Assignment 1

David Blumenthal, Florian Hofer, Werner Nutt, Daniele Porello

1. Array Utility Class, Euclidean Algorithm

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`.

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 *a* (not *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. Euclidean Algorithm

The algorithm `gcd(int a, int b)` is a simple version of the Euclidean Algorithm, which computes the greatest common divisor of two positive integers `a` and `b`:

```
int gcd(int a, int b) {
    while (a != b) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}
```

Write down proofs for the following statements:

1. The algorithm `gcd(int a, int b)` terminates for all positive integers `a` and `b`.

2. The algorithm `gcd(int a, int b)` is correct, i. e., always computes the greatest common divisor of `a` and `b`.

(12 Points)

Deliverables. For Question 1, hand in the code that you wrote. For Question 2, hand in a PDF document containing your proofs.
Combine all deliverables into one zip file, which you submit via the OLE website of the course.

Submission: Until Tue, 14 March 2017, 23:55 hrs, to the OLE submission page of

Lab A / Lab B / Lab C