Paradigma: Teile-und-Herrsche

Teile-und-herrsche ist ein allgemeines Prinzip zur Konstruktion von Algorithmen.

Andere Bezeichnungen: Divide-and-conquer, divide et impera

- Zerlege das gegebene Problem in mehrere getrennte Teilprobleme,
  - löse diese einzeln und
  - setze die Lösung des ursprünglichen Problems aus den Teillösungen zusammen.
- Wende diese Technik auf jedes der Teilprobleme an, dann deren Teilprobleme usw., bis die Teilprobleme klein genug sind, um sie explizit zu lösen.
- Strebe an, dass jedes Teilproblem von derselben Art ist, wie das ursprüngliche Problem.

☞ Rekursion

Mergesort

Zerlegung: Zerlege die zu sortierende Folge in zwei möglichst gleich große Teilfolgen und sortiere die Teile einzeln.

Beispiel: 11 8 6 3 7 9 2 15 12 4 wird in zwei Folgen zerlegt: 11 8 6 3 7 und
9 2 15 12 4

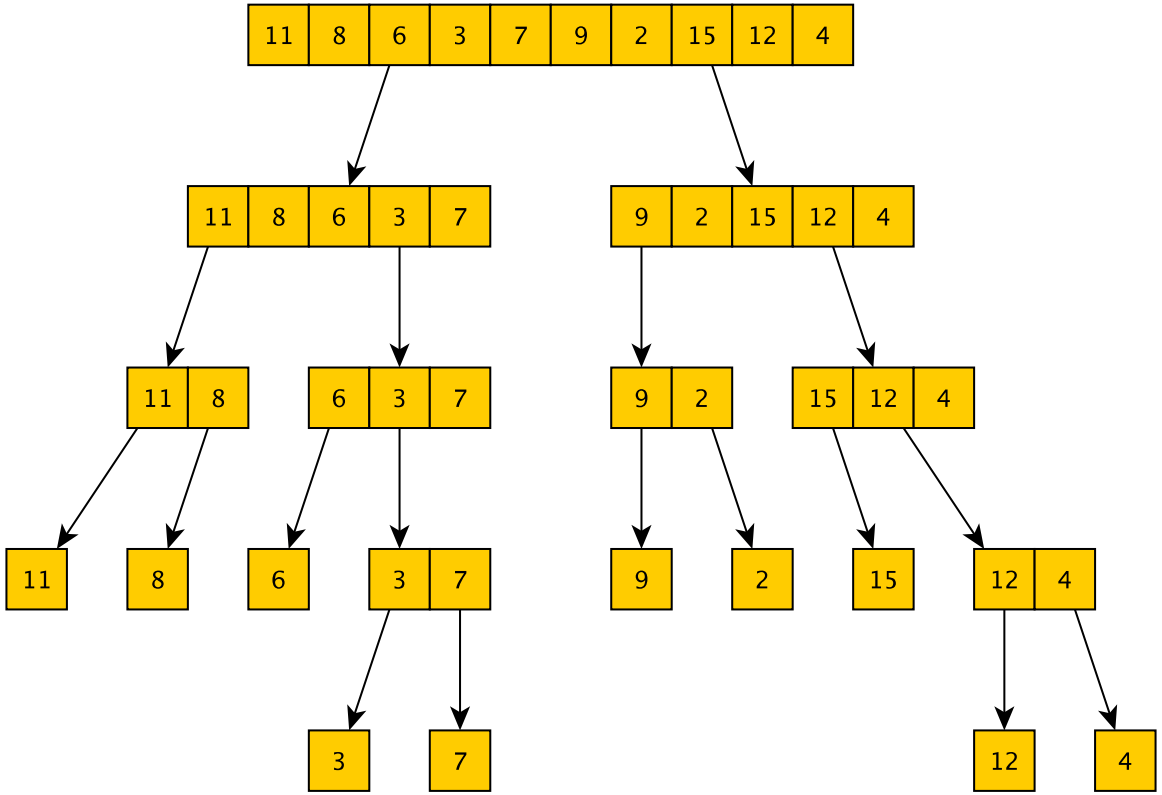
Zusammensetzung: Mische die sortierten Teilfolgen, so dass eine sortierte Gesamtfolge entsteht.

