Assignment 1

Alessandro Artale, Werner Nutt,
Ognjen Savković, Francesco Sportelli

1. Array Utility Class[1]

Instructions: Your assignment should represent your own effort. However, you are not expected to work alone. It is fine to discuss the exercises and try to find solutions together, but each student shall write down and submit his/her solutions separately. It is good academic standard to acknowledge collaborators, so if you worked together with other students, please list their names.
For a programming task, your solution must contain (i) an explanation of your solution to the problem, (ii) the Java code, in a form that we can run it, (iii) instructions how to run it. Also put the source code into your solution document. For all programming tasks, it is not allowed to use any external libraries ("import") if not stated otherwise.
Please, include name, student ID and email address in your submission.

1. Implementation of the basic operations of the class ArrayUtility

Implement in Java the class `ArrayUtility`, which offers basic operations over one-dimensional and two-dimensional arrays. *All* methods *must* be implemented as class methods (i.e., `static` methods). The signature of the methods in the `ArrayUtility` class are the following:

1. `public static int findMax(int[] A, int i, int j)`: returns the maximum value occurring in the array `A` between position i and j.
2. `public static int findMaxPos(int[] A, int i, int j)`: returns the position of the maximum value in the array `A` between position i and j.
3. `public static int findMin(int[] A, int i, int j)`: returns the minimum value in the array `A` between position i and j.
4. `public static int findMinPos(int[] A, int i, int j)`: return the position of the minimum value in the array `A` between position i and j.
5. `public static void swap(int[] A, int i, int j)`: swaps the elements in position `i` and `j` in the array `A`.

[1] Exercises authored by Valeria Fionda, Mouna Kacimi, Werner Nutt, and Simon Razniewski in the academic year 2012/13

6. `public static void shiftRight(int[] A, int i, int j)`: shifts to the right all the elements of the array `A` starting from position `i` and until position `j` (i.e., moves the element in position k to position $k+1$ for all $i \leq k < j$, and leaves position i unchanged).

7. `public static void shiftLeft(int[] A, int i, int j)`: shifts to the left all the elements of the array `A`, from position `j` down to position `i` (i.e., moves the element in position k to position $k-1$ for all $i < k \leq j$, and leaves the position j unchanged).

8. `public static int[] createRandomArray(int size, int min, int max)`: creates and returns an array of size `size`, of random elements with values between `min` and `max` (use the `Math.random()` method of Java!).

9. `public static int[][] createRandomMatrix(int rows, int cols, int min, int max)`: creates and returns a two-dimensional array with `rows` rows and `cols` columns of random elements with values between `min` and `max` (use the `Math.random()` method of Java!).

10. `public static int[] copyArray(int[] A)`: returns an array that is a copy of `A`.

11. `public static int[][] copyMatrix(int[][] A)`: returns a two-dimensional array that is a copy of `A`.

12. `public static int findInArray(int[] A, int q)`: returns the position of the number `q` in the array `A` (returns -1 if `q` is not present in `A`).

13. `public static int findInSortedArrary(int[] A, int q)`: returns the position of the number `q` in the *sorted* array `A` (returns -1 if `q` is not present in `A`).

    The method assumes that the array `A` is sorted, it need not be correct if `A` is not sorted. Exploit the fact that the array is sorted to find an efficient algorithm. (Remember the Google interview questions!)

14. `public static int findFirstInSortedArrary(int[] A, int q)`: returns the *first* position where the number `q` occurs in the *sorted* array `A` (returns -1 if `q` is not present in `A`).

    As before, the method assumes that the array `A` is sorted and need not be correct if `A` is not sorted. Again, exploit the fact that the array is sorted to find an efficient algorithm.

(18 Points)

2. Running Time Comparison — Maxsort

Add to your class `ArrayUtility` two `static` methods implementing the algorithm *Maxsort*, that takes an unsorted array of integer numbers as input and sorts it in descending order, by repeatedly doing the following:

- first, it searches in the whole array for the greatest element;
- it then puts this element to the beginning of the array;
- then, it searches the whole array excluding the first element for the greatest value, and puts it to the second position.

Implement the algorithm according to two different strategies:

- By using the method `shiftRight(int[] A, int i, int j)`: if the maximum element is found in position j and needs to be put into position i, then (i) shift `A` to the right, starting from position i, while remembering the element in position j that will be overridden; (ii) copy the remembered element to position i.
- By using the method `swap(int[] A, int i, int j)`: if the maximum element is found in position i and needs to be put into position j, then use `swap` to exchange the element in position i with the element in position j.

The perform tests to find out which of the two implementations is faster. Is there an array size for which the running times crosse over? (A size N would be such a cross-over point if for inputs of size less than N, the running times of one algorithm are better, while for inputs of size greater than N, the running times of the other algorithm are better.)
To perform your measurements, write a test class that

1. creates random arrays of size $n = 10$, 100, 1000, etc., and
2. for each array created, sorts it using the two implementations of *Maxsort* and measures the running times; to measure the running time use the Java method `System.nanoTime()` in the following way:

```
long startTime = System.nanoTime();
 ...  the code being measured ...
long estimatedTime = System.nanoTime()-startTime;
```

(12 Points)

Deliverables. For each question, hand in the code that you wrote. For Question 2, write also a short report where you describe your measurements and the results you observed.

Submission: Until Mon, 9 March 2015, 11:55 pm, to

`dsa-submissions AT inf DOT unibz DOT it.`