Datenstrukturen und Algorithmen
6. Globalübung
Teil 1: Sortierverfahren

Stefan Dollase

20. Mai 2020

Quicksort – Aufgabe

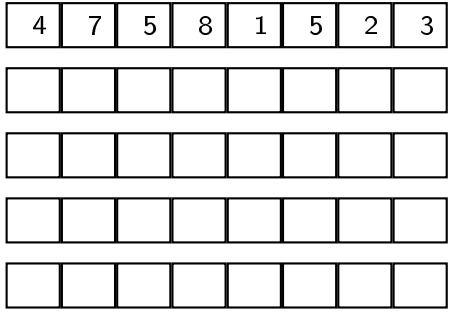
```
1  void quickSort(int[] E, int left, int right) {
2    if (left < right) {
3      int i = partition(E, left, right);
4      quickSort(E, left, i - 1); // sort left
5      quickSort(E, i + 1, right); // sort right
6    }
7  }
8  int partition(int[] E, int left, int right) {
9    int ppos = right, pivot = E[ppos];
10   while (true) {
11     while (left < right && E[left] < pivot) left++;
12     while (left < right && E[right] >= pivot) right--;
13     if (left >= right) break;
14     swap(E[left], E[right]);
15   }
16   swap(E[left], E[ppos]);
17   return left;
18 }
```

Sortieren Sie das folgende Array mithilfe von Quicksort. Geben Sie dazu das Array nach jeder Partition-Operation an und markieren Sie das jeweils verwendete Pivot-Element. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

4	7	5	8	1	5	2	3

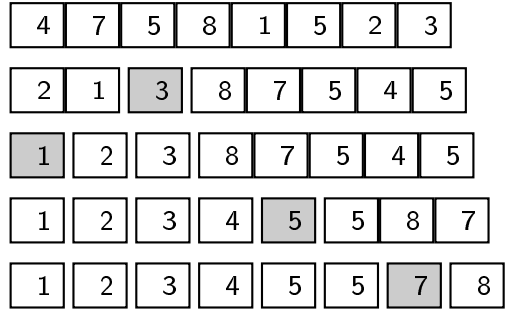
Quicksort – Aufgabe

```
1  void quickSort(int[] E, int left, int right) {
2    if (left < right) {
3      int i = partition(E, left, right);
4      quickSort(E, left, i - 1); // sort left
5      quickSort(E, i + 1, right); // sort right
6    }
7  }
8  int partition(int[] E, int left, int right) {
9    int ppos = right, pivot = E[ppos];
10   while (true) {
11     while (left < right && E[left] < pivot) left++;
12     while (left < right && E[right] >= pivot) right--;
13     if (left >= right) break;
14     swap(E[left], E[right]);
15   }
16   swap(E[left], E[ppos]);
17   return left;
18 }
```



Quicksort – Aufgabe

```
1  void quickSort(int[] E, int left, int right) {
2    if (left < right) {
3      int i = partition(E, left, right);
4      quickSort(E, left, i - 1); // sort left
5      quickSort(E, i + 1, right); // sort right
6    }
7  }
8  int partition(int[] E, int left, int right) {
9    int ppos = right, pivot = E[ppos];
10   while (true) {
11     while (left < right && E[left] < pivot) left++;
12     while (left < right && E[right] >= pivot) right--;
13     if (left >= right) break;
14     swap(E[left], E[right]);
15   }
16   swap(E[left], E[ppos]);
17   return left;
18 }
```



Quicksort – Eigenschaften

- Vorgehen
  - Tausche zunächst Einträge in den richtigen Bereich
  - Sortiere anschließend beide Bereiche rekursiv
- nicht in-place (Rekursion benötigt Speicher und Rekursionstiefe abhängig von der Eingabe)
- nicht stabil (es gibt stabile Varianten)
- $W(n) \in \Theta(n^2)$

Quicksort – Variante 1

Beliebige aber feste Wahl der Position des Pivot

- In der Vorlesung: Pivot ist immer an der Position `right`
- Beispiele für Alternativen:
  - Pivot ist immer an der Position `left`
  - Pivot ist immer an der Position $\frac{\texttt{left}+\texttt{right}}{2}$
  - ...
- Worst-Case weiterhin konstruierbar
- Daher: $W(n) \in \Theta(n^2)$

