, 3, 5, 1, 2, 7, 4]. The pivot P is 8. The left pointer l points to the first element (8), and the right pointer r points to the last element (4). Arrows above the array indicate the movement of pointers: l moves right, and r moves left.

2) Es ist:  $l \geq r$ . Hier tauschen wir r und p

8 6 3 5 1 2 7 4

Diagram showing the state after the first partitioning step. The pivot P (8) is now at the position of the right pointer r (index 7). The left pointer l is at index 0. A curved arrow indicates the swap between the element at index 0 (8) and the element at index 7 (4).

3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$

4 6 3 5 1 2 7 8

Diagram showing the state after the second partitioning step. The pivot P is 4. The left pointer l points to the first element (4), and the right pointer r points to the last element (8). Arrows above the array indicate the movement of pointers: l moves right, and r moves left.

4 6 3 5 1 2 7 8

Diagram showing the state after the second partitioning step. The pivot P (4) is now at the position of the right pointer r (index 5). The left pointer l is at index 0. Arrows above the array indicate the movement of pointers: l moves right, and r moves left.

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8 6 3 5 1 2 7 4

$l \rightarrow \rightarrow \rightarrow \rightarrow r$

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$

8 6 3 5 1 2 7 4

$l, r$

3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$

4 6 3 5 1 2 7 8

$l \rightarrow \rightarrow \rightarrow r$

4)  $a[l] \geq P$  und  $a[r] < P$

4 6 3 5 1 2 7 8

$l \rightarrow \rightarrow \rightarrow r$

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8 6 3 5 1 2 7 4

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$

8 6 3 5 1 2 7 4

3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$

4 6 3 5 1 2 7 8

4)  $a[l] \geq P$  und  $a[r] < P$

4 6 3 5 1 2 7 8

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8 6 3 5 1 2 7 4      4 2 3 5 1 6 7 8

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$

8 6 3 5 1 2 7 4

3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$

4 6 3 5 1 2 7 8

4)  $a[l] \geq P$  und  $a[r] < P$

4 6 3 5 1 2 7 8

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$  8 6 3 5 1 2 7 4 4 2 3 5 1 6 7 8 5) Solange  $a[r] \geq P$  und  $a[l] < P$

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$  8 6 3 5 1 2 7 4

3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$  4 6 3 5 1 2 7 8

4)  $a[l] \geq P$  und  $a[r] < P$  4 6 3 5 1 2 7 8

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

1) Solange  $a[r] \geq P$  und  $a[l] < P$  8 6 3 5 1 2 7 4 4 2 3 5 1 6 7 8 5) Solange  $a[r] \geq P$  und  $a[l] < P$

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$  8 6 3 5 1 2 7 4

3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$  4 6 3 5 1 2 7 8

4)  $a[l] \geq P$  und  $a[r] < P$  4 6 3 5 1 2 7 8



- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:

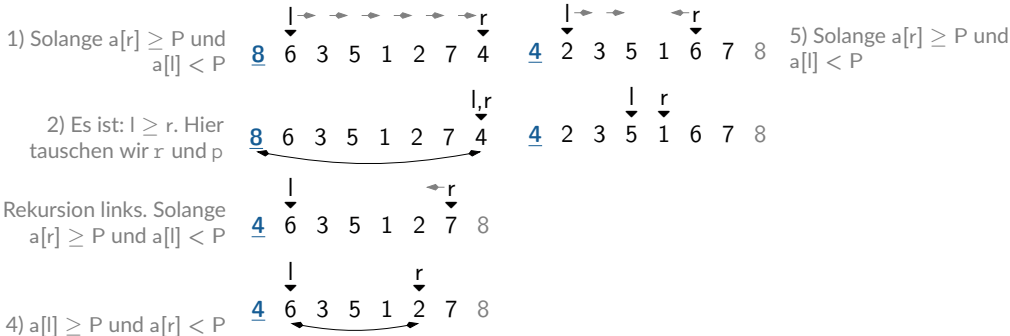
1) Solange  $a[r] \geq P$  und  $a[l] < P$  8 6 3 5 1 2 7 4 4 2 3 5 1 6 7 8 5) Solange  $a[r] \geq P$  und  $a[l] < P$

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$  8 6 3 5 1 2 7 4

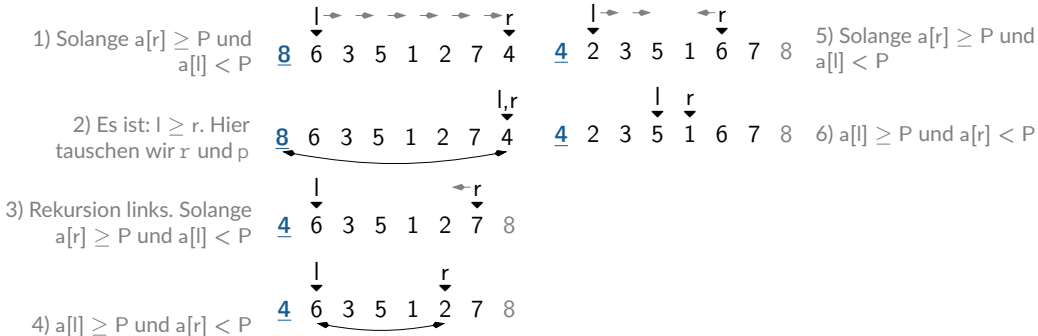
3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$  4 6 3 5 1 2 7 8

4)  $a[l] \geq P$  und  $a[r] < P$  4 6 3 5 1 2 7 8

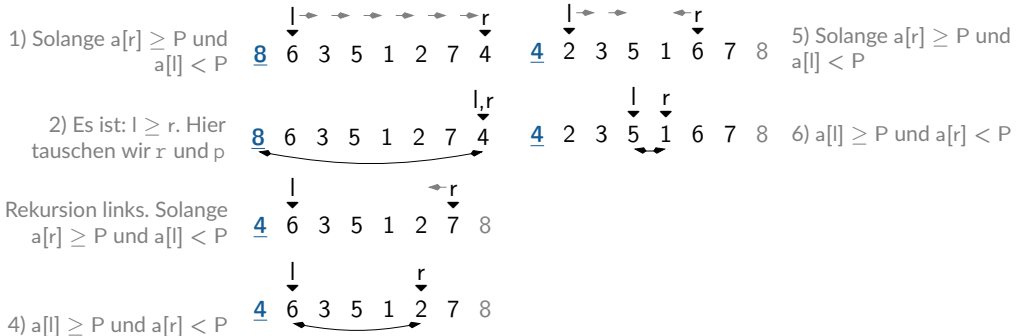
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:



