9

Fifo Heaps

Tutorium Neun

9

Fifo Heaps

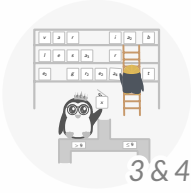
Tutorium Neun

Theorie

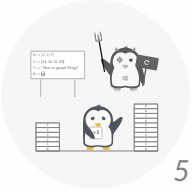
2

13

Grundlagen



3 & 4



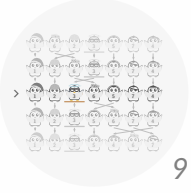
5

6

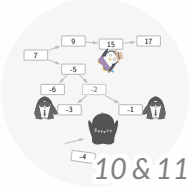
Vertiefungen

7 & 12

8

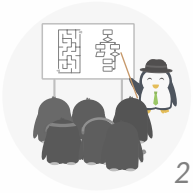


9



10 & 11

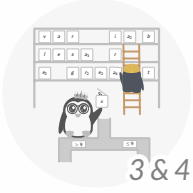
Theorie



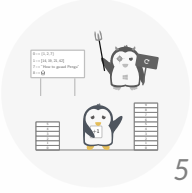
2

13

Grundlagen



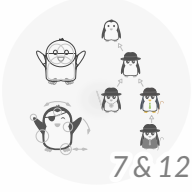
3 & 4



5

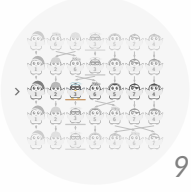
6

Vertiefungen

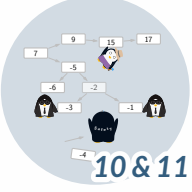


7 & 12

8



9



10 & 11

Kurzwiederholung

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:
  Bubblesort Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

  Bubblesort Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.

  Insertionsort Sortiere erstes unsortiertes Element in sortierten Teil ein.

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

  Bubblesort Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.

  Insertionsort Sortiere erstes unsortiertes Element in sortierten Teil ein.

  Selectionsort Setze kleinsten unsortierten Elements ans Ende des sortierten Teils.

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

Bubblesort	Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.
Insertionsort	Sortiere erstes unsortiertes Element in sortierten Teil ein.
Selectionsort	Setze kleinsten unsortierten Elements ans Ende des sortierten Teils.
Mergesort	Aufteilen bis einelementig, wiederholtes mergen sortierter Teillisten.

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

Bubblesort	Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.
Insertionsort	Sortiere erstes unsortiertes Element in sortierten Teil ein.
Selectionsort	Setze kleinsten unsortierten Elements ans Ende des sortierten Teils.
Mergesort	Aufteilen bis einelementig, wiederholtes mergen sortierter Teillisten.
Quicksort	Pivot ans Ende, l solange $<$, r solange $\geq$. Bei treffen, tausche Pivot.

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

Bubblesort	Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.
Insertionsort	Sortiere erstes unsortiertes Element in sortierten Teil ein.
Selectionsort	Setze kleinsten unsortierten Elements ans Ende des sortierten Teils.
Mergesort	Aufteilen bis einelementig, wiederholtes mergen sortierter Teillisten.
Quicksort	Pivot ans Ende, l solange <, r solange ≥. Bei treffen, tausche Pivot.
Heapsort	Baue Heap (heapify, Eltern ≤ Kinder), entferne Wurzel und heapify.

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

Bubblesort	Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.
Insertionsort	Sortiere erstes unsortiertes Element in sortierten Teil ein.
Selectionsort	Setze kleinsten unsortierten Elements ans Ende des sortierten Teils.
Mergesort	Aufteilen bis einelementig, wiederholtes mergen sortierter Teillisten.
Quicksort	Pivot ans Ende, l solange <, r solange ≥. Bei treffen, tausche Pivot.
Heapsort	Baue Heap (heapify, Eltern ≤ Kinder), entferne Wurzel und heapify.

- Zusätzlich zwei Suchverfahren:

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

  Bubblesort Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.

  Insertionsort Sortiere erstes unsortiertes Element in sortierten Teil ein.

  Selectionsort Setze kleinsten unsortierten Elements ans Ende des sortierten Teils.

  Mergesort Aufteilen bis einelementig, wiederholtes mergen sortierter Teillisten.

  Quicksort Pivot ans Ende, l solange <, r solange ≥. Bei treffen, tausche Pivot.

  Heapsort Baue Heap (heapify, Eltern ≤ Kinder), entferne Wurzel und heapify.

- Zusätzlich zwei Suchverfahren:

  Lineare Suche Betrachte alle Elemente und vergleiche mit Suchschlüssel.

Kurzwiederholung

- Wir kennen eine ganze Bandbreite an Sortierverfahren:

Bubblesort	Vertausche benachbarte Elemente, solange nicht in Sortierreihenfolge.
Insertionsort	Sortiere erstes unsortiertes Element in sortierten Teil ein.
Selectionsort	Setze kleinsten unsortierten Elements ans Ende des sortierten Teils.
Mergesort	Aufteilen bis einelementig, wiederholtes mergen sortierter Teillisten.
Quicksort	Pivot ans Ende, l solange <, r solange ≥. Bei treffen, tausche Pivot.
Heapsort	Baue Heap (heapify, Eltern ≤ Kinder), entferne Wurzel und heapify.

- Zusätzlich zwei Suchverfahren:

Lineare Suche	Betrachte alle Elemente und vergleiche mit Suchschlüssel.
Binäre Suche	Sortierte Liste, wenn Mitte ≠, prüfe wiederholt linke/rechte Hälfte.

Präsenzaufgabe

Präsenzaufgabe

1

FIFO-Freuden

Präsenzaufgabe

1

FIFO-Freuden

In dieser Aufgabe sollen Sie Ihre eigene dynamische Datenstruktur implementieren. Hierbei soll es sich um eine Queue handeln, welche nach dem FIFO-Prinzip (First In — First Out) arbeitet, und `int` Werte speichert. Intern benötigen Sie hier zusätzlich zur Klasse Queue eine (innere) Klasse um die gespeicherten Elemente repräsentieren zu können. Hierfür können Sie die Vorlage `Element.java` auf Moodle verwenden, sowie die Vorlage `Queue.java`, die bereits das Grundgerüst enthält. Implementieren Sie nun zusätzlich die folgenden Methoden der Klasse `Queue`:

- Die Methode `public void enqueue(int value)` soll ein neues Element mit dem übergebenen wert am Ende der Queue einreihen.
- Die Methode `public boolean dequeue()` soll das vorderste Element aus der Queue entfernen.

Sie finden Beispiele für die Operation in Abbildung. Verwenden Sie für Ihre Imple-

Präsenzaufgabe

1

FIFO-Freuden

In dieser Aufgabe sollen Sie Ihre eigene dynamische Datenstruktur implementieren. Hierbei soll es sich um eine Queue handeln, welche nach dem FIFO-Prinzip (First In — First Out) arbeitet, und `int` Werte speichert. Intern benötigen Sie hier zusätzlich zur Klasse Queue eine (innere) Klasse um die gespeicherten Elemente repräsentieren zu können. Hierfür können Sie die Vorlage `Element.java` auf Moodle verwenden, sowie die Vorlage `Queue.java`, die bereits das Grundgerüst enthält. Implementieren Sie nun zusätzlich die folgenden Methoden der Klasse `Queue`:

- Die Methode `public void enqueue(int value)` soll ein neues Element mit dem übergebenen wert am Ende der Queue einreihen.
- Die Methode `public boolean dequeue()` soll das vorderste Element aus der Queue entfernen.

Sie finden Beispiele für die Operation in Abbildung. Verwenden Sie für Ihre Implementierung keine Arrays oder vorgefertigte dynamische Datenstrukturen, sondern implementieren Sie die Queue selbst als (einfach) verkettete Liste.

