In this exercise we study sorting algorithms.

Excercise 3a. (Optional exercise. Implement, test and compare the performance of Insertion sort against Arrays.sort() for an array of strings, 0.5p)

Implement an insertion sort function with the similar function prototype than Java library's `Arrays.sort()`, e.g.

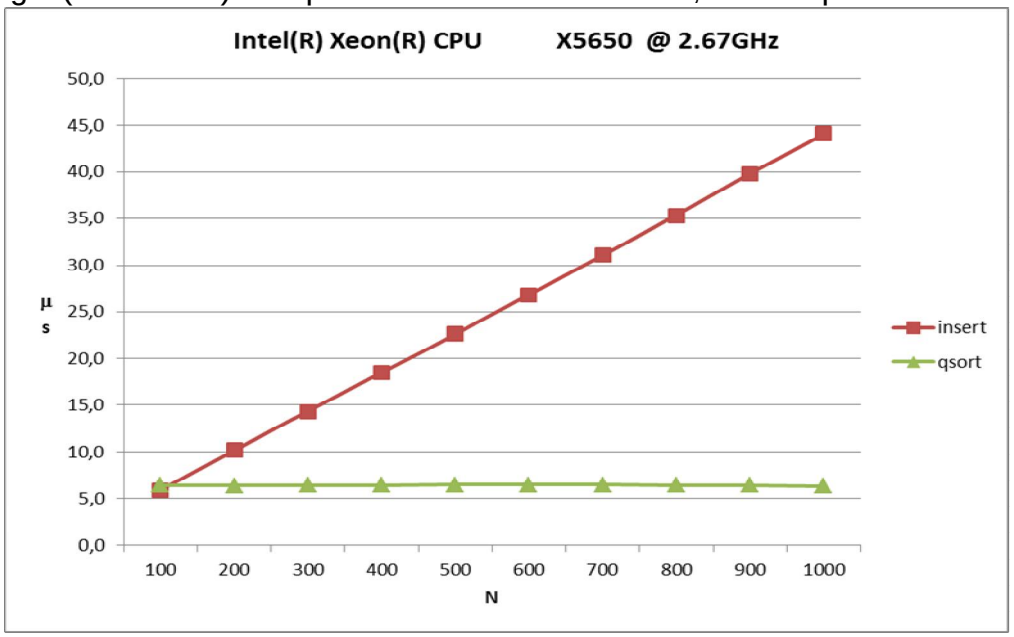
```
static void sort(Object[] a);
```

where a is the array of the data to be sorted.

After you have implemented the sort function, you can test it using random array of strings (length 32 characters).

After checking that the insertion sort function works correctly, measure the execution time of the insertion sort function and compare it to the execution time of `Arrays.sort()`[1] library function. Plot the execution times of both the insertion sort and Java library sort against array sizes from 1000 elements to 10000 in steps of 1000 elements.

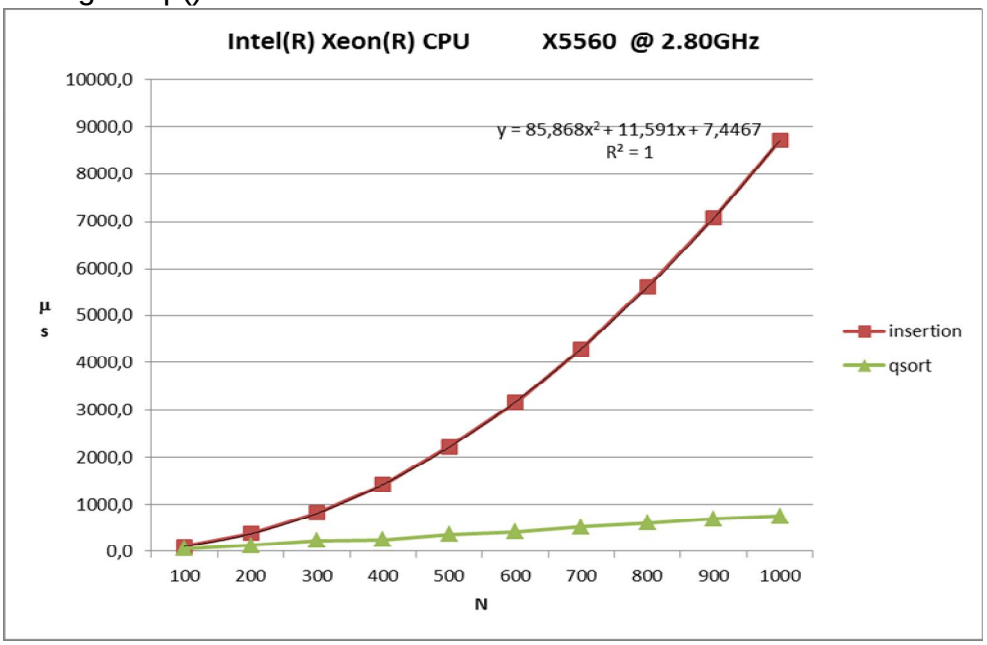
Be careful how to setup the measurement environment when measuring those execution times. Material to be sorted must be always random. If you are sorting already sorted array you get (unrealistic) best performance characteristics, like the picture below.



From the picture we can see that the performance of insertion sort is O(n) when the array is already sorted, which is the best performance characteristics of that sort algorithm. Also interesting is that with a small array (100 elements) to be sorted, execution time of the insertion sort is smaller than the execution time of the quicksort (actually if the array is already sorted, insertion sort is really faster than quicksort because no element movements are needed at all, only one pass of element checking is needed).

[1] Which is QuickSort based

If you make is sure that the array is always random, then you should get performance curves like below. In order to help to maintain the randomness of the array to be sorted, a new method, setup() is added to the Test interface in StopWatch measurement class. This setup() method is called before the test() method is called, but the execution time is not measured during setup().



From the picture we can clearly see that the performance of insertion sort is really $O(n^2)$.

Excercise 3b. (Optional exercise. Implement, test and compare the performance of bubble sort against Insertion Sort (or qsort()) for an array of strings, 0,25p)

Operation of Bubble sort is very easy to understand. Good description can be found from `http://en.wikipedia.org/wiki/Bubble_sort`. Pseudocode (describes the operation of the algorithm, but is not a real programming language) description is here

```
procedure bubbleSort( A : list of sortable items )
  repeat
    swapped = false
    for i = 1 to length(A) - 1 inclusive do:
      if A[i-1] > A[i] then
        swap( A[i-1], A[i] )
        swapped = true
      end if
    end for
  until not swapped
end procedure
```

Bubble sort compares every two element on the array and swaps their positions until no swaps are needed anymore (the array is sorted).

Implement the bubble sort as described by the given pseudocode. Test that it works.

Then compare the execution times of bubble sort to the execution time of insertion sort with different sizes of arrays (e.g. 100, 200, 300,… ,1000). Just for fun compare insertion sort performance to bubble sort also in case when array is already sorted.