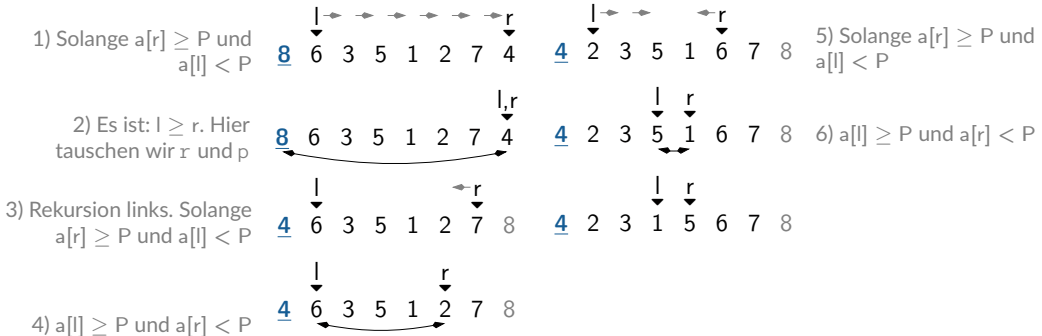
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:



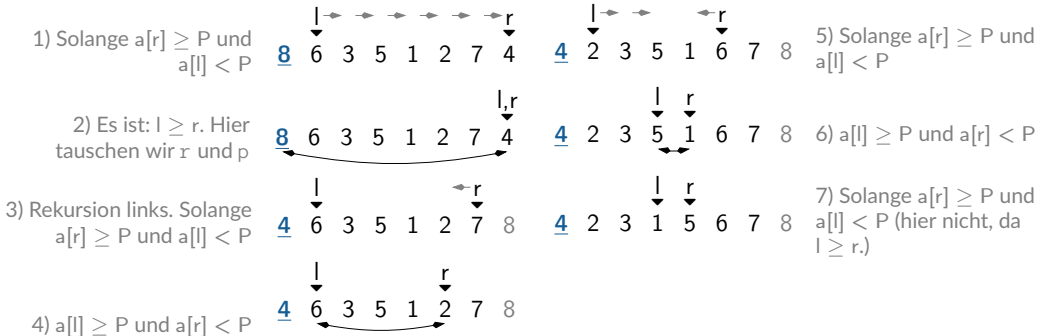
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:



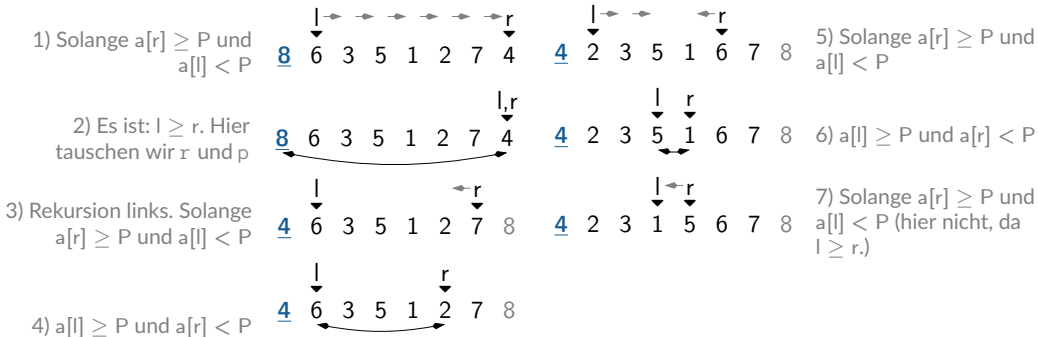
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:



- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:



- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen P:



- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen **P**:

1) Solange  $a[r] \geq P$  und  $a[l] < P$

8       $\downarrow$     $\rightarrow$     $\rightarrow$     $\rightarrow$     $\rightarrow$     $\rightarrow$     $\rightarrow$     $\downarrow$   
 6   3   5   1   2   7   4

2) Es ist:  $l \geq r$ . Hier tauschen wir  $r$  und  $p$

8 6 3 5 1 2 7 4

$l, r$

3) Rekursion links. Solange  $a[r] \geq P$  und  $a[l] < P$

↓                  ← r  
4    6    3    5    1    2    7    8

4)  $a[l] \geq P$  und  $a[r] < P$

4       $\downarrow$                        $\downarrow$   
 6   3   5   1   2   7   8  
 ←—————→

4 2 3 5 1 6 7 8

4 2 3 5 1 6 7 8

l r

↔

4 2 3 1 5 6 7 8

4 2 3 1 5 6 7 8

l,r

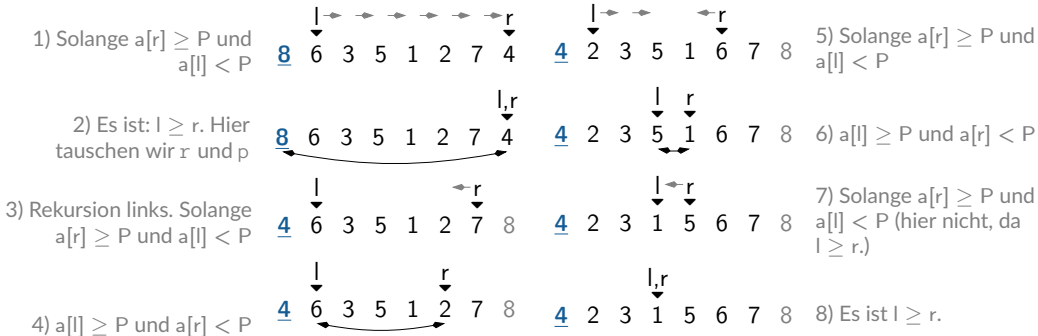
5) Solange  $a[r] \geq P$  und  $a[l] < P$

