10

Ich sortiere das. Ich, als Sortierer.

Ordnung 10

Florian Sihler ○ KW 4

Präsenzaufgabe

Bitte anstellen… als …

2

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 Rückgabetyp 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 ~100 Lines Musterlösung verkraftbar

- 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;
    // ...
}
```

Präsenzaufgabe - Lösung

Element - Alternative

```
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;
  }
}
```

Präsenzaufgabe - Lösung

Queue - Konstruktor

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

    public Queue() {
        this.first = null;
        this.last = null;
        this.length = 0;
    }
}
```

Präsenzaufgabe - Lösung

Queue - Enqueue

```
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++;
    }
}
```

Präsenzaufgabe - Lösung

Queue - 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) {
            this.first = this.first.next; // Verschieben
            length--;
            if(length == 0)
                this.last = null;
            return true;
        }
        return false;
    }
}
```

Präsenzaufgabe - Lösung — Queue - getLength & 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 "[]";

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

Präsenzaufgabe - Lösung

- 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

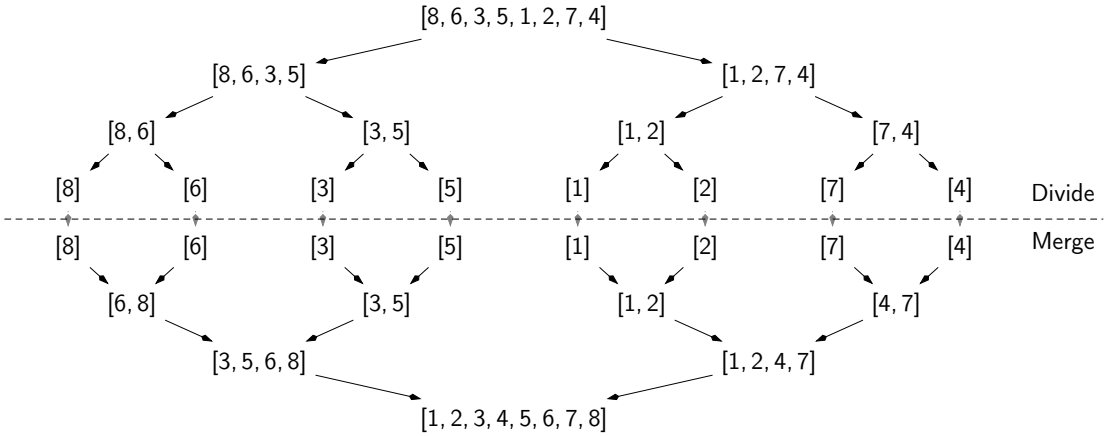
Sortieren Sie die Zahlenfolge $8, 6, 3, 5, 1, 2, 7, 4$ grafisch mit den folgenden Sortieralgorithmen:

- 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.

Übungsblatt 10 - Aufgabe 1

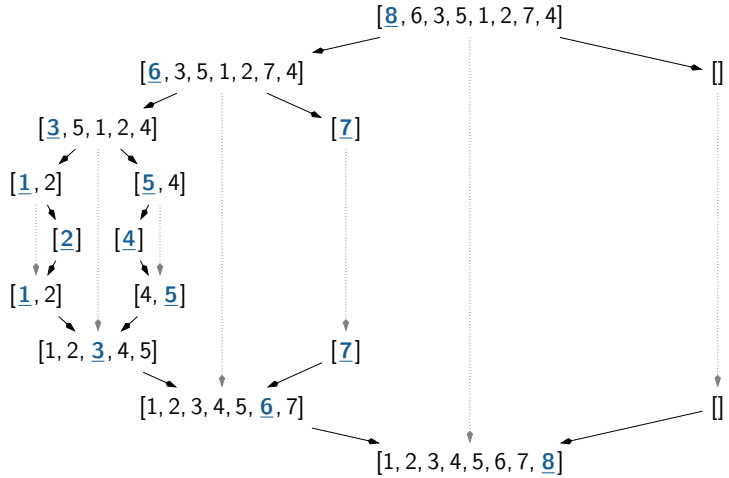
Mergesort



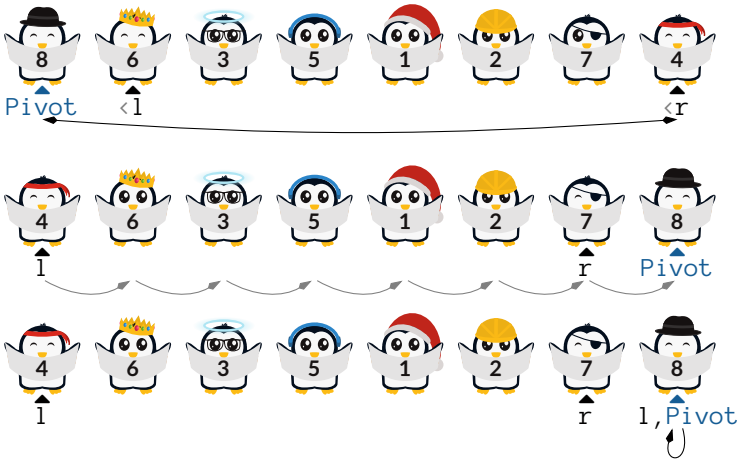
Übungsblatt 10 - Aufgabe 1 Quicksort

- 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.

Übungsblatt 10 - Aufgabe 1 — Quicksort – Listenversion



Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue



Nach der Vorlesung muss das Pivotelement ans Ende!

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

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

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

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) steppsies. And the swappsies.

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

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.

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

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.

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

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