Quicksort – Variante 2

Zufällige Wahl der Position des Pivot

- In der Vorlesung: Pivot ist immer an der Position `right`
- Alternative: Zufällige Wahl der Position des Pivot aus dem Intervall $[\texttt{left}, \texttt{right}]$
- Worst-Case nicht mehr so leicht konstruierbar
- Daher: $W(n) \in \Theta(n \cdot \log(n))$
- Beispiel für sinnvolle Anwendung eines probabilistischen Algorithmus

Bubblesort – Aufgabe

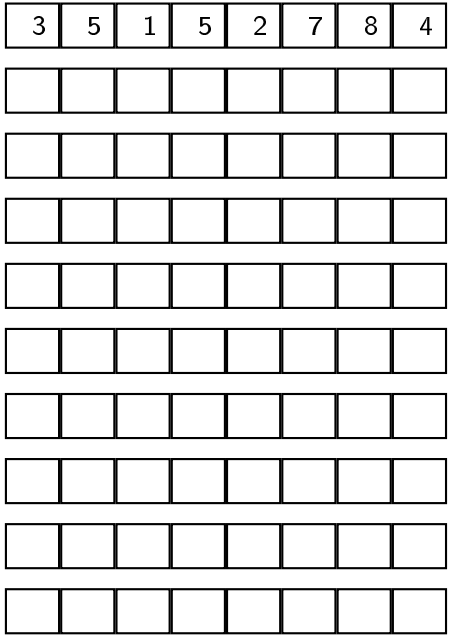
```
1  void bubbleSort(int[] E) {
2    int n = E.length;
3    while (n > 1) {
4      int j = 1;
5      for (int i = 0; i < n - 1; i++) {
6        if (E[i] > E[i + 1]) {
7          swap(E[i], E[i + 1]);
8          j = i + 1;
9        }
10     }
11     n = j;
12   }
13 }
```

Sortieren Sie das folgende Array mithilfe von Bubblesort. Geben Sie dazu das Array nach jeder Swap-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

3	5	1	5	2	7	8	4

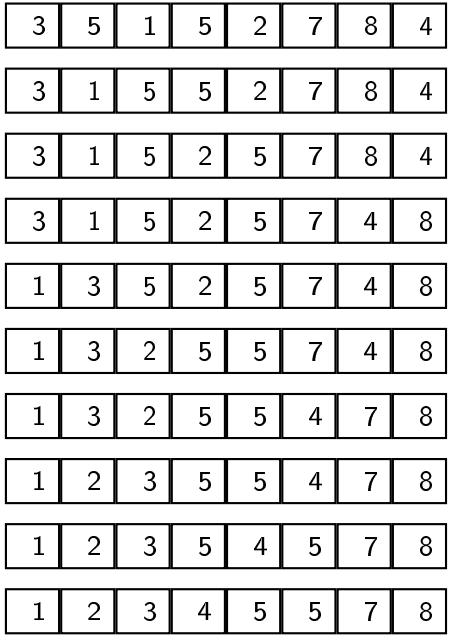
Bubblesort – Aufgabe

```
1  void bubbleSort(int[] E) {
2    int n = E.length;
3    while (n > 1) {
4      int j = 1;
5      for (int i = 0; i < n - 1; i++) {
6        if (E[i] > E[i + 1]) {
7          swap(E[i], E[i + 1]);
8          j = i + 1;
9        }
10     }
11     n = j;
12   }
13 }
```



Bubblesort – Aufgabe

```
1  void bubbleSort(int[] E) {
2    int n = E.length;
3    while (n > 1) {
4      int j = 1;
5      for (int i = 0; i < n - 1; i++) {
6        if (E[i] > E[i + 1]) {
7          swap(E[i], E[i + 1]);
8          j = i + 1;
9        }
10     }
11     n = j;
12   }
13 }
```

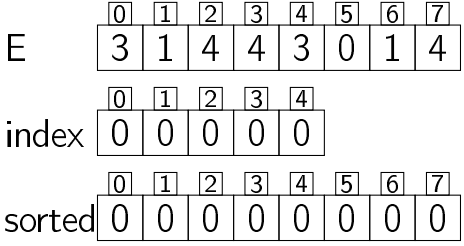


Bubblesort – Eigenschaften

- Vorgehen
  - Jeder Durchlauf der äußeren Schleife spühlt das verbleibende größte Element nach oben
  - Tauscht nur benachbarte Elemente
- in-place
- stabil (bei Gleichheit wird nicht getauscht)
- $W(n) \in \Theta(n^2)$

Countingsort – Beispiel

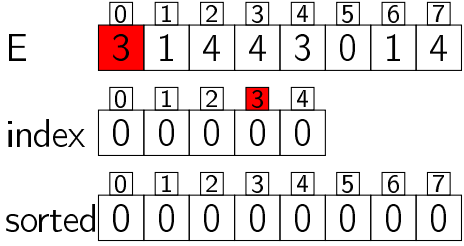
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Speicher initialisieren