6)  $a[l] \geq P$  und  $a[r] < P$

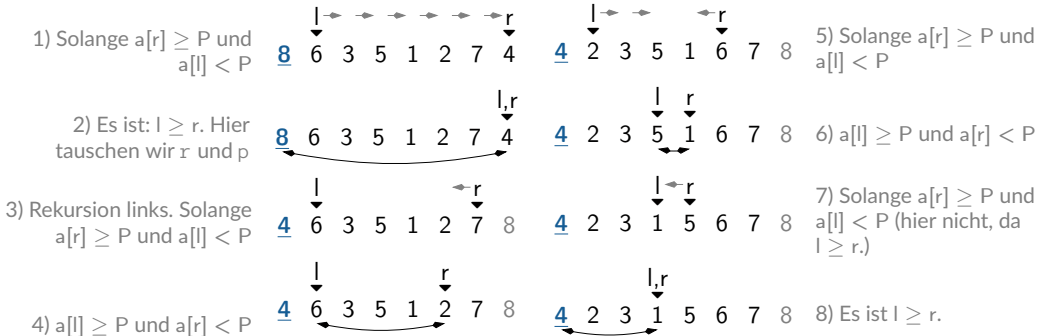
7) Solange  $a[r] \geq P$  und  $a[l] < P$  (hier nicht, da  $l > r$ .)



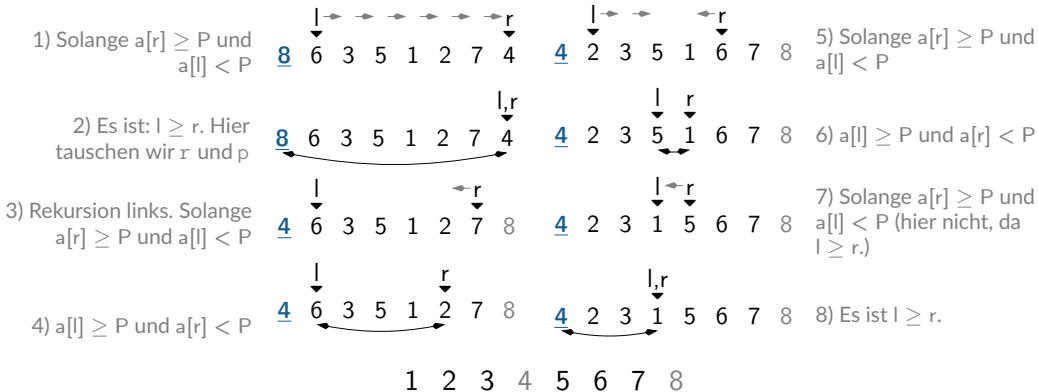
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen **P**:



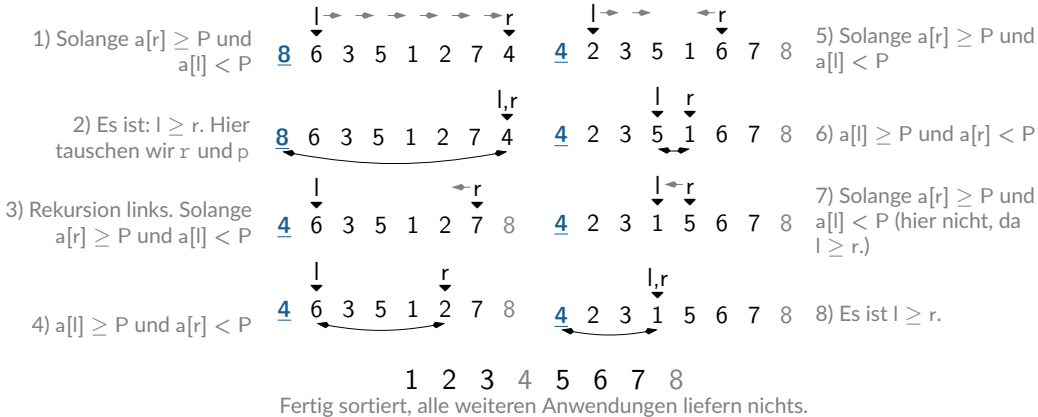
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen **P**:



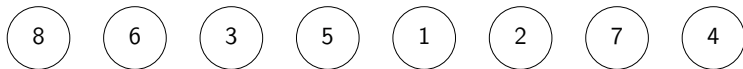
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen **P**:

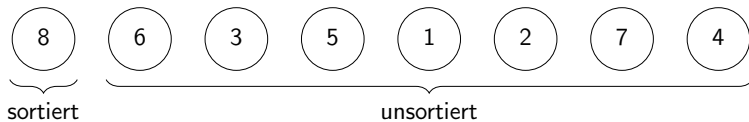


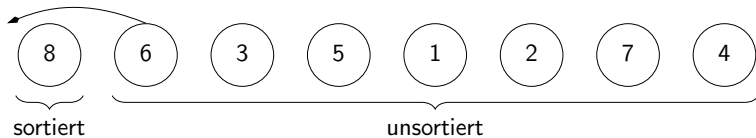
- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen **P**:



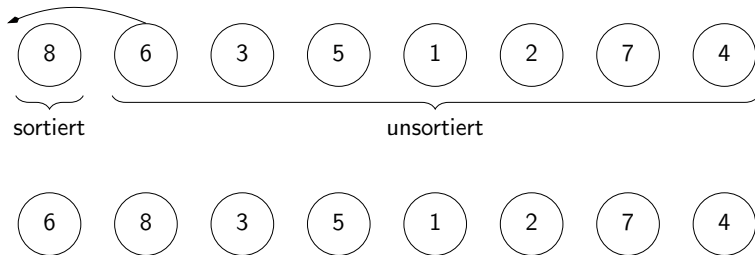


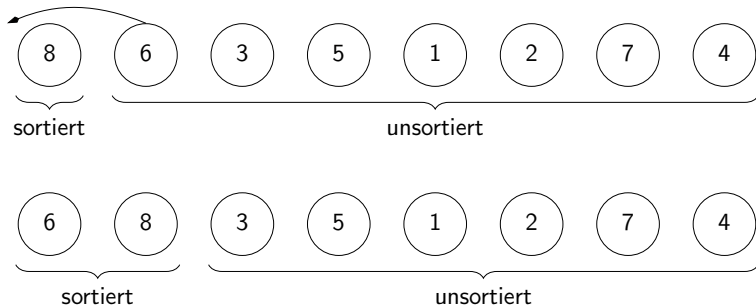


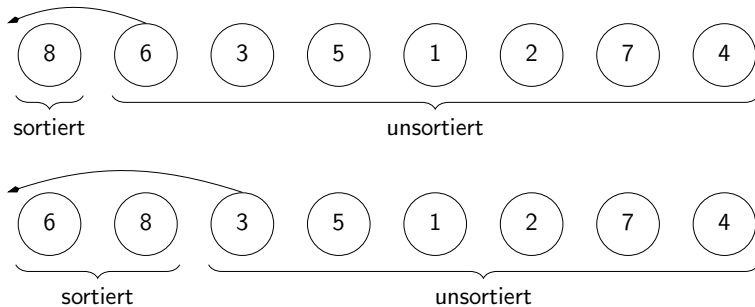


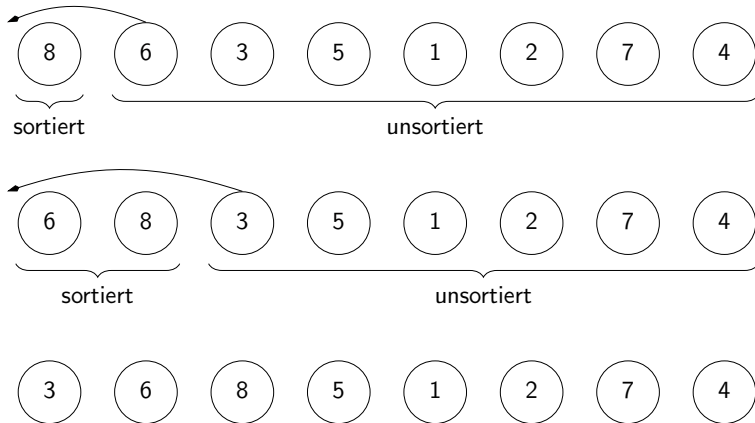


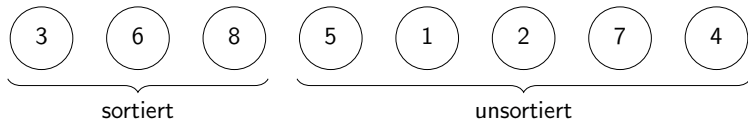
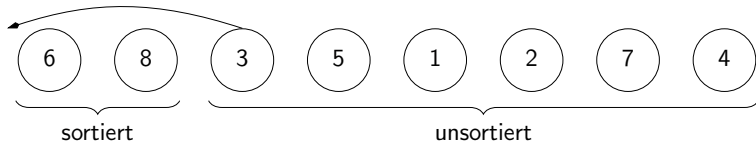
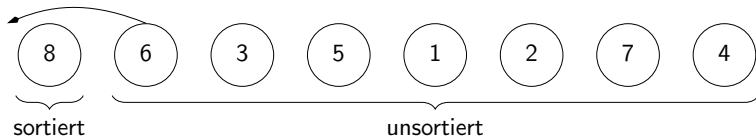