Beispiel: Die sortierten Teilfolgen lauten: 3 6 7 8 11 und 2 4 9 12 15

Sortierte Mischung ergibt 2 3 4 6 7 8 9 11 12 15

Explizite Lösung: Teilfolgen der Länge eins sind bereits sortiert.

Zerlegung


11 8 6 3 7 9 2 15 12 4
11 8 6 3 7
9 2 15 12 4
11 8
6 3 7
9 2
15 12 4
11
8
6
3 7
9
2
15
12 4
3
7
12
4

Algorithmus Zerlegung

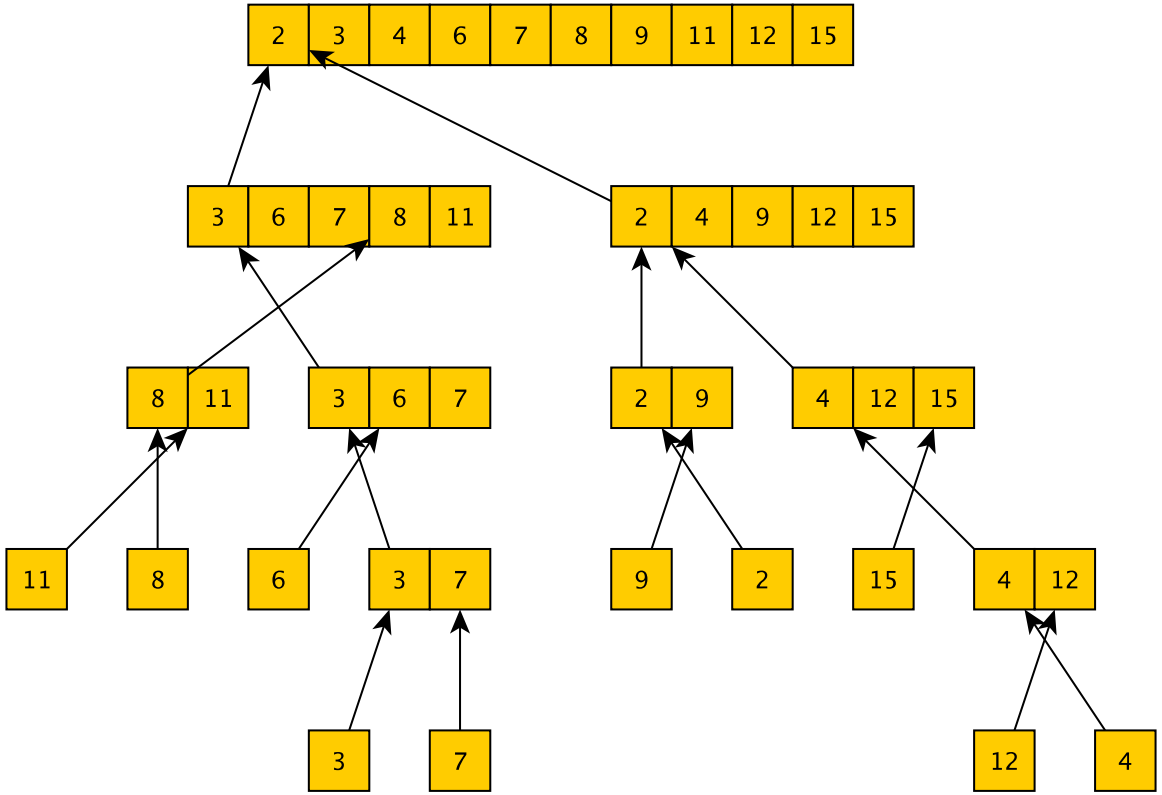
```
/*
 * Sortiere den Bereich des Feldes a zwischen il und ir
 * il ist das erste Element am linken Rand des zu sortierenden Bereichs
 * ir ist das erste Element am rechten Rand, das nicht mehr zum Bereich gehoert
 */
static void mergesort(int[] a, int il, int ir) {
    int imid = (il+ir)/2;          // mittleres Element

    if (ir-il<=1) {                // Bereich ist schon sortiert, wenn nicht mehr als
        return;                    // ein Wert zu sortieren ist.
    }

    mergesort(a, il, imid);        // sortiere linken Teil
    mergesort(a, imid, ir);        // sortiere rechten Teil

    /* jetzt mischen, siehe unten */
}
```

