Script generated by TTT

Title: groh: profile1 (08.07.2016)

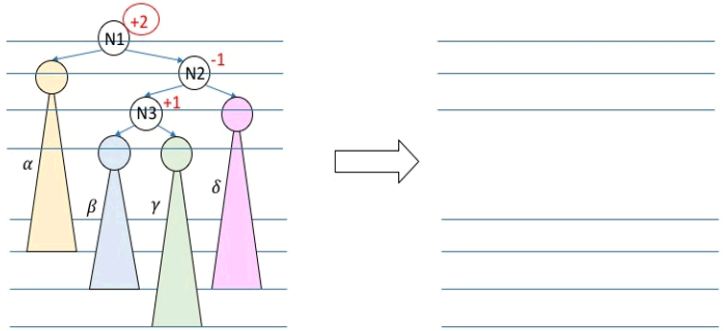
Date: Fri Jul 08 13:04:42 CEST 2016

Duration: 82:12 min

Pages: 51

Aufgabe

AVL Bäume:
In einem AVL Baum ist durch Einfügen in den Teilbaum γ folgender **ungünstiger Zustand entstanden. Korrigieren** Sie ihn durch eine **Doppelrotation nach links! Geben** Sie auch die neuen **Δh Angaben für** A und B **an!**



SelectionSort

- Strategie: Wähle kleinstes Element in verbliebener Eingabesequenz und verschiebe es ans Ende der Ausgabesequenz

```
public void selectionSort(int[] a){
    int n = a.length;
    for(int i=0; i<n; i++){
        //move min({a[i],a[i+1],...,a[n-1]}) to position i:
        for(int j=i; j<n; j++){
            if(a[i] > a[j]) //if current element a[j] smaller
                            //than current min-candidate a[i]
                swap(a[i],a[j]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
(mit besserer Minimumstrategie geht auch $O(n \log n)$)

SelectionSort

- Strategie: Wähle kleinstes Element in verbliebener Eingabesequenz und verschiebe es ans Ende der Ausgabesequenz

```
public void selectionSort(int[] a){
    int n = a.length;
    for(int i=0; i<n; i++){
        //move min({a[i],a[i+1],...,a[n-1]}) to position i:
        for(int j=i; j<n; j++){
            if(a[i] > a[j]) //if current element a[j] smaller
                            //than current min-candidate a[i]
                swap(a[i],a[j]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
(mit besserer Minimumstrategie geht auch $O(n \log n)$)

SelectionSort

- Strategie: Wähle kleinstes Element in verbliebener Eingabesequenz und verschiebe es ans Ende der Ausgabesequenz

```
public void selectionSort(int[] a){
    int n = a.length;
    for(int i=0; i<n; i++){
        //move min({a[i],a[i+1],...,a[n-1]}) to position i:
        for(int j=i; j<n; j++){
            if(a[i] > a[j]) //if current element a[j] smaller
                            //than current min-candidate a[i]
                swap(a[i],a[j]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
  (mit besserer Minimumstrategie geht auch $O(n \log n)$)

SelectionSort

- Strategie: Wähle kleinstes Element in verbliebener Eingabesequenz und verschiebe es ans Ende der Ausgabesequenz

```
public void selectionSort(int[] a){
    int n = a.length;
    for(int i=0; i<n; i++){
        //move min({a[i],a[i+1],...,a[n-1]}) to position i:
        for(int j=i; j<n; j++){
            if(a[i] > a[j]) //if current element a[j] smaller
                            //than current min-candidate a[i]
                swap(a[i],a[j]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
  (mit besserer Minimumstrategie geht auch $O(n \log n)$)

SelectionSort

```
public void selectionSort(int[] a){
    int n = a.length;
    for(int i=0; i<n; i++){
        //move min({a[i],a[i+1],...,a[n-1]}) to position i:
        for(int j=i; j<n; j++){
            if(a[i] > a[j]) //if current element a[j] smaller
                            //than current min-candidate a[i]
                swap(a[i],a[j]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
1	10	19	5	14	3
1	5	19	10	14	3
1	**3**	19	10	14	5
1	3	10	19	14	5

1	3	**5**	19	14	10
1	3	5	14	19	10
1	3	5	**10**	19	14
1	3	5	10	**14**	19
1	3	5	10	14	**19**

[3]

SelectionSort

```
public void selectionSort(int[] a){
    int n = a.length;
    for(int i=0; i<n; i++){
        //move min({a[i],a[i+1],...,a[n-1]}) to position i:
        for(int j=i; j<n; j++){
            if(a[i] > a[j]) //if current element a[j] smaller
                            //than current min-candidate a[i]
                swap(a[i],a[j]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
1	10	19	5	14	3
1	5	19	10	14	3
1	**3**	19	10	14	5
1	3	10	19	14	5

1	3	**5**	19	14	10
1	3	5	14	19	10
1	3	5	**10**	19	14
1	3	5	10	**14**	19
1	3	5	10	14	**19**

[3]