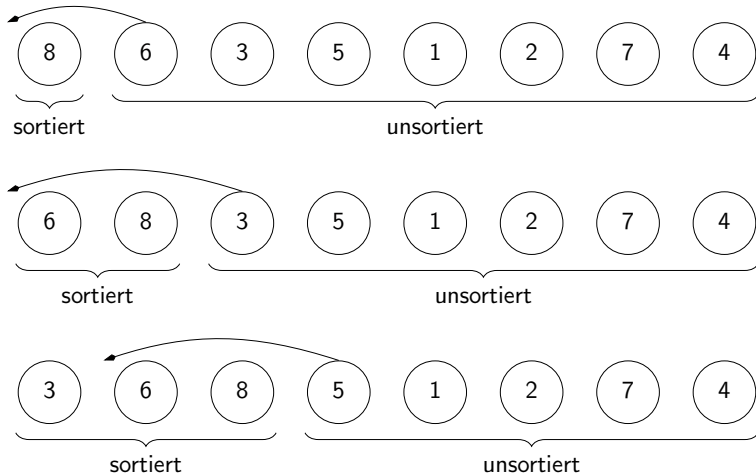




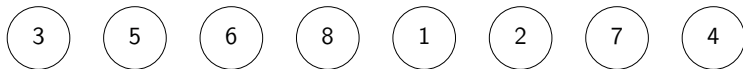




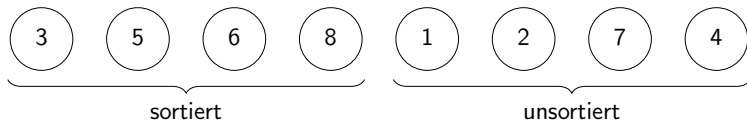


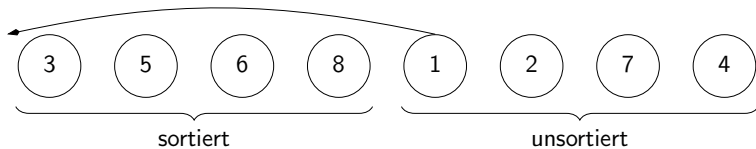


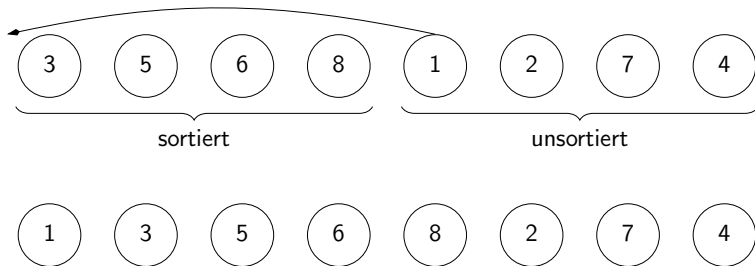


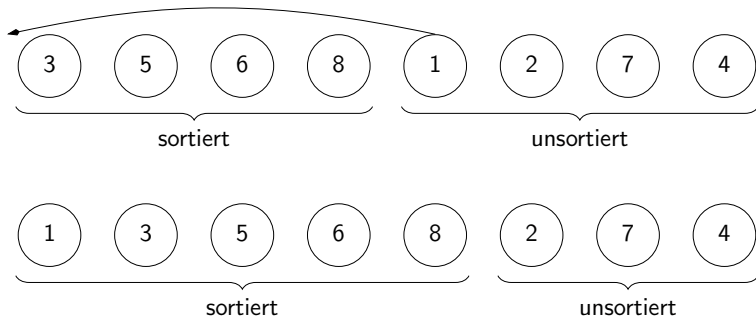


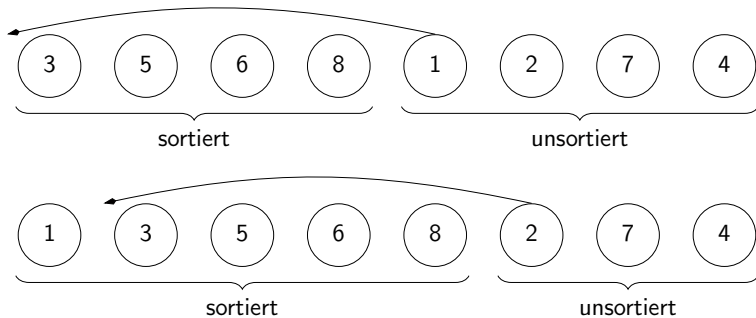


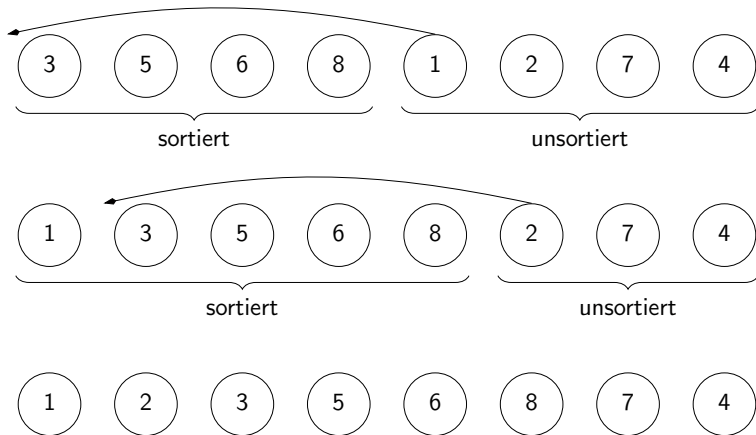


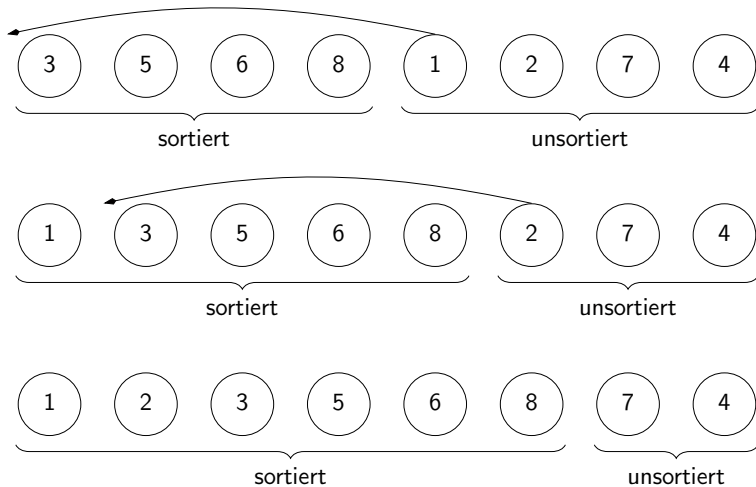


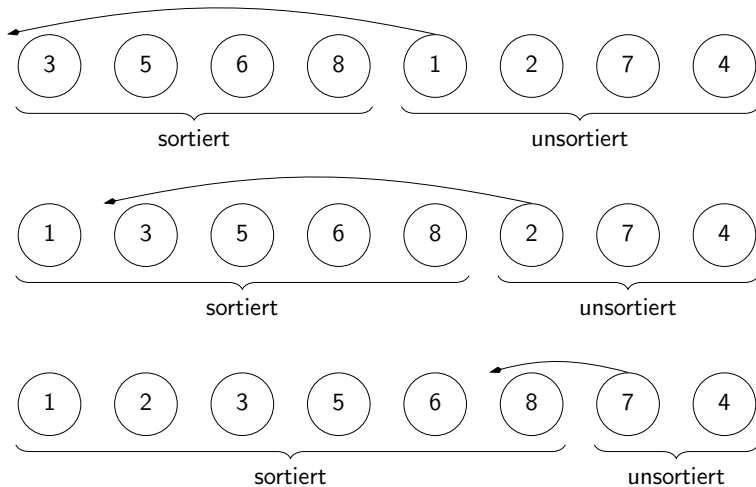








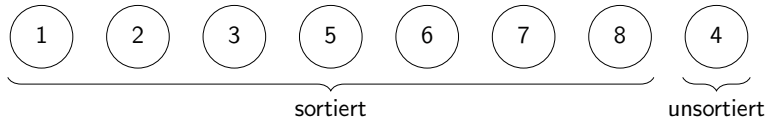


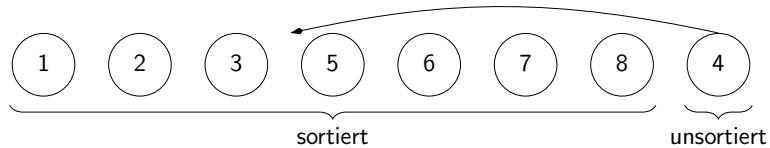


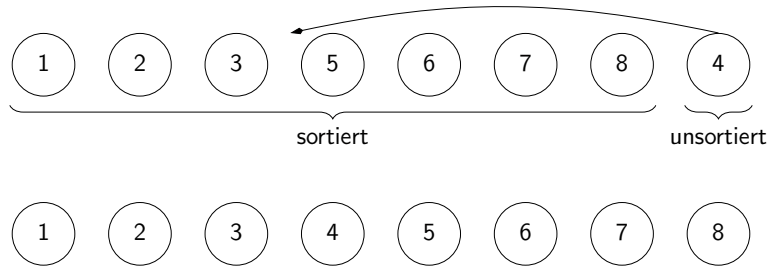


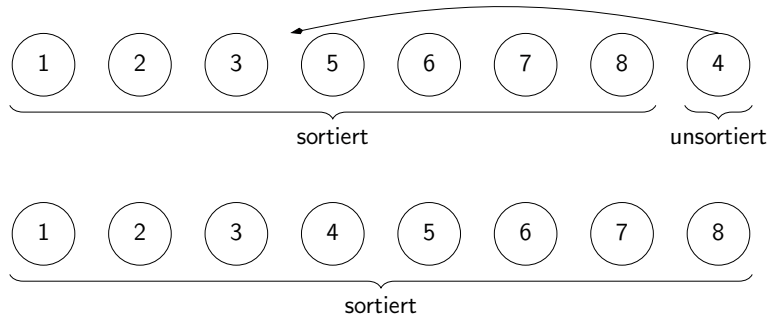


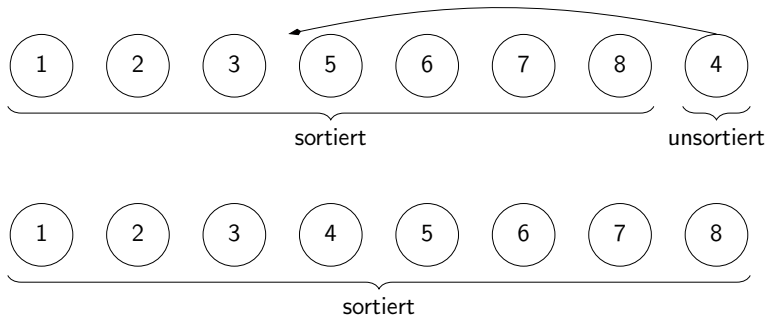












- Theoretisch kann man auch alle „Zwischentauschschritte“ einzeichnen.

# Übungsblatt 10 - Aufgabe 2



## Übungsblatt 10 - Aufgabe 2

In dieser Aufgabe sollen Sie den *Bubblesort* Algorithmus implementieren. Ihre Implementierung soll die Wrapper-Methode `public static void bubbleSort(int[] array, int n)` anbieten, welche das zu sortierende Array und dessen Länge als Parameter übernimmt. Implementieren Sie nun zusätzlich eine rekursive Hilfsmethode mit geeigneten Parametern, die den *Bubblesort* Algorithmus ausführt.

Verwenden Sie in ihrer Lösung keine Schleifen und auch keine vorgefertigten Sortierfunktionen wie z.B. `Arrays.sort()`!

## Übungsblatt 10 - Aufgabe 2

In dieser Aufgabe sollen Sie den *Bubblesort* Algorithmus implementieren. Ihre Implementierung soll die Wrapper-Methode `public static void bubbleSort(int[] array, int n)` anbieten, welche das zu sortierende Array und dessen Länge als Parameter übernimmt. Implementieren Sie nun zusätzlich eine rekursive Hilfsmethode mit geeigneten Parametern, die den *Bubblesort* Algorithmus ausführt.

Verwenden Sie in ihrer Lösung keine Schleifen und auch keine vorgefertigten Sortierfunktionen wie z.B. `Arrays.sort()`!



Wir betrachten einmal eine naive Implementation von Bubblesort und eine optimierte Version, welche eine Abbruchbedingung benutzt.

# Übungsblatt 10 - Aufgabe 2

# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {
```

}

# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {  
    for (int i = arr.length; i > 1; i--) {  
  
  
  
  
  
  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {  
    for (int i = arr.length; i > 1; i--) {  
        for (int j = 0; j < i - 1; j++) {  
  
  
        }  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {  
    for (int i = arr.length; i > 1; i--) {  
        for (int j = 0; j < i - 1; j++) {  
            if (arr[j] > arr[j + 1]) { // swap(arr, j, j + 1)  
                // swap logic  
            }  
        }  
    }  
}
```



# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {  
    for (int i = arr.length; i > 1; i--) {  
        for (int j = 0; j < i - 1; j++) {  
            if (arr[j] > arr[j + 1]) { // swap(arr, j, j + 1)  
                int tmp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = tmp;  
            }  
        }  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {  
    for (int i = arr.length; i > 1; i--) {  
        for (int j = 0; j < i - 1; j++) {  
            if (arr[j] > arr[j + 1]) { // swap(arr, j, j + 1)  
                int tmp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = tmp;  
            }  
        }  
    }  
}
```

- Wir werden zweimal Rekursion benutzen um die For-Schleifen zu re-präsentieren: [BubbleSort.java](#).

# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {  
    for (int i = arr.length; i > 1; i--) {  
        for (int j = 0; j < i - 1; j++) {  
            if (arr[j] > arr[j + 1]) { // swap(arr, j, j + 1)  
                int tmp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = tmp;  
            }  
        }  
    }  
}
```

Florians Didaktik Hoffnungsversuch: Eigentlich machen wir Rekursion nicht *nur um irgendwas iteratives hinzubekommen*. Es gibt schöne, wundervolle Probleme die sich mit Rekursion wunderschön und elegant lösen lassen. Tolle... wirklich tolle... schnief.

- Wir werden zweimal Rekursion benutzen um die For-Schleifen zu repräsentieren: [BubbleSort.java](#).



# Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```
public static void bubbleSort(int[] arr) {  
    for (int i = arr.length; i > 1; i--) {  
        for (int j = 0; j < i - 1; j++) {  
            if (arr[j] > arr[j + 1]) { // swap(arr, j, j + 1)  
                int tmp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = tmp;  
            }  
        }  
    }  
}
```

```
static void swap(int[] array, int i, int j) {  
    int tmp = array[i];  
    array[i] = array[j];  
    array[j] = tmp;  
}
```

Florians Didaktik Hoffnungsversuch: Eigentlich machen wir Rekursion nicht *nur um irgendwas iteratives hinzubekommen*. Es gibt schöne, wundervolle Probleme die sich mit Rekursion wunderschön und elegant lösen lassen. Tolle... wirklich tolle... schnief.

- Wir werden zweimal Rekursion benutzen um die For-Schleifen zu repräsentieren: [BubbleSort.java](#).



# Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
  
  
  
  
  
  
  
  
  
}
```

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
  
  
    }  
}
```

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```

## Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
  
        bubbleSort(array, n - 1);  
    }  
}
```