Präsenzaufgabe

1

FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

Präsenzaufgabe

1

FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

- `void enqueue(int value)`, welche den Wert ans Ende der Queue einreiht

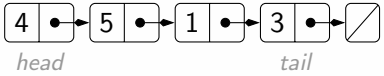
Präsenzaufgabe

1

FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

- `void enqueue(int value)`, welche den Wert ans Ende der Queue einreiht



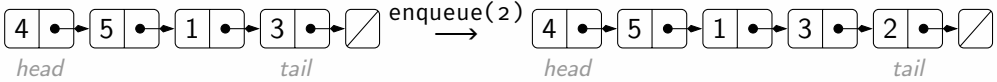
Präsenzaufgabe

1

FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

› `void enqueue(int value)`, welche den Wert ans Ende der Queue einreiht



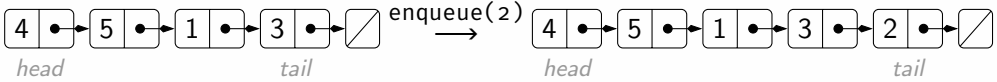
Präsenzaufgabe

1

FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

- `void enqueue(int value)`, welche den Wert ans Ende der Queue einreiht



- `boolean dequeue()`, welche den vordersten Wert entfernt

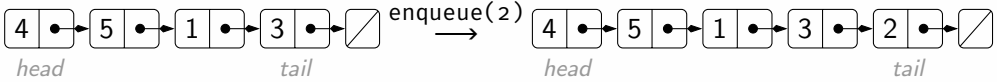
Präsenzaufgabe

1

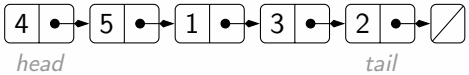
FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

- `void enqueue(int value)`, welche den Wert ans Ende der Queue einreiht



- `boolean dequeue()`, welche den vordersten Wert entfernt



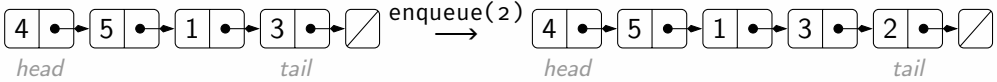
Präsenzaufgabe

1

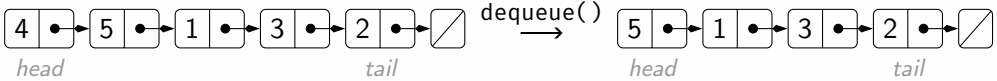
FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

- `void enqueue(int value)`, welche den Wert ans Ende der Queue einreiht



- `boolean dequeue()`, welche den vordersten Wert entfernt

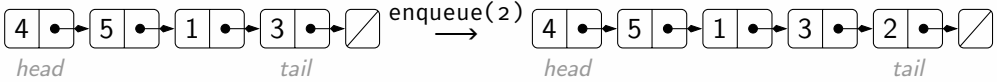


Präsenzaufgabe

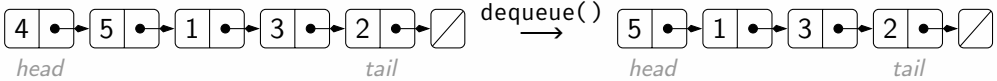
1 FIFO-Freuden

Implementieren Sie eine Queue, die `int` Werte speichert. Verwenden Sie dazu `Element.java` und `Queue.java` als Grundgerüst und implementieren Sie:

- `void enqueue(int value)`, welche den Wert ans Ende der Queue einreiht



- `boolean dequeue()`, welche den vordersten Wert entfernt



Verwenden Sie keine Arrays oder vorgefertigte dynamische Datenstrukturen, sondern eine eigene, (einfach) verkettete Liste.

Adding something to a Queue

Adding something to a Queue

```java
public class Queue {
    private Element first;
    private Element last;
    private int length;
    // ...

    public void enqueue(int value) {
        // TODO
    }
}
```

Adding something to a Queue

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

    public void enqueue(int value) {




    }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);









   }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);
      if(length == 0)

      else



   }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);
      if(length == 0)   Spezialfall beim Einfügen: Isse leer?

      else




   }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);
      if(length == 0)   Spezialfall beim Einfügen: Isse leer?
         this.first = next;
      else




   }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);
      if(length == 0)   Spezialfall beim Einfügen: Isse leer?
         this.first = next;
      else
         this.last.setNextElement(next);


   }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);
      if(length == 0)   Spezialfall beim Einfügen: Isse leer?
         this.first = next;
      else
         this.last.setNextElement(next);
      this.last = next;

   }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);
      if(length == 0)   Spezialfall beim Einfügen: Isse leer?
         this.first = next;
      else
         this.last.setNextElement(next);
      this.last = next;   In jedem Fall: Verschiebe den Tail

   }
}
```

Adding something to a Queue

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

   public void enqueue(int value) {
      Element next = new Element(value);
      if(length == 0)   Spezialfall beim Einfügen: Isse leer?
         this.first = next;
      else
         this.last.setNextElement(next);
      this.last = next;   In jedem Fall: Verschiebe den Tail
      length++;
   }
}
```

Moving out

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









}
```

Moving out

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

   public boolean dequeue() {




   }
}
```

Moving out

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

   public boolean dequeue() {
      if(length > 0) {



      }

   }
}
```

Moving out

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

   public boolean dequeue() {
      if(length > 0) {
         this.first = this.first.getNextElement();


      }

   }
}
```

Moving out

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

   public boolean dequeue() {
      if(length > 0) {
         this.first = this.first.getNextElement();
         length--;

      }

   }
}
```

Moving out

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

   public boolean dequeue() {
      if(length > 0) {
         this.first = this.first.getNextElement();
         length--;
         return true;
      }

   }
}
```

Moving out

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

   public boolean dequeue() {
      if(length > 0) {
         this.first = this.first.getNextElement();
         length--;
         return true;    Es gab etwas zu Entfernen!
      }

   }
}
```

Moving out

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

   public boolean dequeue() {
      if(length > 0) {
         this.first = this.first.getNextElement();
         length--;
         return true;    Es gab etwas zu Entfernen!
      }
      return false;
   }
}
```