SelectionSort

```
public void selectionSort(int[] a){
    int n = a.length;
    for(int i=0; i<n; i++){
         //move min({a[i],a[i+1],...,a[n-1]}) to position i:
        for(int j=i; j<n; j++){
             if(a[i] > a[j]) //if current element a[j] smaller
                             //than current min-candidate a[i]
                 swap(a[i],a[j]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
1	10	19	5	14	3
1	5	19	10	14	3
1	**3**	19	10	14	5
1	3	10	19	14	5

1	3	**5**	19	14	10
1	3	5	14	19	10
1	3	5	**10**	19	14
1	3	5	10	**14**	19
1	3	5	10	14	**19**

[3]

InsertionSort

- Strategie: Füge nächstes Element aus Eingabesequenz an die richtige Stelle der Ausgabesequenz ein

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //((in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
  (mit besserer Einfügestrategie geht auch $O(n \log^2 n)$)

InsertionSort

- Strategie: Füge nächstes Element aus Eingabesequenz an die richtige Stelle der Ausgabesequenz ein

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //((in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
  (mit besserer Einfügestrategie geht auch $O(n \log^2 n)$)

InsertionSort

- Strategie: Füge nächstes Element aus Eingabesequenz an die richtige Stelle der Ausgabesequenz ein

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //((in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
  (mit besserer Einfügestrategie geht auch $O(n \log^2 n)$)

InsertionSort

- Strategie: Füge nächstes Element aus Eingabesequenz an die richtige Stelle der Ausgabesequenz ein

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //(in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

- Laufzeit: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
  (mit besserer Einfügestrategie geht auch $O(n \log^2 n)$)

InsertionSort

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //(in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
5	10	19	1	14	3
5	**10**	19	1	14	3
5	10	**19**	1	14	3
5	10	1	19	14	3

5	1	10	19	14	3
1	5	10	19	14	3
1	5	10	**14**	19	3
1	...	←	...	3	19
1	**3**	5	10	14	19

[3]

InsertionSort

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //(in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
5	10	19	1	14	3
5	**10**	19	1	14	3
5	10	**19**	1	14	3
5	10	1	19	14	3

5	1	10	19	14	3
1	5	10	19	14	3
1	5	10	**14**	19	3
1	...	←	...	3	19
1	**3**	5	10	14	19

[3]

InsertionSort

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //(in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
5	10	19	1	14	3
5	**10**	19	1	14	3
5	10	**19**	1	14	3
5	10	1	19	14	3

5	1	10	19	14	3
1	5	10	19	14	3
1	5	10	**14**	19	3
1	...	←	...	3	19
1	**3**	5	10	14	19

[3]

Aufgabe

Gegeben sei ein **Durchlauf** des Sortieralgorithmus **InsertionSort** auf dem Array `int[] a={12, 3, 5, 23, 7, 19, 8, 4, 17, 99}`.
Ergänzen Sie die fehlenden **drei Zeilen sinnvoll**!

```
12  | 3, 5, 23, 7, 19, 8, 4, 17, 99
3, 12  | 5, 23, 7, 19, 8, 4, 17, 99
3, 5, 12  | 23, 7, 19, 8, 4, 17, 99
3, 5, 12, 23  | 7, 19, 8, 4, 17, 99
                                      |
3, 5, 7, 12, 19, 23  | 8, 4, 17, 99
                                      |
                                      |
3, 4, 5, 7, 8, 12, 17, 19, 23  | 99
3, 4, 5, 7, 8, 12, 17, 19, 23, 99  |
```


Aufgabe

Gegeben sei ein **Durchlauf** des Sortieralgorithmus **InsertionSort** auf dem Array `int[] a={12, 3, 5, 23, 7, 19, 8, 4, 17, 99}`.
Ergänzen Sie die fehlenden **drei Zeilen sinnvoll**!

```
12  | 3, 5, 23, 7, 19, 8, 4, 17, 99
3, 12  | 5, 23, 7, 19, 8, 4, 17, 99
3, 5, 12  | 23, 7, 19, 8, 4, 17, 99
3, 5, 12, 23  | 7, 19, 8, 4, 17, 99
                                      |
3, 5, 7, 12, 19, 23  | 8, 4, 17, 99
                                      |
                                      |
3, 4, 5, 7, 8, 12, 17, 19, 23  | 99
3, 4, 5, 7, 8, 12, 17, 19, 23, 99  |
```


Aufgabe

```
12  | 3, 5, 23, 7, 19, 8, 4, 17, 99
3, 12  | 5, 23, 7, 19, 8, 4, 17, 99
3, 5, 12  | 23, 7, 19, 8, 4, 17, 99
3, 5, 12, 23  | 7, 19, 8, 4, 17, 99
3, 5, 7, 12, 23  | 19, 8, 4, 17, 99
3, 5, 7, 12, 19, 23  | 8, 4, 17, 99
3, 5, 7, 8, 12, 19, 23  | 4, 17, 99
3, 4, 5, 7, 8, 12, 19, 23  | 17, 99
3, 4, 5, 7, 8, 12, 17, 19, 23  | 99
3, 4, 5, 7, 8, 12, 17, 19, 23, 99  |
```


Aufgabe

```
12  | 3, 5, 23, 7, 19, 8, 4, 17, 99
3, 12  | 5, 23, 7, 19, 8, 4, 17, 99
3, 5, 12  | 23, 7, 19, 8, 4, 17, 99
3, 5, 12, 23  | 7, 19, 8, 4, 17, 99
3, 5, 7, 12, 23  | 19, 8, 4, 17, 99
3, 5, 7, 12, 19, 23  | 8, 4, 17, 99
3, 5, 7, 8, 12, 19, 23  | 4, 17, 99
3, 4, 5, 7, 8, 12, 19, 23  | 17, 99
3, 4, 5, 7, 8, 12, 17, 19, 23  | 99
3, 4, 5, 7, 8, 12, 17, 19, 23, 99  |
```

InsertionSort

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //((in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
5	10	19	1	14	3
5	**10**	19	1	14	3
5	10	**19**	1	14	3
5	10	1	19	14	3

5	1	10	19	14	3
1	5	10	19	14	3
1	5	10	**14**	19	3
1	...	←	...	3	19
1	**3**	5	10	14	19

[3]

InsertionSort

