Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2015

Práctico 1 - Parte 1

Objetivo: repaso de algoritmos elementales (ejercicios 1 y 4), familiarizarse con la notación **for** *(2 y 3) y con los algoritmos de ordenación más sencillos (5 y 6) y sus variantes (8, 9, 11, 15 y 16). Diferenciar el* ***qué*** *y el* ***cómo*** *de un algoritmo (4 y 9). Aprender cómo se ejecuta un algoritmo (5, 6, 9 y 11) y a contar sus operaciones (7 y 10). Reflexionar sobre la posibilidad del cómputo simultáneo o paralelo (12). Experimentar los beneficios de la abstracción para resolver problemas relacionados (13 y 14). El ejercicio 4 ya fue mencionado en el teórico. Además de resolverse acá en pseudo código, se implementará en el lenguaje c en el laboratorio.*

Dificultad estimada: *Primer nivel de dificultad: 1, 2, 3, 4, 5 y 6. Segundo: 7. Tercero: 8 y 12. Cuarto: 9, 10, 11, 16. Quinto: 13 y 14. Sexto: 15.*

1. Escribí un algoritmo para resolver cada uno de los siguientes problemas sobre un arreglo a de posiciones 1 a n, utilizando **do**
   (a) Inicializar cada componente del arreglo con el valor 0.
   (b) Inicializar el arreglo con los primeros n números naturales positivos.
   (c) Inicializar el arreglo con los primeros n números naturales impares.
2. Transformá cada uno de los algoritmos anteriores en uno equivalente que utilice **for** ... **to** .
3. Ahora, en uno equivalente que utilice **for** ... **downto** .
4. Escribí un algoritmo que reciba un arreglo a de posiciones 1 a n y determine si el arreglo recibido está ordenado o no. Explicá en palabras **qué** hace el algoritmo. Explicá en palabras **cómo** lo hace.
5. Ordená los siguientes arreglos, utilizando el algoritmo de ordenación por selección visto en clase. Mostrá en cada paso de iteración cuál es el elemento seleccionado y cómo queda el arreglo después de cada intercambio.
   (a) $[7, 1, 10, 3, 4, 9, 5]$ (b) $[5, 4, 3, 2, 1]$ (c) $[1, 2, 3, 4, 5]$
6. Ordená los arreglos del ejercicio 5 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.
7. Calculá de la manera más exacta y simple posible el número de asignaciones a la variable t de los siguientes algoritmos. Las ecuaciones que se encuentran al final del práctico pueden ayudarte.

(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)
```
t := 1;
do t < n
    t := t * 3
od
```

8. En cada uno de los siguientes algoritmos, descifrar qué hacen, explicar cómo lo hacen y reescribirlos asignando nombres adecuados a todos los identificadores

(a)
```
proc p (in/out a: array[1..n] of T)
    var x: nat
    for i:= n downto 2 do
        x:= f(a,i)
        swap(a,i,x)
    od
end proc
```

```
fun f (a: array[1..n] of T, i: nat) ret x: nat
    x:= 1
    for j:= 2 to i do
        if a[j] > a[x] then x:= j fi
    od
end fun
```

(b)
```
proc q (in/out a: array[1..n] of T)
    for i:= n-1 downto 1 do
        r(a,i)
    od
end proc
```

```
proc r (in/out a: array[1..n] of T, in i: nat)
    var j: nat
    j:= i
    do j < n ∧ a[j] > a[j+1] → swap(a,j+1,j)
                               j:= j+1
    od
end proc
```

9. El *cocktail sort* es la siguiente variante bidireccional del *selection sort*

```
proc cocktail_sort (in/out a: array[1..n] of T)
    var minp,maxp,last: nat
    for i:= 1 to n ÷ 2 do
        last:= n+1-i
        minp,maxp:= min_max_pos_between(a,i,last)
        swap(a,i,minp)
        if maxp = i then swap(a,last,minp)
        else swap(a,last,maxp)
        fi
    od
end proc
```

```
fun min_max_pos_between (a: array[1..n] of T, i, last: nat)
    ret minp,maxp: nat
    minp, maxp:= i, i
    for j:= i+1 to last do
        if a[j] < a[minp] then minp:= j
        else if a[j] ≥ a[maxp] then maxp:= j fi
        fi
    od
end fun
```

(a) ¿Qué hace este algoritmo?

