Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2011

Práctico 2: Ordenación y Análisis de Complejidad

Docentes: Daniel Fridlender, Renato Cherini, Diego Lis, y Juan Cruz Rodriguez

1. Ordená los siguientes arreglos, utilizando el algoritmo de ordenación por selección. Mostrá en cada paso de iteración cuál es el elemento seleccionado y cómo queda el arreglo después de cada intercambio.

   a) $[1, 2, 3, 4, 5]$

   b) $[5, 4, 3, 2, 1]$

   c) $[7, 1, 10, 3, 4, 9, 5]$

2. Repetí el ejercicio anterior, pero utilizando el algoritmo de ordenación por inserción. Mostrá cómo queda el arreglo en cada iteración del procedimiento *insert*.

3. Se desea ordenar un arreglo a : **array**$[1..N]$ **of nat**, de forma que los números pares se ubiquen hacia el principio del arreglo, y los números impares hacia el final. A su vez, los números pares (respectivamente los impares) deben quedar ordenados entre sí de mayor a menor (respectivamente de menor a mayor). Por ejemplo, el arreglo $[4, 3, 25, 7, 6, 16, 12, 0]$ se ordena como $[16, 12, 6, 4, 0, 3, 7, 25]$. Escribí un programa que implemente esta ordenación.

4. Se desea acceder de forma *ordenada* a los elementos de un arreglo a : **array**$[1..N]$ **of int** pero sin modificar el propio arreglo. La idea es utilizar una tabla b : **array**$[1..N]$ **of nat** que contenga (secuencialmente) los índices de los elementos según el orden deseado, de manera que $a[b[i]] \leq a[b[i+1]]$ para todo $1 \leq i < N$. Por ejemplo, para $a = [4, 7, 1, 3, 9]$ la tabla es $b = [3, 4, 1, 2, 5]$.

   Escribí un programa que dado un arreglo compute la tabla descripta.

5. Calculá de la manera más exacta y simple posible el número de operaciones de los siguientes programas:

   a)
   ```
   t := 0;
   for i := 1 to N do
       for j := 1 to N^2 do
           for k := 1 to N^3 do t := t + 1
   ```

   b)
   ```
   t := 0;
   for i := 1 to N do
       for j := 1 to i do
           for k := j to N do t := t + 1
   ```

   c)
   ```
   t := 0;
   for i := 1 to N do
       for j := 1 to i do
           for k := j to j + 3 do t := t + 1
   ```

   d)
   ```
   p := 1;
   do p < N
       p := p * 3
   od
   ```

6. Determiná cuáles de las siguientes afirmaciones son verdades y cuáles falsas. Justificá apropiadamente.

   a) $\mathcal{O}(f + g) = \mathcal{O}(max(f, g))$.

   b) Si $s \in \mathcal{O}(f)$ y $r \in \mathcal{O}(g)$ entonces $s + r \in \mathcal{O}(f + g)$.

   c) Si $s \in \mathcal{O}(f)$ y $r \in \mathcal{O}(g)$ entonces $s - r \in \mathcal{O}(f - g)$.

   d) $2^{n+1} \in \mathcal{O}(2^n)$.

   e) $(m + 1)! \in \mathcal{O}(m!)$.

7. Ordená de la forma más precisa posible los $\mathcal{O}$ de las siguientes funciones, donde $0 < \varepsilon < 1$:

$$n^8,\ nlog(n),\ n^{1+\varepsilon},\ (1+\varepsilon)^n,\ n^2/log(n),\ (n^2-n+1)^4$$

8. Ordená de la forma más precisa posible los $\mathcal{O}$ de las siguientes funciones:

$$n \log 2^n,\ 2^n \log n,\ n! \log n,\ 2^n$$

9. Calculá el orden de los siguientes programas:

a)
```
i := 1;
do i ≤ N
    c := i;
    do c > 0
        operación_de_O(1);
        c := c/2;
    od
    i := i + 1;
od
```

b)
```
do n > 0 ∧ m > 0
    if n > m then n, m := m, n mod m
    else n, m := n, m mod n
od
if n = 0 then return m
else return n
```

10. Escribí programas cuyas complejidades sean:

a) $n^2 + 2 \log n$

b) $n^2 \log n$

c) 3^n

11. 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 programa que encuentre la cima de un arreglo dado (asumiendo que efectivamente existe una cima), utilizando la idea de *búsqueda binaria*.

b) Calculá el orden de complejidad del programa.