```
public void insertionSort(int[] a){
    int n = a.length;
    for(int i=1; i<n; i++){
        //move a[i] to the right position:
        for(int j=i-1; j>=0; j--){
            if(a[j] > a[j+1]) //if current element a[j+1]
                              //((in the beginning this is a[i])
                              //is smaller as left neighbor a[j]
                swap(a[j],a[j+1]);  //swap them
        }
    }
}
```

Beispiel

5	10	19	1	14	3
5	10	19	1	14	3
5	**10**	19	1	14	3
5	10	**19**	1	14	3
5	10	1	19	14	3

5	1	10	19	14	3
1	5	10	19	14	3
1	5	10	**14**	19	3
1	...	←	...	3	19
1	**3**	5	10	14	19

[3]

174

MergeSort

- InsertionSort, SelectionSort, (Bubblesort…): Einfach zu implementieren aber quadratische Laufzeit.
  Geht das besser? Antwort: ja, in O(n log n): Mergesort, Quicksort etc.)

- Mergesort: Rekursiver Ansatz: Teile Sequenz in zwei Hälften, sortiere jede Hälfte und merge die Hälften anschließend zusammen:

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
…
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //then merge both sorted parts:
…
}
```


MergeSort

- InsertionSort, SelectionSort, (Bubblesort…): Einfach zu implementieren aber quadratische Laufzeit.
  Geht das besser? Antwort: ja, in O(n log n): Mergesort, Quicksort etc.)

- Mergesort: Rekursiver Ansatz: Teile Sequenz in zwei Hälften, sortiere jede Hälfte und merge die Hälften anschließend zusammen:

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
…
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //then merge both sorted parts:
…
}
```

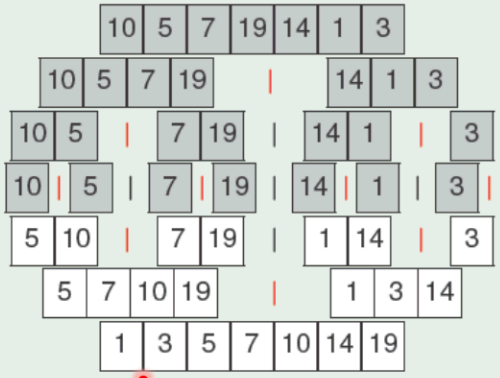
177

MergeSort

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
    if (l == r) return; // we have only one element, nothing to sort;
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //merge both sorted parts:
    int j = l;  //running variable for iterating left part
    int k = m + 1; //running variable for iterating right part
    int[] b = new int[r-l+1]; //intermediate storage for merging result
    for(int i=0; i<r-l+1; i++){ //perform merge:
        if(j > m){ //left part is used up --> use elements of right part
            b[i] = a[k];
            k++;
        } else if(k>r){ //right part is used up --> use elements of left part
            b[i] = a[j];
            j++;
        } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
            b[i] = a[j];
            j++;
        } else { //element from the right part is smaller --> use it
            b[i] = a[k];
            k++;
        }
    }
    for(int i=0; i<r-l+1; i++) //copy b back into respective parts of a:
        a[l+i] = b[i];
}
```

MergeSort

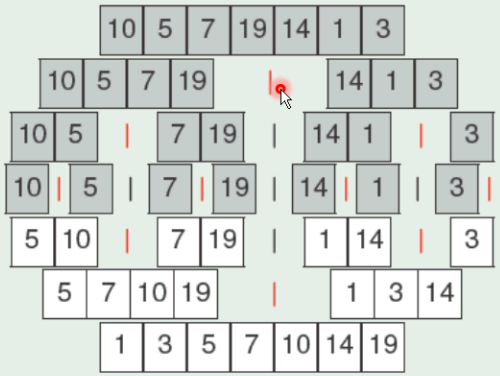
Beispiel



[3]

MergeSort

Beispiel

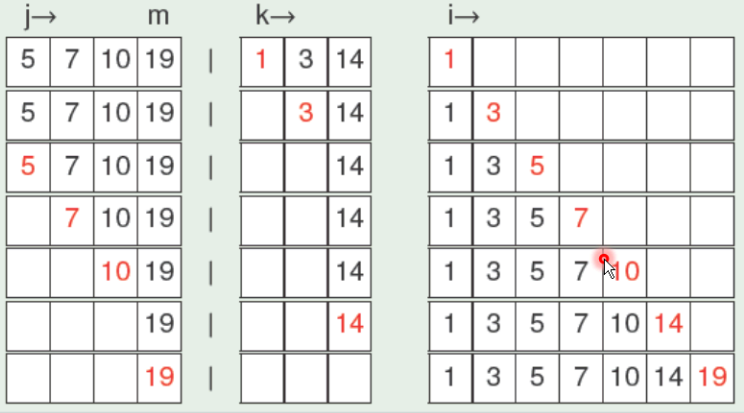


[3]

