ETSINF – Computer Programming – THEORY – Academic year 2016/17
First mid term exam – 10 of April, 2017 – Duration 2 hours

Notice: This exam is evaluated over a grade of 10, but its weight for the final grade is 30%, i.e., **3 points**.

1. [4 points] Given an array `a` of `int` and an integer `x`, write a **recursive** method for returning how many numbers multiple of `x` are stored in the array `a`.

   What to do:

   a) (0.75 points) Profile of the method, with the appropriate parameters for recursively solving the problem.

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

   c) (1.50 points) Java implementation.

   d) (0.50 points) Initial call to the method in order to get the solution for whole array.

   Solution:

   a) A possible solution is to define the method with the following profile,

   ```
   /** Precondition: 0 <= pos */
   public static int multiplesOfX( int[] a, int x, int pos )
   ```

   where `pos` ≥ 0 is a parameter used as index to specify the current position of the array being processed. The result should be the count of how many values multiple of `x` are in the slice of the array from zero to `pos`, `a[0..pos]`.

   b)
   - Trivial case: `pos` < 0, empty array $\rightarrow$ zero is returned.
   - General case: $0 \leq$ `pos` $<$ `a.length`, subarray with one or more elements.
     If `a[pos] % x == 0`, then returns 1 plus the count of values multiple of `x` in the subarray from 0 to `pos`-1, `a[0..pos-1]` else returns just the count of values multiple of `x` in `a[0..pos-1]`.

   c)
   ```
   /** Returns the count of values multiple of x in a[0..pos-1].
    *  Precondition: 0 <= pos */
   public static int multiplesOfX( int[] a, int x, int pos )
   {
       if ( pos < 0 ) {
           return 0;
       } else if (a[pos] % x == 0) {
           return 1 + multiplesOfX(a, x, pos + 1);
       } else {
           return multiplesOfX(a, x, pos + 1);
       }
   }
   ```

   d) For an array `a`, the initial call for considering the whole array must be `multiplesOfX( a, x, a.length-1 )`.

2. [3 points] Given an array of `char` `a` and a single `char` `c`, the following method writes on standard output, line by line, all the prefixes of the character sequence in `a` with length ≥ 1 that do not end with the `char` `c`.

   ```
   public static void prefixes( char[] a, char c )
   {
       for( int i = 0; i < a.length; i++ ) {
           if ( a[i] != c ) {
               for ( int j = 0; j <= i; j++ ) {
                   System.out.print( a[j] );
               }
               System.out.println();
           }
   ```

```
            }
        }
```

As an example, if `a = {'g', 't', 'a', 't', 'c'}`, the prefixes of consecutive lengths are `g`, `gt`, `gta`, `gtat` y `gtatc`. With `a` and `c = 't'` the method writes:

```
g
gta
gtatc
```

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) The input size for this problem is number of elements in the array `a`, i.e., `a.length`.

$$n \equiv \texttt{a.length}$$

b) Possible elementary operations to be considered as critical instructions are: `j <= i`, `j++` or `System.out.print( a[j] )`, but for sake of simplicity let us choose `j <= i`.

c) Yes, the algorithm is sensible to different instances of the problem, i.e., its temporal behaviour varies according to the value of the parameter `c` and the values stored in the array `a`.

The best case is when all the characters stored in the array `a` are equal to the character stored in the parameter `c`.

The worst case is when the character stored in the parameter `c` is not in the array `a`.

d) Both temporal cost functions are obtained, for best and worst cases:

- Best case: $T^b(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ program steps
- Worst case:

$$\begin{aligned} T^w(n) &= 1 + \textstyle\sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i} 1) = 1 + \sum_{i=0}^{n-1}(2 + i) = \\ &= 1 + \textstyle\sum_{i=1}^{n}(1 + i) = 1 + n + \sum_{i=1}^{n} i = \\ &= 1 + n + \frac{n(n+1)}{2} = 1 + \frac{3n}{2} + \frac{n^2}{2} \text{ program steps} \end{aligned}$$

e) Using asymptotic notation:

$$T^b(n) \in \Theta(n) \qquad \text{and} \qquad T^w(n) \in \Theta(n^2) \qquad \Rightarrow \qquad T(n) \in \Omega(n) \cap O(n^2)$$

3. 3 points The following method discovers if, given a positive integer `num`, its literal can be expressed by using a particular base `b`, where $2 \leq$ `b` ≤ 10. For computing if `num` can be expressed in base `b` the method checks if all the digits have a value lower than `b`. For instance, the number 453123 can be a value expressed in bases 6, 7, 8, 9 and 10, because all its digits are strictly lower than any of these bases.

```
/** Precondition: 2 <= b <= 10 and num >= 0 */
public static boolean possibleBase( int num, int b ) {
    if (num == 0) { return true; }
    else {
        int lastDigit = num % 10;
        if ( lastDigit < b ) { return possibleBase( num / 10, b ); }
        else { return false; }
    }
}
```

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.
As it is a recursive method you define a recurrent equation and use it in 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) For this problem the input size can be defined as

(1) the value of `num`, the first parameter,

(2) or as the number of digits of `num`.

b) As it is a recursive method, the critical instruction is the condition for distinguishing between trivial and general cases.

c) The algorithm is sensible to different instances of the problem. It is a search, because it can stop before checking all the digits of `num` if it is found one digit whose value is greater than or equal to the base `b`.

The best case is when `num` cannot be expressed in base `b` and the first digit of `num` is greater than or equal to `b`.

The worst case is when `num` can be expressed in base `b`. All digits of `num` are lower than `b`.

d) Using the value of `num` as the input size: $n \equiv$ num

- Best case: $T^b(n) = 1, \forall n \geq 0$
- Worst case: $T^w(n) = \begin{cases} T^w(n/10) + 1 & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

  $T^w(n) = 1 + T(n/10) = 1 + 1 + T(n/10^2) = 2 + T(\frac{n}{10^2}) = \ldots = k + T(\frac{n}{n^k})$, where $k = \log_{10} n$
  So $T^w(n) = 1 + \log_{10} n$

Using the number of digits of `num` as the input size: $n \equiv numberOfDigits$(num)

- Best case: $T^b(n) = 1, \forall n \geq 0$
- Worst case: $T^w(n) = \begin{cases} T^w(n-1) + 1 & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

  $T^w(n) = 1 + T(n-1) = 1 + 1 + T(n-2) = 2 + T(n-2) = \ldots = n + T(0)$, where n is the number of digits of `num`.
  So $T^w(n) = 1 + n$

e) Using the asymptotic notation:

(1) If $n \equiv \texttt{num}$
$T^b(n) \in \Theta(1)$ and $T^w(n) \in \Theta(\log_{10} n) \equiv \Theta(\log n) \Rightarrow T(n) \in \Omega(1) \cap O(\log n)$

(2) If $n \equiv numberOfDigits(\texttt{num})$
$T^b(n) \in \Theta(1)$ and $T^w(n) \in \Theta(n) \Rightarrow T(n) \in \Omega(1) \cap O(n)$

As one can observe, the temporal cost of the method is the same, although the resulting expression be different. The reader can easily realise the difference is due to the definition of the input size. The method performs a number of program steps which is proportional to the number of digits of `num`, so if the input size is defined as the number of digits, which is a the integer logarithm of `num` plus one, then the temporal cost function obtained is linear. When the value of `num` is used as the input size, then the temporal cost function has a logarithmic behaviour.