12. El *cocktail sort* es una variante bidireccional del *selection sort*, que busca los valores mínimo y máximo en cada paso, para luego ubicarlos en las posiciones apropiadas. Así se reduce el número de pasos en la búsqueda por un factor de 2, sin decrementar el número de comparaciones ni *swaps*.

a) Escribí un programa para este método de ordenación, siguiendo como patrón el *selection sort* visto en clase.

b) Demostrá que esta variante tiene el mismo orden de complejidad que el *selection sort* regular.

13. Recuperá tu solución al ejercicio 25 del Práctico 1 sobre implementación del TAD *lista* con la estructura de datos *lista ligada*.

a) Analizá la complejidad de cada procedimiento y función de tu implementación.

b) Diseñá una estructura de datos que generalice la idea de *lista ligada*, teniendo en mente la posibilidad de implementar un procedimiento *reverse* (que invierte la lista) con orden constante.

Ayuda: Investiga sobre *listas doblemente ligadas* y *listas XOR*. El ejercicio 23 sobre el TAD *pila reversible* también puede sugerir algunas ideas.

c) Reimplementá el TAD *lista*, extendido con la operación *reverse*, con la estructura de datos del ítem anterior.

14. Analizá la eficiencia de tu solución al ejercicio 4.

15. Resolvé las siguientes recurrencias:

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

b) $t(n) = \begin{cases} 1 & \text{si } n = 1 \\ 4 & \text{si } n = 2 \\ 8t(n-1) - 15t(n-2) & \text{si } n > 2 \end{cases}$

c) $t(n) = \begin{cases} 0 & \text{si } 1 \leq n \leq 2 \\ 1 & \text{si } n = 3 \\ 3t(n-2) - 2t(n-3) & \text{si } n > 3 \end{cases}$

d) $t(n) = \begin{cases} 5 & \text{si } n = 1 \\ 3t(n-1) - 2^{n-1} & \text{si } n > 1 \end{cases}$

e) $t(n) = \begin{cases} 0 & \text{si } 1 \leq n \leq 2 \\ 3t(n-1) - 4t(n-3) & \text{si } n > 2 \end{cases}$

f) $t(n) = \begin{cases} 1 & \text{si } 0 \leq n \leq 1 \\ \frac{t(n-1)+t(n-2)+12n-16}{2} & \text{si } n > 2 \end{cases}$

16. Dada la siguiente recurrencia

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

Demostrá que $t(n) \in \mathcal{O}(nlog^2(n))$.

17. Calculá el orden de complejidad de los siguientes programas:

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 n^3 do operación_de_O(1) od
```

b)
```
proc f2(in n : nat)
    for i := 1 to n do
        for j := 1 to i do operación_de_O(3) od
    od
    if n ≤ 0 then skip
    else
        for i := 1 to 4 do f2(n div 2) od
```

c)
```
proc f3(n : nat)
    for j := 1 to 6 do
        if n ≤ 1 then skip
        else
            for i := 1 to 3 do f3(n div 4) od
            for i := 1 to n^4 do operación_de_O(1) od
    od
```

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

```
proc f(in n : nat)
    if n ≤ 1 then skip
    else
        for i := 1 to K do f(n div L) od
        for i := 1 to n^4 do operación_de_O(1) od
```

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

a) $\Theta(n^4 \log n)$

b) $\Theta(n^4)$

c) $\Theta(n^5)$

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

```
fun minimo(i, k : int)  ret m : int
    if i = k then m := a[i]
    else
        j := (i + k) div 2
        m := min(minimo(i, j), minimo(j+1,k))
```

20. 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 $i = 6$).

Utilizá una idea similar a la *búqueda binaria* para escribir un programa que dado un arreglo a : **array**$[1..n]$ que almacena una secuencia de valores que tiene orden cíclico, encuentre el menor elemento de la secuencia (es decir, el valor alojado en la posición i). El programa debe ser de $\mathcal{O}(log(n))$. Analizá la eficiencia en forma detallada.