Merge Step

```
for(int i=0; i<r-l+1; i++){ //perform merge:
    if(j > m){ //left part is used up --> use elements of right part
        b[i] = a[k];
        k++;
    } else if(k>r){ //right part is used up --> use elements of left part
        b[i] = a[j];
        j++;
    } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
        b[i] = a[j];
        j++;
    } else { //element from the right part is smaller --> use it
        b[i] = a[k];
        k++;
    }
```

Beispiel für merge



[3]

Merge Step

```
for(int i=0; i<r-l+1; i++){ //perform merge:
        if(j > m){ //left part is used up --> use elements of right part
            b[i] = a[k];
            k++;
        } else if(k>r){ //right part is used up --> use elements of left part
            b[i] = a[j];
            j++;
        } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
            b[i] = a[j];
            j++;
        } else { //element from the right part is smaller --> use it
            b[i] = a[k];
            k++;
        }
```

Beispiel für merge

j→			m		k→			i→						
5	7	10	19	|	1	3	14	1						
5	7	10	19	|		3	14	1	3					
5	7	10	19	|			14	1	3	5				
	7	10	19	|			14	1	3	5	7			
		10	19	|			14	1	3	5	7	10		
			19	|			14	1	3	5	7	10	14	
			19	|				1	3	5	7	10	14	19

[3]

MergeSort

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
    if (l == r) return; // we have only one element, nothing to sort;
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //merge both sorted parts:
    int j = l;  //running variable for iterating left part
    int k = m + 1; //running variable for iterating right part
    int[] b = new int[r-l+1]; //intermediate storage for merging result
    for(int i=0; i<r-l+1; i++){ //perform merge:
        if(j > m){ //left part is used up --> use elements of right part
            b[i] = a[k];
            k++;
        } else if(k>r){ //right part is used up --> use elements of left part
            b[i] = a[j];
            j++;
        } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
            b[i] = a[j];
            j++;
        } else { //element from the right part is smaller --> use it
            b[i] = a[k];
            k++;
        }
    }
    for(int i=0; i<r-l+1; i++) //copy b back into respective parts of a:
        a[l+i] = b[i];
}
```


MergeSort

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
    if (l == r) return; // we have only one element, nothing to sort;
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //merge both sorted parts:
    int j = l;  //running variable for iterating left part
    int k = m + 1; //running variable for iterating right part
    int[] b = new int[r-l+1]; //intermediate storage for merging result
    for(int i=0; i<r-l+1; i++){ //perform merge:
        if(j > m){ //left part is used up --> use elements of right part
            b[i] = a[k];
            k++;
        } else if(k>r){ //right part is used up --> use elements of left part
            b[i] = a[j];
            j++;
        } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
            b[i] = a[j];
            j++;
        } else { //element from the right part is smaller --> use it
            b[i] = a[k];
            k++;
        }
    }
    for(int i=0; i<r-l+1; i++) //copy b back into respective parts of a:
        a[l+i] = b[i];
}
```

MergeSort

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
    if (l == r) return; // we have only one element, nothing to sort;
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //merge both sorted parts:
    int j = l;  //running variable for iterating left part
    int k = m + 1; //running variable for iterating right part
    int[] b = new int[r-l+1]; //intermediate storage for merging result
    for(int i=0; i<r-l+1; i++){ //perform merge:
        if(j > m){ //left part is used up --> use elements of right part
            b[i] = a[k];
            k++;
        } else if(k>r){ //right part is used up --> use elements of left part
            b[i] = a[j];
            j++;
        } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
            b[i] = a[j];
            j++;
        } else { //element from the right part is smaller --> use it
            b[i] = a[k];
            k++;
        }
    }
    for(int i=0; i<r-l+1; i++) //copy b back into respective parts of a:
        a[l+i] = b[i];
}
```

MergeSort

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
    if (l == r) return; // we have only one element, nothing to sort;
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //merge both sorted parts:
    int j = l;  //running variable for iterating left part
    int k = m + 1; //running variable for iterating right part
    int[] b = new int[r-l+1]; //intermediate storage for merging result
    for(int i=0; i<r-l+1; i++){ //perform merge:
        if(j > m){ //left part is used up --> use elements of right part
            b[i] = a[k];
            k++;
        } else if(k>r){ //right part is used up --> use elements of left part
            b[i] = a[j];
            j++;
        } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
            b[i] = a[j];
            j++;
        } else { //element from the right part is smaller --> use it
            b[i] = a[k];
            k++;
        }
    }
    for(int i=0; i<r-l+1; i++) //copy b back into respective parts of a:
        a[l+i] = b[i];
}
```

Merge Step

```
for(int i=0; i<r-l+1; i++){ //perform merge:
      if(j > m){ //left part is used up --> use elements of right part
          b[i] = a[k];
          k++;
      } else if(k>r){ //right part is used up --> use elements of left part
          b[i] = a[j];
          j++;
      } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
          b[i] = a[j];
          j++;
      } else { //element from the right part is smaller --> use it
          b[i] = a[k];
          k++;
      }
```

Beispiel für merge

j→			m		k→				i→						
5	7	10	19	|	1	3	14		1						
5	7	10	19	|		3	14		1	3					
5	7	10	19	|			14		1	3	5				
	7	10	19	|			14		1	3	5	7			
		10	19	|			14		1	3	5	7	10		
			19	|			14		1	3	5	7	10	14	
			19	|					1	3	5	7	10	14	19