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 = Pivot. Wir verkürzen das in dieser Variante

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

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**r**eater).
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.

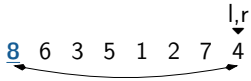
(Der „gleich“-Fall ist an sich beliebig. Man kann also auch `l` für kleiner gleich und `r` für größer als das Pivotelement verwenden. Man muss nur eine der Varianten wählen.)

Übungsblatt 10 - Aufgabe 1 — Quicksort – Nur mit Markern

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

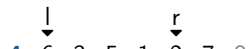
1) Solange a[r] $\geq$ P und a[l] $<$ P

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



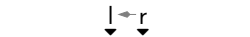
3) Rekursion links. Solange a[r] $\geq$ P und a[l] $<$ P

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

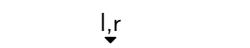


5) Solange a[r] $\geq$ P und a[l] $<$ P

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



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

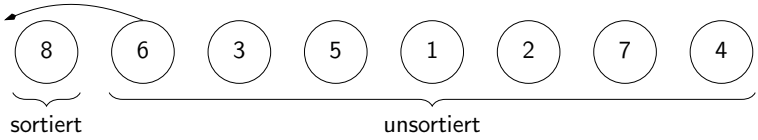


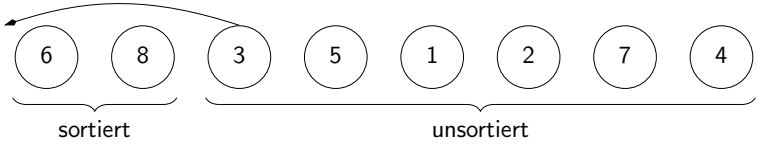
8) Es ist l $\geq$ r.

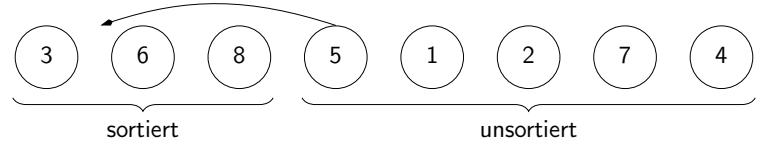
1 2 3 4 5 6 7 8

Fertig sortiert, alle weiteren Anwendungen liefern nichts.

Übungsblatt 10 - Aufgabe 1 Insertionsort

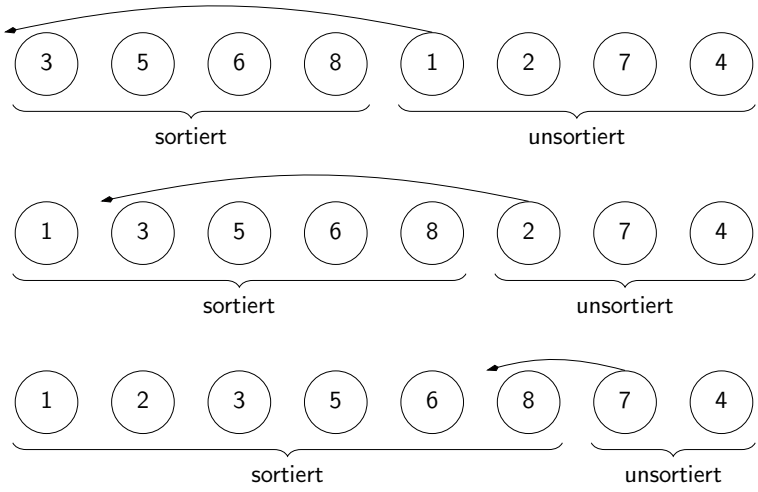


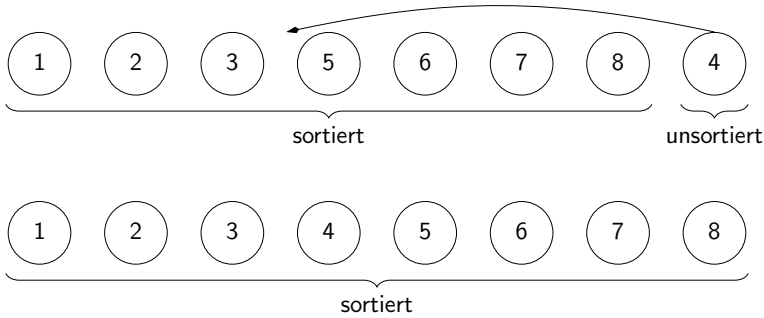




Übungsblatt 10 - Aufgabe 1

Insertionsort, II





- Theoretisch kann man auch alle „Zwischentauschschritte“ einzeichnen.

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

- Iterativ ist das einfach:

```java
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;
            }
        }
    }
}
```

```java
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

```java
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);
    }
}
```

```java
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

- 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

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

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:

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

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

- 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 → 0
  if (w[i] > W) return k(W, i + 1, w, v); // w_i > W_max → 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“)
}
```

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

Knapsack.java

Wieder kein Pingu, weil süß!