(b) ¿Cómo lo hace?

(c) ¿Qué rol cumple el último condicional "**if** maxp = i **then** ..." del algoritmo principal? ¿Qué podría ocurrir si se reemplazara todo ese condicional simplemente por "swap(a,last,maxp)"?

(d) Mostrá cómo funciona el algoritmo ordenando los arreglos del ejercicio 5. Observá en qué casos es relevante la observación del inciso anterior.

10. Analizá la cantidad de operaciones del algoritmo del ejercicio 9 (cocktail_sort).

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

```
proc bubble_sort (in / out a : array[1..n]  of int)
    var i: nat
    var swapped: bool
    swapped := true
    i := 1
    do swapped ∧ i < n →
        swapped := false
        for  j:= n downto  i+1  do
            if a[j] < a[j-1]  then
                swap (a, j, j-1)
                swapped := true
            fi
        od
        i := i + 1
    od
end
```

(a) Mostrá cómo funciona *bubble sort* ordenando los arreglos del ejercicio 5.

(b) 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.

(c) Compará la cantidad de intercambios que realiza, en el peor y en el mejor caso, con respecto a *selection sort*.

12. En los algoritmos visto en los ejercicios 1 y 4, ¿encontrás acciones que podrían realizarse en forma simultánea en vez de secuencial, sin afectar el resultado? ¿Y en el caso del *bubble sort*?

13. Dada una función f, se desea ordenar los elementos de un arreglo a : **array**[1..n] **of T** de manera no decreciente según el valor de la f en cada celda del arreglo. Es decir que el arreglo se considerará ordenado si y sólo si para todo $1 \leq i < n$, f(a[i]) $\leq$ f(a[i+1]).

    ¿Cómo podrías hacer para aplicar cualquiera de los algoritmos de ordenación que conocés (tal vez introduciéndoles algún pequeño cambio) sin necesidad de inventar un algoritmo nuevo?

    ¿Cuántas comparaciones te parece que haría este algoritmo?

14. Aprovechando su algoritmo del ejercicio 13, en caso de que el dominio de la función f sea el conjunto {1, ..., n}, cómo podría hacer para generar un listado ordenado de los elementos del dominio. Es decir, generar la secuencia de los elementos de {1, ..., n} tal que f es no decreciente en esa secuencia.

15. (difícil) ¿Qué función cumple la variable swapped en el *bubble sort*? Se la puede reemplazar por una variable con valores naturales que recuerde en qué posición ocurrió el último swap. ¿Cómo se podría sacar provecho de esta información en la siguiente iteración? Escribir el *bubble sort* optimizado con esta idea.

16. Dar una versión del algoritmo de ordenación por inserción que solamente ordene las posiciones pares entre sí y las impares entre sí, pero sin mezclar elementos de posiciones pares con elementos de posiciones impares. Es decir, solo se están comparando entre sí celdas de posición congruente módulo 2. ¿Ayudaría en algo hacer esto antes de ordenar el arreglo con el *insertion sort*? Esta idea, más elaborada da lugar al algoritmo de ordenación llamado *shell sort*.

En las ecuaciones que siguen $n, m \in \mathbb{N}$ y k es una constante arbitraria:

$$\sum_{i=1}^{n} 1 = n \qquad \sum_{i=m}^{n} (k * a_i) = k * (\sum_{i=m}^{n} a_i)$$

$$\sum_{i=m}^{n} 1 = n - m + 1 \quad \text{si } n \geq m - 1 \qquad \sum_{i=m}^{n} (a_i + b_i) = (\sum_{i=m}^{n} a_i) + (\sum_{i=m}^{n} b_i)$$

$$\sum_{i=1}^{n} i = \frac{n * (n+1)}{2} \qquad \sum_{i=m}^{n} (a_i - b_i) = (\sum_{i=m}^{n} a_i) - (\sum_{i=m}^{n} b_i)$$

$$\sum_{i=1}^{n} i^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \qquad \sum_{i=0}^{n} a_{n-i} = \sum_{i=0}^{n} a_i$$

$$\sum_{i=1}^{n} i^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$$

La última ecuación de la derecha dice simplemente que:

$$a_n + a_{n-1} + \ldots + a_1 + a_0 = a_0 + a_1 + \ldots + a_{n-1} + a_n$$