The Bubblesort

Learning Targets:

- You have understood the principle of the bubble sort
- You can judge how efficient the bubble sort is

1 Sorting by steps

Use pieces of paper in order to do following exercise. Note down following numbers on pieces of paper:

51	**13**	**9**	**44**	**18**	**93**	**25**

1.1 Exercise: sort by size

Sort the pieces of paper according to number size, so that the smallest number is left and the largest number is right.

- ➔ Can you describe what steps you took to sort these numbers?

1.2 Exercise: only swap neighbouring pieces

Place the pieces back into their initial position. Now do the sorting again. **But** you are only allowed to do **one operation**: you can only swap two neighbouring pieces.

1.3 Exercise: systematic from left to right

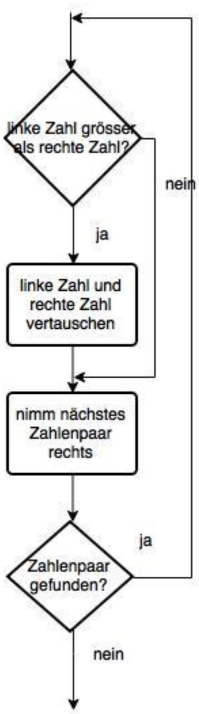
Place the pieces back into their initial position. Sort again by swapping neighbours, however this time we will work systematically from left to right. You swap, if necessary, the first with the second number, then the second with the third and so on.

- ➔ What happens with one single pass?
- ➔ When can you stop and there are no more passes necessary? Note down how many steps you had to take.

2 Definition Bubblesort-Algorithm

The bubble sort algorithm sorts a list of elements ascending, by going through the list from left to right and swapping the neighbouring element (if the left element is larger than the right one). The pass through the list is repeated until no element has to be swapped anymore.

Here’s a flow chart:



2 Efficiency of Bubblesort

2.1 Number of passes

Think how many passes are necessary in a list of *n* elements. Sort following list and note down all intermediate steps :

97 15 33 28 25 11 73

As you might have guessed, *n* passes are sufficient. With each pass at least one element is put into its final position. (Even *n-1* passes are enough, since the last number has no neighbour which can be swapped).

2.2 Number of Comparisons

In a list of *n* numbers there are *n-1* pairs of neighbouring numbers, which must be compared in one pass. And we have *n* passes as a maximum. Therefore *n(n-1)* number of comparisons are necessary.

We can therefore say in general that roughly n^2 number of comparisons exists for a list with *n* elements.

2.3 Effort in the best case

If the list is already sorted, the algorithm compares all *n-1* neighbouring pairs one time and then realizes that there's nothing to be sorted. This is the quickest and best case, needing *n-1* comparisons and 0 swaps.

Source: ETH-Script, August 2015