Moving out

Queue.java

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

   public boolean dequeue() {
      if(length > 0) {
         this.first = this.first.getNextElement();
         length--;
         return true;    Es gab etwas zu Entfernen!
      }
      return false;
   }
}
```

Übungsblatt 9

Aufgabe 1: Heap Sort

Aufgabe 1: Heap Sort

In dieser Aufgabe sollen Sie das folgende Array händisch aufsteigend mit dem Heap Sort Algorithmus sortieren:

$$[0, -1, 9, 4]$$

a) *Phase 1: Geben Sie den schrittweise den entsprechenden Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist. Orientieren Sie sich dabei an der Darstellung aus der Vorlesung.*

b) *Phase 2: Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um. Markieren Sie jeweils wo nach dem Herausnehmen des Head die Heap-Eigenschaft verletzt wurde und das entsprechende Resultat der Heapify Operation.*

Der Aufbau eines Heaps

Der Aufbau eines Heaps

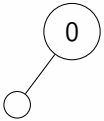
a) Geben Sie den schrittweise den Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist.

$$[0, -1, 9, 4]$$

Der Aufbau eines Heaps

a) Geben Sie den schrittweise den Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist.

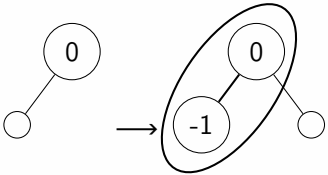
$[0, -1, 9, 4]$



Der Aufbau eines Heaps

a) Geben Sie den schrittweise den Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist.

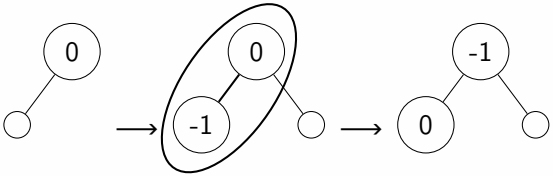
$[0, -1, 9, 4]$



Der Aufbau eines Heaps

a) Geben Sie den schrittweise den Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist.

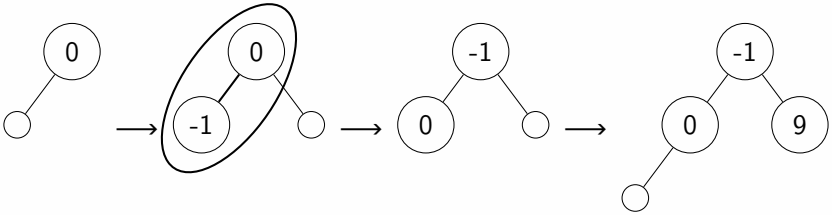
$[0, -1, 9, 4]$



Der Aufbau eines Heaps

a) Geben Sie den schrittweise den Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist.

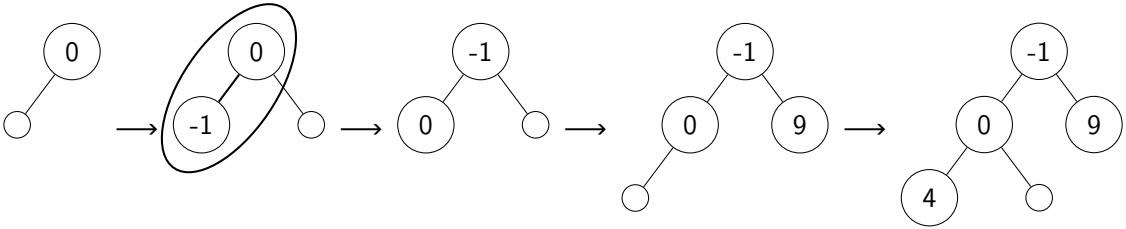
$[0, -1, 9, 4]$



Der Aufbau eines Heaps

a) Geben Sie den schrittweise den Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist.

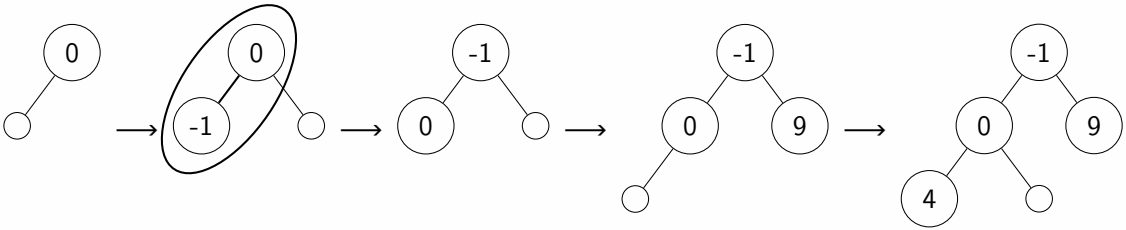
$[0, -1, 9, 4]$



Der Aufbau eines Heaps

a) Geben Sie den schrittweise den Heap an und markieren Sie jeweils wo die Heap-Eigenschaft verletzt ist.

$[0, -1, 9, 4]$



Hier haben wir einen Min-Heap, ein Max-Heap würde genau so funktionieren!

