In Datenknoten:

```
public Datenknoten sortiertEinfuegen(Datenelement de){
    if (inhalt.istKleiner(de)){
        naechsterRechts = naechsterRechts.sortiertEinfuegen(de);
    }
    else {
        naechsterLinks = naechsterLinks.sortiertEinfuegen(de);
    }
    return this;
}
```

In Abschluss:

```
public Datenknoten sortiertEinfuegen(Datenelement de){
      return new Datenknoten(this, new Abschluss(), de);
}
```

Teste dein Programm, indem du mehrere Kunden einfügst (in einer Testklasse) und anschließend inorder ausgibst. Implementiere die Methode maximaleTiefeBestimmen() und überprüfe die Struktur des Baumes bei verschiedener Einfügereihenfolge.

maximaleTiefeBestimmen in Kundenverwaltung:

```
public int maximaleTiefeBestimmen(){
    return bst.maximaleTiefeBestimmen();
}
```

maximaleTiefeBestimmen in BST:

```
public int maximaleTiefeBestimmen(){
    return wurzel.maximaleTiefeBestimmen();
}
```

maximaleTiefeBestimmen in Baumelement:

```
public abstract int maximaleTiefeBestimmen();
```

maximaleTiefeBestimmen in Datenknoten:

```
public int maximaleTiefeBestimmen(){
        int r = naechsterRechts.maximaleTiefeBestimmen();
        int l = naechsterLinks.maximaleTiefeBestimmen();
        if (r>l) return r+1;
        else return l+1;
}
```

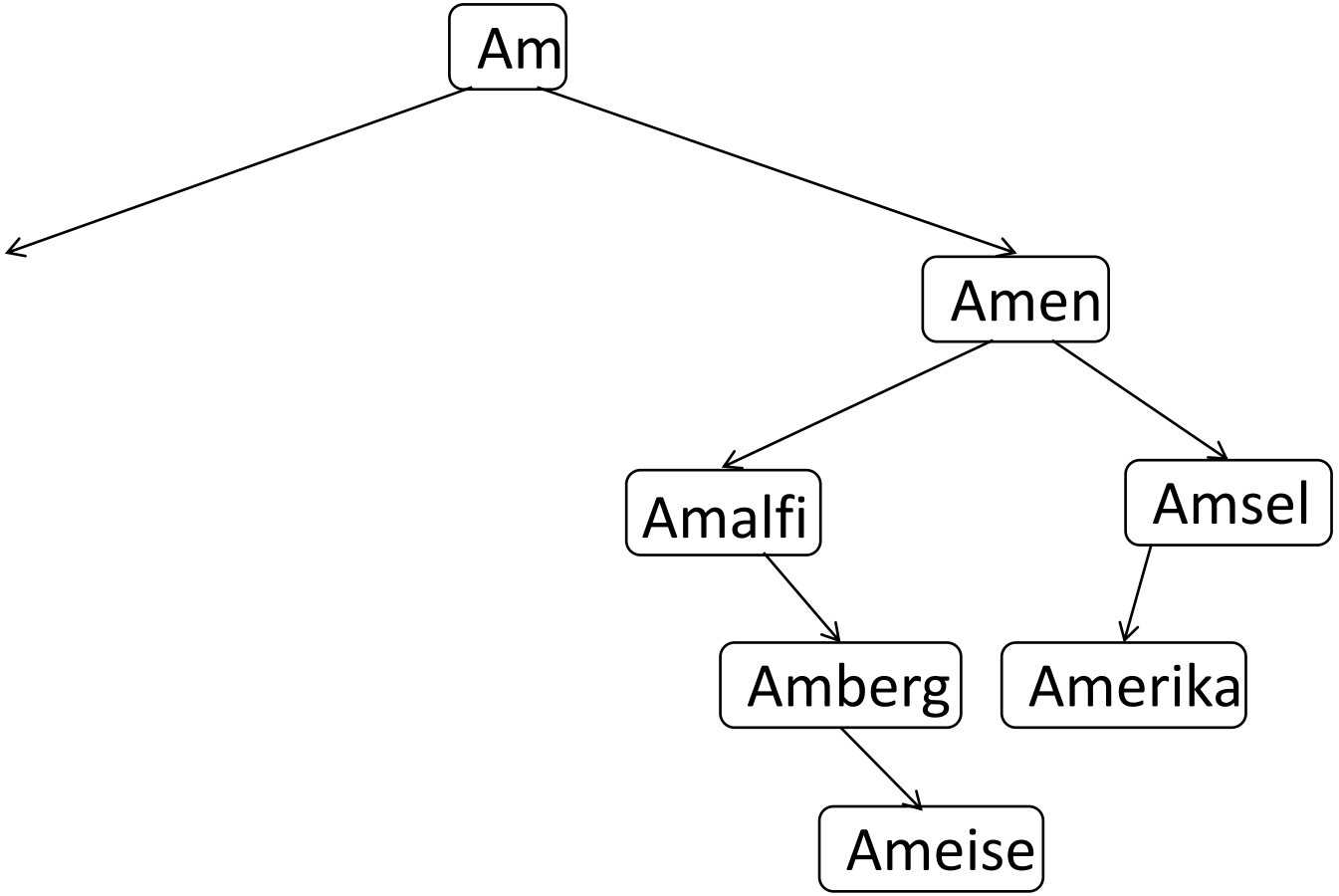
maximaleTiefeBestimmen in Abschluss:

```
public int maximaleTiefeBestimmen(){
        return -1;
    }
```

