PRG first quiz - ETSInf

Date: March, 21st 2011. Duration: 1 hour and 30 minutes

NOTE: You must answer on separate sheets. It is not necessary to hand in this sheet.

1. (3 points) The following **iterative** method computes the product of a row vector `x` by a square matrix `a`, giving as a result another row vector. Since `a` is a square matrix, both vectors, the row vector `x` and the resulting vector, have the same length, that in this case is the number of rows of the matrix `a`.

```
/**  x.length = a.length  and 'a' is a square matrix */
static double [] product( double x[], double a[][] )
{
    double r[] = new double [ a.length ];

    for( int i=0; i < r.length; i++ ) {
        r[i] = 0.0;
        for( int j=0; j < x.length; j++ )
            r[i] += x[j] * a[j][i];
    }
    return r;
}
```

For example, given $x = (2, 1, 3)$, a row vector with dimension 1×3, and $a = \begin{pmatrix} 4 & 3 & 2 \\ 2 & 5 & 1 \\ 1 & 0 & 3 \end{pmatrix}$, a 3×3 matrix, the product $x \times a$ is $r = (13, 11, 14)$, a row vector of dimension 1×3.

Answer the following:

1) Determine the input size of the problem and write the expression that represents it.
2) 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.
3) Identify if there exist significant instances, that is, does the number of times the critical instruction is repeated depend on the values stored in the arrays? In the affirmative case indicate the best and worst cases.
4) 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 $T^w(n)$.
5) Express the results using asymptotic notation.

Solution:

1. The input size is the number of rows of the input matrix, `a.length`, which coincides with the length of input vector `x`.

   $n =$ `a.length`.

2. This is a problem which needs to go through all components of the arrays, therefore, for the same size of problem no instances are significant.

3. Taking as a measure unit the program step, the temporal cost function can be expressed as: $T(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(1 + n) = 1 + n + n^2$.

   If we choose a critical instruction as the measure unit, concretely `r[i] += x[j] * a[j][i]`, the most inner assignment, the temporal cost function is: $T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n^2$.

4. Expressed using asymptotic notation: $T(n) \in \Theta(n^2)$.

2. (3.5 points) We propose the following **iterative** method for obtaining the position of the odd number within the vector of integers `v` most near the end.

```
static int positionOfLastOddNumber( int[] v )
{
    int i = v.length-1;
    while( i >= 0 && v[i]%2 == 0 ) i--;
    return i;
}
```

Answer the same points enumerated in the first question.

Solution:

a) The input size of the problem is the number of elements in the array `v` and the expression which represents it is `v.length`. From now on we refer to it as $n =$ `v.length`.

b) It is a linear search problem, concretely an iterative sequential descending search, so there exist significant instances.

   The *best case* is given when the last component of the vector is odd, that is,

   `(v[v.length-1]%2) != 0`.

   The *worst case* is given when all the components of the vector are even, that is,

   `(v[v.length-1]%2) == 0` $\forall i : 0 \leq i <$ `v.length`

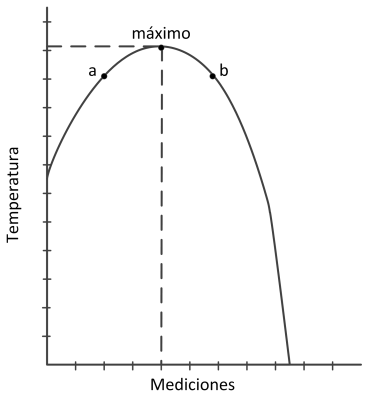
c) We opted to choose the critical instruction as measure unit for estimating the cost. The only instruction that can be considered as critical is the loop condition: `i>= 0 && v[i]%2 == 0`. It is always executed once more than any other instruction within the loop.

