First mid term lab exam – PRG – ETSInf. Academic year 2015-16
April 11th, 2016. Duration 1 hour

(Notice: This exam is evaluated over 10 points, but its weight in the final grade of the subject is 0,8 points).

NAME:	LAB GROUP:

1. [2 points] **Answer** the following questions about the first lab practice.

   a) If there are 30 disks in the `origin` needle, what do we have to do with the 29 disks that are on the disk with the widest diameter before moving it to the `target` needle?

   > **Solution:** Move them to the `temporary` needle.

   b) When the method `hanoi()` is executed for moving more than one disk, to which case, trivial or general, is associated the first call to the method `moveDisk( origin, target )`?

   > **Solution:** The first call to `moveDisk( origin, target )` is associated to the trivial case.

2. [3 points] The recursive method `posFracSep( String )` studied in lab practice 2 determines the position of the delimiter of the decimal part of a well-formatted real number stored in a `String` . Both the dot `.` and the comma `,` are accepted as delimiters.

   What to do: complete the recursive method `posFracSep( String )` with the necessary instructions to ensure it is correct, by taking into account that the desired solution performs a descending recursive decomposition of the `String` given as input parameter. In other words, at each step the input string is split in the two parts, the last character in the string and the slice from the position 0 up to the position before the last (`s.length()-2`).

```
/** Returns the position of the character used as delimiter of the decimal part or
  * -1 if it doesn't appear in the string.
  * @param s String with the well-formatted real number
  * @return int position of the delimiter or -1 if it doesn't appear
  * PRECONDITION: s contains a well-formatted real number.
  */
public static int posFracSep( String s )
{
    int last = s.length() - 1;
    if ( s.charAt(last) == '.' || s.charAt(last) == ',' ) { return last; }
    else { return posFracSep( s.substring( 0, last ) ); }
}
```

   Reminder: `s.substring(i, j)` returns another `String` object with the substring of `s` containing the characters from position `i` up to `j-1`.

Solution:

```
public static int posFracSep( String s )
{
    if (s.length() == 0) {
        return -1;
    } else {
        int last = s.length() - 1;
```

```
            if ( s.charAt(last) == '.' || s.charAt(last) == ',' ) { return last; }
            else { return posFracSep( s.substring( 0, last ) ); }
        }
    }
```

3. 5 points We need to measure the running time in average of an algorithm for sorting arrays of integers. The way of executing such method for an array `a` is the following:

```
MeasurableAlgorithms.sort(a);
```

For improving the accuracy it is better to run the algorithm several times and get the average after a given number of repetitions (`numReps`).

Additionally, we have available the following methods for creating arrays of a given size with different arrangements:

```
private static int[] createRandomArray( int s )
private static int[] createSortedArrayInAscendingOrder( int s )
private static int[] createSortedArrayInDescendingOrder( int s )
```

and we can use the following method provided by the system for reading the time of the system with high precision as you saw in lab practice 3:

```
long t = System.nanoTime();
```

What to do: write a method with the following header:

```
public static double averageTimeInMillis( int s, int numReps )
```

that returns as a result the running time in milliseconds for the average case of the method `MeasurableAlgorithms.sort( int [] )` for a given input size `s` and a number of repetitions `numReps`.

You have to choose the most appropriate method for creating arrays according to the case you are going to measure the behaviour of the method. And it is important to remind you that it has no sense to sort an already sorted array.

Solution:

```
public static double averageTimeInMillis( int s, int numReps )
{
    final double NANOS_MILIS = 1e6;
    int [] a;
    long tStart, tEnd;
    double accum = 0;
    for( int i = 0; i < numReps; i++ ) {
        a = createRandomArray(s);
        tStart = System.nanoTime();
        MeasurableAlgorithms.sort(a);
        tEnd = System.nanoTime();
        accum += tEnd - tStart;
    }
    return accum / (numReps * NANOS_MILIS);
}
```