Löschen eines Datenknotens: S. 71

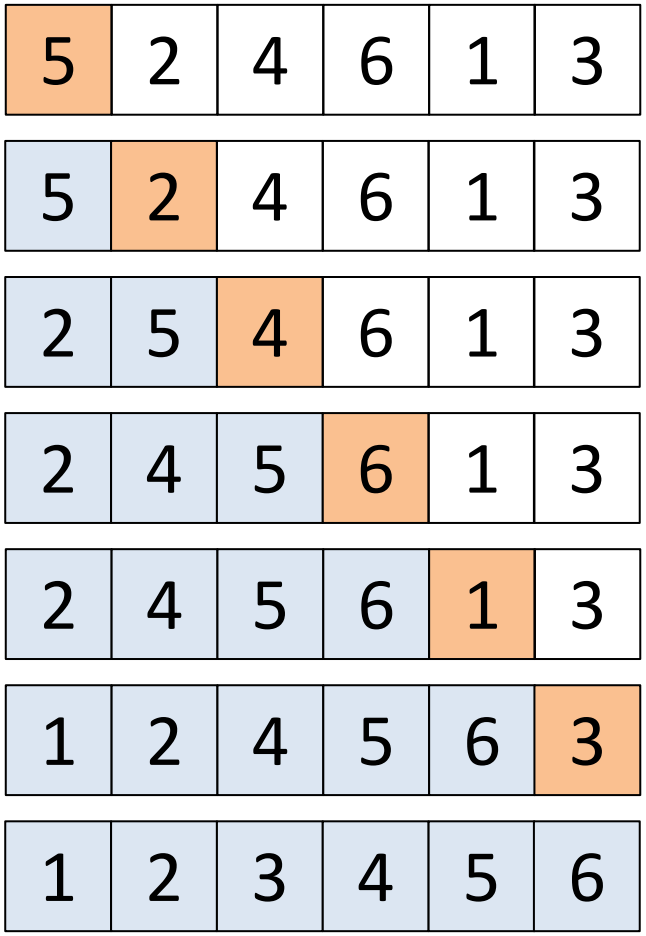
In Gruppenarbeit: S.72 / 3

Finde alle Wörter, die mit "Inf" beginnen.
Diskutiert insb., ob ein Binärbaum hier die geeignete Datenstruktur ist.