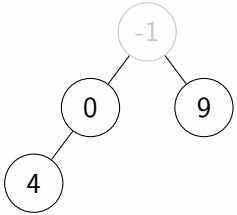
Der Abbau eines Miep

Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.

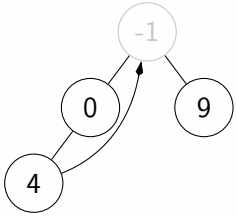
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



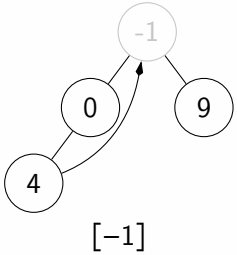
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



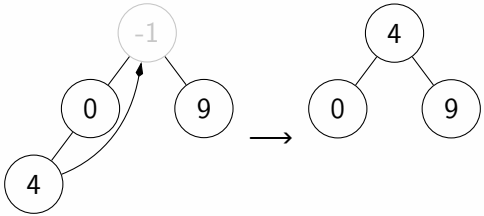
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



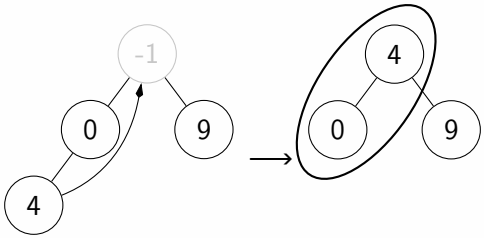
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



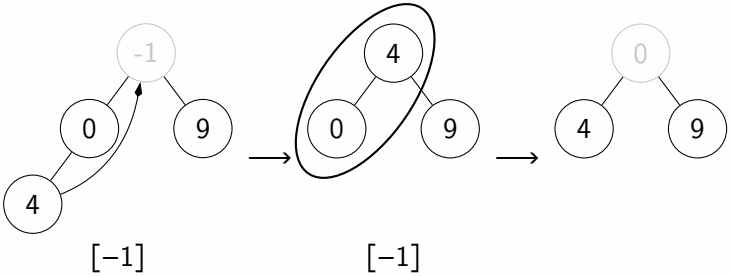
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



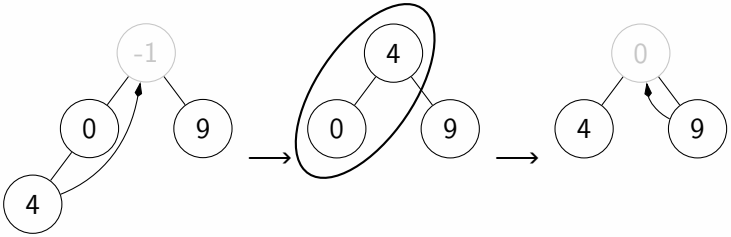
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



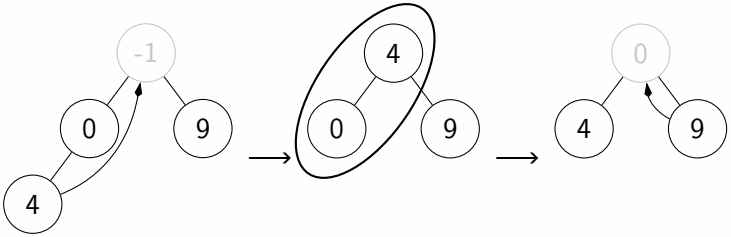
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



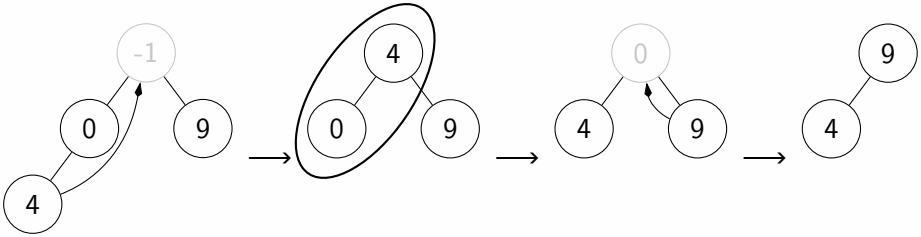
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



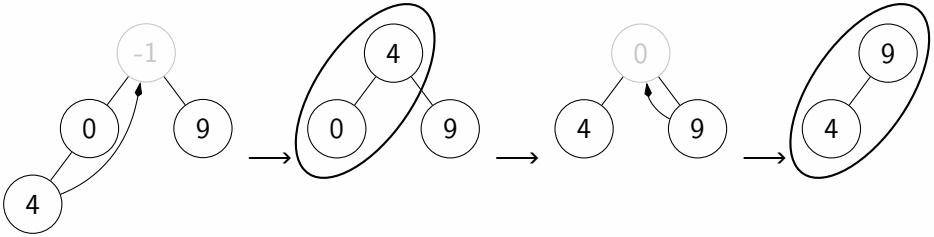
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



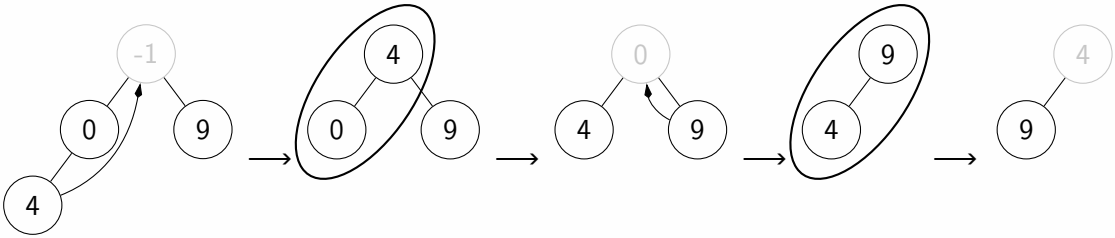
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



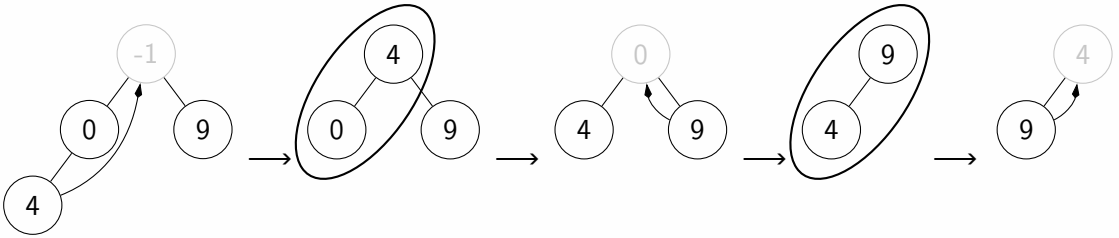
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



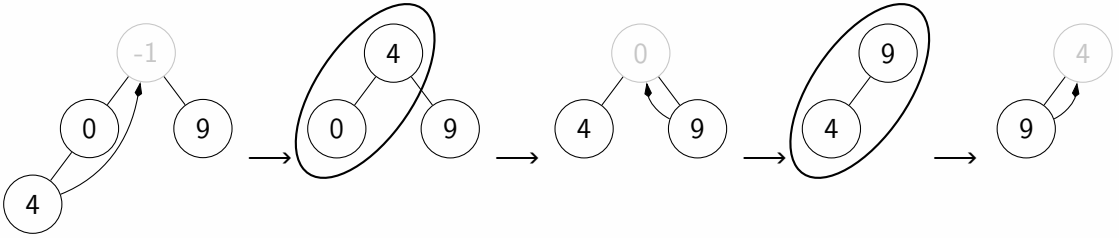
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



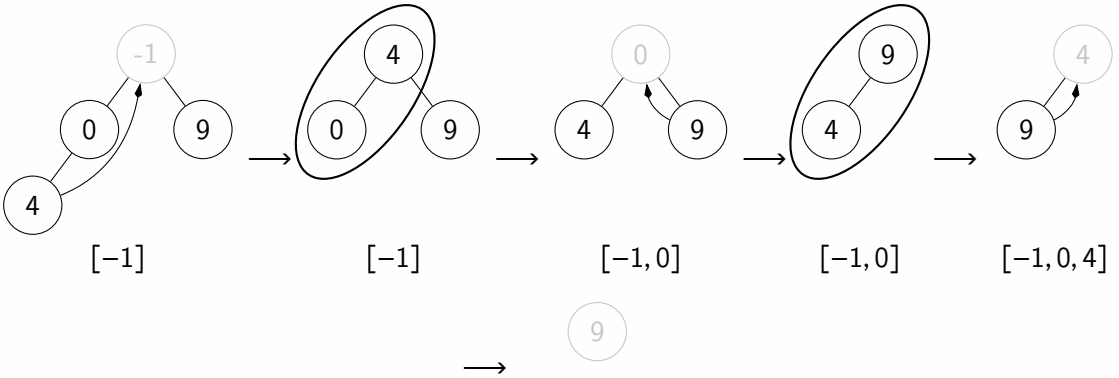
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



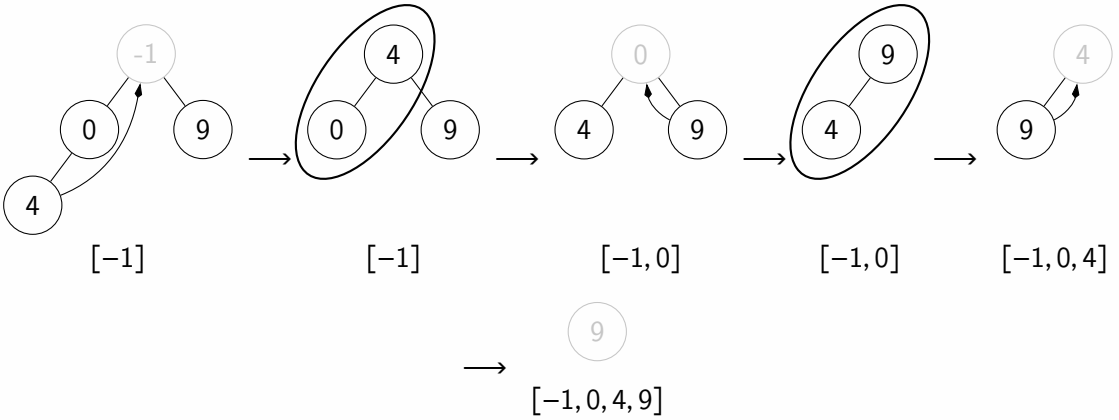
Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



Der Abbau eines Miep

b) Wandeln Sie den Heap aus Phase 1 schrittweise in ein sortiertes Array um.



Aufgabe 2: Rekursive Sortieralgorithmen: Shaker Sort

Aufgabe 2: Rekursive Sortieralgorithmen: Shaker Sort

In dieser Aufgabe sollen Sie einen rekursiven Sortieralgorithmus selbst implementieren. Wir betrachten eine Variante des Bubble-Sort Algorithmus, bei dem das Array in jedem Sortierschritt alternierend von vorne und hinten durchlaufen wird. Nach jedem Durchlaufen werden die bereits sortieren Elemente vorne und hinten nicht mehr betrachtet werden. Implementieren Sie Shaker Sort in Java und testen Sie Ihre Lösung an mindestens einem Beispiel.

Helping Swap

Helping Swap

