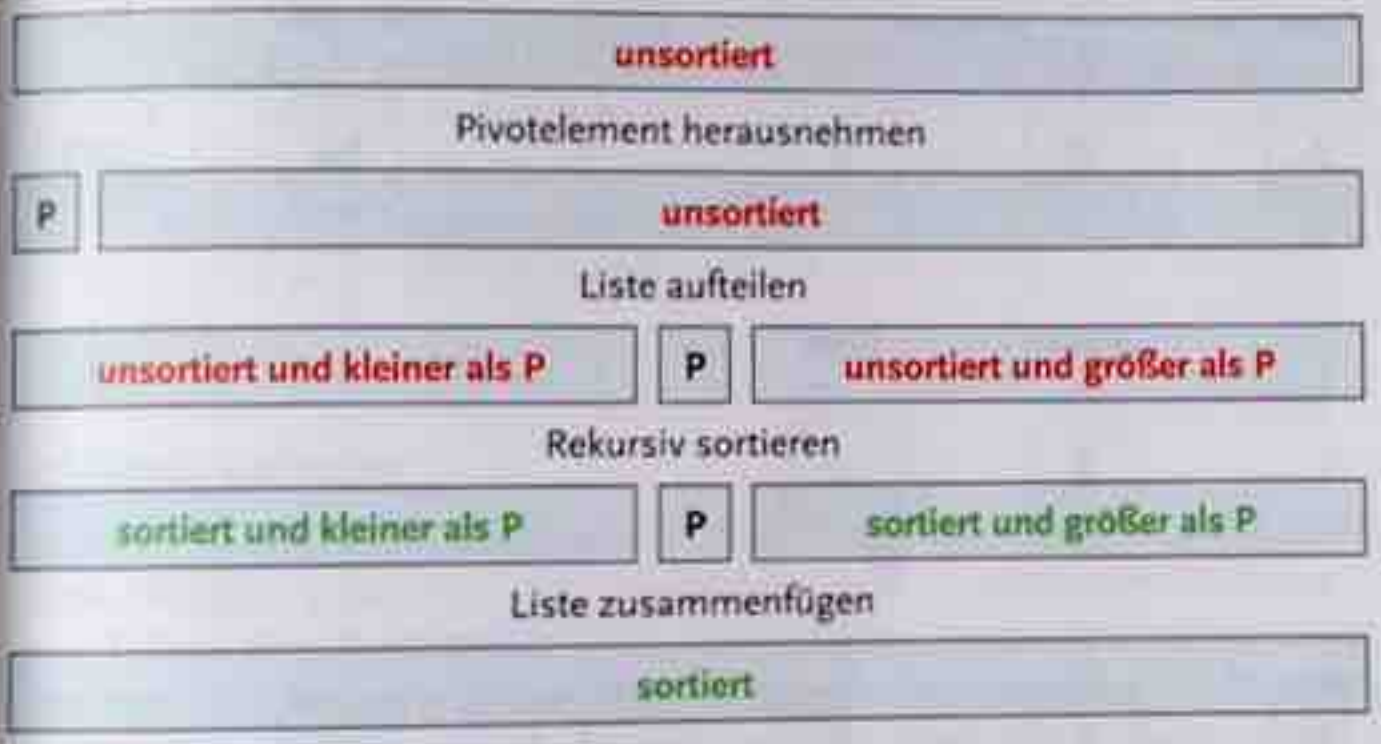
Quicksort

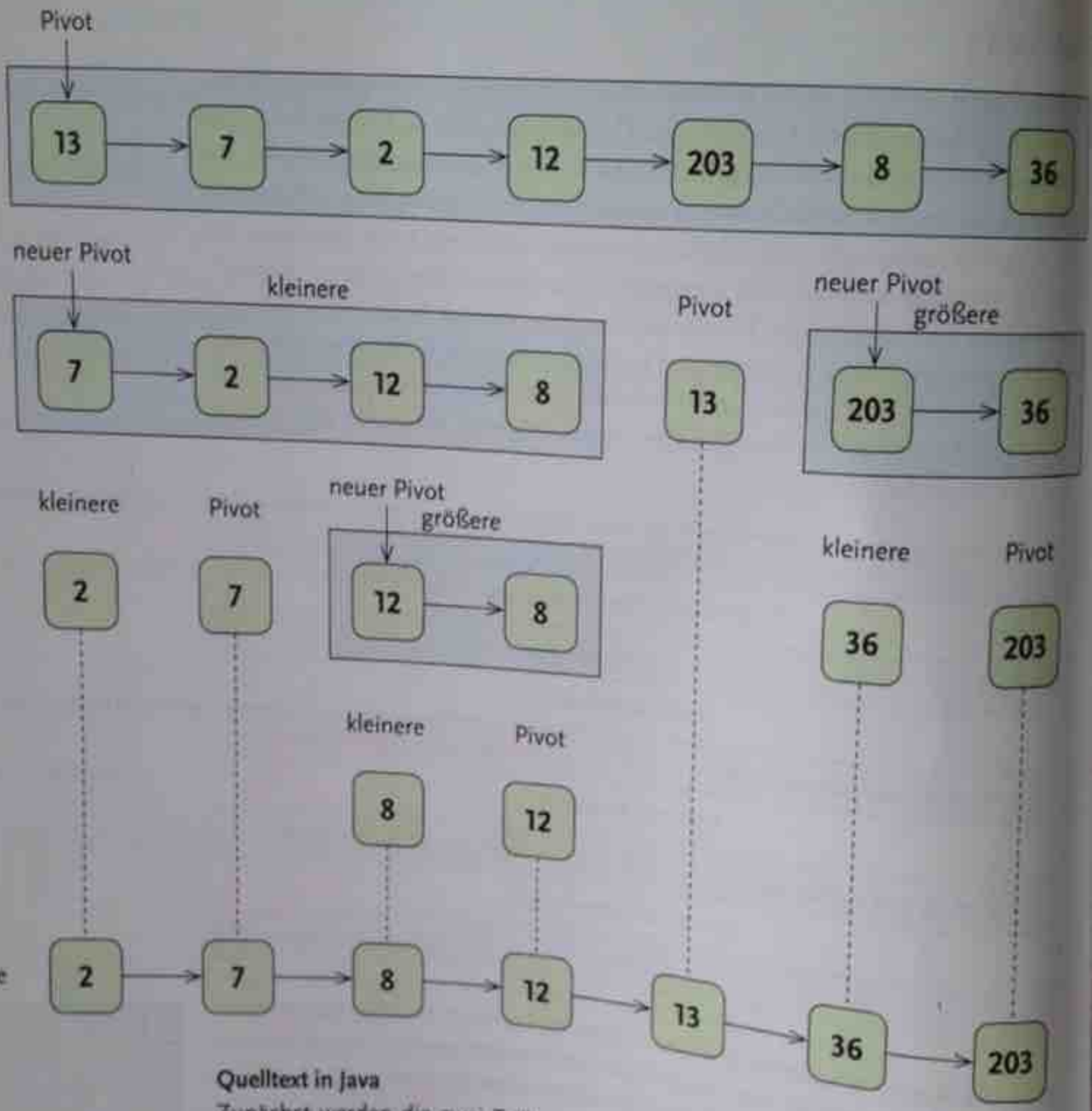
Während beim Sortieren durch Einfügen die zu sortierende Folge mehrfach durchlaufen und dabei jeweils ein Element in die Ordnung gebracht wird, wählt der Quicksort-Algorithmus eine andere Vorgehensweise. Er bringt zunächst alle Elemente, die kleiner als ein frei gewähltes Element (Pivotelement) sind, in eine linke Teilfolge und alle, die größer sind, nach rechts. Es liegt jetzt zwar noch lange keine Sortierung vor, aber die Elemente befinden sich zumindest schon mal in der richtigen „Region". Nun werden nach dem selben Prinzip die Elemente, die sich in der linken Teilfolge befinden, sortiert. Dieses Vorgehen wird so lange wiederholt, bis das Problem trivial ist, also nur noch ein Element sortiert werden muss. Man verfährt jeweils genauso mit den Elementen der rechten Teilfolge. Dieses Vorgehen entspricht dem Prinzip Teile und Herrsche.



Am Beispiel wird die rekursive Arbeitsweise des Quicksortalgorithmus' deutlich. Das erste Listenelement, die 13, wird als Pivotelement gewählt. Diese Wahl ist grundsätzlich beliebig, d. h. man hätte auch jedes andere Element als Pivot auswählen können.

Nun werden die restlichen Zahlen auf zwei Listen verteilt. Die eine Liste nimmt alle Elemente auf, die kleiner als 13 sind und die andere alle größeren. Auf diese beiden Listen wird nun rekursiv wiederum der Quicksortalgorithmus angewandt, d. h. beispielsweise für die Liste der kleineren Elemente, dass die 7 als Pivot fungiert und die verbleibenden drei Zahlen in zwei weitere Listen aufgeteilt werden. Schließlich gelangt man zu einer Aufteilung, bei der die Listen von kleineren und größeren Elementen nur noch eine Zahl enthalten oder sogar leer sind. Dann muss natürlich nicht mehr sortiert werden, der Rekursionsanker ist erreicht. Am Ende werden alle sortieren Teillisten und Pivotelemente wieder zu einer sortierten Gesamtliste zusammengefügt.

H Für die Effizienz des Quicksortalgorithmus' ist die Wahl des Pivots wesentlich. Immer das erste Element zu wählen, wäre beispielsweise sehr ungünstig, wenn die Zahlen bereits sortiert vorliegen.