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```



## Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
        move(array, 0, n);  
        bubbleSort(array, n - 1);  
    }  
}
```

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```

## Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
        move(array, 0, n);  
        bubbleSort(array, n - 1);  
    }  
}  
  
private static void move(int[] array, int i, int n) {
```

```
    public static void bubbleSort(int[] arr) {  
        for (int n = arr.length; n > 1; n--) {  
            for (int i = 0; i < n - 1; i++) {  
                if (arr[i] > arr[i + 1])  
                    swap(arr, i, i + 1);  
            }  
        }  
    }
```

## Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
        move(array, 0, n);  
        bubbleSort(array, n - 1);  
    }  
}  
  
private static void move(int[] array, int i, int n) {  
    if (i < n - 1) {  
  
    }  
}
```

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```

## Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
        move(array, 0, n);  
        bubbleSort(array, n - 1);  
    }  
}  
  
private static void move(int[] array, int i, int n) {  
    if (i < n - 1) {  
  
        move(array, i + 1, n);  
    }  
}
```

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```

## Übungsblatt 10 - Aufgabe 2

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
        move(array, 0, n);  
        bubbleSort(array, n - 1);  
    }  
}  
  
private static void move(int[] array, int i, int n) {  
    if (i < n - 1) {  
        if (array[i] > array[i + 1])  
            swap(array, i, i + 1);  
        move(array, i + 1, n);  
    }  
}
```

```
public static void bubbleSort(int[] arr) {  
    for (int n = arr.length; n > 1; n--) {  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr, i, i + 1);  
        }  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde ([OptimizedBubbleSort.java](#)):

## Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde (`OptimizedBubbleSort.java`):

```
public static void bubbleSort(int[] array, int n) {  
  
  
  
  
  
  
}
```



# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde ([OptimizedBubbleSort.java](#)):

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde ([OptimizedBubbleSort.java](#)):

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
  
        bubbleSort(array, n - 1);  
    }  
}
```

# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde ([OptimizedBubbleSort.java](#)):

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
        if(!move(array, 0, n, false)) return; // kein Tausch mehr?  
        bubbleSort(array, n - 1);  
    }  
}
```



# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde ([OptimizedBubbleSort.java](#)):

```
public static void bubbleSort(int[] array, int n) {
    if (n > 1) {
        if(!move(array, 0, n, false)) return; // kein Tausch mehr?
        bubbleSort(array, n - 1);
    }
}

private static boolean move(int[] array, int i, int n, boolean swapped) {
    if(i < n - 1) {

    }

}
```

# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde (**OptimizedBubbleSort.java**):

```
public static void bubbleSort(int[] array, int n) {
    if (n > 1) {
        if(!move(array, 0, n, false)) return; // kein Tausch mehr?
        bubbleSort(array, n - 1);
    }
}

private static boolean move(int[] array, int i, int n, boolean swapped) {
    if(i < n - 1) {

        return move(array, i + 1, n, swapped);
    }
}
```

# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde (**OptimizedBubbleSort.java**):

```
public static void bubbleSort(int[] array, int n) {
    if (n > 1) {
        if(!move(array, 0, n, false)) return; // kein Tausch mehr?
        bubbleSort(array, n - 1);
    }
}

private static boolean move(int[] array, int i, int n, boolean swapped) {
    if(i < n - 1) {

        return move(array, i + 1, n, swapped);
    }
    return swapped;
}
```

# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde (**OptimizedBubbleSort.java**):

```
public static void bubbleSort(int[] array, int n) {
    if (n > 1) {
        if(!move(array, 0, n, false)) return; // kein Tausch mehr?
        bubbleSort(array, n - 1);
    }
}

private static boolean move(int[] array, int i, int n, boolean swapped) {
    if(i < n - 1) {
        if (array[i] > array[i+1]) {

            }
        return move(array, i + 1, n, swapped);
    }
    return swapped;
}
```



# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde (**OptimizedBubbleSort.java**):

```
public static void bubbleSort(int[] array, int n) {
    if (n > 1) {
        if(!move(array, 0, n, false)) return; // kein Tausch mehr?
        bubbleSort(array, n - 1);
    }
}

private static boolean move(int[] array, int i, int n, boolean swapped) {
    if(i < n - 1) {
        if (array[i] > array[i+1]) {
            swap(array, i, i + 1);
        }
        return move(array, i + 1, n, swapped);
    }
    return swapped;
}
```

# Übungsblatt 10 - Aufgabe 2

- Wir fügen eine Variable hinzu, ob getauscht wurde (**OptimizedBubbleSort.java**):

```
public static void bubbleSort(int[] array, int n) {  
    if (n > 1) {  
        if(!move(array, 0, n, false)) return; // kein Tausch mehr?  
        bubbleSort(array, n - 1);  
    }  
}  
  
private static boolean move(int[] array, int i, int n, boolean swapped) {  
    if(i < n - 1) {  
        if (array[i] > array[i+1]) {  
            swap(array, i, i + 1);  
            swapped = true;  
        }  
        return move(array, i + 1, n, swapped);  
    }  
    return swapped;  
}
```

# Übungsblatt 10 - Aufgabe 3

# Übungsblatt 10 - Aufgabe 3

1. Was versteht man unter einem stabilen Sortieralgorithmus?

# Übungsblatt 10 - Aufgabe 3

## 1. Was versteht man unter einem stabilen Sortieralgorithmus?

Ein *stabiler* Sortieralgorithmus verändert nicht die Reihenfolge von Elementen mit dem gleichen *Sortierschlüssel*.

(Beispiel: Wir sortieren zuerst nach Alter, danach nach Alphabet. Ein stabiler Sortieralgorithmus garantiert, dass Personen mit gleichem Alter immernoch alphabetisch sortiert sind.)

# Übungsblatt 10 - Aufgabe 3

## 1. Was versteht man unter einem stabilen Sortieralgorithmus?

Ein *stabiler* Sortieralgorithmus verändert nicht die Reihenfolge von Elementen mit dem gleichen *Sortierschlüssel*.

(Beispiel: Wir sortieren zuerst nach Alter, danach nach Alphabet. Ein stabiler Sortieralgorithmus garantiert, dass Personen mit gleichem Alter immernoch alphabetisch sortiert sind.)