Countingsort – Beispiel

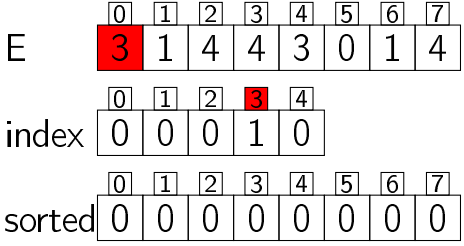
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

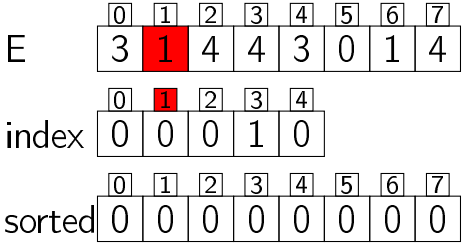
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

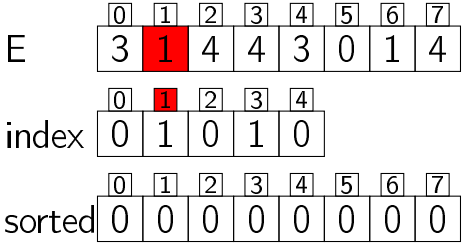
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

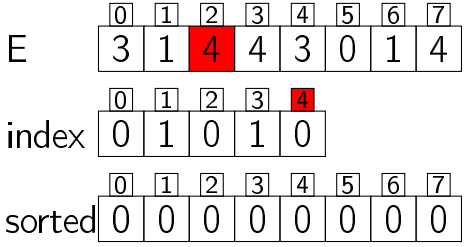
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

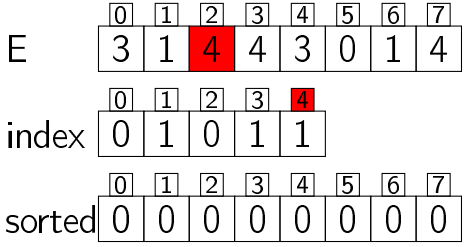
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

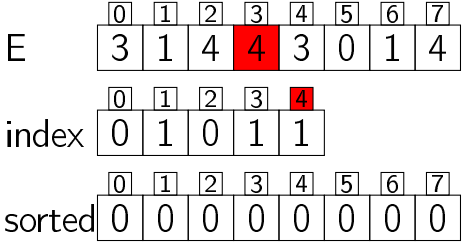
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

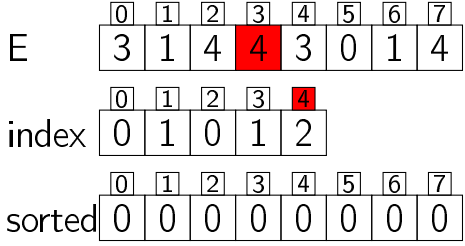
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

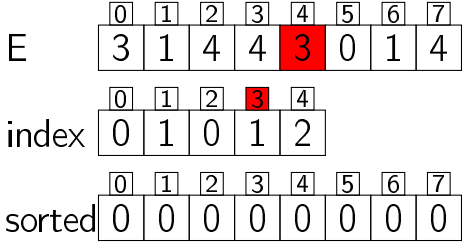
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

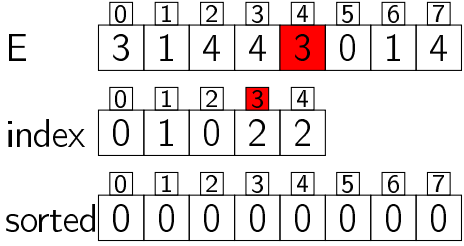
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

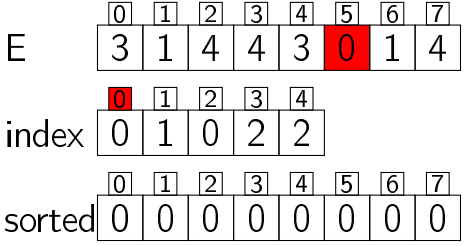
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

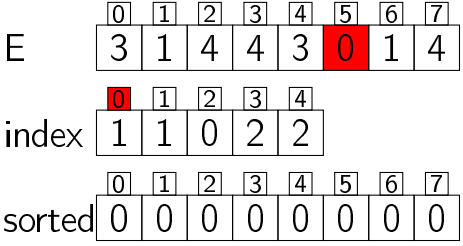
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

