PRG - ETSInf. THEORY. Academic Year 2012/2013
Recovering of first partial exam
June 17th, 2013. Duration: 1h 50min.

1. 2.5 points In order to find out if a non-negative number `n` can be expressed in a given base `b` ($2 \leq b \leq 10$), that's enough to check all their digits are lower than the base `b`.

   Let us see an example, the number 453123 could be a value represented in the following bases: 6, 7, 8, 9 and 10 because all its digits are strictly lower than any of these possible bases.

   To be done:

   Implement a **recursive** method using the Java programming language that given `n` and `b` as parameters solves the posed problem. You have to specify the trivial case and the general case for the recursion.

   Solution:

```
/** Returns if it possible that 'n' is a number in base 'b'
  * PRECONDITION: 2<=b<=10 and n >=0 */

public static boolean possibleInBase( int n, int b )
{
    if ( n == 0 ) {   // trivial case
        return true;
    } else {          // general case
        int lastDigit = n%10;
        if ( lastDigit < b )
            return possibleInBase( n/10, b );
        else
            return false;
    }
}
```

2. 2.0 points Given an array `a` of real numbers and a given value `x`, it is needed to discover, **recursively**, how many components of `a` are greater than `x` from the position `pos` up to the end (`a[pos .. a.length-1]`). The profile of the method must be the following:

```
public static int countGreaterThan( double[] a, double x, int pos )
```

   To be done:

   - Implement the **recursive** method specifying the trivial case and the general case of the recursion.
   - Write the initial call for calculating how many elements of the whole `a` are greater than `x`.

   Solution:

```
/**
  * Computes how many elements greater than 'x' exist in 'a'
  * from pos up to the end.
  */
public static int countGreaterThan( double[] a, double x, int pos )
{
    if ( pos >= a.length )
        return 0;                                  // trivial case
    else if( a[pos] > x )
        return 1+countGreaterThan( a, x, pos+1 ); // general case
    else
        return countGreaterThan( a, x, pos+1 );   // general case
}
```

Initial call: `int num = countGreaterThan( v, x, 0 );` given that `v` and `x` fulfil the preconditions given by the method profile.

3. 3.0 points Given a squared matrix `m` of real numbers, the following method allows to discover whether it is a lower triangular matrix, i.e., all its values stored in positions upper to the main diagonal are zeroes.

```
/**
  * Checks if a given squared matrix is a lower triangular one, i.e.,
  * all its values in positions higher than the main diagonal are zeroes.
  */
public static boolean isLowerTriangular( double[][] m )
{
    boolean itIs = true;

    for( int i=0; i < m.length && itIs; i++ )
        for( int j=i+1; j < m.length && itIs; j++ )
            itIs = (m[i][j] == 0);

    return itIs;
}
```

To be done: analyse the temporal cost of the given algorithm by following the next steps:

a) Determine the input size of the problem and write the expression that represents it.

> **Solution:** The input size is $n =$ `m.length`, i.e., the number of rows or columns, both are equal because the matrix is squared.

b) Choose a unit of measure for estimating the cost (program step or critical instruction), determine which elementary operations can be considered as critical instruction and select one of them.

> **Solution:** The following instructions can be used as critical instructions or program steps:
>
> - the condition of the most inner loop: `( j < m.length && itIs )`

- the auto increment of the variable used to control the loop: `( j++ )`
- the body of the most inner loop: `itIs = (m[i][j] == 0);`

We prefer to use the condition of the most inner loop, but we can select `( j < m.length )` too.

c) Identify if there exist significant instances. In the affirmative case indicate the best and worst cases.

Solution: The method solves a search problem because of the use of the `boolean` variable `itIs` in the conditions of both loops. So we case say it has significant instances.

Best case: When `m[0][1]!=0` both loops end their execution after their first iteration.

Worst case: When the matrix is a lower triangular matrix, all the values stored in positions higher than the main diagonal are zeroes. In this case both loops do all the possible iterations.

d) Obtain the mathematical expression as precise as be possible for the temporal cost function $T(n)$. If there exist significant instances then both functions must be obtained, one for the best case $T^b(n)$ and another for the worst case $T^w(n)$.

Solution: Taking the condition of the most inner loop as a reference of program step we get:

- *Best case:* $T^m(n) = 1$ program step.
- *Worst case:* $T^p(n) = \sum_{i=0}^{n-1}(\sum_{j=i+1}^{n-1} 1) = \sum_{i=0}^{n-1}(n-i-1) = n^2/2 - n/2$ program steps.

e) Express the results using asymptotic notation.

Solution: $T(n) \in \Omega(1)$, $T(n) \in O(n^2)$.

4. 2.5 points The following recursive method checks if two `String` objects equally long are symmetric. We can say two string of chars are symmetric when the first char of one of them is equal to the last char of the other, and the second char of one of them is equal to the char previous to the last one of the other, and so on.

Let us see an example, `"HOLA"` and `"OLAH"` are symmetric, whereas `"HOLA"` and `"ALAH"` are not.

```
/**
  * Checks if 'a' and 'b' are symmetric
  * PRECONDITION: a.length()==b.length()
  */
public static boolean symmetric( String a, String b )
{
    if ( a.length() == 0 ) {
        return true;
    } else {
        int last = b.length()-1;
        return a.charAt(0)==b.charAt(last) &&
```

```
                symmetric( a.substring(1), b.substring(0,last) );
    }
}
```

To be done: analyse the temporal cost of the method, knowing that **all the operations of the class** `String` used inside the method **have constant temporal cost**.

a) Determine the input size of the problem and write the expression that represents it.

> **Solution:** The input size is the length of each input `String` (`n=a.length()`).

b) Choose a unit of measure for estimating the cost (program step or critical instruction), determine which elementary operations can be considered as critical instruction and select one of them.

> **Solution:** This is a recursive method, so the critical instruction is the condition used for distinguishing the trivial case against the general case: `a.length() == 0`.

c) Identify if there exist significant instances. In the affirmative case indicate the best and worst cases.

> **Solution:** Yes, they exist significant instances because the method could end the recursion before reaching the trivial case. When the first pair of chars to be compared is not equal the method does not continue the recursive call. So we can say it performs a search, and a search always has significant instances.
>
> The *best case* is when the first pair of chars to be compared are not equal, none recursive call is executed. The *worst case* is when both strings of chars are symmetric.

d) Obtain the mathematical expression as precise as be possible for the temporal cost function $T(n)$. If there exist significant instances then both functions must be obtained, one for the best case $T^b(n)$ and another for the worst case $T^w(n)$. Since this is a recursive method we recommend to use the substitution method.

> **Solution:**
>
> - Best case: $T^m(n) = 1$ program steps.
> - Worst case: $T(n) = 1$ if $n = 0$ and $T(n) = T(n-1)+1$ if $n > 0$. Then, $T^p(n) = T(n) = n+1$ program steps.

e) Express the results using asymptotic notation.

> **Solution:** $T(n) \in \Omega(1)$, $T(n) \in O(n)$.