marc-oliver pahl

Informatik II – Kapitel 12

„Sortier-Algorithmen“

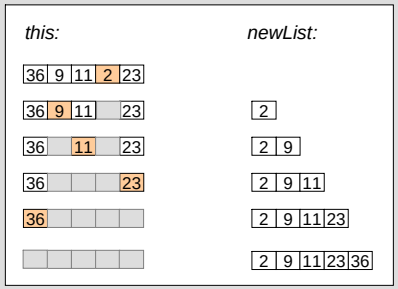
Zusammenfassung des Kapitel 12
Küchlin, Weber, Einführung in die Informatik, 2.Auflage

11.6.2004

- Greedy
  - SelectionSort (durch Auswahl)
  - InsertionSort (durch Einfügen)
  - BubbleSort (durch Vertauschen)
- Divide & Conquer
  - o Hard Split/ Easy Join
    - QuickSort
  - o Easy Split/ Hard Join
    - MergeSort

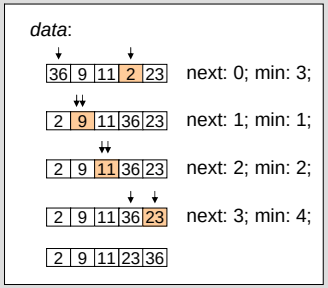

```
public OrderedList selectionSort(){
        int min;
        OrderedList newList = new OrderedList();

        while( head != null ){
                min = findMin();
                deleteElem(min);
                newList.insertLast(min);
        }
        return newList;
}
```

Anmerkung:
Diese Implementierung baut eine *neue Liste* auf, man könnte zum Sortieren auch die *Verkettungs-reihenfolge* auf der bestehenden Liste *ändern*.
Letztendlich läuft es aber auf fast dasselbe hinaus, da die *nicht mehr verlinkten ListNodes* automatisch durch den *garbage collector* im Speicher freigegeben werden.




```
Comparable[] data;
public void selectionSort(){
        int min;

        for( int next = 0; next < data.length-1; next++ ){
                min = findMin( next, data.length-1 );
                swap( next, min);
        }
}
```

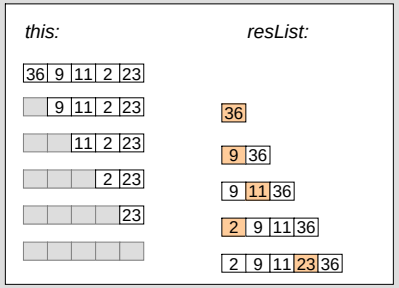
Anmerkung:
„Diese Implementierung führt nur **Vertauschungen** durch. Daher ist die **Reihung** sehr gut geeignet.“



Greedy.InsertionSort (*auf Liste*) 5

marc-oliver pahl

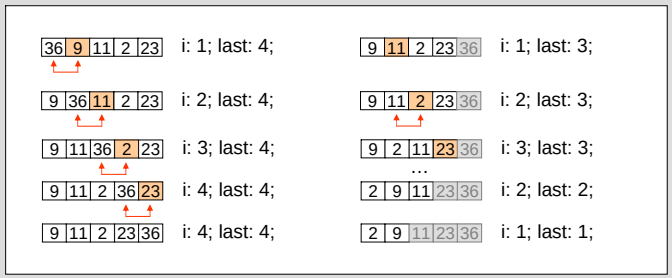
```
public OrderedList insertionSort(){
        Comparable first;
        OrderedList resList = new OrderedList();

        while( head != null ){
                first = takeFirst();
                resList.insertSorted(first);
        }
        return resList;
}
```



Greedy.BubbleSort (*auf Reihung*) 6

marc-oliver pahl

```
Comparable[] data;
public void bubbleSort(){
        for( int last = data.length-1; last > 0; last- - )
            for( int i = 1; i <= last; i++ ){
                if ( data[i-1].compareTo( data[i] ) > 0 ) swap( i-1, i);
}
```