Quelltext in Java

Zunächst werden die zwei Teillisten für die größeren und kleineren Elemente erzeugt. Das erste Element der Originalliste wird zum Pivot und kann aus der Liste entfernt werden. Alle restlichen Elemente werden mithilfe der while-Schleife durch Vergleich mit dem Pivotelement auf die beiden Teillisten verteilt. Die Originalliste ist nun leer. Dann folgt der Rekursionsschritt, in dem beide Teillisten sortiert werden. An die leere Originalliste werden schließlich zuerst die sortierte Liste aller kleineren Elemente, dann das Pivotelement und zuletzt die sortierte Liste aller größeren Elemente angehängt. Damit ist die Originalliste sortiert.

```
public void quicksort(List<Integer> liste)
{
  if(laenge(liste) > 1)                    //Hilfsmethode laenge
                                           (pl:List<ContentType>):int
  {
     List<Integer> kleinere = new List<Integer>();
     List<Integer> groessere = new List<Integer>();
     liste.toFirst();
     int pivot = liste.getContent();
```

Rekursionsanker

```
        liste.remove();
        while (!liste.isEmpty())
        {
            int akt = liste.getContent();
            if(akt < pivot)
            {
                kleinere.append(akt)
            }
            else
            {
                groessere.append(akt);
            }
            liste.remove();
        }
        quicksort(kleinere);
        quicksort(groessere);
        liste.concat(kleinere);
        liste.append(pivot);
        liste.concat(groessere);
    }
}
```

Rekursionsschritt

Effizienzbetrachtung

Im besten Fall teilt das Pivotelement die restlichen Elemente in zwei genau gleich große Teilfolgen. In jedem Schritt wird die Anzahl der zu sortierenden Elemente in einer Teilliste also mindestens halbiert. Ähnlich wie bei der binären Suche benötigt man dann $\log_2 n$ Rekursionsebenen. In jeder Ebene müssen allerdings die Elemente mit dem Pivot verglichen werden, um sie in die passende Teilliste einsortieren zu können. Damit ergibt sich für n Elemente ein Aufwand von etwa $n \cdot \log_2 n$.

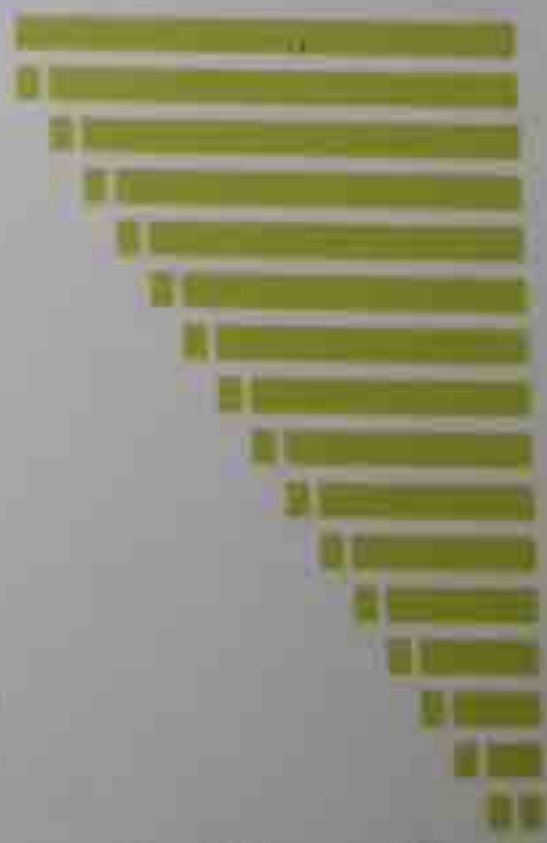
Rekursionsstruktur im besten Fall ...

Auch im mittleren Fall benötigt der Quicksortalgorithmus nur etwa $n \cdot \log_2 n$ Vergleiche. Dies lässt sich durch praktische Untersuchungen oder durch theoretische Begründungen belegen.

Der schlechteste Fall liegt vor, wenn immer das kleinste oder das größte Element als Pivot ausgewählt wird. Dann entstehen als Teillisten immer eine leere und eine nur um ein Element kleinere Liste. So hat man im ersten Schritt n–1 Vergleiche, im zweiten Schritt n–2 usw. Insgesamt benötigt der Quicksortalgorithmus im schlechtesten Fall also

$$(n-1) + (n-2) + \ldots + \frac{1}{2}(n-1)n = \frac{1}{2}n^2 - \frac{1}{2}n$$

Vergleiche (Gaußsche Summenformel). Damit ist die Laufzeit im schlechtesten Fall quadratisch.



... und im schlechtesten Fall.

Damit ist der Quicksortalgorithmus, zumindest im mittleren Fall, entschieden besser als der Algorithmus zum Sortieren durch Einfügen. Die Logarithmusfunktion $\log_2 n$ wächst viel langsamer als die lineare Funktion n, sodass auch $n \cdot \log_2 n$ langsamer wächst als n^2.