Öffne Buecherei-Vorlage:

In einer Bücherei werden Bücher in einem geordneten Binärbaum verwaltet. Von jedem Buch wird die ISBN, der Autor und der Titel gespeichert. Der Baum ist nach den ISBN geordnet.

- Ergänze in den Klassen BST und Datenknoten Attribute und Konstruktor.
- Vervollständige in der Klasse Datenknoten die Methode alleDatenGeben() so, dass eine entsprechend des Schlüsselwerts geordnete Ausgabe erfolgt.
- Implementiere eine Methode buchSuchen(int nr) in der Klasse Buecherei und die dazu nötigen Methoden in den anderen Klassen. Diese Methode sucht ein Buch, dessen ISBN den Wert nr hat, und gibt es zurück.
- Implementiere eine Methode tiefeBestimmen(int nr) in der Klasse Buecherei und die dazu nötigen Methoden in den anderen Klassen. Diese Methode gibt die Tiefe des Datenknotens, der das Buch mit der entsprechenden ISBN enthält, zurück. Ist kein Buch mit der entsprechenden ISBN enthalten, soll -1 zurückgegeben werden.

In BST:

```
private Baumelement wurzel;

public BST(){
    wurzel = new Abschluss();
}
```

In Datenknoten:

```
private Baumelement naechsterLinks, naechsterRechts;
private Datenelement inhalt;

public Datenknoten(Baumelement nL, Baumelement nR,
                   Datenelement inh){
    naechsterLinks = nL;
    naechsterRechts = nR;
    inhalt = inh;
}
```

Rumpf von alleDatenGeben in Datenknoten: Inorder

```
return naechsterLinks.alleDatenGeben()
        + inhalt.datenGeben() + "\n"
        + naechsterRechts.alleDatenGeben();
```

buchSuchen in Buecherei:

```
public Datenelement buchSuchen(int nr){
    return bst.inhaltSuchen(new Buch(nr, "", ""));
}
```

inhaltSuchen in BST:

```
public Datenelement inhaltSuchen(Datenelement de){
    return wurzel.inhaltSuchen(de);
}
```

inhaltSuchen in Baumelement:

```
public abstract Datenelement inhaltSuchen(Datenelement de);
```

inhaltSuchen in Datenknoten:

```
public Datenelement inhaltSuchen(Datenelement de){
      if (inhalt.istGleich(de)){
          return inhalt;
      }
      else {
          if (inhalt.istKleiner(de)){
              return naechsterRechts.inhaltSuchen(de);
          }
          else {
              return naechsterLinks.inhaltSuchen(de);
          }
      }
  }
```

inhaltSuchen in Abschluss:

```
public Datenelement inhaltSuchen(Datenelement de){
        return null;
}
```

tiefeBestimmen in Buecherei:

```
public int tiefeBestimmen(int nr){
        return bst.tiefeBestimmen(new Buch(nr, "", ""));
}
```

tiefeBestimmen in BST:

```
public int tiefeBestimmen(Datenelement de){
        return wurzel.tiefeBestimmen(de, 0);
}
```

tiefeBestimmen in Baumelement:

```
public abstract int tiefeBestimmen(Datenelement de, int tiefe);
```

tiefeBestimmen in Datenknoten:

```
public int tiefeBestimmen(Datenelement de, int tiefe){
        if (inhalt.istGleich(de)){
          return tiefe;
      }
      else {
         if (inhalt.istKleiner(de)){
            return naechsterRechts.tiefeBestimmen(de, tiefe+1);
          }
          else {
             return naechsterLinks.tiefeBestimmen(de, tiefe+1);
          }
       }
    }
```

tiefeBestimmen in Abschluss:

```
public int tiefeBestimmen(Datenelement de, int tiefe){
    return -1;
}
```

maximaleTiefeBestimmen in Buecherei:

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

maximaleTiefeBestimmen in BST:

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

maximaleTiefeBestimmen in Baumelement:

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

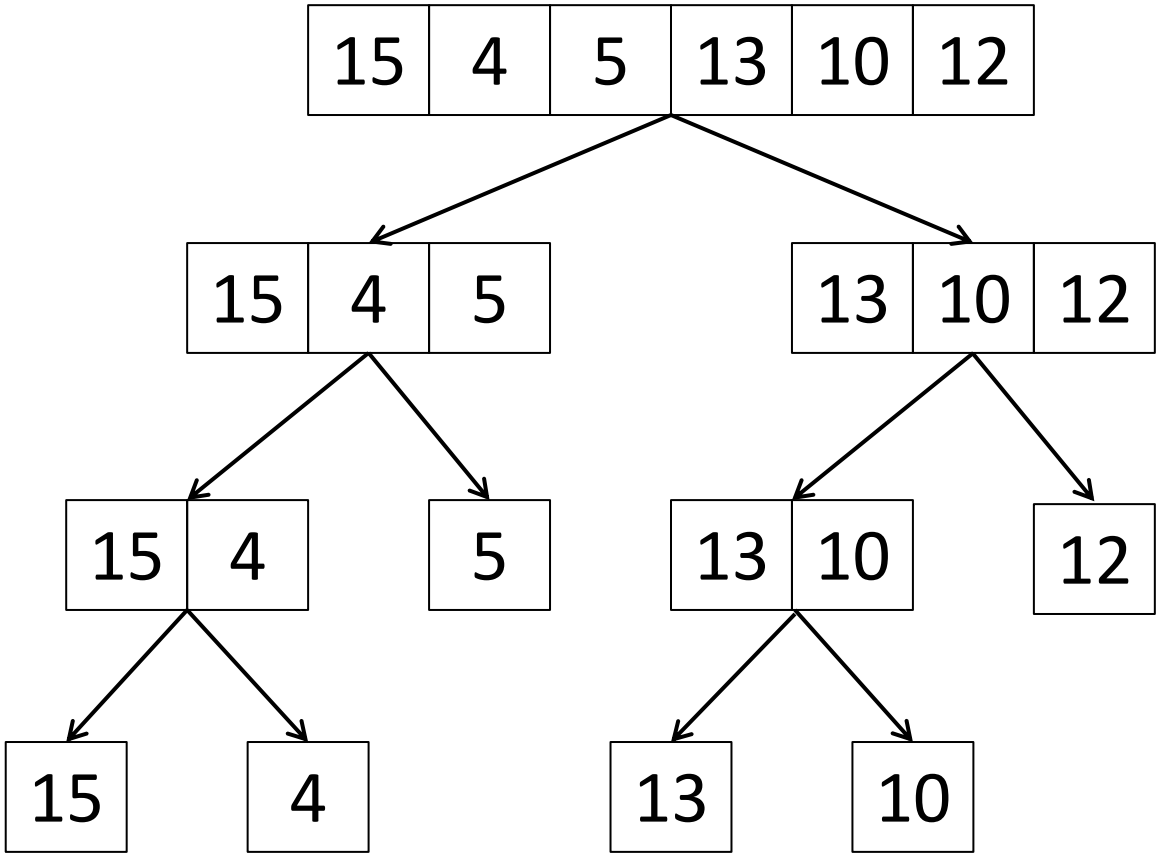
maximaleTiefeBestimmen in Datenknoten:

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

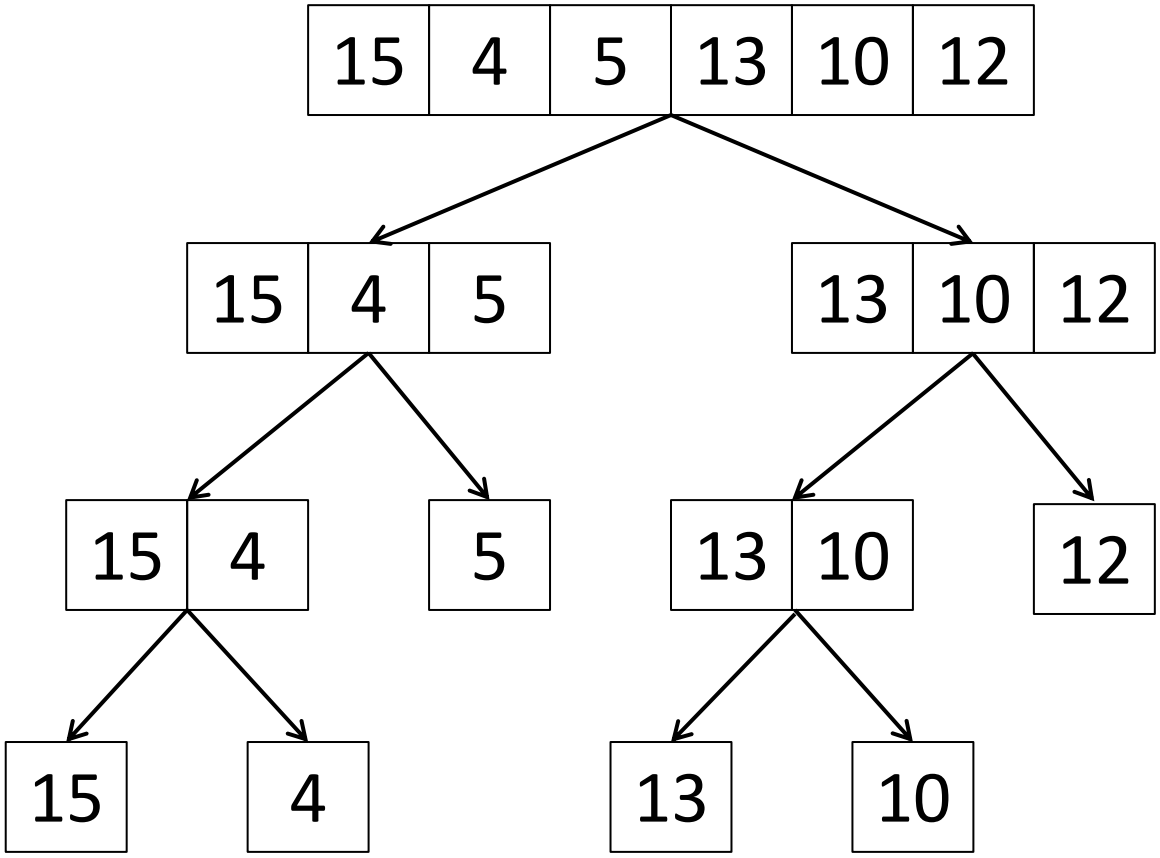
maximaleTiefeBestimmen in Abschluss:

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

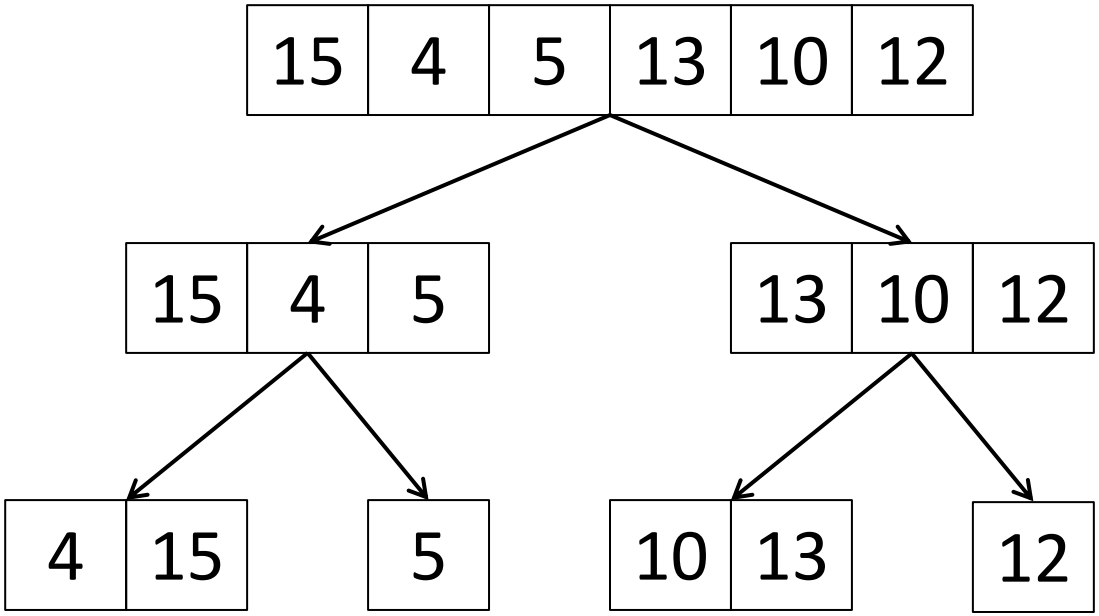
S. 66/6 Divide et impera: Mergesort



S. 66/6 Divide et impera: Mergesort



S. 66/6 Divide et impera: Mergesort



S. 66/6 Divide et impera: Mergesort