Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2013

Práctico 1 - Parte 1

1. Escribí tres programas para resolver cada uno de los siguientes problemas sobre un arreglo a de tamaño N: uno que utilice **do** , un segundo que utilice **for** ... **to** y un tercero que utilice **for** ... **downto** .

   a) Inicializar cada componente del arreglo con el valor 0.

   b) Inicializar el arreglo con los primeros N números naturales.

   c) Inicializar el arreglo con los primeros N números naturales impares.

2. 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*) $[7, 1, 10, 3, 4, 9, 5]$

3. 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.

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

4. 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.

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 $1 \leq i < N$. 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. 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
       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
   ```

7. Analizá la cantidad de operaciones de los programas que escribiste en los ejercicios 3, 4 y 5.

8. 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*) $[7, 1, 10, 3, 4, 9, 5]$

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

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 algunos arreglos (por ejemplo, los del ejercicio 8).

   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*.