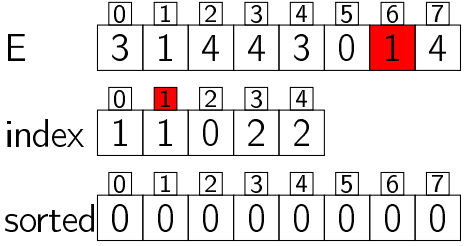
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

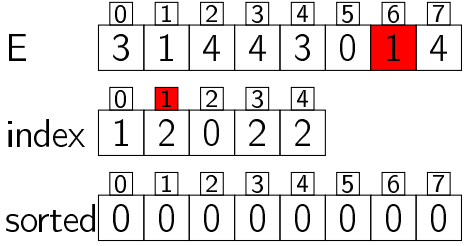
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

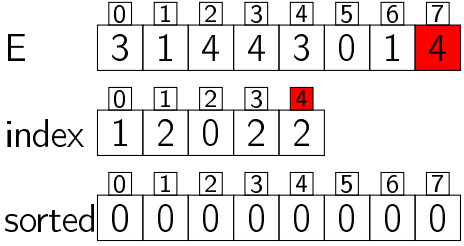
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

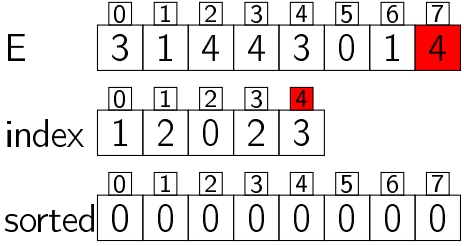
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

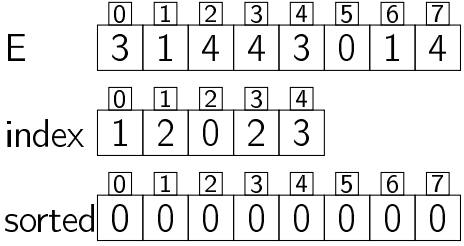
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl zählen

Countingsort – Beispiel

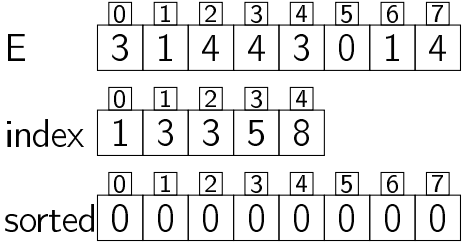
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl summieren

Countingsort – Beispiel

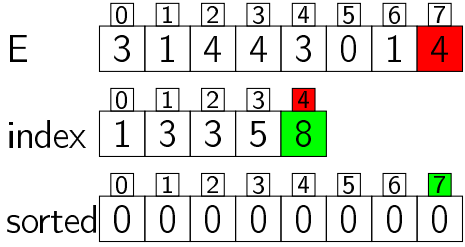
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Anzahl summieren

Countingsort – Beispiel

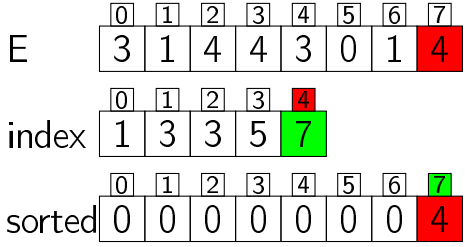
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

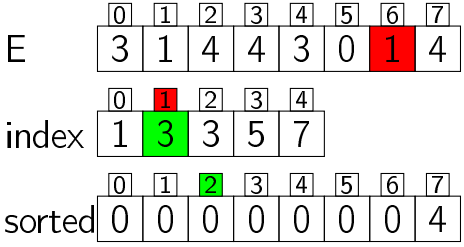
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

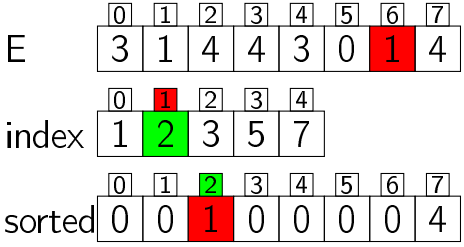
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

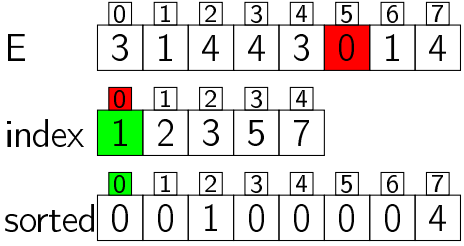
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