Mischen


2 3 4 6 7 8 9 11 12 15
3 6 7 8 11
2 4 9 12 15
8 11
3 6 7
2 9
4 12 15
11
8
6
3 7
9
2
15
4 12
3
7
12
4

Algorithmus Mischen

```
static void mergesort(int[] a, int il, int ir) {
    int imid = (il+ir)/2;

    /* zerlegen, siehe oben */

    int[] atmp = new int[ir-il];      // Hilfsfeld, um sortierte Mischung zu speichern
    int   itmp = 0;                   // Index fuer Hilfsfeld
    int   jl = il;                    // Index fuer linken sortierten Teil von a
    int   jr = imid;                  // Index fuer rechten sortierten Teil von a

    while (jl < imid && jr < ir) {    // solange es in den sortierten Teilen noch Werte gibt
        if (a[jl] <= a[jr]) {         // nehme den kleineren aus dem linken Teil
            atmp[itmp] = a[jl];       // uebertrage ihn nach atmp
            jl++;                     // und gehe im linken Teil zum naechsten Wert
        }
        else {                        // nehme kleineren aus rechtem Teil
            atmp[itmp] = a[jr];       // uebertrage ihn nach atmp
            jr++;                     // und gehe im rechten Teil zum naechsten Wert
```

```
        }
        itmp++;                          // auf jeden Fall ist in atmp jetzt ein Wert mehr
    }

    // jetzt ist im linken oder im rechten sortierten Teil
    // kein Wert mehr
    while (jl < imid) {                  // uebertrage Reste vom linken Teil nach atmp
        atmp[itmp] = a[jl];
        jl++; itmp++;
    }
    while (jr < ir) {                    // uebertrage Reste vom rechten Teil nach atmp
        atmp[itmp] = a[jr];
        jr++; itmp++;
    }

    // jetzt liegt die sortierte Mischung des linken und rechten Teils in atmp
    // kopiere die Inhalte von atmp nach a
    for (int i=0 ; i<atmp.length ; i++) {
       a[il+i] = atmp[i];
    }
}
```