[3]

Merge Step

```
for(int i=0; i<r-l+1; i++){ //perform merge:
      if(j > m){ //left part is used up --> use elements of right part
          b[i] = a[k];
          k++;
      } else if(k>r){ //right part is used up --> use elements of left part
          b[i] = a[j];
          j++;
      } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
          b[i] = a[j];
          j++;
      } else { //element from the right part is smaller --> use it
          b[i] = a[k];
          k++;
      }
```

Beispiel für merge

j→			m		k→				i→						
5	7	10	19	|	1	3	14		1						
5	7	10	19	|		3	14		1	3					
5	7	10	19	|			14		1	3	5				
	7	10	19	|			14		1	3	5	7			
		10	19	|			14		1	3	5	7	10		
			19	|			14		1	3	5	7	10	14	
			19	|					1	3	5	7	10	14	19

[3]

180

Merge Step

```
for(int i=0; i<r-l+1; i++){ //perform merge:
      if(j > m){ //left part is used up --> use elements of right part
          b[i] = a[k];
          k++;
      } else if(k>r){ //right part is used up --> use elements of left part
          b[i] = a[j];
          j++;
      } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
          b[i] = a[j];
          j++;
      } else { //element from the right part is smaller --> use it
          b[i] = a[k];
          k++;
      }
```

Beispiel für merge

j→			m		k→				i→						
5	7	10	19	|	1	3	14		1						
5	7	10	19	|		3	14		1	3					
5	7	10	19	|			14		1	3	5				
	7	10	19	|			14		1	3	5	7			
		10	19	|			14		1	3	5	7	10		
			19	|			14		1	3	5	7	10	14	
			19	|					1	3	5	7	10	14	19

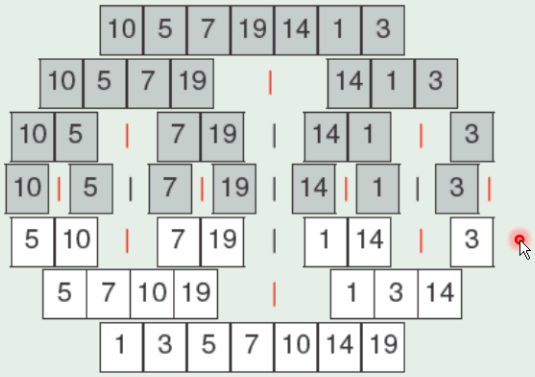
[3]

MergeSort

```
public void mergeSort(int[] a, int l, int r){ //sort from position l to r
    if (l == r) return; // we have only one element, nothing to sort;
    int m = (r + l)/2; // choose mid
    mergeSort(a, l, m); // sort left part
    mergeSort(a, m + 1, r); // sort right part
    //merge both sorted parts:
    int j = l;  //running variable for iterating left part
    int k = m + 1; //running variable for iterating right part
    int[] b = new int[r-l+1]; //intermediate storage for merging result
    for(int i=0; i<r-l+1; i++){ //perform merge:
        if(j > m){ //left part is used up --> use elements of right part
            b[i] = a[k];
            k++;
        } else if(k>r){ //right part is used up --> use elements of left part
            b[i] = a[j];
            j++;
        } else if(a[j]<=a[k]){ //element from the left part is smaller --> use it
            b[i] = a[j];
            j++;
        } else { //element from the right part is smaller --> use it
            b[i] = a[k];
            k++;
        }
    }
    for(int i=0; i<r-l+1; i++) //copy b back into respective parts of a:
        a[l+i] = b[i];
}
```

MergeSort

Beispiel



[3]

Vergleichsbasiertes Sortieren allgemein

- Angegebene Laufzeiten (InsertionSort, SelectionSort : $O(n^2)$, MergeSort: $O(n \log n)$) waren worst case Laufzeiten.
- *Frage*: geht es noch besser als $O(n \log n)$ im worst case?
  Anwort: Ohne Annahmen über die Menge der zu sortierenden Elemente und nur auf der Basis von Vergleichen geht es NICHT besser als $O(n \log n)$ im worst case.
- Aber: Wenn Keys der Elemente Zahlen aus [0, K-1] → verwende BucketSort:
  -- Array b mit K Elementen, jedes Array-Element ist beliebige Sequenz (bspw. verkettete Liste).
  -- für jedes e aus der zu sortierenden Sequenz:
  b(key(e)).pushBack(e)
  -- konkateniere alle Sequenzen b[i]

  Laufzeit: $\Theta(n + K)$ (natürlich nur gut wenn $K \in o(n \log n)$ (K ist „kleiner“ als n log n))

Vergleichsbasiertes Sortieren allgemein

- Angegebene Laufzeiten (InsertionSort, SelectionSort : $O(n^2)$, MergeSort: $O(n \log n)$) waren worst case Laufzeiten.
- *Frage*: geht es noch besser als $O(n \log n)$ im worst case?
  Anwort: Ohne Annahmen über die Menge der zu sortierenden Elemente und nur auf der Basis von Vergleichen geht es NICHT besser als $O(n \log n)$ im worst case.
- Aber: Wenn Keys der Elemente Zahlen aus [0, K-1] → verwende BucketSort:

  -- Array b mit K Elementen, jedes Array-Element ist beliebige Sequenz (bspw. verkettete Liste).