## 2. Welche der vier auf diesem Übungsblatt vorgekommenen Sortieralgorithmen sind stabil und welche sind instabil?

# Übungsblatt 10 - Aufgabe 3

## 1. Was versteht man unter einem stabilen Sortialgorithmus?

Ein *stabiler* Sortialgorithmus verändert nicht die Reihenfolge von Elementen mit dem gleichen *Sortierschlüssel*.

(Beispiel: Wir sortieren zuerst nach Alter, danach nach Alphabet. Ein stabiler Sortialgorithmus garantiert, dass Personen mit gleichem Alter immernoch alphabetisch sortiert sind.)

## 2. Welche der vier auf diesem Übungsblatt vorgekommenen Sortialgorithmen sind stabil und welche sind instabil?

Insertionsort, Mergesort und Bubblesort sind stabil, Quicksort ist instabil.

# Übungsblatt 10 - Aufgabe 3

## 1. Was versteht man unter einem stabilen Sortieralgorithmus?

Ein *stabiler* Sortieralgorithmus verändert nicht die Reihenfolge von Elementen mit dem gleichen *Sortierschlüssel*.

(Beispiel: Wir sortieren zuerst nach Alter, danach nach Alphabet. Ein stabiler Sortieralgorithmus garantiert, dass Personen mit gleichem Alter immernoch alphabetisch sortiert sind.)

## 2. Welche der vier auf diesem Übungsblatt vorgekommenen Sortieralgorithmen sind stabil und welche sind instabil?

Insertionsort, Mergesort und Bubblesort sind stabil, Quicksort ist instabil.

(Genau genommen kann man aber jeden Algorithmus mit entsprechendem Zusatzaufwand stabil und nicht stabil implementieren. Wir halten uns hier an die Vorlesungsimplementation.)



# Übungsblatt 10 - Zusatzaufgabe 4/6

# Übungsblatt 10 - Zusatzaufgabe 4/6

In dieser Aufgabe sollen Sie sich mit dem 0-1-Rucksackproblem beschäftigen. Hier gibt es einen Rucksack mit einer maximalen Tragekapazität  $W_{\max}$ , sowie  $n$  Gegenstände welche jeweils ein Gewicht  $w_i$  und einen Wert  $v_i$  haben. Die gesuchte Lösung ist der maximale Wert  $V$  einer Teilmenge von Gegenständen, deren Gesamtgewicht nicht größer als  $W_{\max}$  ist.

Das Problem lässt sich durch folgende rekursive Formel definieren:

$$k(W_{\max}, i) = \begin{cases} 0 & \text{für } i > n \\ k(W_{\max}, i + 1) & \text{für } w_i > W_{\max} \\ \max(k(W_{\max}, i + 1), k(W_{\max} - w_i, i + 1) + v_i) & \text{sonst} \end{cases}$$

Implementieren Sie die Methode `public static int knapsack(int W, int[] weights, int[] values)` welche die maximale Tragekapazität des Rucksacks, sowie zwei Integer Arrays für die Gewichte und Werte der Gegenstände als Parameter übernimmt, und den maximalen Wert der Gegenstände zurückgibt die im Rucksack transportiert werden können. Implementieren Sie dazu eine rekursive Hilfsmethode mit geeigneten Parametern, welche intern von der Wrapper-Methode aufgerufen wird.

# Zusatzaufgabe 4/6

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
  
  
  
  
  
  
  
  
  
}
```

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
  
}
```

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
    return k(W, 0, weights, values);  
}
```

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
    return k(W, 0, weights, values);  
}
```

```
private static int k(int W, int i, int[] w, int[] v) {  
  
  
  
  
  
  
  
  
  
}
```



## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
    return k(W, 0, weights, values);  
}
```

```
private static int k(int W, int i, int[] w, int[] v) {  
    if (i >= w.length) return 0; // beginnt bei 0:  $i > n \rightarrow 0$   
  
}
```

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
    return k(W, 0, weights, values);  
}
```

```
private static int k(int W, int i, int[] w, int[] v) {  
    if (i >= w.length) return 0; // beginnt bei 0:  $i > n \rightarrow 0$   
    if (w[i] > W) return k(W, i + 1, w, v); //  $w_i > W_{\max} \rightarrow k(W_{\max}, i + 1)$   
  
}
```

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
    return k(W, 0, weights, values);  
}
```

```
private static int k(int W, int i, int[] w, int[] v) {  
    if (i >= w.length) return 0; // beginnt bei 0:  $i > n \rightarrow 0$   
    if (w[i] > W) return k(W, i + 1, w, v); //  $w_i > W_{\max} \rightarrow k(W_{\max}, i + 1)$   
    else return Math.max(k(W, i + 1, w, v), // sonst: max(„ohne“,  
        k(W - w[i], i + 1, w, v) + v[i]); // „mit“)  
}
```

## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
    return k(W, 0, weights, values);  
}
```

Wir nutzen `w.length` für `n` und müssen beachten, dass es bei 0 losgeht.

```
private static int k(int W, int i, int[] w, int[] v) {  
    if (i >= w.length) return 0; // beginnt bei 0:  $i > n \rightarrow 0$   
    if (w[i] > W) return k(W, i + 1, w, v); //  $w_i > W_{\max} \rightarrow k(W_{\max}, i + 1)$   
    else return Math.max(k(W, i + 1, w, v), // sonst: max(„ohne“,  
        k(W - w[i], i + 1, w, v) + v[i]); // „mit“)  
}
```



## Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {  
    if(weights.length != values.length) return -1;  
    return k(W, 0, weights, values);  
}
```

Wir nutzen `w.length` für `n` und müssen beachten, dass es bei 0 losgeht.

```
private static int k(int W, int i, int[] w, int[] v) {  
    if (i >= w.length) return 0; // beginnt bei 0:  $i > n \rightarrow 0$   
    if (w[i] > W) return k(W, i + 1, w, v); //  $w_i > W_{\max} \rightarrow k(W_{\max}, i + 1)$   
    else return Math.max(k(W, i + 1, w, v), // sonst: max(„ohne“,  
        k(W - w[i], i + 1, w, v) + v[i]); // „mit“)  
}
```



Knapsack.java





Wieder kein Pingu,



Wieder kein Pingu, weil süß!