```
public class ShakerSort {



}
```

Helping Swap

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) {



    }
}
```

Helping Swap

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) {
      int tmp = arr[i];
      arr[i] = arr[j];
      arr[j] = tmp;
    }
}
```

Recursive Bubble-Up

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }









}
```

Recursive Bubble-Up

```
public class ShakerSort {
    private static void swap(int[] arr, int i, int j) { ... }

    static boolean moveUp(int[] arr, int i, int end, boolean swapped) {




    }
}
```

Recursive Bubble-Up

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }

   static boolean moveUp(int[] arr, int i, int end, boolean swapped) {
      if (i >= end)
         return swapped;









   }
}
```

Recursive Bubble-Up

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }

   static boolean moveUp(int[] arr, int i, int end, boolean swapped) {
      if (i >= end)
         return swapped;
      if (arr[i] < arr[i - 1]) {


      }

   }
}
```

Recursive Bubble-Up

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }

   static boolean moveUp(int[] arr, int i, int end, boolean swapped) {
      if (i >= end)
         return swapped;
      if (arr[i] < arr[i - 1]) {
         swap(arr, i, i - 1);

      }

   }
}
```

Recursive Bubble-Up

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }

   static boolean moveUp(int[] arr, int i, int end, boolean swapped) {
      if (i >= end)
         return swapped;
      if (arr[i] < arr[i - 1]) {
         swap(arr, i, i - 1);
         swapped = true;
      }

   }
}
```

Recursive Bubble-Up

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }

   static boolean moveUp(int[] arr, int i, int end, boolean swapped) {
      if (i >= end)
         return swapped;
      if (arr[i] < arr[i - 1]) {
         swap(arr, i, i - 1);
         swapped = true;
      }
      return moveUp(arr, i+1, end, swapped);
   }
}
```

Recursive Bubble-Up

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }

   static boolean moveUp(int[] arr, int i, int end, boolean swapped) {
      if (i >= end)
         return swapped;
      if (arr[i] < arr[i - 1]) {
         swap(arr, i, i - 1);
         swapped = true;
      }
      return moveUp(arr, i+1, end, swapped);
   }
}
```

Was müsste man anpassen um immer mit `i + 1` zu vergleichen?

Recursive Bubble-Down

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean swapped)
      { ... }









}
```

Recursive Bubble-Down

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean swapped)
      { ... }

   static boolean moveDown(int[] arr, int i, int end, boolean swapped) {
      if (i <= end)
         return swapped;
      if (arr[i] < arr[i - 1]) {
         swap(arr, i, i - 1);
         swapped = true;
      }
      return moveDown(arr, i-1, end, swapped);
   }
}
```

Recursive Bubble-Down

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean swapped)
      { ... }

   static boolean moveDown(int[] arr, int i, int end, boolean swapped) {
      if (i <= end)
         return swapped;
      if (arr[i] < arr[i - 1]) {
         swap(arr, i, i - 1);
         swapped = true;
      }
      return moveDown(arr, i-1, end, swapped);
   }
}
```

I shake it up, I shake it down!

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean s) { ... }
   static boolean moveDown(int[] arr, int i, int end, boolean s) { ... }
```

I shake it up, I shake it down!

```
public class ShakerSort {
    private static void swap(int[] arr, int i, int j) { ... }
    static boolean moveUp(int[] arr, int i, int end, boolean s) { ... }
    static boolean moveDown(int[] arr, int i, int end, boolean s) { ... }

    public static void shakerSort(int[] array, int n) {




    }
}
```

I shake it up, I shake it down!

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean s) { ... }
   static boolean moveDown(int[] arr, int i, int end, boolean s) { ... }

   public static void shakerSort(int[] array, int n) {
      int offset = array.length - n;









   }
}
```

I shake it up, I shake it down!

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean s) { ... }
   static boolean moveDown(int[] arr, int i, int end, boolean s) { ... }

   public static void shakerSort(int[] array, int n) {
      int offset = array.length - n;
      if (n <= array.length / 2)
         return;









   }
}
```

I shake it up, I shake it down!

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean s) { ... }
   static boolean moveDown(int[] arr, int i, int end, boolean s) { ... }

   public static void shakerSort(int[] array, int n) {
      int offset = array.length - n;
      if (n <= array.length / 2)
         return;
      else if (!moveUp(array, offset + 1, n, false))

      else if (!moveDown(array, n - 1, offset, false))


   }
}
```

I shake it up, I shake it down!

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean s) { ... }
   static boolean moveDown(int[] arr, int i, int end, boolean s) { ... }

   public static void shakerSort(int[] array, int n) {
      int offset = array.length - n;
      if (n <= array.length / 2)
         return;
      else if (!moveUp(array, offset + 1, n, false))
         return;
      else if (!moveDown(array, n - 1, offset, false))
         return;

   }
}
```

I shake it up, I shake it down!

```
public class ShakerSort {
   private static void swap(int[] arr, int i, int j) { ... }
   static boolean moveUp(int[] arr, int i, int end, boolean s) { ... }
   static boolean moveDown(int[] arr, int i, int end, boolean s) { ... }

   public static void shakerSort(int[] array, int n) {
      int offset = array.length - n;
      if (n <= array.length / 2)
         return;
      else if (!moveUp(array, offset + 1, n, false))
         return;
      else if (!moveDown(array, n - 1, offset, false))
         return;
      shakerSort(array, n - 1);
   }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }









}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {









    }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {
        Random random = new Random();









    }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {
        Random random = new Random();
        int length = random.nextInt(11) + 10;









    }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {
        Random random = new Random();
        int length = random.nextInt(11) + 10;
        int[] array = new int[length];









    }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {
        Random random = new Random();
        int length = random.nextInt(11) + 10;
        int[] array = new int[length];
        for (int i = 0; i < length; i++)
            array[i] = random.nextInt(100);



    }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {
        Random random = new Random();
        int length = random.nextInt(11) + 10;
        int[] array = new int[length];
        for (int i = 0; i < length; i++)
            array[i] = random.nextInt(100);
        System.out.println("Unsortiert: " + Arrays.toString(array));


    }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {
        Random random = new Random();
        int length = random.nextInt(11) + 10;
        int[] array = new int[length];
        for (int i = 0; i < length; i++)
            array[i] = random.nextInt(100);
        System.out.println("Unsortiert: " + Arrays.toString(array));
        shakerSort(array, length);

    }
}
```

Some sample Code

ShakerSort.java

```
public class ShakerSort {
    public static void shakerSort(int[] array, int n) { ... }

    public static void main(String[] args) {
        Random random = new Random();
        int length = random.nextInt(11) + 10;
        int[] array = new int[length];
        for (int i = 0; i < length; i++)
            array[i] = random.nextInt(100);
        System.out.println("Unsortiert: " + Arrays.toString(array));
        shakerSort(array, length);
        System.out.println("Sortiert: " + Arrays.toString(array));
    }
}
```

Aufgabe 3: Laplacescher Entwicklungssatz

Aufgabe 3: Laplacescher Entwicklungssatz

In dieser Aufgabe sollen Sie nochmal die Implementierung rekursiver Funktionen üben. Dazu betrachten wir die rekursive Berechnung der Determinante einer quadratischen $n \times n$ *Matrix* M*. Die Determinante gibt an, wie sich das Volumen bei der durch die Matrix beschriebenen linearen Abbildung ändert, und ist ein Hilfsmittel bei der Lösung linearer Gleichungssysteme. Sie kann für kleine Matrizen* ($1 \leq n \leq 10$) *rekursiv über den Laplaceschen Entwicklungssatz nach der ersten Spalte berechnet werden:*

$$\det(M) = \begin{cases} m_{1,1} & \text{falls } n = 1 \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) & \text{für } n \geq 2 \end{cases}$$

wobei $n \times n$ *die Dimensionalität der Matrix* M, $M_{-i,1}$ *die Matrix, die aus* M *entsteht, wenn die erste Spalte und die* i*-te Zeile entfernt werden (samt anschließendem Zusammenschieben) und* $m_{i,j}$ *das element in der* i*-ten Zeile und* j*-ten Spalte seien.*
In den Materialien zu diesem Übungsblatt finden Sie die Datei `Determinante.java`*, die bereits alle notwendigen Hilfsmethoden enthält. Sie sollen die vorgegebene Methode* `int det(int[][] mat)` *implementieren.*

Translating a formula… 1:1 (caring for zeros)

