Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2017

Práctico 1 - Parte 3

ejercicios para el lunes 27/3

1. El siguiente algoritmo calcula el mínimo elemento de un arreglo a : **array**$[1..n]$ **of nat** mediante la técnica de programación *divide y vencerás*. Analizá la eficiencia de $minimo(1, n)$.

```
fun minimo(a : array[1..n] of nat, i, k : nat)  ret m : nat
    if i = k then m := a[i]
    else
        j := (i + k) div 2
        m := min(minimo(a, i, j), minimo(a,j+1,k))
    fi
end fun
```

2. Dado un arreglo a : **array**$[1..N]$ **of nat** se define una *cima* de a como un valor k en el intervalo $1, \ldots, N$ tal que $a[1..k]$ está ordenado crecientemente y $a[k..N]$ está ordenado decrecientemente.
   (a) Escribí un algoritmo que determine si un arreglo dado tiene cima.
   (b) Escribí un algoritmo que encuentre la cima de un arreglo dado (asumiendo que efectivamente tiene una cima) utilizando una búsqueda secuencial, desde el comienzo del arreglo hacia el final.
   (c) Escribí un algoritmo que resuelva el mismo problema del inciso anterior utilizando la idea de *búsqueda binaria*.
   (d) Calculá y compará el orden de complejidad de ambos algoritmos.

3. Calculá el orden de complejidad de los siguientes algoritmos:

(a)
```
proc f1(in n : nat)
    if n ≤ 1 then skip
    else
        for i := 1 to 8 do f1(n div 2) od
        for i := 1 to n³ do t := 1 od
```

(b)
```
proc f2(in n : nat)
    for i := 1 to n do
        for j := 1 to i do t := 1 od
    od
    if n > 0 then
        for i := 1 to 4 do f2(n div 2) od
```

ejercicios para el miércoles 29/3

4. Ordená utilizando $\sqsubset$ e $\approx$ los órdenes de las siguientes funciones. No calcules límites, utilizá las propiedades algebraicas.
   (a) $n \log 2^n \qquad 2^n \log n \qquad n! \log n \qquad 2^n$
   (b) $n^4 + 2\log n \qquad \log(n^{n^4}) \qquad 2^{4\log n} \qquad 4^n \qquad n^3 \log n$
   (c) $\log n! \qquad n \log n \qquad \log(n^n)$

5. Si $\lim_{n\to\infty} h(n) = \infty$, ¿cuáles de las siguientes afirmaciones resultan verdaderas?
   (a) si $f(n) \sqsubset g(n))$, entonces $f(h(n)) \sqsubset g(h(n))$,
   (b) si $f(h(n)) \sqsubset g(h(n))$, entonces $f(n) \sqsubset g(n)$,
   (c) si $f(n) \sqsubset g(n)$, entonces $h(f(n)) \sqsubset h(g(n))$.

ejercicios adicionales

6. Dada la siguiente recurrencia

$$t(n) = \begin{cases} 4 & \text{si } n = 1 \\ 2t(n/2) + 2nlog_2(n) & \text{si } n > 1 \end{cases}$$

Demostrá que $t(n)$ es del orden de $nlog^2(n)$. Ayuda: primero demostralo para $n = 2^m$.

7. Sean K y L constantes, y f el siguiente procedimiento:

   proc *f*(**in** n : **nat**)
       if $n \leq 1$ **then skip**
       else
           for $i := 1$ **to** K **do** *f*(n **div** L) **od**
           for $i := 1$ **to** n^4 **do** *operación_de_*$\mathcal{O}(1)$ **od**

   Determiná posibles valores de K y L de manera que el procedimiento tenga orden:

   (a) $n^4 \log n$ (b) n^4 (c) n^5

8. Escribí algoritmos cuyas complejidades sean (asumiendo que el lenguaje no tiene multiplicaciones ni logaritmos, o sea que no podés escribir **for** i:= 1 **to** $n^2 + 2\log n$ **do** ... **od**):

   (a) $n^2 + 2\log n$ (b) $n^2 \log n$ (c) 3^n

9. Una secuencia de valores $x_1, ..., x_n$ se dice que tiene *orden cíclico* si existe un i con $1 \leq i \leq n$ tal que $x_i < x_{i+1} < \ldots < x_n < x_1 < \ldots < x_{i-1}$. Por ejemplo, la secuencia $5, 6, 7, 8, 9, 1, 2, 3, 4$ tiene orden cíclico (tomando $i = 6$).

   (a) Escribí un algoritmo que determine si un arreglo almacena una secuencia de valores que tiene orden cíclico o no.
   (b) Escribí un algoritmo que dado un arreglo a : **array**$[1..n]$ **of nat** que almacena una secuencia de valores que tiene orden cíclico, realice una búsqueda secuencial en el mismo para encontrar la posición del menor elemento de la secuencia (es decir, la posición i).
   (c) Escribí un algoritmo que resuelva el problema del inciso anterior utilizando la idea de *búsqueda binaria*.
   (d) Calculá y compará el orden de complejidad de ambos algoritmos.

10. Calculá el orden de complejidad del siguiente algoritmo:

    proc *f3*(n : **nat**)
        for $j := 1$ **to** 6 **do**
            if $n \leq 1$ **then skip**
            else
                for $i := 1$ **to** 3 **do** *f3*(n **div** 4) **od**
                for $i := 1$ **to** n^4 **do** $t := 1$ **od**
        od