Computer Programming – ETSINF – Academic year 2017/2018
First midterm exam of Theory – April 16, 2018 – Duration: 2 hours

Note: Max grade in this exam 10 points. Weight in the final grade of the subject **3 points**.

1. [4 points] Let `v` be an array of `String` and let `n` be a natural number, write a **recursive** method that returns how many elements in array `v` are `n` characters long. Some examples:

Contents of array `v`	**Value of `n`**	**Returned value**
{"barco", "autobus", "tren", "moto", "bici"}	4	3
Same array	5	1
Same array	3	0

What to do:

a) (0.75 points) Profile of the method with the appropriate parameters for recursively solving the problem. Add the necessary pre-conditions for the parameters.

b) (1.25 points) Define trivial and general cases.

c) (1.50 points) Write the Java code for the recursive method.

d) (0.50 points) Initial call to the recursive method for processing the whole array. Use a wrapper method if you prefer it.

Solution:

a)
```
/** Precondition: n >= 0 && 0 <= pos <= v.length */
public static int wordCount( String[] v, int n, int pos )
```

It is expected that the method will return how many `String` objects whose length is equal to `n` are stored in the subarray from the first position (indexed by 0) up to the one indexed by `pos`. The method does not check if `pos` is a not valid index, i.e. out of range [0 . . . `pos`].

b)
- Trivial case: `pos < 0`: Empty subarray, 0 is returned.
- General case: `0 <= pos < v.length`: Subarray of size `pos+1`.

$$\texttt{wordCount(v, n, pos)} = \begin{cases} \texttt{wordCount}(v, n, pos-1) & if\ \texttt{v[pos].length() != n} \\ \texttt{wordCount}(v, n, pos-1)+1 & if\ \texttt{v[pos].length() == n} \end{cases}$$

c)
```
public static int wordCount( String[] v, int n, int pos )
{
    if ( pos < 0 ) return 0;

    return wordCount( v, n, pos-1 ) + (v[pos].length() == n ? 1 : 0 );
}
```

d) The initial call is inside a wrapper method:

```
public static int wordCount( String[] v, int n )
{
    return wordCount( v, n, v.length-1 );
}
```

2. [3 points] Let `m` be an square matrix of integers whose elements are all positive values. The following method checks which rows fulfil that the sum of its components is lower than `maxVal`, and if it is the case, prints on screen the value of such sum.

```
/** Precondition: m is an square matrix filled with positive values
 *  and maxVal > 0
```

```
 */
public static void sums( int[][] m, int maxVal )
{
    int n = m.length;
    for( int i=0; i < n; i++ ) {
        int sum = m[i][0];
        int j = 1;
        while( j < n && sum < maxVal ) {
            sum += m[i][j];
            j++;
        }
        if ( sum < maxVal ) {
            System.out.println( "Row " + i + ": " + sum );
        }
    }
}
```

What to do:

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, is the critical instruction 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 each case if the answer to the previous question was yes and a unique expression if the answer was no.

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) $n =$ `m.length`

b) Both `( j < n )` and `( j < n && sum < maxVal )` can be chosen as critical instructions.

c) Yes, the method is sensible to different instances of the problem.

Best case: `m[i][0] > maxVal` for all rows, i.e. the first element of each row is greater than `maxVal`, or in other words, all the elements of the first column are greater than `maxVal`.

Worst case: The sum of the elements of one row is lower than `maxVal` for all the rows.

d) **Best case:** $T^b(n) = \sum_{i=0}^{n-1} 1 = 1 + n$
Worst case: $T^w(n) = \sum_{i=0}^{n-1}(1 + \sum_{j=1}^{n-1} 1) = \sum_{i=0}^{n-1} n = 1 + n^2$

e) $T^b(n) \in \Theta(n)$ and $T^w(n) \in \Theta(n^2) \quad \Rightarrow \quad T(n) \in \Omega(n) \cap O(n^2)$

3. 3 points Given the following recursive method:

```
public static double testMethod( double[] v, int left, int right )
{
    if ( left > right ) return 1.0;

    int middle = (left + right) / 2;
    return v[middle]
```

```
                * testMethod( v, left,     middle-1 )
                * testMethod( v, middle+1, right    );
        }
```

What to do:

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, is the critical instruction 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 each case if the answer to the previous question was yes and a unique expression if the answer was no.

In this case a recursive method must be analysed, so it is preferable to write the recursive equation of the temporal cost and apply the substitution method.

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) $n =$ `right-left+1`

b) Critical instruction: `left > right`

c) No

d) For simplicity it is considered that at every recursive call the input size is divided by 2.

$$T(n) = \begin{cases} 2 \cdot T(n/2) + 1 & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

Applying the substitution method:

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + 1 \\ &= 2^2 \cdot T(n/2^2) + 2^1 + 2^0 \\ &= 2^3 \cdot T(n/2^3) + 2^2 + 2^1 + 2^0 \\ &= 2^3 \cdot T(n/2^3) + 2^3 - 1 \\ &\dots \\ &= 2^k \cdot T(n/2^k) + 2^k - 1 \end{aligned}$$

$1 \leq n/2^k < 2 \quad \Rightarrow \quad k = \lfloor \log_2(n) \rfloor$, so $T(n) = 2^{\lfloor \log_2(n) \rfloor} \cdot T(n/2^{\lfloor \log_2(n) \rfloor}) + 2^{\lfloor \log_2(n) \rfloor} - 1$

Approximating $2^{\lfloor \log_2(n) \rfloor} \approx n$, then $T(n) = n \cdot T(1) + n - 1 = n \cdot (2 \cdot T(0) + 1) + n - 1$

As in the trivial case $T(0) = 1$, the result is: $T(n) = 4n - 1$

e) $T(n) \in \Theta(n)$