  -- für jedes e aus der zu sortierenden Sequenz:

  b(key(e)).pushBack(e)

  -- konkateniere alle Sequenzen b[i]

Laufzeit: $\Theta(n + K)$ (natürlich nur gut wenn $K \in o(n\ log\ n)$ (K ist „kleiner“ als n log n))

Vergleichsbasiertes Sortieren allgemein

- Angegebene Laufzeiten (InsertionSort, SelectionSort : $O(n^2)$, MergeSort: $O(n \log n)$) waren worst case Laufzeiten.
- *Frage*: geht es noch besser als $O(n \log n)$ im worst case?
  Anwort: Ohne Annahmen über die Menge der zu sortierenden Elemente und nur auf der Basis von Vergleichen geht es NICHT besser als $O(n \log n)$ im worst case.
- Aber: Wenn Keys der Elemente Zahlen aus [0, K-1] → verwende BucketSort:

  -- Array b mit K Elementen, jedes Array-Element ist beliebige Sequenz (bspw. verkettete Liste).

  -- für jedes e aus der zu sortierenden Sequenz:

  b(key(e)).pushBack(e)

  -- konkateniere alle Sequenzen b[i]

Laufzeit: $\Theta(n + K)$ (natürlich nur gut wenn $K \in o(n\ log\ n)$ (K ist „kleiner“ als n log n))

Vergleichsbasiertes Sortieren allgemein

- Angegebene Laufzeiten (InsertionSort, SelectionSort : $O(n^2)$, MergeSort: $O(n \log n)$) waren worst case Laufzeiten.
- *Frage*: geht es noch besser als $O(n \log n)$ im worst case?
  Anwort: Ohne Annahmen über die Menge der zu sortierenden Elemente und nur auf der Basis von Vergleichen geht es NICHT besser als $O(n \log n)$ im worst case.
- Aber: Wenn Keys der Elemente Zahlen aus [0, K-1] → verwende BucketSort:

  -- Array b mit K Elementen, jedes Array-Element ist beliebige Sequenz (bspw. verkettete Liste).

  -- für jedes e aus der zu sortierenden Sequenz:

  b(key(e)).pushBack(e)

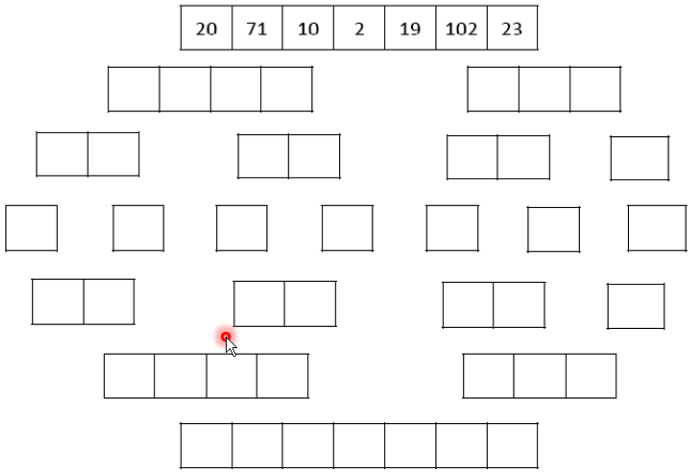
  -- konkateniere alle Sequenzen b[i]

Laufzeit: $\Theta(n + K)$ (natürlich nur gut wenn $K \in o(n\ log\ n)$ (K ist „kleiner“ als n log n))

Aufgabe

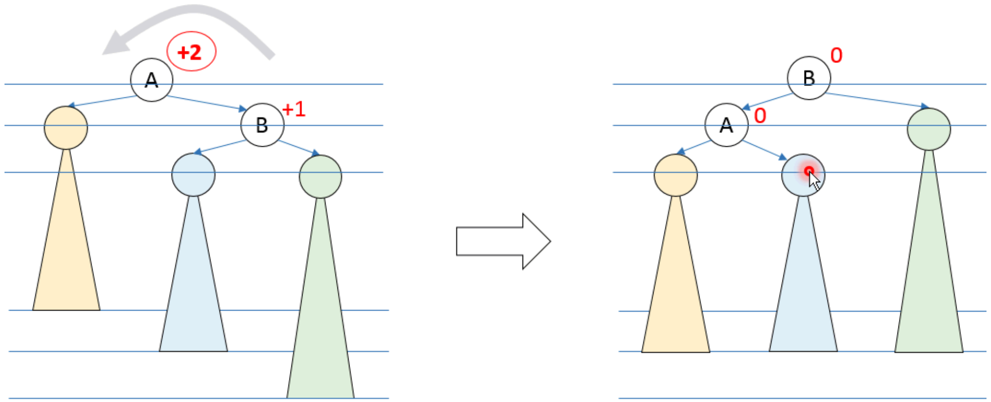
Mergesort

Stellen Sie im bereitgestellten **Schema** die **Sortierung** der Sequenz (20, 71, 10, 2, 19, 102, 23) mit **Mergesort dar**!