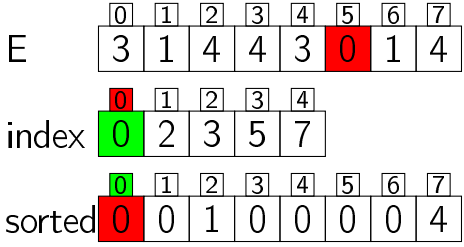
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

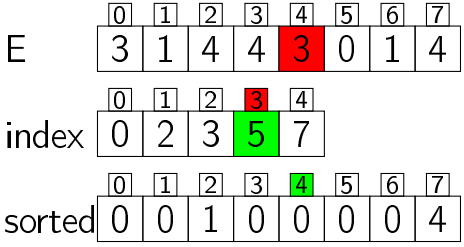
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

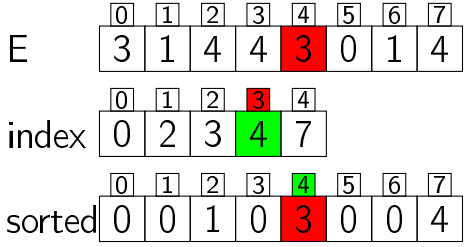
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

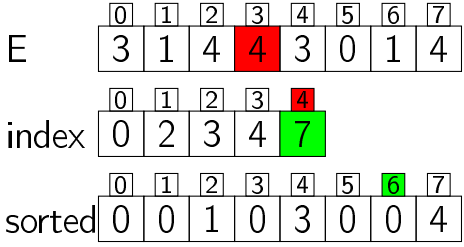
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

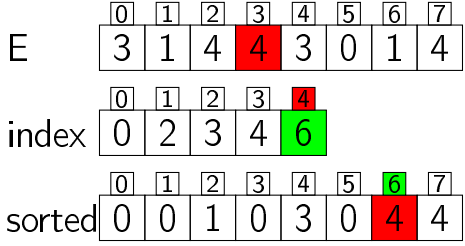
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

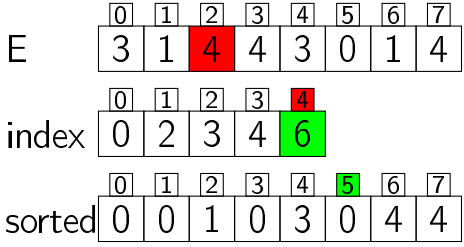
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

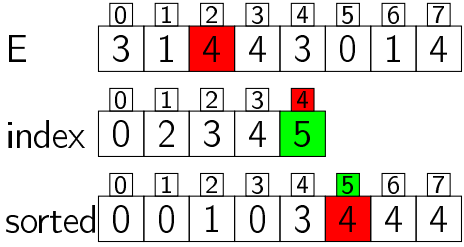
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

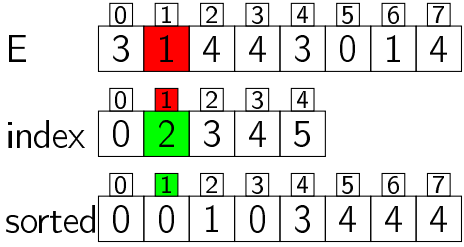
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

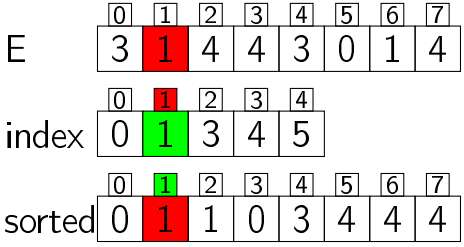
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

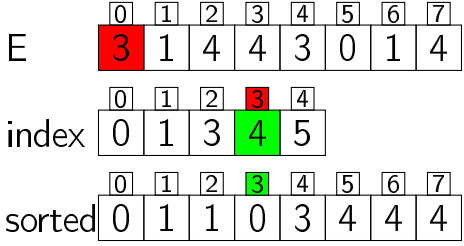
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

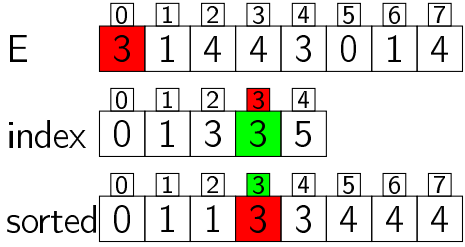
```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Beispiel