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Translating a formula… 1:1 (caring for zeros)

```
private static int det(int[][] mat) {



}
```

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Translating a formula… 1:1 (caring for zeros)

```
private static int det(int[][] mat) {
    if(mat.length == 1) {

    } else {









    }
}
```

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Translating a formula… 1:1 (caring for zeros)

```
private static int det(int[][] mat) {
    if(mat.length == 1) {
        return mat[0][0];
    } else {




    }
}
```

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Translating a formula… 1:1 (caring for zeros)

```
private static int det(int[][] mat) {
    if(mat.length == 1) {
        return mat[0][0];
    } else {
        int result = 0;




    }
}
```

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Translating a formula… 1:1 (caring for zeros)

```
private static int det(int[][] mat) {
   if(mat.length == 1) {
      return mat[0][0];
   } else {
      int result = 0;
      for(int i = 0; i < mat.length; i++) {


      }

   }
}
```

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Translating a formula… 1:1 (caring for zeros)

```
private static int det(int[][] mat) {
   if(mat.length == 1) {
      return mat[0][0];
   } else {
      int result = 0;
      for(int i = 0; i < mat.length; i++) {
         result += (int)Math.pow(-1, i+1) * mat[i][0]
                   * det(reduceMatrix(mat, i));
      }

   }
}
```

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Translating a formula… 1:1 (caring for zeros)

```java
private static int det(int[][] mat) {
   if(mat.length == 1) {
      return mat[0][0];
   } else {
      int result = 0;
      for(int i = 0; i < mat.length; i++) {
         result += (int)Math.pow(-1, i+1) * mat[i][0]
                   * det(reduceMatrix(mat, i));
      }
      return result;
   }
}
```

$$\det(M) = \begin{cases} m_{1,1} \\ \sum_{i=1}^{n} (-1)^{i+1} m_{i,1} \det(M_{-i1}) \end{cases}$$

Aussicht: Übungsblatt 10

Aufgabe 1: Bäume

Aufgabe 1: Bäume

- Wir können Bäume als rekursive Datenstruktur beschreiben

Aufgabe 1: Bäume

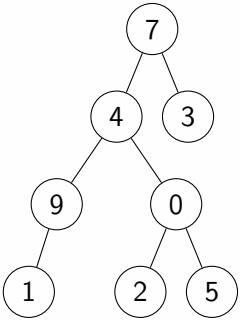
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums

Aufgabe 1: Bäume

- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum

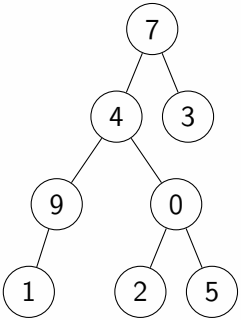
Aufgabe 1: Bäume

- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



Aufgabe 1: Bäume

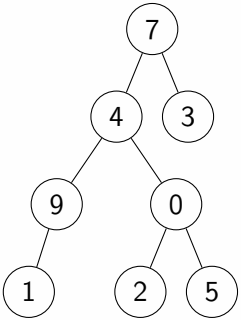
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



- Breitendurchlauf:

Aufgabe 1: Bäume

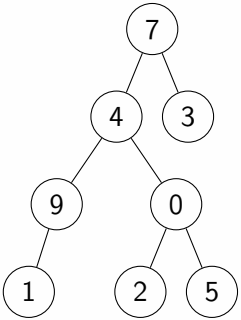
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



- Breitendurchlauf:
  7, 4, 3, 9, 0, 1, 2, 5

Aufgabe 1: Bäume

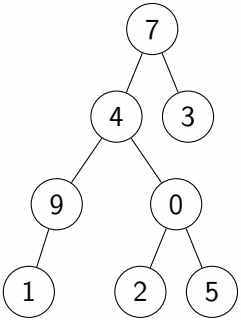
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



- Breitendurchlauf:
  7, 4, 3, 9, 0, 1, 2, 5
- Prä-Order: (Knoten, links, rechts)

Aufgabe 1: Bäume

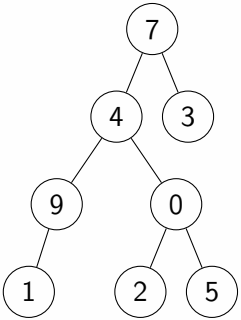
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



- Breitendurchlauf:
  7, 4, 3, 9, 0, 1, 2, 5
- Prä-Order: (Knoten, links, rechts)
  7, 4, 9, 1, 0, 2, 5, 3

Aufgabe 1: Bäume

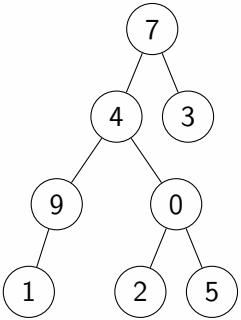
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



- Breitendurchlauf:
  7, 4, 3, 9, 0, 1, 2, 5
- Prä-Order: (Knoten, links, rechts)
  7, 4, 9, 1, 0, 2, 5, 3
- In-Order: (links, Knoten, rechts)

Aufgabe 1: Bäume

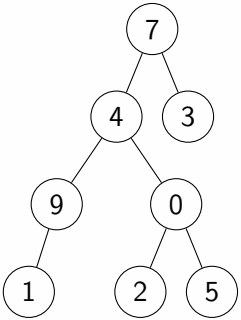
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



- Breitendurchlauf:
  7, 4, 3, 9, 0, 1, 2, 5
- Prä-Order: (Knoten, links, rechts)
  7, 4, 9, 1, 0, 2, 5, 3
- In-Order: (links, Knoten, rechts)
  1, 9, 4, 2, 0, 5, 7, 3

Aufgabe 1: Bäume

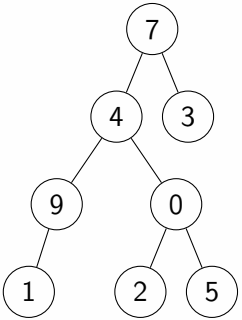
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



- Breitendurchlauf:
  7, 4, 3, 9, 0, 1, 2, 5
- Prä-Order: (Knoten, links, rechts)
  7, 4, 9, 1, 0, 2, 5, 3
- In-Order: (links, Knoten, rechts)
  1, 9, 4, 2, 0, 5, 7, 3
- Post-Order: (links, rechts, Knoten)

Aufgabe 1: Bäume

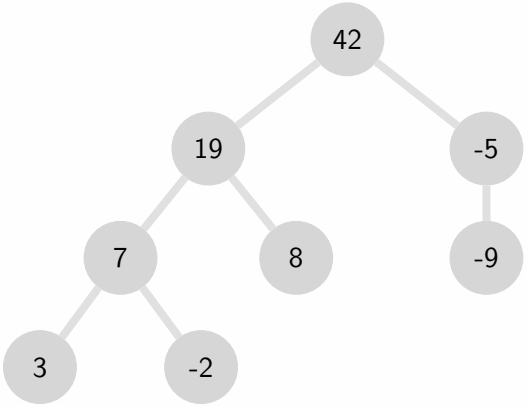
- Wir können Bäume als rekursive Datenstruktur beschreiben
  - Für Binärbäume haben wir einen Knoten als Wurzel eines Teilbaums
  - Einen linken Teilbaum und einen rechten Teilbaum



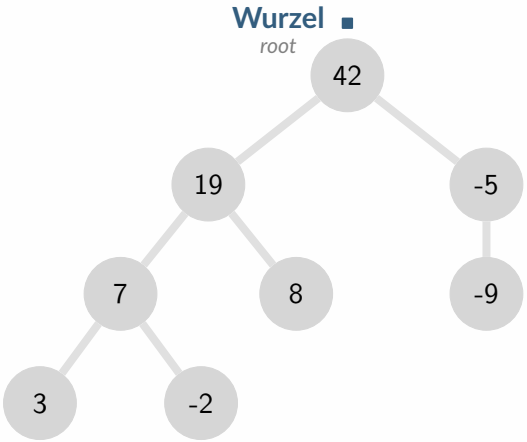
- Breitendurchlauf:
  7, 4, 3, 9, 0, 1, 2, 5
- Prä-Order: (Knoten, links, rechts)
  7, 4, 9, 1, 0, 2, 5, 3
- In-Order: (links, Knoten, rechts)
  1, 9, 4, 2, 0, 5, 7, 3
- Post-Order: (links, rechts, Knoten)
  1, 9, 2, 5, 0, 4, 3, 7

Exkurs: Ein wenig fantastische Heaps gefällig?

