PRG first quiz - ETSInf

Date: May 7th, 2012. Duration: 2 hours and 30 minutes

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

1. (2 points) Write a recursive method with one argument, a non negative integer, for writing on standard output the descending sequence of values from n to zero, and the ascending sequence from zero to n. Each sequence on a different line, as it is shown in the following example. If the method is invoked with `n=14`, the output is the following:

   ```
   14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
   ```

   Notice that the numbers of a sequence are in the same line.

   Solution:

   ```
   public class Problem1 {

       public static void main( String[] args )
       {
           sequences( 14 );
           System.out.println();
       }

       /** n>=0 */
       private static void sequences( int n )
       {
           if ( n > 0 ) {
               System.out.print( n + " " );
               sequences( n-1 );
               System.out.print( " " + n );
           } else {
               System.out.print( "0\n0" );
           }
       }
   }
   ```

2. (2 points) Write a recursive method that given an integer array, `v`, two indexes for indicating the starting and ending positions, `begin >= 0` and `end < V.length`, and an integer `x`, returns `true` if the sum of each pair of symmetric items is equal to `x`, otherwise returns `false`. In other words, the method will return `true` if the condition `v[begin+i]+v[end-i]==x` is satisfied for `0<=i<=(begin+end)/2`.

   For example:

   If `x = 10, begin = 1, end = v.length-2` and `v = {1,2,3,4,5,6,7,8,9}` returns `true`.

   If `x = 10, begin = 1, end = v.length-2` and `v = {1,2,4,4,5,6,7,8,9}` returns `false`.

   If `x = 6, begin = 0, end = v.length-1` and `v = {1,2,3,3,4,5}` returns `true`.

   Solution:

```
public class Problem2 {

    public static void main( String[] args )
    {
        int[] a = {1,2,3,4,5,6,7,8,9};
        int[] b = {1,2,4,4,5,6,7,8,9};
        int[] c = {1,2,3,3,4,5};

        System.out.println( method( a, 1, a.length-2, 10 ) );
        System.out.println( method( b, 1, b.length-2, 10 ) );
        System.out.println( method( c, 0, c.length-1,  6 ) );
    }

    public static boolean method( int[] v, int begin, int end, int x )
    {
        if ( begin > end )
            return true;
        else if ( begin == end )
            return (2*v[begin])==x;
        else if ( v[begin]+v[end] != x )
            return false;
        else
            return method( v, begin+1, end-1, x );
    }
}
```

3. (2 points) Given the following method:

```
/** n>=0 */
public static void binary( int n ) {
    if ( n > 0 ) binary( n/2 );
    System.out.print( n % 2 );
}
```

Required:

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

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.

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

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 $T^w(n)$.

e) Express the results using asymptotic notation.

Solution:

1. The input size of the problem is the value of the argument, i.e., `n`.

2. There are no significant instances.

3. Given the following recurrence equation: $T(n) = \begin{cases} T(n/2) + k & \text{if } n > 0 \\ k' & \text{if } n = 0 \end{cases}$ and applying the substitution method, we have: $T(n) = T(n/2)+k = T(n/2^2)+2k = \ldots = T(n/2^i)+ik$. When the trivial case is reached $i \approx log_2 n$, the cost is $T(n/2^i) \rightarrow T(0) = k'$, and $T(n) \approx k' + klog_2 n$.

4. Using asymptotic we have: $T(n) \in \Theta(\log n)$, the temporal cost is logarithmic with respect to the input size.

4. (3 points) Let `m` be a nonnegative integer matrix. In order to check if the sum of the values in the main diagonal is greater than a positive value given as argument, we have available the following two methods:

```
public class Problem4
{
    /** for all i, j: 0<=i<m.length, 0<=j<m.length, m[i][j]>=0 y val>=0 */
    public static boolean method1( int[][] m, int val )
    {
        int s = 0;
        for( int i=0; i < m.length; i++ ) s += m[i][i];
        return s > val;
    }
    /** for all i, j: 0<=i<m.length, 0<=j<m.length, m[i][j]>=0 y val>=0 */
    public static boolean method2( int[][] m, int val )
    {
        int s = 0;
        for( int i=0; i < m.length && s <= val; i++ ) {
            for( int j=0; j <= i && s <= val; j++ ) {
                if ( i == j ) s += m[i][j];
            }
        }
        return s > val;
    }
}
```

Required:

a) **For each method:**
   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. 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.

b) Describe briefly the differences between both methods.

c) How could you modify the first method in order to improve it?

Solution:

a) For each method:

1. For both methods the input size of the problem is the same: the number of rows `m.length`, which is equal to the number of columns. From now on $n =$ `m.length`.
2. The first method is an array traversal's algorithm, so there are no significant instances.

   The second one is a search algorithm, so for a given input size the behaviour of the algorithm may differ depending on the values contained in the matrix.

   Best case appears when `m[0][0] > val`, worst case appears when $\sum_{i=0}^{n-1} \texttt{m}[i][i] < \texttt{val}$.
3. We select the critical instruction as unit measure for obtaining the temporal cost function.

   In the first method we use `s += m[i][i]` as the critical instruction and obtain the following temporal cost function: $T(n) = \sum_{i=0}^{n-1} 1 = n$.

   In the second method, we use `(i == j)` as the critical instruction. Then, in the best case it runs once, so $T^m(n) = 1$. In the worst case it is repeated the maximum possible number of times, so $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(i+1) = (n^2+n)/2$.
4. First method the temporal cost is linear: $T(n) \in \Theta(n)$

   Second method:

   - Best case is constant: $T^m(n) \in \Theta(1)$
   - Worst case is quadratic: $T^w(n) \in \Theta(n^2)$

   In general: $T(n) \in \Omega(1) \cap O(n^2)$

b) To determine whether the sum of the diagonal elements of the square matrix `m` is greater than `val`, `method1()` accesses to each element once, while `method2()` also checks all items below the diagonal.

The first method is a traversal algorithm, adds up all the diagonal elements directly. However, the second method is a search algorithm, adds up the elements of the diagonal while the sum is less than `val`, i.e., ends before accessing all the values in the inferior triangular matrix just when the sum is greater than `val`.

c) The first method can be improved if we convert it into a search algorithm, just substituting the loop condition with `i < m.length && s <= val`.

5. (1 point) The following table shows running time, in milliseconds, of an unknown algorithm for a set of different input sizes.

```
# Input size   Running time (ms)
#---------------------------------------
      1000        49.78
      2000       202.33
      3000       454.42
      4000       804.03
      5000      1270.28
      6000      1841.47
      7000      2506.30
      8000      3253.62
      9000      4141.05
     10000      5277.99
```

a) From an asymptotic point of view, which could be the typical function that best fits the values of the column `running time`?

b) In an approximate way, could you give us an estimation of the running time of the unknown algorithm for an input size equal to 20000?

Solution: The temporal cost function which best fits the values in the table is a quadratic function with respect to the input size. If n represents the input size, then $T(n) \in \Theta(n^2)$. It can be checked that when the input size is multiplied by 2 the running time is multiplied by 4. For example:

$$\frac{T(4000)}{T(2000)} = \frac{804.03}{202.33} = 3.97 \approx 4$$

For a given input size equal to 20000, the running time will be approximately 4×5277.99, i.e. $T(20000) \approx 4 \times T(10000) \approx 20000$ ms.