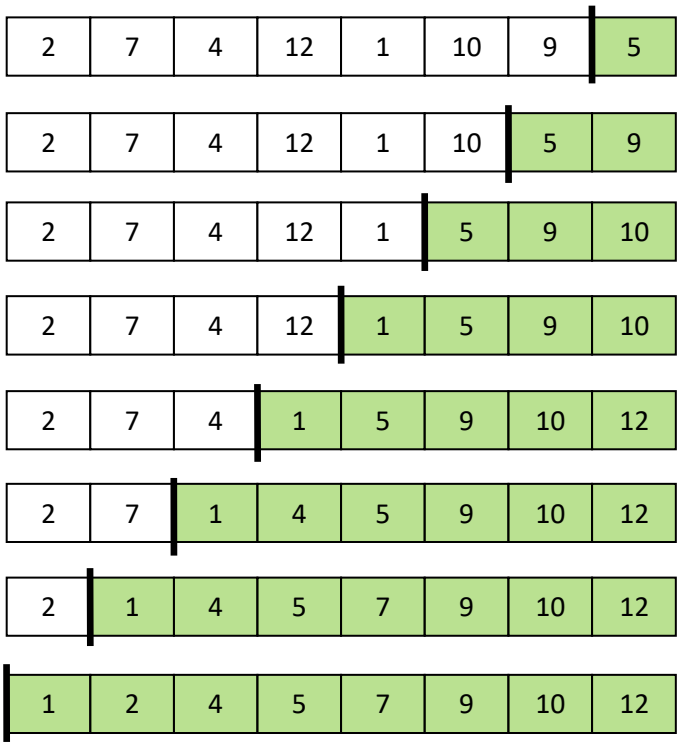
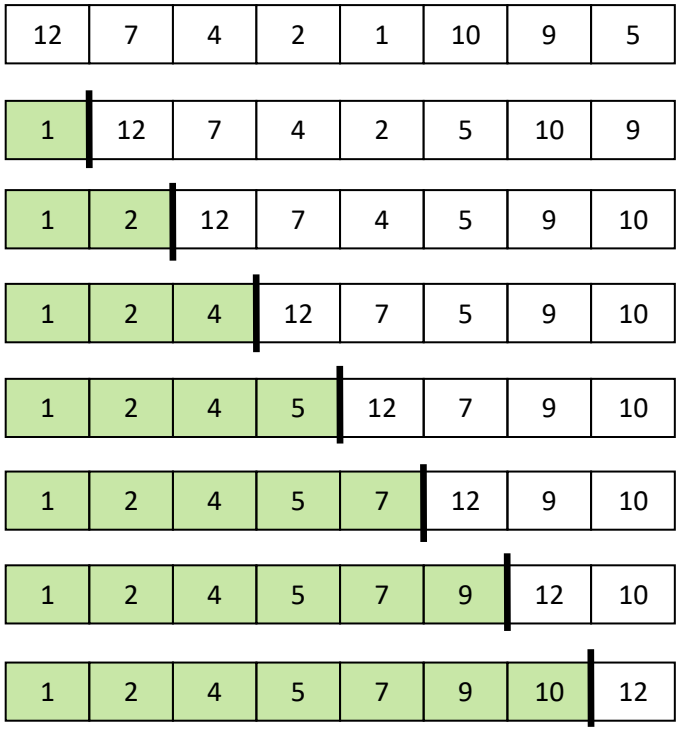
PWr Week 6
Data Structures and Algorithms,