Am
Amen
Amalfi
Amsel
Amberg
Amerika
Ameise

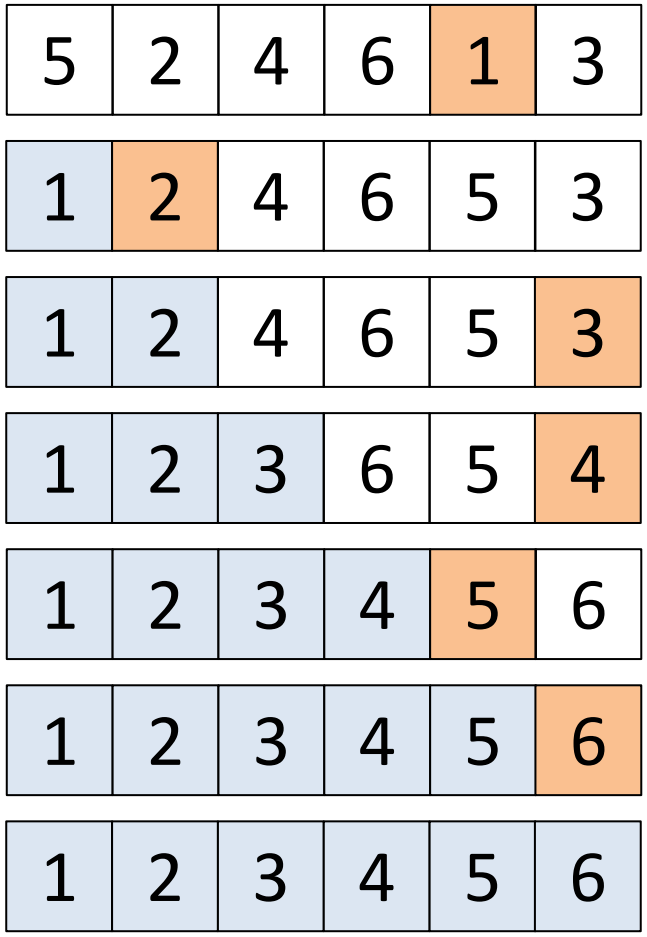
Einschub: Sortieralgorithmen

- Insert Sort:
  Elemente des noch nicht sortierten 2. Teils der Liste werden in den ersten Teil der Liste einsortiert.



