OOP: Übung 8 10.06.16

Christoph Van Heteren-Frese (Mi 8 Uhr)
John Nguyen & Anastasia Isaeva

Aufgabe 38: Stabile Sortierverfahren 10 Punkte

Bubble Sort ist ein stabiler Sortierverfahren, aber man muss bei der Implementierung auf die Vergleichsoperation (Zeile 10) achten. Bubblesort ist stabil, wenn der Vergleich von zwei benachbarten Elementen überprüft, ob das erste Element *echt größer* als das zweite Element. Wenn die Operation $\geq$ ist, dann ist Bubblesort nicht mehr stabil, denn die Positionen von zwei gleichen Elementen werden dann getauscht.

```
1   def bubblesort(A):
2       # end index of unsorted part
3       end = len(A) - 1
4       # while still unsorted
5       while end > 0:
6           last_change = 0
7           # for each elem in unsorted
8           for i in range(end):
9               # if out of order
10              if A[i] > A[i+1]:
11                  # swap
12                  A[i], A[i+1] = A[i+1], A[i]
13                  # note index
14                  last_change = i
15          # update new end index
16          end = last_change
```

Merge Sort ist ein stabiler Sortierverfahren, wenn man wie beim Bubblesort die Vergleichsoperation (Zeile 6) berücksichtigt. An dieser Stelle wird überprüft, welches Element von zwei Listen in die neue verschmolzene Liste eingefügt werden soll. Im Bezüg auf die ursprüngliche Reihenfolge kommen die Elemente in X vor den Elementen in Y vor. Das heißt, mit der $\leq$ Operation wird diese Reihenfolge beim gleichen Elementen belassen. Falls diese Operation $>$ ist, dann ist der Algorithmus nicht stabil.

```
1   def mergesort(A):
2       def merge(X, Y):
3           result = []
```

```
4            # while elements in both lists
5            while X and Y:
6               result.append(X.pop(0) if X[0] <= Y[0] else Y.pop(0))
7            # consume remaining elems
8            return result + (X if X else Y)
9
10        # base case
11        if len(A) <= 1: return A
12
13        # split lists & merge
14        mid   = len(A)//2
15        left  = mergesort(A[:mid])
16        right = mergesort(A[mid:])
17        return merge(left, right)
```

Insertion Sort ist auch stabil, wenn die passende Operation verwendet wird. In der Schleife an Zeile 9 wird das aktuelles Element e in die sortierte Teilliste eingefügt. An Zeile 11 wird die richtige Position innerhalb dieser Liste berechnet. Falls ein Element x, das gleich e ist, schon in der sortierten Liste liegt,

---

```
1     def insertionsort(A):
2        # sorted list
3        result = []
4        # while elements
5        while A:
6           # first elem
7           e = A.pop(0)
8           # for x in sorted
9           for x in result:
10             # if elem smaller
11             if e < x:
12                # insert before x
13                result.insert(result.index(x), e)
14                break
15          else:
16             # add to end
17             result.append(e)
18
19       return result
```