   In the *best case* it only runs once and the temporal cost function in this case is $T^m(n) = 1$. In the *worst case* it will be repeated as many times as the number of elements in the array plus one time more (when it is false), and the temporal cost function is $T^p(n) = \sum_{i=0}^{n} 1 = n + 1$.

d) Using asymptotic notation: $T^m(n) \in \Theta(1)$ and $T^p(n) \in \Theta(n)$. So, $T(n) \in \Omega(1)$ and $T(n) \in O(n)$, that is, the temporal cost function in general is bounded by a constant as the lower bound and by a linear function depending on the input size as the upper bound.

3. (3.5 points) We have at our disposal a succession of temperature measurements distributed as a parabolic function. The measurements are stored in a vector `v` of floating point numbers. The following recursive method obtains the maximum value of the curve by scanning the vector `v`.

For such curves, it is known that if for a given value the anterior value and the posterior one are smaller than it, then we are at the peak measured. On the contrary they exist two possibilities, the first one is when the previous value is higher and the following one is lower, in this case we're on the right of the maximum, and the other one is when the previous value is lower and the next one is higher, then we're on the left of the maximum. Another situation is impossible in this type of curves. The figure shows how the points `a` and `b` (before and after peak, respectively) are lower than the maximum point.



```
static double maxTemp( double[] v, int start, int end )
{
    if ( start == end )
        return v[start];
    else if ( start == end-1 )
        return Math.max( v[start], v[end] );
    else {
        int theHalf = (end+start)/2;
        if ( v[theHalf] > v[theHalf-1] && v[theHalf] > v[theHalf+1] )
            return v[theHalf];
        else if ( v[theHalf] < v[theHalf-1] && v[theHalf] > v[theHalf+1] )
            return maxTemp( v, start, theHalf );
        else
            return maxTemp( v, theHalf, end );
    }
}
```

The initial invocation is: `double maxValue = maxTemp( v, 0, v.length-1 );`

Answer the following:

1) Determine the input size of the problem and write the expression that represents it.
2) 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.
3) Identify if there exist significant instances, that is, does the number of times the critical instruction is repeated depend on the values stored in the arrays? In the affirmative case indicate the best and worst cases.
4) 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 $T^w(n)$.

   We recommend to use the substitution method.
5) Express the results using asymptotic notation.

Solution:

1. The input size is the number of elements in the subarray `v[start:end]`, so, the input size can be expressed as $n =$ `end-start+1`.

2. It's a search problem, concretely the binary search, so there exist significant instances. The best case is when the maximum is located in the central position of the input vector, that is, in the position `(start+end)/2` when `start=0` and `end=v.length-1`. The worst case is when it is necessary to do the maximum number of recursive calls to find the peak, that is, when the input size is reduced until the trivial case is reached.

3. In the *best-case* the temporal cost function $T^m(n) = c$. For obtaining the temporal cost function in the *worst-case* we use the recurrence equation:

$$T^p(n) = \begin{cases} T^p(\frac{n}{2}) + k & \text{if } n > 2 \\ k' & \text{if } n = 1 \text{ or } n = 2 \end{cases}$$

   and applying the substitution method:

$$T^p(n) = T^p(\frac{n}{2}) + k = T^p(\frac{n}{2^2}) + 2k = \ldots = T^p(\frac{n}{2^i}) + i \cdot k$$

   when the trivial case is reached (input size = 1 or 2), we have $\underbrace{\frac{n}{2^i} = 1 \Rightarrow i = \log_2 n}$ when the input size is 1 and $\underbrace{\frac{n}{2^i} = 2 \Rightarrow n = 2^i * 2 \Rightarrow i = \log_2 n + \log_2 2 = \log_2 n + 1}$ when the input size is 2. So $T^p(n) \approx k' + k \cdot \log_2 n$.

4. Expressing the results by using asymptotic notation we have $T^m(n) \in \Theta(1)$ and $T^p(n) \in \Theta(\log_2 n)$, so $T(n) \in \Omega(1)$ and $T(n) \in O(\log_2 n)$. It means that the behavior of the algorithm is lower bounded by constant and upper bounded by a logarithmic function as the input size increases.