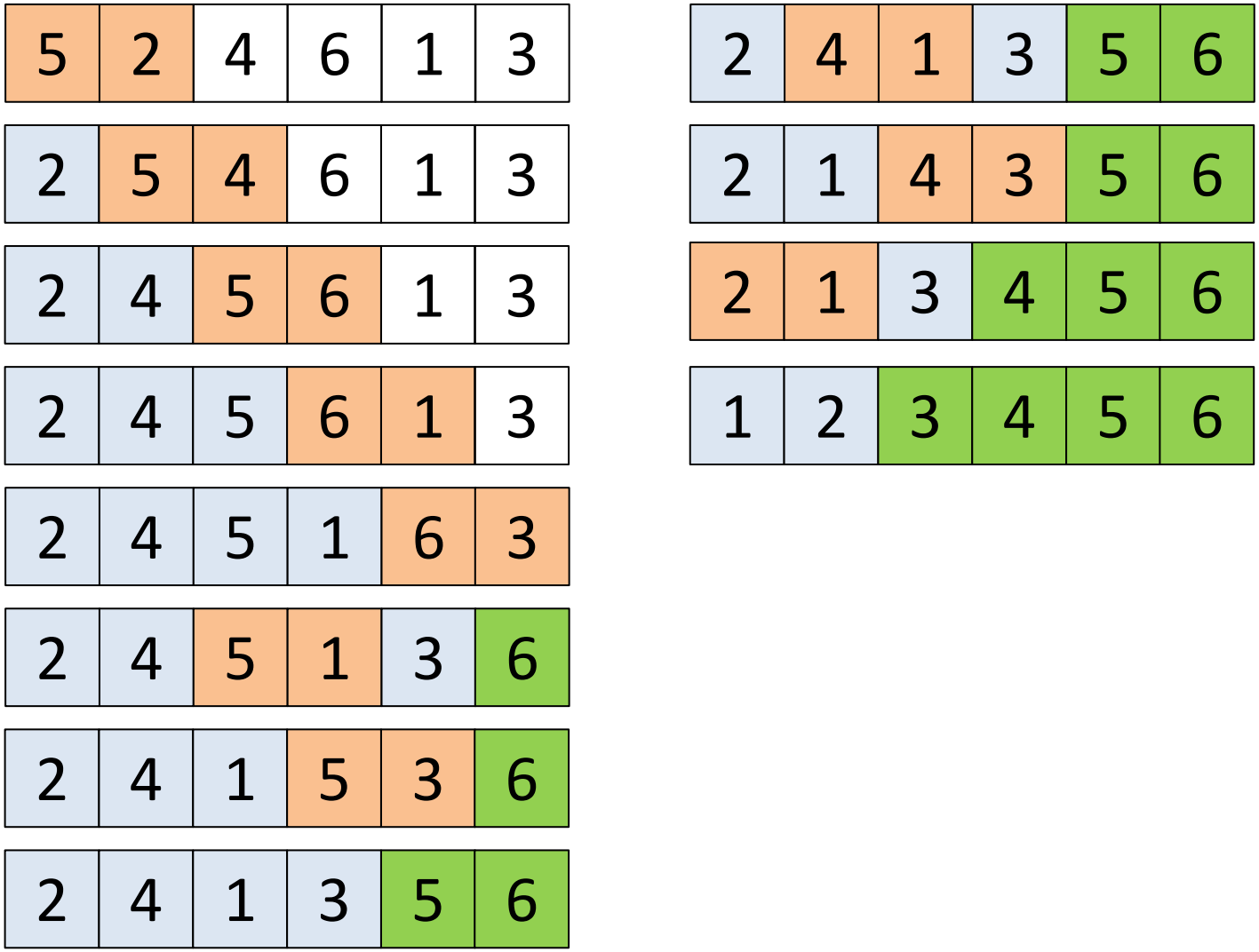
- Selection Sort:

  Das jeweils kleinste Element des noch nicht sortierten 2. Teils der Liste wird in den ersten Teil der Liste einsortiert.