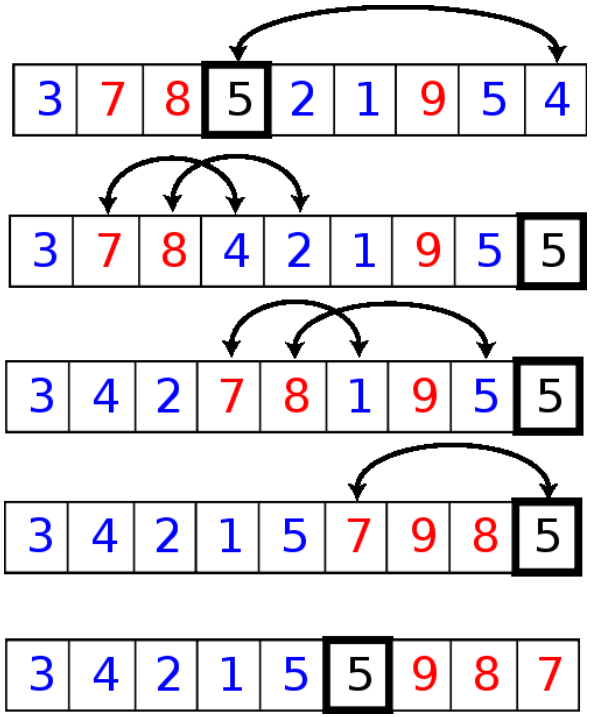
Eigenschaften von Mergesort

- Gesamtaufwand: Zeit $O(n \log n)$
  - Mischen zweier Teilfolgen der Länge n_l bzw. n_r in Zeit $O(n_l + n_r)$
  - Zerlegung produziert ausgeglichenen Baum der Höhe $O(\log n)$ aus Teilproblemen
  - Auf jeder Ebene Aufwand von $O(n)$ für das Mischen der Teilfolgen
- Keine In-Place Sortierung
- Mergesort ist *stabil*, d.h. gleiche Werte behalten in der sortierten Folge ihre relative Reihenfolge.

Quicksort

- Eines der bekanntesten Sortierverfahren ist Quicksort.
- basiert ebenfalls auf Teile und herrsche
- stammt von C. A. R. Hoare aus dem Jahre 1960
- Idee:
  - Wähle ein Pivotelement aus.
  - Bringe alle Werte, die kleiner als das Pivotelement sind, auf die linke Seite des Pivotelements und alle größeren Werte auf die rechte Seite.
  - Sortiere (rekursiv) die linke und rechte Seite.

Quicksort In-Place



- Wähle Pivotelement (hier 5) und vertausche Pivotelement mit dem Element am rechten Rand.
- Von links nach rechts bis vor das Pivotlement:
  - Wenn Element $\leq$ Pivotelement, dann tausche nach links.
  - Zielposition für einen Tausch ist zunächst der linke Rand, aber für jedes Element $\leq$ Pivotelement rückt die Zielposition um eins nach rechts.
- Tausche Pivotelement an die richtige Position.
- Sortiere (rekursiv) linken (blau) und rechten Teil (rot).

Quicksort: Algorithmus

```
private static void quicksort(int[] a, int il, int ir) {
    int imid = (il+ir)/2;      // Index Pivotelement
    int pivot;                 // Wert des Pivotelements
    int ipneu;                 // Index fuer Tausch und Endposition Pivotelement

    if (ir-il<=1) {            // nicht mehr als ein Element zu sortieren?
        return;                // ja, dann sind wir fertig.
    }

    pivot = a[imid];
    swap(a,imid,ir-1);         // vertausche Pivotelement mit dem rechten Rand

    ipneu = il;                // Ziel fuer Tauschvorgaenge liegt zunaechst am linken Rand
    for (int i=il; i < ir-1; i++) {
        if (a[i]<=pivot) {     // Ein Element gefunden, dass vom Pivotelement stehen muss?
            swap(a,ipneu,i);   // ja, dann tausche nach links
            ipneu++;           // Index fuer naechsten Tausch erhoehen
        }
```

```
    }
    swap(a,ir-1,ipneu);         // Pivotelement an die richtige Position platzieren

    quicksort(a,il,ipneu);      // alles links vom Pivotelement sortieren
    quicksort(a,ipneu+1,ir);    // alles rechts vom Pivotlelement sortieren
}
```

Quicksort: Eigenschaften

- Worst Case: Das Pivotelement ist z.B. stets das kleinste Element. Dann ist die linke Teilfolge leer und die rechte Teilfolge enthält nur ein Element weniger.
  Aufwand im Worst Case: $O(n^2)$
- Average Case: Das Pivotelement teilt die Folge in ungefähr gleich große Teilfolgen.
  Aufwand im Average Case: $O(n \log n)$
- Vergleich zu Mergesort:
  - Bei Mergesort fällt der Aufwand beim Zusammensetzen an, bei Quicksort bei der Zerlegung.
  - Bei gleichlangen Folgen ist die Zerlegung bei Quicksort effizienter als das Zusammensetzen bei Mergesort.
  - Dafür zerlegt Mergesort stets optimal, Quicksort nicht.
- In-Place Sortierung
- Quicksort ist nicht stabil.