Introduction

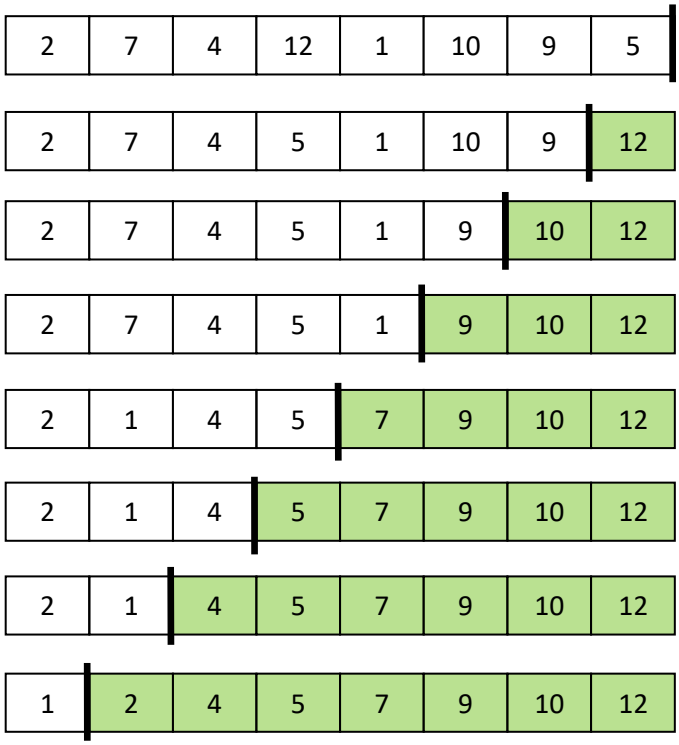
There are many simple sorting algorithms. One of them is an insert sort algorithm. It has many variants, let consider the one in which the data is sorted in increasing order, and the sorted part grows from the end of the array. In this case we have to insert a next element from unsorted left part (the one on the far right) to the sorted part on proper position. So the sequence on states of an example array will be as follows (before the first step we can assume, that last element is sorted):



Another simple algorithm is a bubble sort algorithm. It has many variants, let consider the one in which the data is sorted in increasing order, and the sorted part grows from the begin of the array. the idea of the algorithm is based on comparing neighboring elements, moving from the end of the table to the beginning and swapping the elements, if they are in an incorrect order. After this step the first element will be on correct final position. In next step we do exactly the same operations, but without last comparison, because the first element is set correct. So in every next steps the number of comparisons decrease by one. The states of an example array of values is presented on next page.



The last simple sorting algorithm in this task is the select sort algorithm. The idea of this sorting is to find the minimal (or maximal) part of the unsorted part and attach it to the previously sorted part. In the case of an array, the attachment consists in the exchange with the element in the next position to be inserted. A variant we consider here will be such, that data will be sorted in increasing order and the sorted part will grow from right side. In this case the states of an example array of values will be as follows:



Task list

Laboratory – **List 5 (version 1)**

1. For a document from Laboratory - List 4 prepare a function `getWeights()` which will create and return an array of integer number taken from links of the document (in an order in which they was in list of links).
2. Prepare a method `showArray()`, which will present a state of an array of integers in one line, every number separated by one space (no space after last number).
3. Implement function `void insertSort()` which sort (using insertsort) in **increasing** order in which the sorted part of array grows from **right side** (starting from the higher index). Print state of the array (using `showArray`) before you start sort it and after each execution of outer loop.
4. Implement function **`void`** `bubbleSort()` which sort (using bubble sort) in **increasing** order in which the sorted part of array grows from **left side** (starting from the zero's index). Print state of the array (using `showArray`) before you start sort it and after each execution of outer loop.
5. Implement function **`void`** `selectSort()` which sort (using selection sort) in **increasing** order in which the sorted part of array grows from **right side** (starting from the zero's index). Print state of the array (using `showArray`) before you start sort it and after each execution of outer loop.

For 100 points present solutions for this list till Week 7.
For 80 points present solutions for this list till Week 8.
For 50 points present solutions for this list till Week 9.
After Week 9 the list is closed.

Appendix
It have to work all operation from previous list but additionally the following:

If a line has a format:
`bubblesort`
your program has to call `bubbleSort()` for current document.

If a line has a format:
`insertsort`
your program has to call `insertSort()` for current document.

If a line has a format:
`selectsort`
your program has to call `selectSort()` for current document.

The example from previous task is also proper for this list of tasks. A simple test for sorting:

```
#Test for Lab5
go 10
ld doc1
link=c(10) and link=b(7)
write also link=z end finish link=a(20)
eod
add f(8)
add g(4)
add e(3)
show
bubblesort
insertsort
ha
```

The output have to be:

```
START
!go 10
!ld doc1
!add f(8)
true
!add g(4)
true
!add e(3)
true
!show
Document: doc1
a(20) b(7) c(10) e(3) f(8) g(4) z(1)
!bubblesort
20 7 10 3 8 4 1
1 20 7 10 3 8 4
1 3 20 7 10 4 8
1 3 4 20 7 10 8
1 3 4 7 20 8 10
1 3 4 7 8 20 10
1 3 4 7 8 10 20
!insertsort
20 7 10 3 8 4 1
20 7 10 3 8 1 4
```

```
20 7 10 3 1 4 8
20 7 10 1 3 4 8
20 7 1 3 4 8 10
20 1 3 4 7 8 10
1 3 4 7 8 10 20
!ha
END OF EXECUTION
```