```
1  int[] countingSort(int[] E, int nrOfKeys) {
2    int[] index = new int[nrOfKeys];
3    int[] sorted = new int[E.length];
4    for (int j = 0; j < E.length; j++) {
5      index[E[j]]++;
6    }
7    for (int i = 1; i < index.length; i++) {
8      index[i] += index[i - 1];
9    }
10   for (int j = E.length - 1; j >= 0; j--) {
11     sorted[index[E[j]] - 1] = E[j];
12     index[E[j]]--;
13   }
14   return sorted;
15 }
```



- Einträge kopieren

Countingsort – Eigenschaften

- Nur bei bekanntem und endlichem Wertebereich anwendbar (k verschiedene Schlüssel)
- Vorgehen
  - Zähle wie oft jeder Schlüssel vorkommt
  - Rechne Häufigkeiten in Zielpositionen um
  - Kopiere Einträge an Zielpositionen (Rückwärts für Stabilität)
- nicht in-place (Array `sorted` benötigt $O(n)$ zusätzlichen Speicher)
- stabil
- $W(n) \in \Theta(n + k)$

Datenstrukturen und Algorithmen
6. Globalübung
Teil 2: Binäre Suchbäume

Tobias Winkler

20. Mai 2020

Was sind BST und wozu sind sie zu gebrauchen?

Terminologie: *Binärer Suchbaum = Binary Search Tree = BST*

General-purpose Datenstruktur zum Verwalten von Zahlen(multi)mengen

- Suchen
- Sortiert ausgeben
- Minimum/Maximum bestimmen
- Vor-/Nachgänger bestimmen
- Einfügen/Löschen

Unsere Vision: Alle Operationen in $o(n)$ Zeit (außer sortiertes Ausgeben), Speicherbedarf Datenstruktur $O(n)$.

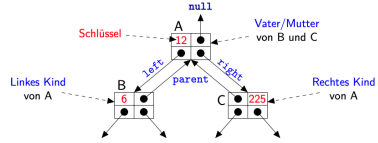