D&C.QuickSort (*auf Reihung*) 7

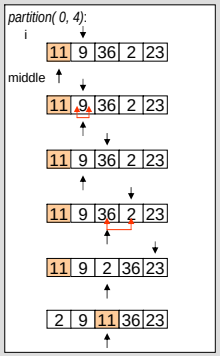
marc-oliver pahl

Hard Split/ Easy Join:

- Wähle ein Pivot-Element p; (in unserer Implementierung das erste El.)
- Teile die Folge ohne p in „<=p“ und „>p“;
- Fahre rekursiv mit den entstandenen Teilmengen fort bis diese einelementig sind;

Bentley Split:

```
private int partition(int left, int right) {
   int middle = left ;
   int pivot = data[left];
   for(int i = left + 1; i <= right; i++) {
     if (data[i] < pivot) {
       swap(++middle, i);
     }
   }
   swap(left, middle); // Pivot an die richtige Position
   return middle; // Position des Pivot
 }
```



D&C.MergeSort (*auf Liste*) 8

marc-oliver pahl

Easy Split/ Hard Join:

- Teile die Folge in der Mitte;
- Fahre rekursiv mit den entstandenen Teilmengen fort bis diese einelementig oder leer sind;
- Baue die Listen auf dem Rückweg durch die Rekursion sortiert zusammen.

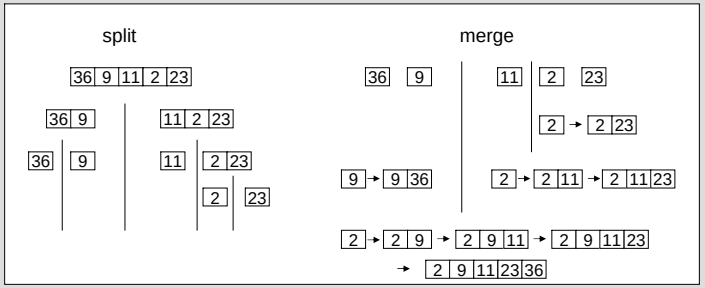
```
mergeSort( ) {
   „Wenn selbst nicht leer: Teile an (this.length-1) / 2 in aList und bList.“
   aList.mergeSort();
   bList.mergeSort();

   merge( aList, bList );
}

merge( List a, List b ) {
   „Betrachte jeweils das erste Element von a und b und lösche das Kleinere aus seiner Liste
und füge es als Nächstes an die neue Liste an.“
}
```

D&C.MergeSort (*auf Liste*)

marc-oliver pahl

```
mergeSort( ) {
   „Wenn selbst nicht leer: Teile an (this.length-1) / 2 in aList und bList.“
  aList.mergeSort();
  bList.mergeSort();

  merge( aList, bList );
}
```

```
merge( List a, List b ) {
   „Betrachte jeweils das erste Element von a und b und
lösche das Kleinere aus seiner Liste und füge es als Nächstes
an die neue Liste an.“
}
```



Zugrunde liegende Datenstruktur

marc-oliver pahl

Anhand der unterschiedlichen Struktur der Algorithmen bieten sich auch unterschiedliche Verfahren für unterschiedliche **zugrunde-liegende Datenstrukturen** an:

- **Verkettete Liste**:
  - Kein direkter Elementzugriff (-)
  - Einfügen an beliebiger Stelle möglich, ohne viele Elemente kopieren zu müssen (+)

  Daher besonders geeignet für **InsertionSort** und **MergeSort**.

- **Reihung**:
  - Einfügen von Elementen unter Beibehaltung der Sortierung erfordert das Kopieren vieler Elemente (alle nach der Einfügestelle) (-)
  - Direkter Element-Zugriff (+)

  Daher besonders geeignet für **SelectionSort**, **BubbleSort** und **QuickSort**, wo es keine Einfügungen gibt.