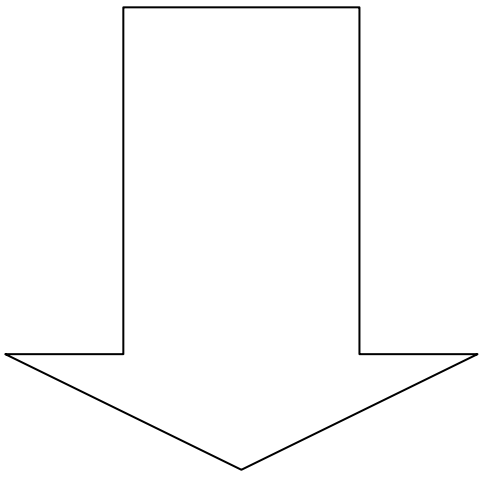
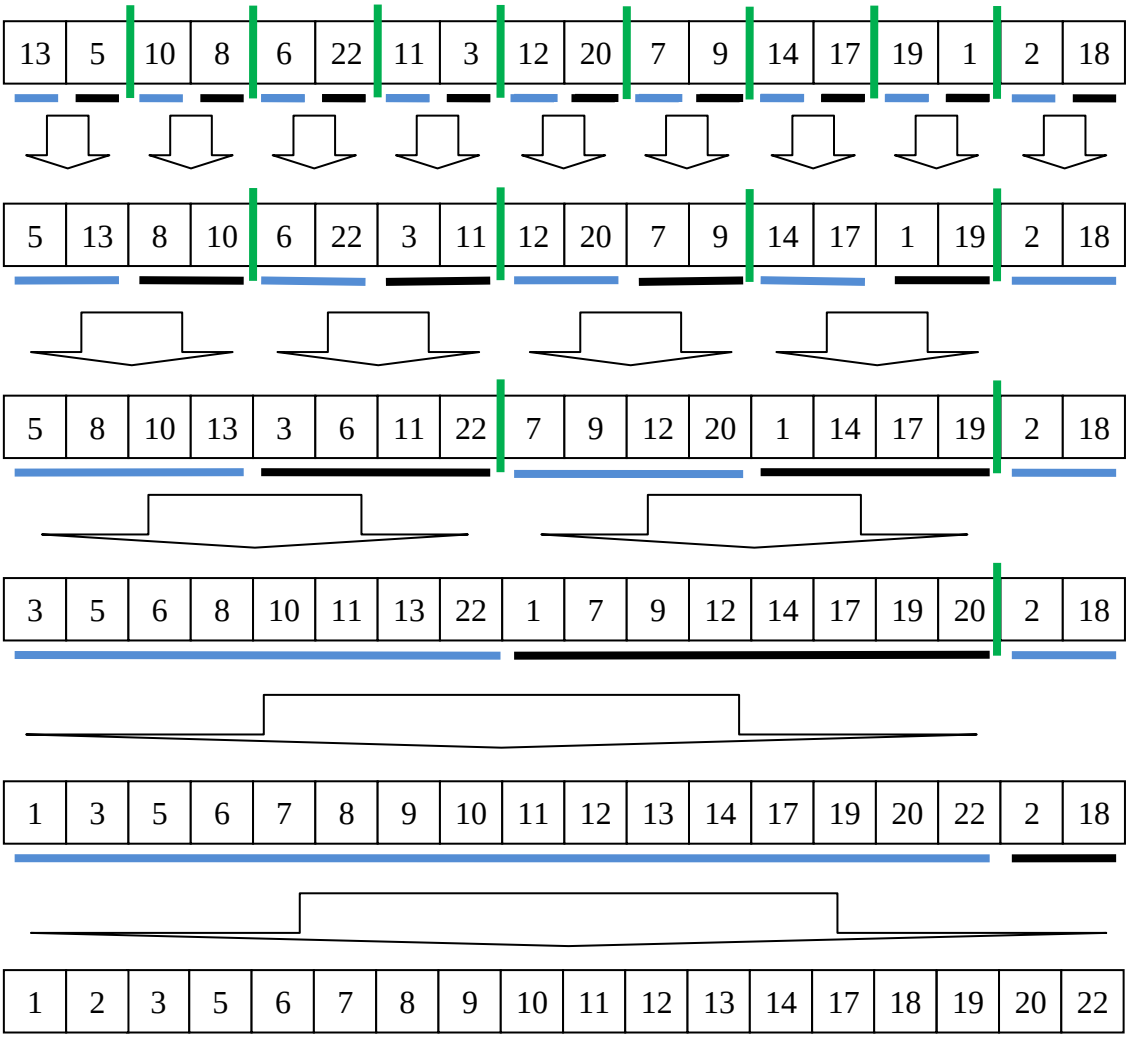
PWr Week 5
Algorithms and Data Structures
Laboratory – **List 5**

