Computer Programming – Theory – ETSINF – Academic year 2018/2019
Retake of first mid term exam – June 11th, 2019 – duration two hours

Note: The maximum mark for this exam is 10 points, but its weight in the final grade of PRG is **3 points**.

1. $\boxed{\text{4 points}}$ Given any integer $n > 0$, you have to design and implement a recursive method that writes on standard output the figures of n in the reverse order, followed by the figures of n in the direct order. With two examples you will better catch the idea. If $n = 4873$ then the method must show 37844873, if $n = 48$, then the method must show 8448, and if $n = 4$ then it must show 44.

   To do:

   a) (0.75 points) Profile of the method with the corresponding precondition.

   b) (1.25 points) Trivial case and general case.

   c) (2 points) Implementation in Java.

Solution:

a) A possible solution is:

```
/** Precondition: n > 0 */
public static void showFigures( int n )
```

b)
- Trivial case: `n` ≤ 9, only one figure.
- General case: `n` ≥ 10, more than one figure.

c)
```
/** Precondition: n > 0 */
public static void showFigures( int n )
{
    if ( n <= 9 ) {
        System.out.print( n + "" + n );
    } else {
        int figure = n % 10;
        System.out.print(figure);
        showFigures( n / 10 );
        System.out.print(figure);
    }
}
```

2. $\boxed{\text{3 points}}$ An *upper triangular matrix* is an square matrix with all the elements below the main diagonal equal to zero. The iterative method `isUpperTriangular` checks if any matrix `m` has set to zero all the elements under the main diagonal.

```
/** Precondition: m is an square matrix of integers. */
public static boolean isUpperTriangular( int [][] m )
{
    boolean res = true;
    int i = m.length - 1;
    while( i >= 0 && res ) {
        int j = 0;
        while( j < i && m[i][j] == 0 ) { j++; }
        if ( j < i ) { res = false; }
        else { i--; }
    }
    return res;
}
```

You have to analyse this method by following these steps:

a) (0.25 points) Describe the input size of the problem and give an expression for it.

b) (0.50 points) Choose a critical instruction for using it as reference for counting program steps.

c) (0.75 points) Is the method sensible to different instances of the problem for the same input size? In other words, will the critical instruction be repeated more or less times depending on the input data for the same input size?
If the answer is yes describe best and worst cases.

d) (1.00 points) Obtain an expression of the temporal cost function for both best and worst cases ($T^b(n)$ and $T^w(n)$) if the answer to the previous question was yes, otherwise a unique expression of temporal cost function $T(n)$.

e) (0.50 points) Use the asymptotic notation for expressing the behaviour of the temporal cost function for large enough values of the input size.

Solution:

a) The input size of the problem can be expressed by `m.length` as it refers to the number of rows `m`:

$$n = m.length$$

b) As we have to nested loops, we will use the guard of the most inner loop:

```
j < i && m[i][j] == 0
```

c) Yes.

The **best case** is when the matrix is not an upper triangular one and the first element checked is different from zero. According the implementation of the method, `m[ m.length-1 ][0]` is the first element compared with zero.

The **worst case** is then the matrix is an upper triangular matrix, so all the values under the main diagonal will be visited and compared with zero.

d)

$$T^b(n) = 1$$

$$\begin{aligned}
T^w(n) &= 1\Big|_{\substack{i=n-1\\ j=0}} + 1\Big|_{\substack{i=n-1\\ j=1}} + 1\Big|_{\substack{i=n-1\\ j=2}} + \cdots + 1\Big|_{\substack{i=n-1\\ j=i}} + \\
&+ 1\Big|_{\substack{i=n-2\\ j=0}} + 1\Big|_{\substack{i=n-2\\ j=1}} + 1\Big|_{\substack{i=n-2\\ j=2}} + \cdots + 1\Big|_{\substack{i=n-2\\ j=i}} + \\
&\cdots \\
&+ 1\Big|_{\substack{i=2\\ j=0}} + 1\Big|_{\substack{i=2\\ j=1}} + 1\Big|_{\substack{i=2\\ j=2}} + \\
&+ 1\Big|_{\substack{i=1\\ j=0}} + 1\Big|_{\substack{i=1\\ j=1}} + \\
&+ 1\Big|_{\substack{i=0\\ j=0}} = \\
&= n\Big|_{i=n-1} + (n-1)\Big|_{i=n-2} + (n-2)\Big|_{i=n-3} + \cdots + 3\Big|_{i=2} + 2\Big|_{i=1} + 1\Big|_{i=0} = \\
&= \sum_{k=1}^{n} k = \frac{n*(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}
\end{aligned}$$

e)

$$T^b(n) \in \Omega(1) \quad \texttt{and} \quad T^w(n) \in O(n^2) \quad \Rightarrow \quad T(n) \in \Omega(1) \cap O(n^2)$$

3. $\boxed{\text{3 points}}$ Given an square matrix `m` of integers, the following recursive method computes the sum of all the elements of any $n \times n$ sub-matrix with origin in $(0, 0)$. Note that if we need to sum all the elements of the matrix the initial call to the method must be `add(m,m.length)`

```
/** Precondition: m is an square matrix of integers, and 0 <= n <= m.length */
public static int add( int[][] m, int n ) {
    if ( n == 0 ) {
        return 0;
    }
    else
    {
        int s = m[n-1][n-1];
        for( int i = 0; i < n-1; i++ ) {
            s = s + m[n-1][i] + m[i][n-1];
        }
        return s + add( m, n-1 );
    }
}
```

You have to carry out the same steps asked in the previous problem, but, in this case, we suggest to use the substitution method in the fourth step, i.e. for obtaining the temporal cost function. The weight in points for the five steps is the same.

Solution:

a) $n = n$, because n is the parameter that defines the size of the square sub-matrix.

b) $n == 0$ for controlling the recursion, and $i < n - 1$ for controlling the loop in the general case.

c) No, this method does not depend on the values stored in the matrix `m`.

d) The loop for any value of $n > 0$ has a linear cost, so we will use n program steps. So, by applying the substitution method:

$$\begin{aligned}
T(n) &= n + T(n-1) \\
&= n + (n-1) + T(n-2) \\
&= n + (n-1) + (n-2) + T(n-3) \\
&\quad \cdots \\
&= n + (n-1) + (n-2) + (n-3) \cdots + 3 + 2 + T(1) \\
&= n + (n-1) + (n-2) + (n-3) \cdots + 3 + 2 + 1 + T(0) \\
&= n + (n-1) + (n-2) + (n-3) \cdots + 3 + 2 + 1 + 1 \\
&= \sum_{k=1}^{n} k = \frac{n * (n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}
\end{aligned}$$

e) $T(n) \in O(n^2)$