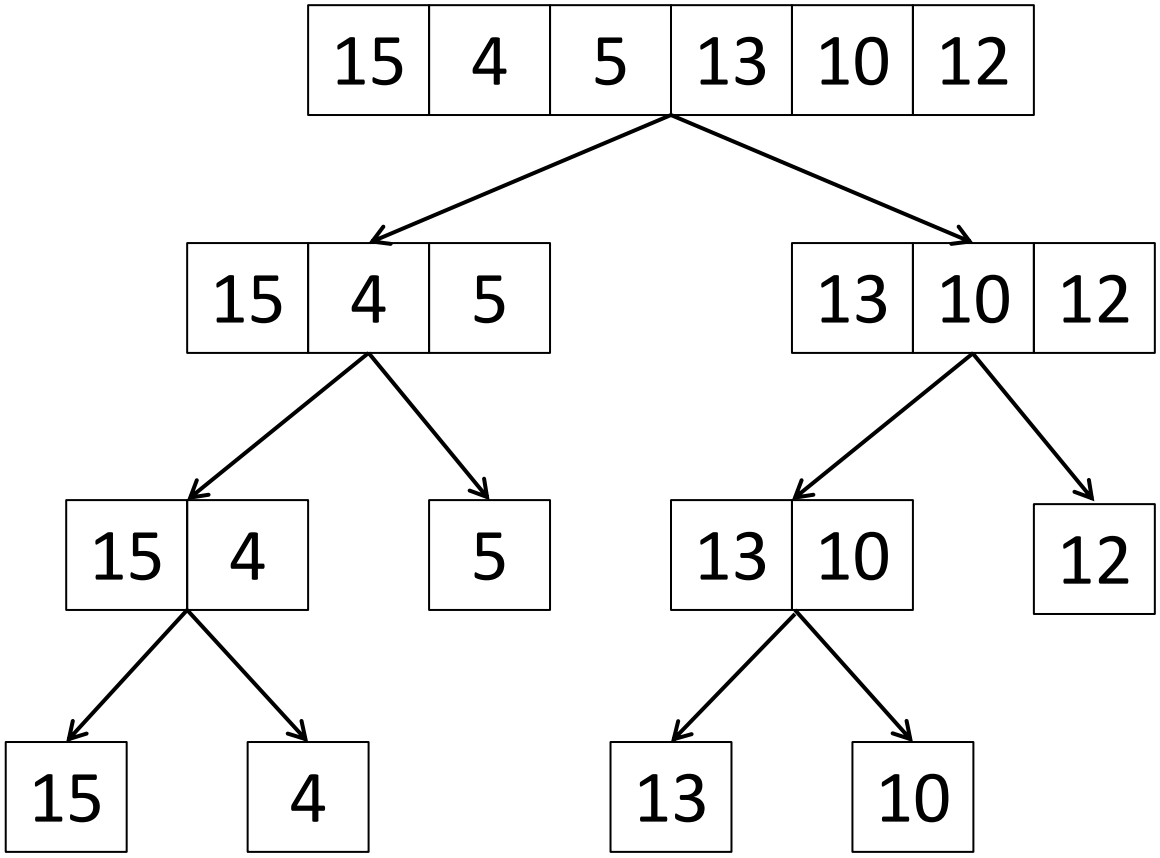


- Bubble Sort:

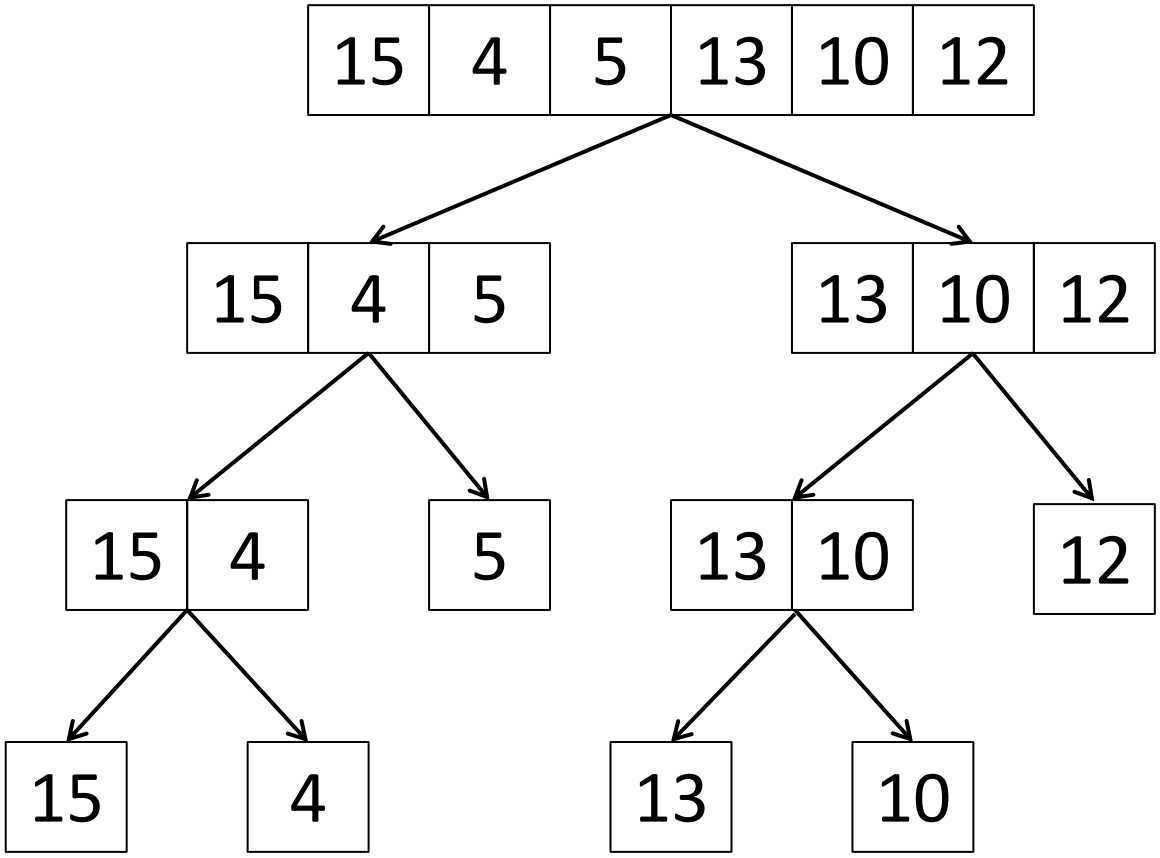
  Nachbarelemente, die in der falschen Reihenfolge stehen, werden so lange vertauscht, bis das Feld sortiert ist.



