Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2012

Práctico 1

Docentes: Silvia Pelozo, Felipe Espósito, Emmanuel Gunther y Leandro Ramos

1. Escribí un programa que inicialice cada componente de un arreglo a de tamaño N con el valor 0. Especificá su comportamiento dando una pre y postcondición, y un invariante del ciclo.

2. Analizá el siguiente programa y especificá su comportamiento dando una pre y postcondición, y un invariante del ciclo:

   var a : **array**$[1..N]$ **of nat**
   var n, m : **nat**

   $\{Pre:\ ?\ \}$
   $n, m := 1, 0;$
   do $n \leq N$
   $\quad \{Inv:\ ?\ \}$
   $\quad$ **if** $a[n] \leq m$ **then skip**;
   $\quad$ **else** $m := a[n]$;
   $\quad n := n + 1;$
   od
   $\{Post:\ ?\ \}$

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

4. 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 disminuir el número de comparaciones ni intercambios.

   Escribí un programa para este método de ordenación, siguiendo el esquema de *selection sort* visto en clase.

5. Se desea acceder de forma *ordenada* a los elementos de un arreglo a : **array**$[1..N]$ **of int** pero sin modificar el 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, usando el algoritmo de ordenación por selección para determinar los elementos de la tabla.

6. 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.

7. Calculá de la manera más exacta y simple posible el número de operaciones sobre la variable t (asignaciones) de los siguientes programas:

   a)
   $t := 0;$
   for $i := 1$ **to** N **do**
   $\quad$ **for** $j := 1$ **to** N^2 **do**
   $\quad\quad$ **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)
```
t := 1;
do t < N
    t := t * 3
od
```

8. Analizá la cantidad de operaciones de los programas que escribiste en los ejercicios 4, 6 y 5.

9. Ordená los siguientes arreglos utilizando el algoritmo de ordenación por inserción. Mostrá en cada paso de iteración las comparaciones e intercambios realizados hasta ubicar el elemento en su posición.

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

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

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

10. Modificá el programa del ejercicio 5 para que utilice el algoritmo de inserción.

11. Otro algoritmo de ordenación es *bubble sort*, que recorre varias veces el arreglo comparando e intercambiando (si corresponde) elementos adyacentes. En cada recorrida al menos un elemento queda en su posición definitiva al final de la lista. La lista está ordenada cuando al recorrer la lista no se realizaron intercambios.

```
proc bubble_sort (in / out a : array[1..N] of int)
    var i, j: nat
    var swapped: bool

    swapped := true
    j := N - 1
    do swapped ∧ j ≥ 1
        swapped := false
        for i:= 1 to j do
            if a[i] > a[i+1] then
                swap (a, i, i+1)
                swapped := true
            fi
        od
        j := j - 1
    od
```

    Identificá el mejor y el peor caso para el algoritmo, es decir, el tipo de arreglos en que se realizan el menor y el mayor número de comparaciones (respectivamente), y el orden de cada uno de ellos.

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

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

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

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

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

15. Calculá el orden de los siguientes programas:

a)
```
i := 1;
do i ≤ N
    c := i;
    do c > 1
        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
```

16. Escribí programas cuyas complejidades sean:

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

b) $n^2 \log n$

c) 3^n

17. 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)* $\Theta(n^5)$

18. 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))
```

19. 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.

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.