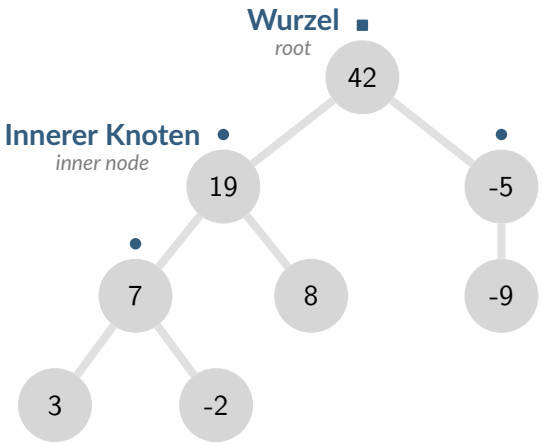
Exkurs: Ein wenig fantastische Heaps gefällig?



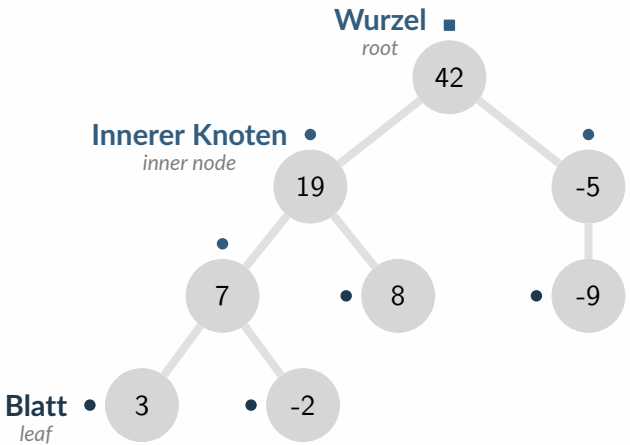
Exkurs: Ein wenig fantastische Heaps gefällig?



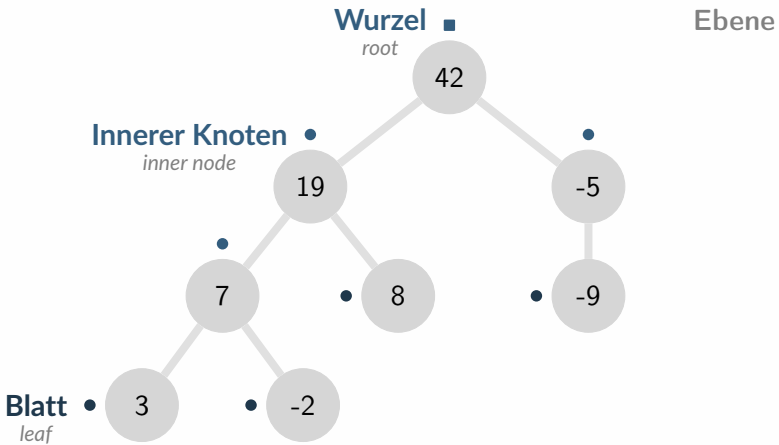
Exkurs: Ein wenig fantastische Heaps gefällig?



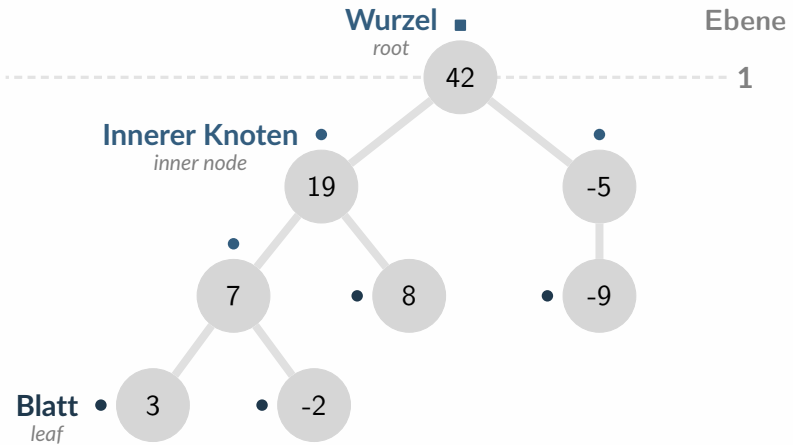
Exkurs: Ein wenig fantastische Heaps gefällig?



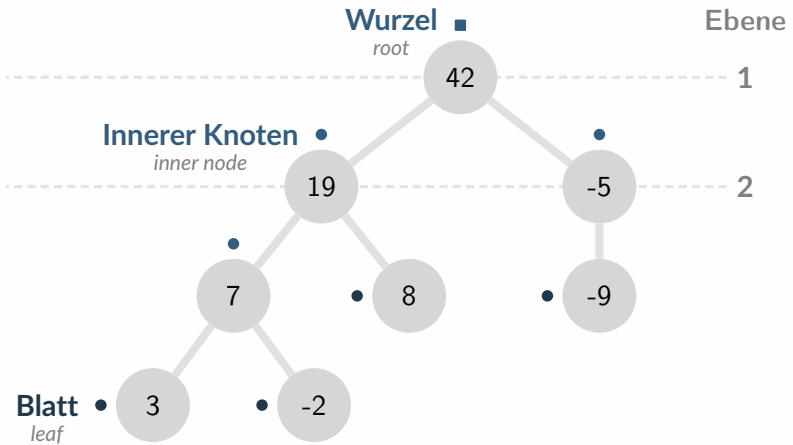
Exkurs: Ein wenig fantastische Heaps gefällig?



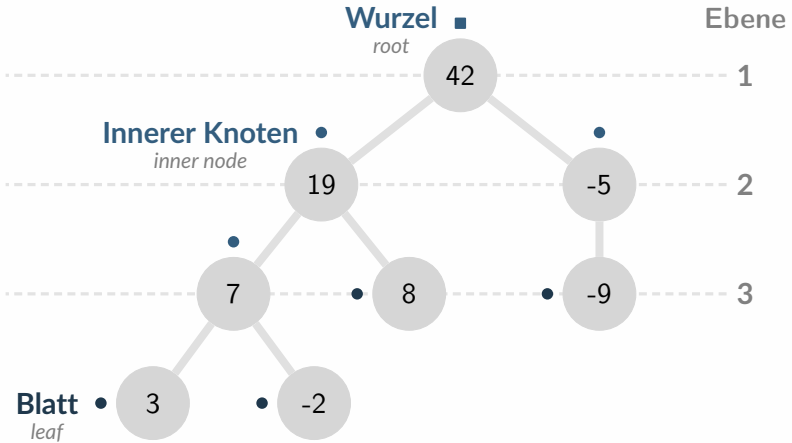
Exkurs: Ein wenig fantastische Heaps gefällig?