1. Implement function **`void`** `showArray(`**`int`** `array[], `**`int`** `size)` which present the array in one line: after every value put a comma.
2. (2pts) Implement function **`void`** `insertSort(`**`int`** `array[], `**`int`** `size)` 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 after each execution of outer loop.
3. (2pts) Implement function **`void`** `bubbleSort(`**`int`** `array[], `**`int`** `size)` 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 after each execution of outer loop.
4. (3pts) Implement iterative version on mergesort algorithm. Choose a version in which first we assume every one-element array is sorted. Then we merge every pair of consecutive one-element arrays into two-element sorted arrays. In next step every pair of consecutive two-element arrays into four-element sorted arrays, and so on. which present the array in one line: after every value put a comma. Implement function **`void`** `mergeSortIter(`**`int`** `array[], `**`int`** `size)` for sorting in increasing order. Print state of the array after each execution of outer loop.
5. (3pts) Compare times of the sorting algorithms from tasks 2-4 for 100, 1.000, 2.000, 5.000, 10.000, 50.000, 100.000 numbers. Is it consistent with the theory? Write your own another `main()` function to do that. Prepare charts with time results.

For **10 points** present solutions for this list till **Week 6**.
For 8 **points** present solutions for this list till **Week 7**.
For 5 **points** present solutions for this list till **Week 8**.
After Week 8 the list is closed.

Graphical explanation for iterative mergesort (on next page):




13 5 10 8 6 22 11 3 12 20 7 9 14 17 19 1 2 18
5 13 8 10 6 22 3 11 12 20 7 9 14 17 1 19 2 18
5 8 10 13 3 6 11 22 7 9 12 20 1 14 17 19 2 18
3 5 6 8 10 11 13 22 1 7 9 12 14 17 19 20 2 18
1 3 5 6 7 8 9 10 11 12 13 14 17 19 20 22 2 18
1 2 3 5 6 7 8 9 10 11 12 13 14 17 18 19 20 22

Appendix 1

The solution of task 2,3,4 will be automated tested with tests from console of presented below format.

If a line starts from '#' sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

If a line has a format:
`IS` *n*
in next lines there will be *n* integer numbers separated by space or newline. The array have to be sorted using `insertSort` function, writing on the console state of array after every big step of the algorithm. Before start the function write in one line "insertSort" and initial state of the array in next line.

If a line has a format:
`BS` *n*
in next lines there will be *n* integer numbers separated by space or newline. The array have to be sorted using `bubbleSort` function, writing on the console state of array after every big step of the algorithm. Before start the function write in one line "bubbleSort" and initial state of the array in next line.

If a line has a format:
`MI` *n*
in next lines there will be *n* integer numbers separated by space or newline. The array have to be sorted using `mergeSortIter` function, writing on the console state of array after every big step of the algorithm. Before start the function write in one line "mergeSortIter" and initial state of the array in next line.

If a line has a format:
`HA`
your program has to end the execution, writing as the last line "END OF EXECUTION".
Every test ends with this line.
For example for input test:

```
IS 5
6 9 4 7 5
BS 4
3 9 6 8
MI 18
13 5 10 8 6 22 11 3 12 20 7 9 14 17 19 1 2 18
MI 1
13
MI 2
1 3
HA
```

The output have to be:

```
START
!IS 5
6,9,4,7,5,
6,9,4,5,7,
6,9,4,5,7,
6,4,5,7,9,
4,5,6,7,9,
!BS 4
3,9,6,8,
3,6,9,8,
3,6,8,9,
3,6,8,9,
!MI 18
13,5,10,8,6,22,11,3,12,20,7,9,14,17,19,1,2,18,
5,13,8,10,6,22,3,11,12,20,7,9,14,17,1,19,2,18,
5,8,10,13,3,6,11,22,7,9,12,20,1,14,17,19,2,18,
3,5,6,8,10,11,13,22,1,7,9,12,14,17,19,20,2,18,
1,3,5,6,7,8,9,10,11,12,13,14,17,19,20,22,2,18,
1,2,3,5,6,7,8,9,10,11,12,13,14,17,18,19,20,22,
!MI 1
13,
!MI 2
1,3,
1,3,
!HA
END OF EXECUTION
```