BST vs Heap

Gemeinsamkeiten:

- Beides sind Binärbäume
- Die Knoten tragen Schlüssel mit Werten aus $\mathbb{Z}$ (zumindest in dieser VL)

Unterschiede:

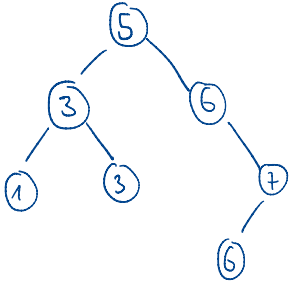
- Implementierung
  - BST: Mittels *Array*
  - Heap: Mittels *Pointer-Struktur*:



- Schlüsselordnung: Schlüssel jedes Knotens ist ...
  - BST: mindestens so groß wie alle Schlüssel im linken Teilbaum, höchstens so groß wie alle im rechten.
  - Heap: mindestens so groß wie Schlüssel der (max. 2) Kindknoten.

Sortieren und die Inorder-Traversierung

Lemma: Inorder-Traversierung eines BST liefert Schlüssel in aufsteigend sortierter Reihenfolge.



Sortieren und die Inorder-Traversierung

Lemma: Inorder-Traversierung eines BST liefert Schlüssel in aufsteigend sortierter Reihenfolge.

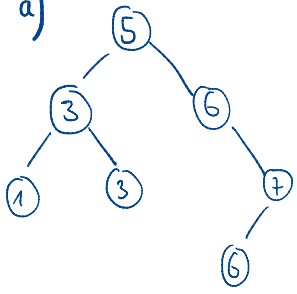
Beweis: Per Induktion nach der Höhe h.

- IA: $h = 0$. Die Aussage gilt für den BST, der nur aus der Wurzel besteht.
- IV: Gelte die Aussage für alle $h' \leq h$ für beliebiges aber festes $h \in \mathbb{N}$.
- IS: $h \to h + 1$. Betrachte einen BST der Höhe $h + 1$.
  - Dann haben der linke und der rechte Teilbaum unter der Wurzel höchstens Höhe h, falls diese Teilbäume existieren.
  - Nach IV liefert Inorder-Traversierung dieser Teilbäume deren Schlüssel in aufsteigender Reihenfolge.