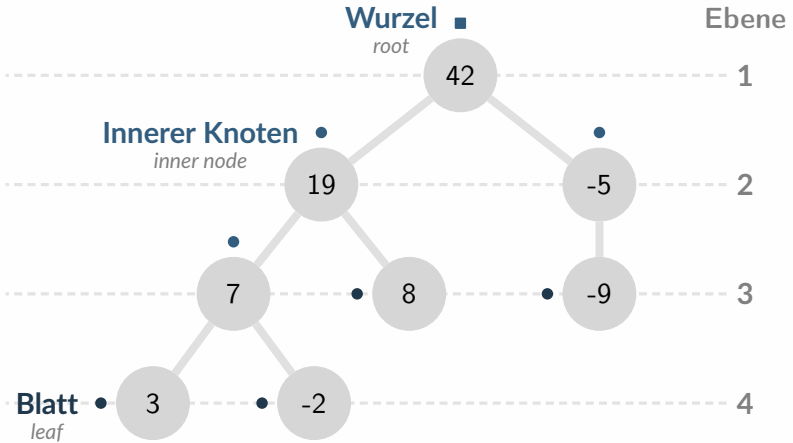
Exkurs: Ein wenig fantastische Heaps gefällig?



Exkurs: Ein wenig fantastische Heaps gefällig?



Exkurs: Ein wenig fantastische Heaps gefällig?



Aufgabe 2: Self-Overriding Ring Buffer

Aufgabe 2: Self-Overriding Ring Buffer

- Bei einem Ring Buffer, zeigt ein Element nie auf `null`

Aufgabe 2: Self-Overriding Ring Buffer

- Bei einem Ring Buffer, zeigt ein Element nie auf `null`
- Komplizierter wird es, wenn nicht werte in Elementen überschrieben sondern neue Elemente eingesetzt werden müssen

Aufgabe 2: Self-Overriding Ring Buffer

- Bei einem Ring Buffer, zeigt ein Element nie auf `null`
- Komplizierter wird es, wenn nicht werte in Elementen überschrieben sondern neue Elemente eingesetzt werden müssen
- Versucht kategorisch, Skizzen zur Hilfestellung zu nehmen

Aufgabe 2: Self-Overriding Ring Buffer

- Bei einem Ring Buffer, zeigt ein Element nie auf `null`
- Komplizierter wird es, wenn nicht werte in Elementen überschrieben sondern neue Elemente eingesetzt werden müssen
- Versucht kategorisch, Skizzen zur Hilfestellung zu nehmen
  - Insbesondere wenn mehrere Zeiger im Spiel sind, ist das hilfreich

Aufgabe 2: Self-Overriding Ring Buffer

- Bei einem Ring Buffer, zeigt ein Element nie auf `null`
- Komplizierter wird es, wenn nicht werte in Elementen überschrieben sondern neue Elemente eingesetzt werden müssen
- Versucht kategorisch, Skizzen zur Hilfestellung zu nehmen
  - Insbesondere wenn mehrere Zeiger im Spiel sind, ist das hilfreich
  - So können auch Randfälle identifiziert werden

Aufgabe 2: Self-Overriding Ring Buffer

- Bei einem Ring Buffer, zeigt ein Element nie auf `null`
- Komplizierter wird es, wenn nicht werte in Elementen überschrieben sondern neue Elemente eingesetzt werden müssen
- Versucht kategorisch, Skizzen zur Hilfestellung zu nehmen
  - Insbesondere wenn mehrere Zeiger im Spiel sind, ist das hilfreich
  - So können auch Randfälle identifiziert werden
- Gebt euch Mühe, sauberen Code abzugeben

Aufgabe 2: Self-Overriding Ring Buffer

- Bei einem Ring Buffer, zeigt ein Element nie auf `null`
- Komplizierter wird es, wenn nicht werte in Elementen überschrieben sondern neue Elemente eingesetzt werden müssen
- Versucht kategorisch, Skizzen zur Hilfestellung zu nehmen
  - Insbesondere wenn mehrere Zeiger im Spiel sind, ist das hilfreich
  - So können auch Randfälle identifiziert werden
- Gebt euch Mühe, sauberen Code abzugeben
- Kommentare werden gern gesehen

Abschließendes

Zusammenfassend

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

```
void preorder(Node n) {
   if(n == null) return;
   // process n
   preorder(n.left);
   preorder(n.right);
}
```

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

```
void preorder(Node n) {
   if(n == null) return;
   // process n
   preorder(n.left);
   preorder(n.right);
}
```

```
void inorder(Node n) {
   if(n == null) return;
   inorder(n.left);
   // process n
   inorder(n.right);
}
```

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

```
void preorder(Node n) {
   if(n == null) return;
   // process n
   preorder(n.left);
   preorder(n.right);
}
```

```
void inorder(Node n) {
   if(n == null) return;
   inorder(n.left);
   // process n
   inorder(n.right);
}
```

```
void postorder(Node n) {
   if(n == null) return;
   postorder(n.left);
   postorder(n.right);
   // process n
}
```

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

```
void preorder(Node n) {
   if(n == null) return;
   // process n
   preorder(n.left);
   preorder(n.right);
}
```

```
void inorder(Node n) {
   if(n == null) return;
   inorder(n.left);
   // process n
   inorder(n.right);
}
```

```
void postorder(Node n) {
   if(n == null) return;
   postorder(n.left);
   postorder(n.right);
   // process n
}
```

  - Auch lassen sich Regeln für Heaps so beschreiben

Zusammenfassend

- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

```
void preorder(Node n) {
   if(n == null) return;
   // process n
   preorder(n.left);
   preorder(n.right);
}
```

```
void inorder(Node n) {
   if(n == null) return;
   inorder(n.left);
   // process n
   inorder(n.right);
}
```

```
void postorder(Node n) {
   if(n == null) return;
   postorder(n.left);
   postorder(n.right);
   // process n
}
```

  - Auch lassen sich Regeln für Heaps so beschreiben
- Listen (als spezielle Bäume) und Bäume sind vielseitig und elementar

Zusammenfassend

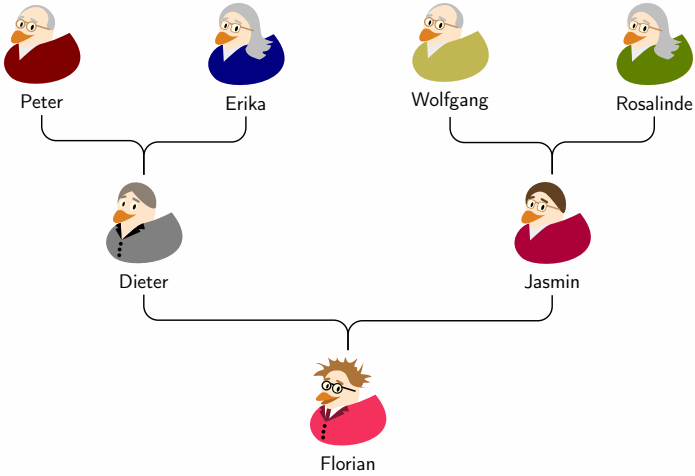
- Rekursion wird uns so schnell nicht verlassen
  - Viele Datenstrukturen sind rekursiv, was wir ausnutzen können
  - Für die verschiedenen Tiefen-Traversierungen beispielsweise:

```
void preorder(Node n) {
   if(n == null) return;
   // process n
   preorder(n.left);
   preorder(n.right);
}
```

```
void inorder(Node n) {
   if(n == null) return;
   inorder(n.left);
   // process n
   inorder(n.right);
}
```

```
void postorder(Node n) {
   if(n == null) return;
   postorder(n.left);
   postorder(n.right);
   // process n
}
```

  - Auch lassen sich Regeln für Heaps so beschreiben
- Listen (als spezielle Bäume) und Bäume sind vielseitig und elementar
  - Auch bei ihrer Handhabung hilft Übung!


Peter
Erika
Wolfgang
Rosalinde
Dieter
Jasmin
Florian