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, 10, 15 y 16). Diferenciar el* ***qué*** *y el* ***cómo*** *de un algoritmo (4). Aprender cómo se ejecuta un algoritmo (5, 6, 8 y 10) y a contar sus operaciones (7, 9 y 14). Reflexionar sobre la posibilidad del cómputo simultáneo o paralelo (11). Experimentar los beneficios de la abstracción para resolver problemas relacionados (12 y 13). 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: 11. Cuarto: 8, 9, 10, 16. Quinto: 12, 13, 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) $[1, 2, 3, 4, 5]$ (b) $[5, 4, 3, 2, 1]$ (c) $[7, 1, 10, 3, 4, 9, 5]$
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 operaciones sobre la variable t (asignaciones) 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. 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.

   (a) Escribí un algoritmo que encuentre el mínimo y el máximo de un arreglo a de 1 a N.

   (b) Escribí un algoritmo que encuentre las **posiciones** del mínimo y del máximo del arreglo entre las posiciones i y $N - i + 1$ (asumiendo que $i < N - i + 1$, o sea $2 * i - 1 < N$). Hacé un gráfico del arreglo para entender el significado de estas posiciones.

   (c) Lo mismo que en el inciso anterior, pero que en caso de ser **iguales todos** los elementos del arreglo entre las posiciones i y $N - i + 1$, las posiciones encontradas para el mínimo y máximo deben ser diferentes.

   (d) Escribí el *cocktail sort*. Te conviene seguir el esquema de la **versión abreviada** del *selection sort*.

   (e) Mostrá cómo funciona el algoritmo ordenando los arreglos del ejercicio 5.

9. Analizá la cantidad de operaciones del cocktail sort que escribiste en el ejercicio 8 (cocktail).

10. 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*.

11. 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?

12. 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 $[16, 12, 6, 4, 0, 3, 7, 25]$.

    ¿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?

13. 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 $1 \leq i < N$. Por ejemplo, para $a = [4, 7, 1, 3, 9]$ la tabla es $b = [3, 4, 1, 2, 5]$.

    ¿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?

14. ¿Te animás a predecir la cantidad de operaciones de los algoritmos que escribiste en los ejercicios 12 (pares_impares) y 13 (tabla) sin hacer "nuevamente" las cuentas? ¿Cómo lo justificarías?

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:

$$\sum_{i=1}^{N} 1 = N \qquad \sum_{i=M}^{N} (k * a_i) = k * (\sum_{i=M}^{N} a_i)$$

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

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

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

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

La última ecuación de la derecha dice simplemente que:

$$a_N + a_{N-1} + \ldots + a_1 + a_0 = a_0 + a_1 + \ldots + a_{N-1} + a_N$$