  - Die BST-Eigenschaft zusammen mit der Inorder-Reihenfolge (linker Teilbaum - Wurzel - rechter Teilbaum) liefert die Aussage.

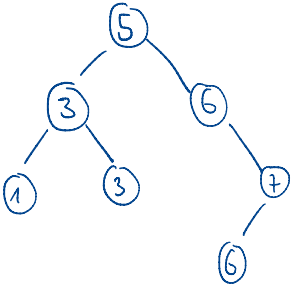
Die Suche nach dem Nachfolger

Gegeben sei ein BST, der einen Knoten `node` enthält.
Wie finden wir den Nachfolger von `node` in der Inorder-Traversierung?

2 Fälle a)



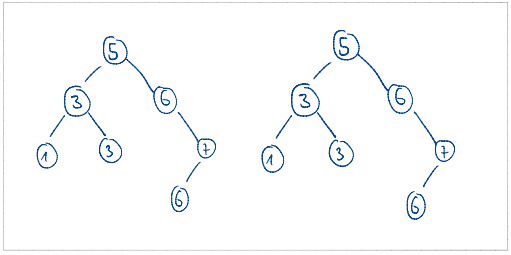
b)



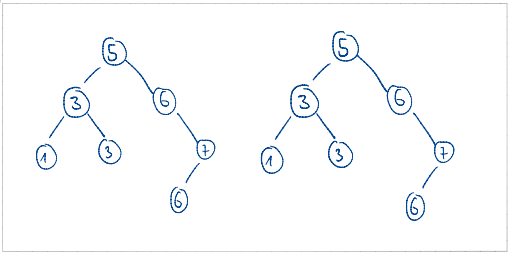
Löschen im BST

3 Fälle

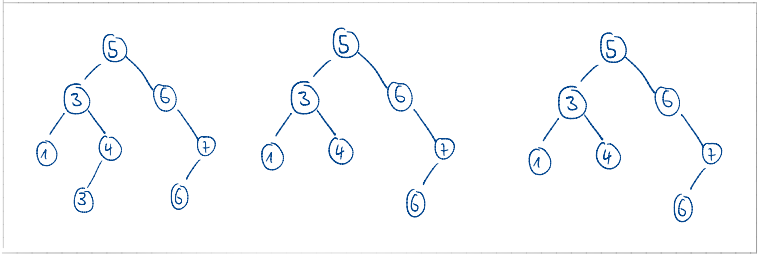
a)



b)



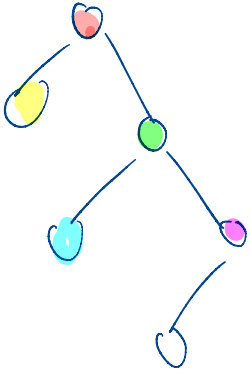
c)



Wenn es doch so einfach wäre ...

Die Operationen sind *linear in der Höhe h* des BSTs. Ist $h \in o(n)$?

Rotieren

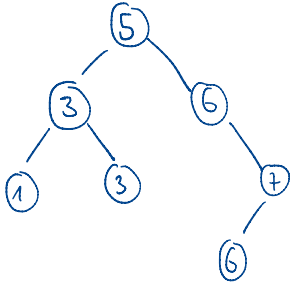


AVL Bäume

AVL (Adelson-Velski-Landis) Bäume sind BSTs mit einer zusätzlichen Eigenschaft. Definiere für jeden Knoten `node`:

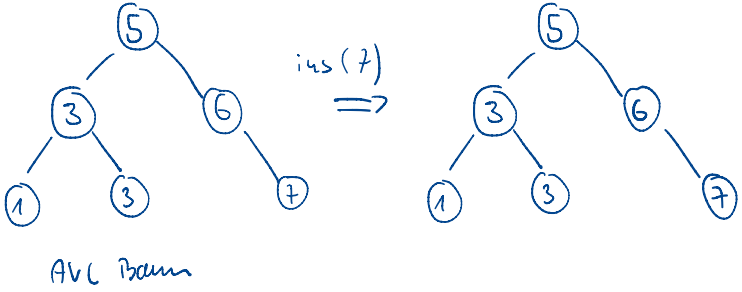
$$\mathit{balance}(\texttt{node}) := \mathit{hoehe}(\mathit{rechterTeilbaum}) - \mathit{hoehe}(\mathit{linkerTeilbaum})$$

AVL Baum: $|\mathit{balance}(\texttt{node})| \leq 1$ für alle Knoten `node`.

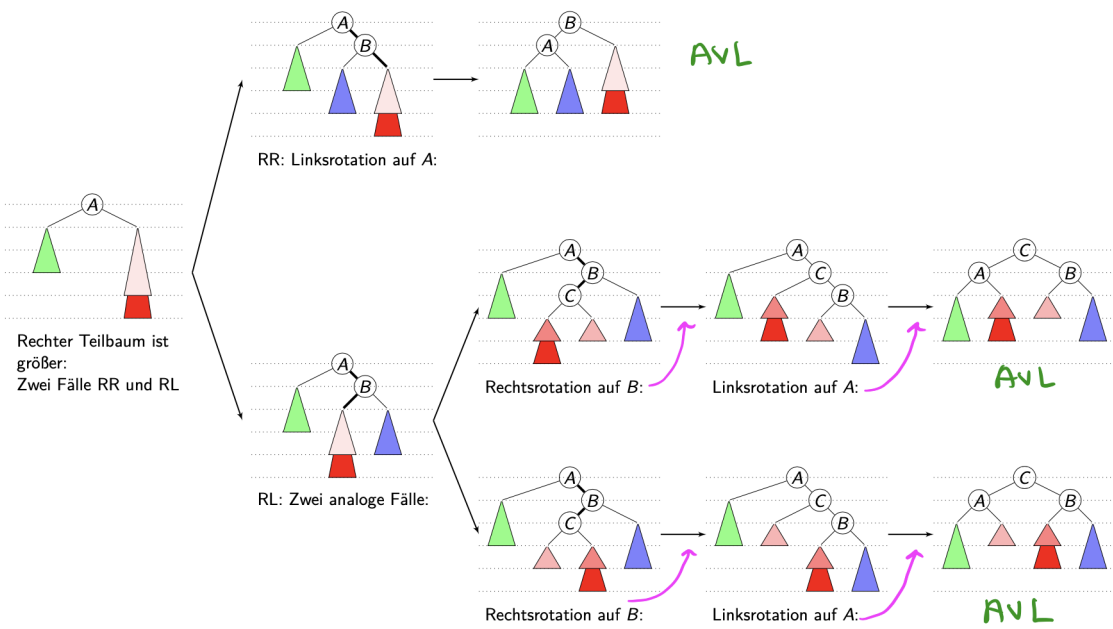


Endlich am Ziel?

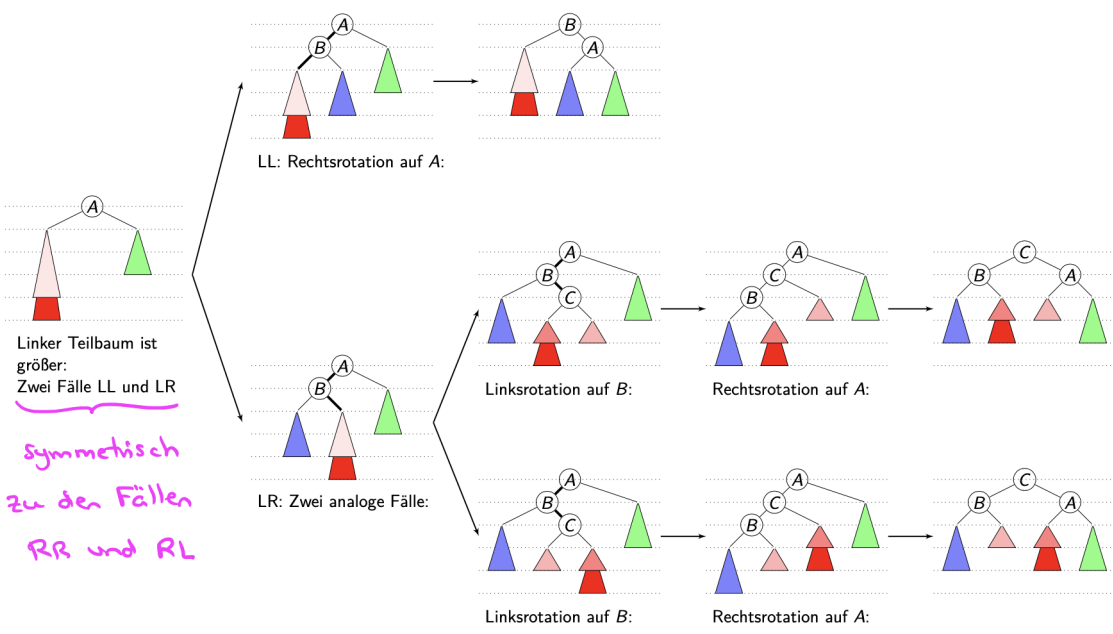
Satz: Ein AVL Baum mit n Knoten hat Höhe $h \in O(\log(n))$.
Ist dadurch unsere Vision realisiert?



Balancierung nach Einfügen/Löschen



Balancierung nach Einfügen/Löschen



Balancierung nach Einfügen/Löschen