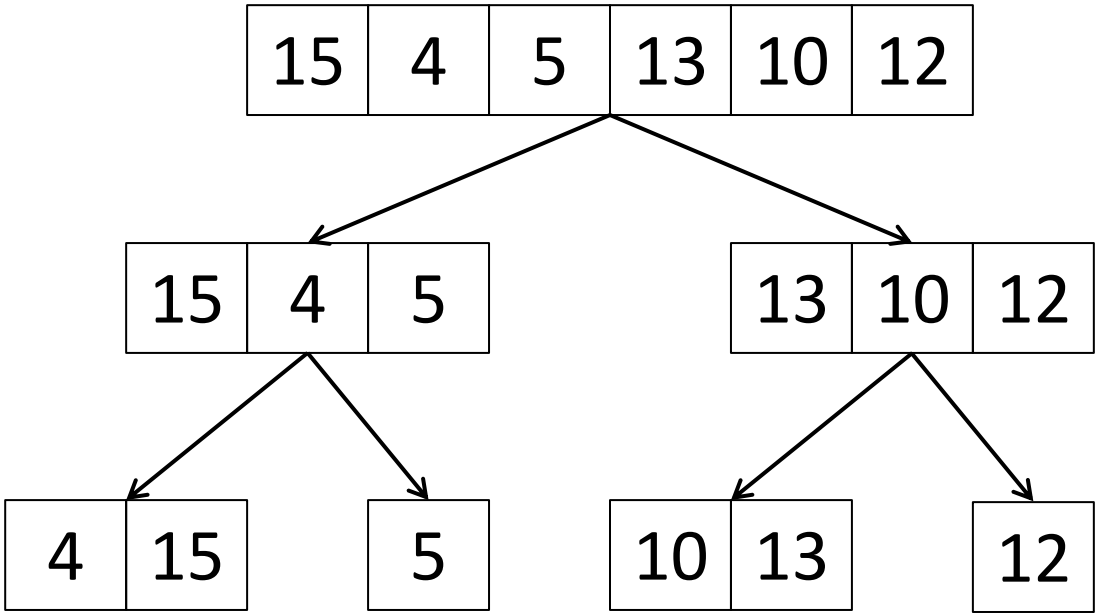
S. 66/6 Divide et impera: Mergesort



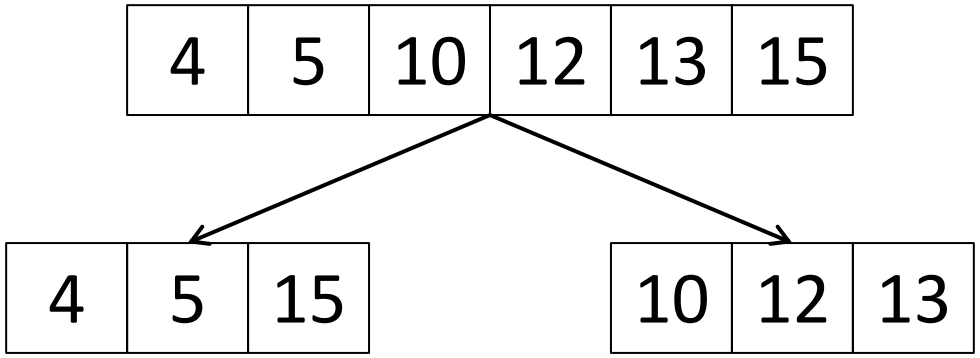
S. 66/6 Divide et impera: Mergesort



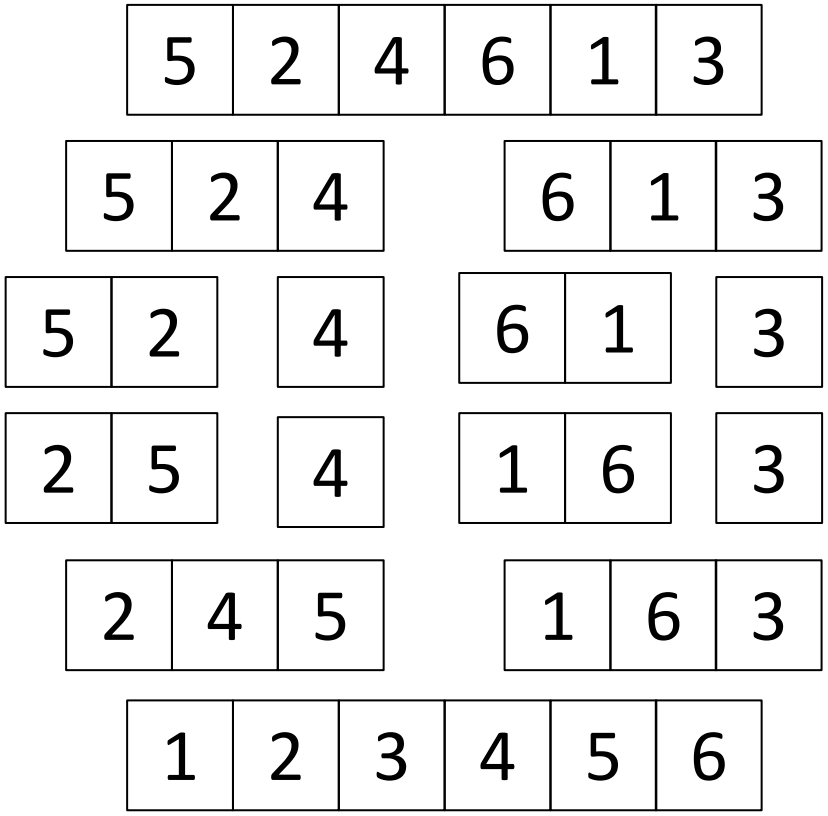
S. 66/6 Divide et impera: Mergesort



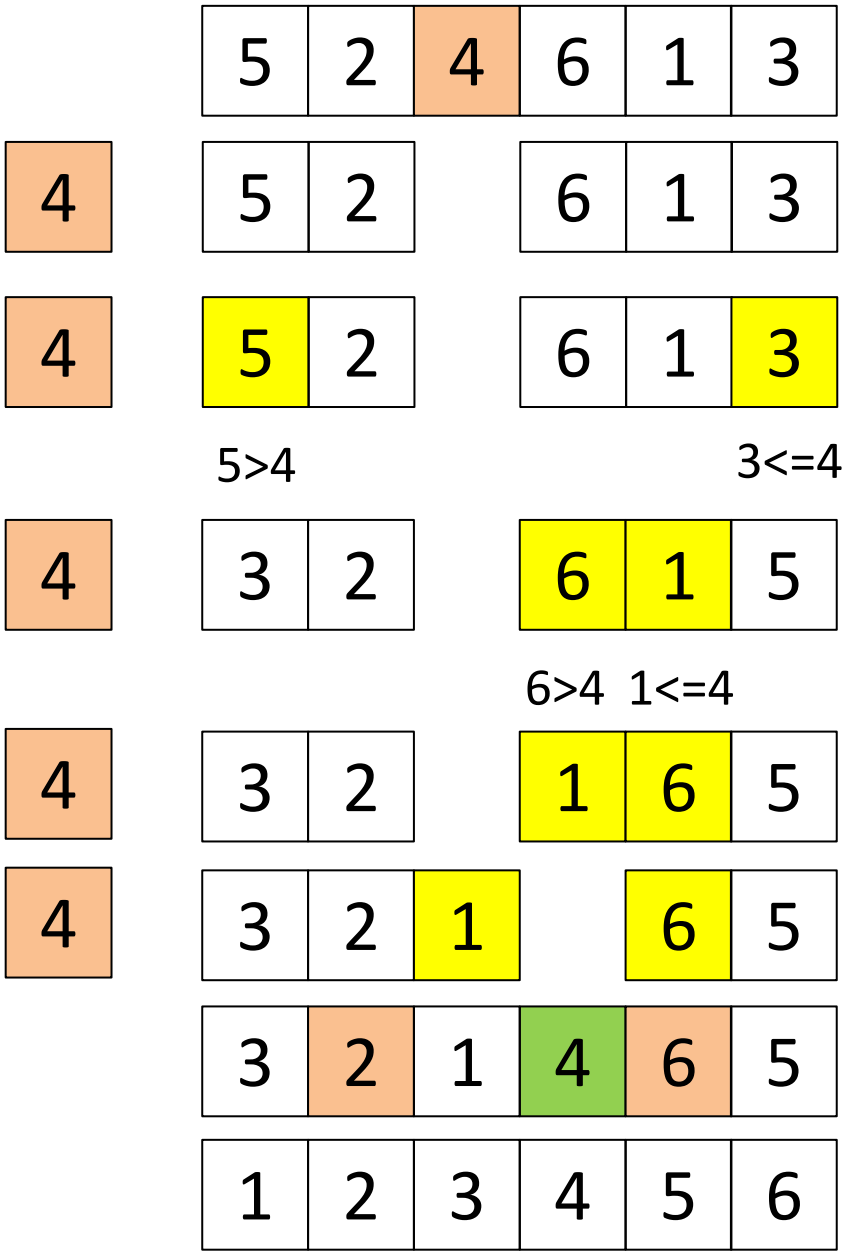
S. 66/6 Divide et impera: Mergesort



- Merge Sort:
  Teile in zwei gleich große Teillisten und sortiere diese getrennt.



- Quick Sort:
  Aus der unsortierten Liste wird ein beliebiges Element p ausgewählt und die übrigen Elemente auf zwei Listen verteilt, von denen die eine nur Elemente kleiner gleich p, die andere nur Elemente größer gleich p enthalten darf.


5 2 4 6 1 3
4
5 2 6 1 3
4
5 2 6 1 3
5>4
3<=4
4
3 2 6 1 5
6>4 1<=4
4
3 2 1 6 5
4
3 2 1 6 5
3 2 1 4 6 5
1 2 3 4 5 6

Laufzeit	Best Case	Average Case	Worst Case
Insert Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$
Quick Sort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$