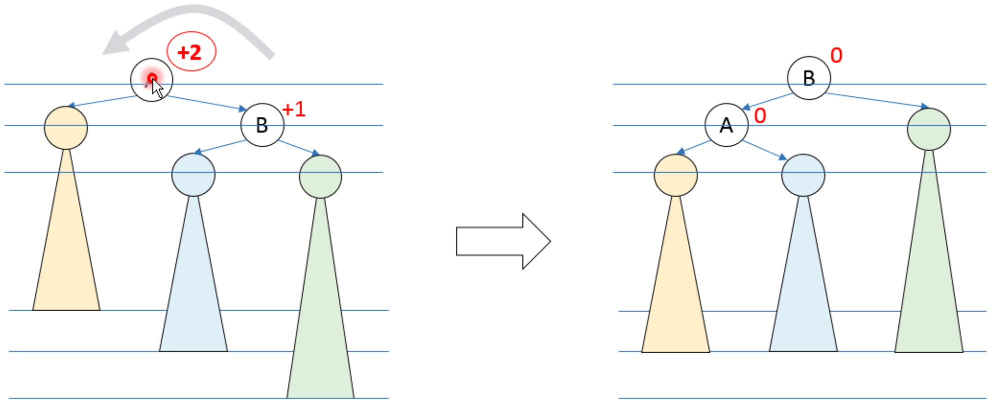
AVL Bäume: Rotationen

- Rotation nach links:



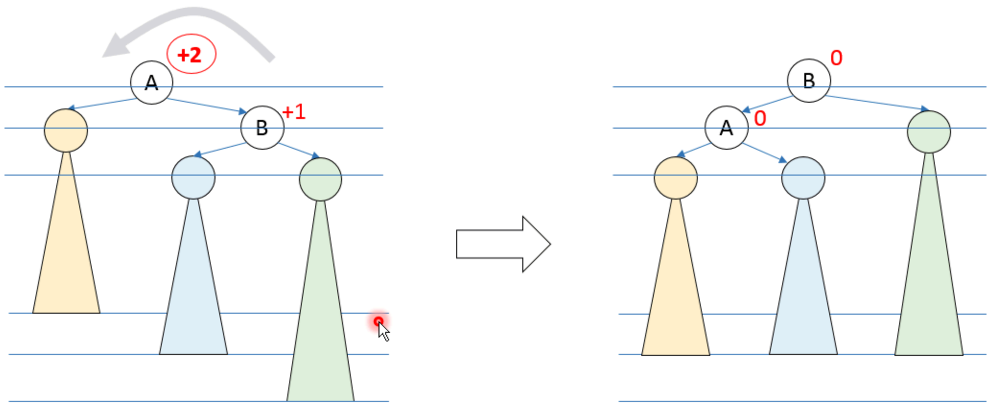
AVL Bäume: Rotationen

- Rotation nach links:



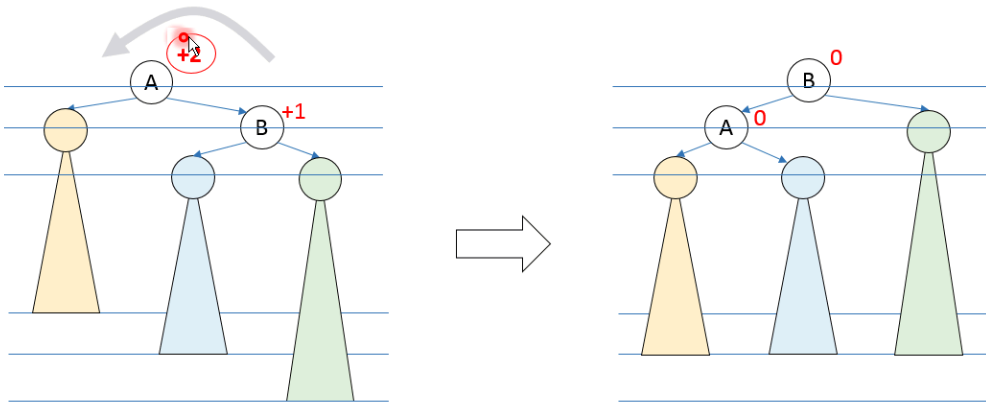
AVL Bäume: Rotationen

- Rotation nach links:

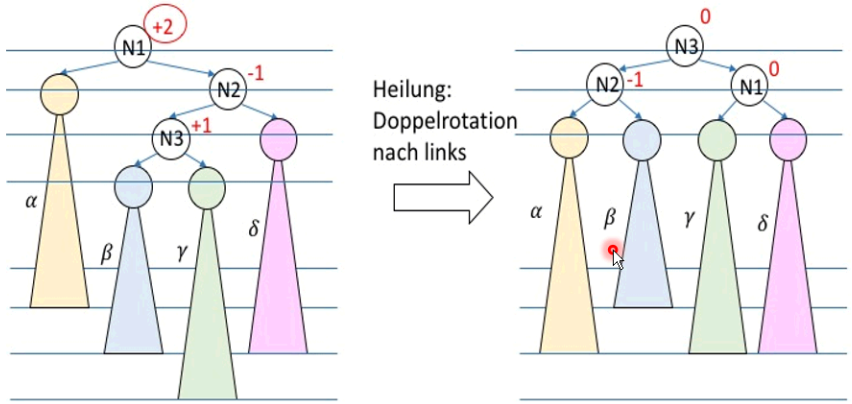


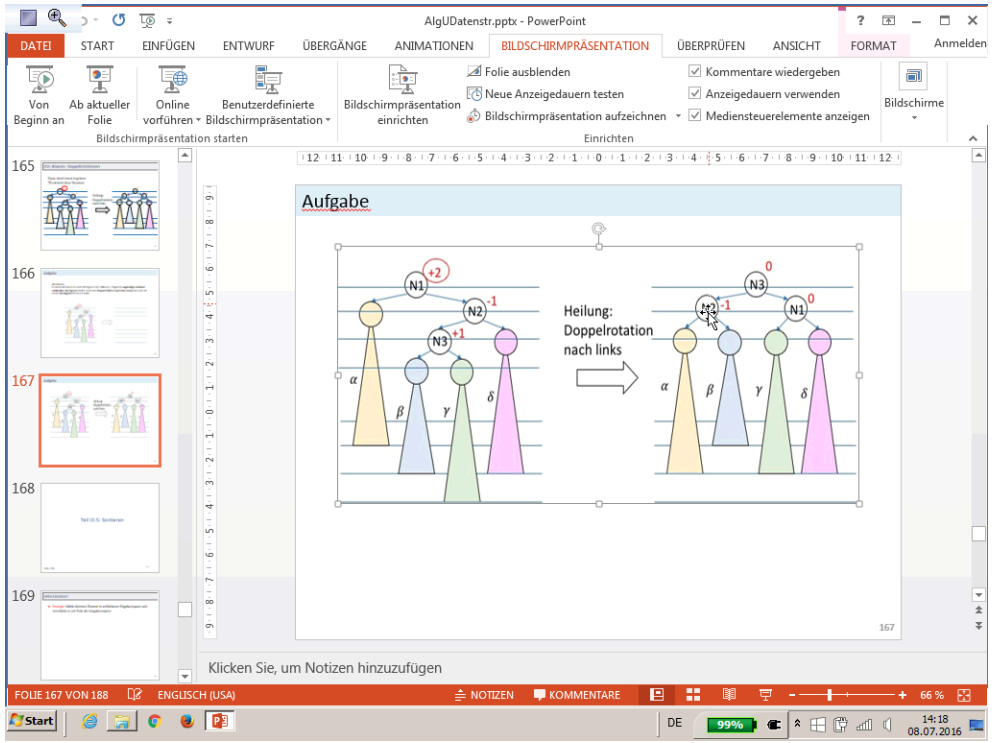
AVL Bäume: Rotationen

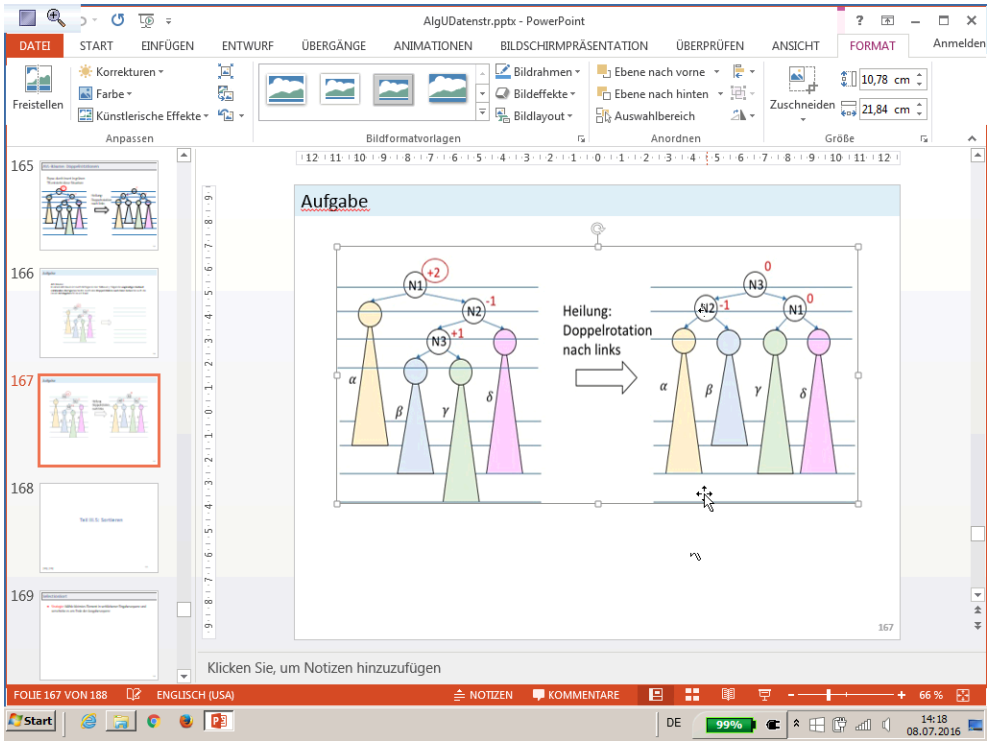
- Rotation nach links:



Aufgabe







AVL Bäume

- Lösung: Balance des Baumes erhalten (Nach jeder Operation checken, ggf. Rebalancierung)
- viele verschiedene Ansätze: AVL Bäume, (a,b)-Bäume, rot-schwarz-Bäume etc.

- AVL Bäume: in jedem Knoten Höhenunterschiede $\Delta h = h_r - h_l$ des rechten und linken Teilbaums speichern. Ziel: für alle Knoten soll stets $\Delta h \in \{-1,0,1\}$ sein.
- Bei insert und remove: Δh-Werte nach oben bis zur Wurzel anpassen
- Rebalancierungs-Methoden wenn $|\Delta h| \geq 2$:
  - Rotationen nach rechts oder links,
  - Doppelrotationen nach rechts oder links.