Algoritmos y Estructuras
de Datos II

EJERCICIOS RESUELTOS - PRÁCTICO 1 - ORDENACIÓN ELEMENTAL

Pedro Villar

1 Ejercicio 1

Escribí algoritmos para resolver cada uno de los siguientes problemas sobre un arreglo a de posiciones 1 a n, utilizando do. Elegí en cada caso entre estos dos encabezados el que sea más adecuado:

```
proc nombre (in/out a: array [1..n] of nat)
    ...
end proc
```

```
proc nombre (out a: array [1..n] of nat)
    ...
end proc
```

(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.

(d) Incrementar las posiciones impares del arreglo y dejar intactas las posiciones pares.

1.1 Solución (a)

```
proc inicializarConCero (out a: array [1..n] of nat)
    i := 1
    do i <= n ->
    a[i] := 0
    i := i + 1
    od
end proc
```

1.2 Solución (b)

```
proc inicializarConNaturales (out a: array [1..n] of nat)
    i := 1
    do i <= n ->
    a[i] := i
    i := i + 1
    od
end proc
```

1.3 Solución (c)

```
proc inicializarConImpares (out a: array [1..n] of nat)
    i := 1
    do i <= n ->
    a[i] := 2 * i - 1
    i := i + 1
    od
end proc
```

1.4 Solución (d)

```
proc incrementarImpares (in/out a: array [1..n] of nat)
    i := 1
    do i <= n ->
    if i mod 2 = 1 then
        a[i] := a[i] + 1
    fi
    i := i + 1
    od
end proc
```

2 Ejercicio 2

Transformá cada uno de los algoritmos anteriores en uno equivalente que utilice **for...to**

2.1 Solución (a)

```
proc inicializarConCero (out a: array [1..n] of nat)
    for i := 1 to n do
    a[i] := 0
    od
end proc
```

2.2 Solución (b)

```
proc inicializarConNaturales (out a: array [1..n] of nat)
    for i := 1 to n do
    a[i] := i
    od
end proc
```

2.3 Solución (c)

```
proc inicializarConImpares (out a: array [1..n] of nat)
    for i := 1 to n do
    a[i] := 2 * i - 1
    od
end proc
```

2.4 Solución (d)

```
proc incrementarImpares (in/out a: array [1..n] of nat)
    for i := 1 to n do
    if i mod 2 = 1 then
        a[i] := a[i] + 1
    fi
    od
end proc
```

3 Ejercicio 3

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.

3.1 Solución

```
fun estaOrdenado (a: array [1..n] of nat) ret r: bool
    r := true
    for i := 1 to n - 1 do
    if a[i] > a[i + 1] then
        r := false
    else
        skip
    fi
    od
end fun
```

¿Qué hace el algoritmo? El algoritmo recibe un arreglo de números naturales y retorna un valor booleano que indica si el arreglo está ordenado o no.

¿Cómo lo hace? El algoritmo recorre el arreglo desde la primera posición hasta la penúltima, comparando cada elemento con el siguiente. Si encuentra un elemento mayor que el siguiente, retorna **false**, indicando que el arreglo no está ordenado. Si recorre todo el arreglo sin encontrar un elemento mayor que el siguiente, retorna **true**, indicando que el arreglo está ordenado.

4 Ejercicio 4

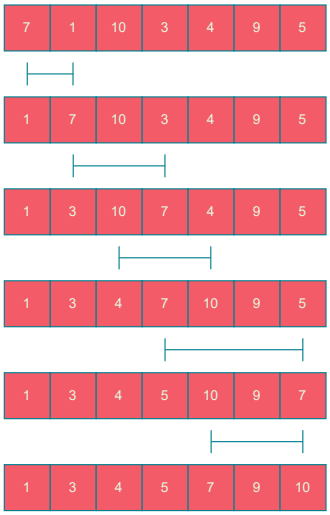
Ordená los siguientes arreglos, utilizando el algoritmo de ordenación por selección visto en clase. Mostrá en cada paso de iteración cuál es el elemento seleccionado y cómo queda el arreglo después de cada intercambio.

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

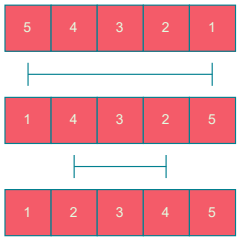
(b) $[5, 4, 3, 2, 1]$

(c) $[1, 2, 3, 4, 5]$

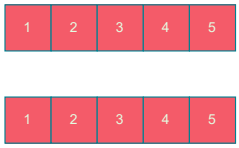
4.1 Solución (a)



4.2 Solución (b)



4.3 Solución (c)



5 Ejercicio 5

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.

```
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
    od
    od
od
```

```
t := 0
for i := 1 to n do
    for j := 1 to i do
    for k := j to j+3 do
        t := t + 1
    od
    od
od
```

5.1 Solución (a)

```
ops(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
    od
    od
od)
```

Escribo el bucle en forma de sumatorias

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{n^2}\right)\sum_{k=1}^{n^3} t := t+1\Bigg)\Bigg)$$

Ahora aplico la función ops a la secuencia:

$$ops(t := 0) + \sum_{i=1}^{n}\left(ops\left(\sum_{j=1}^{n^2}\sum_{k=1}^{n^3} t := t+1\right)\right)$$

$= \{\ ops = \sum_{j=1}^{n^2}$ es una sumatoria de n^2 elementos, por lo que vale n^2 $\}$

$$ops(t := 0) + \sum_{i=1}^{n}\left(n^2 \cdot ops\left(\sum_{k=1}^{n^3} t := t+1\right)\right)$$

$= \{\ ops(t := 0)$ es una asignación, por lo que vale 1 $\}$

$$1 + \sum_{i=1}^{n}\left(n^2 \cdot ops\left(\sum_{k=1}^{n^3} t := t+1\right)\right)$$

= { $ops\left(\sum_{k=1}^{n^3} t := t+1\right)$ es una sumatoria de n^3 elementos, por lo que vale n^3 }

$$1 + \sum_{i=1}^{n} \left(n^2 \cdot n^3\right)$$

= { Resuelvo la sumatoria interna }

$$1 + \sum_{i=1}^{n} \left(n^5\right)$$

= { Resuelvo la sumatoria externa }

$$1 + n \cdot n^5$$

= { Resuelvo la multiplicación }

$$1 + n^6$$

5.2 Solución (b)

```
ops(t := 0
for i := 1 to n do
    for j := 1 to i do
    for k := j to j+3 do
        t := t + 1
    od
    od
od)
```

Escribo el bucle en forma de sumatorias

$$\sum_{i=1}^{n} \left(\sum_{j=1}^{i} \right) \sum_{k=j}^{j+3} t := t+1 \Bigg) \Bigg)$$

Ahora aplico la función ops a la secuencia:

$$ops(t := 0) + \sum_{i=1}^{n} \left(ops \left(\sum_{j=1}^{i} \sum_{k=j}^{j+3} t := t+1 \right) \right)$$

= { $ops = \sum_{j=1}^{i}$ es una sumatoria de i elementos, por lo que vale i }

$$ops(t := 0) + \sum_{i=1}^{n} \left(i \cdot ops \left(\sum_{k=j}^{j+3} t := t+1 \right) \right)$$

= { $ops(t := 0)$ es una asignación, por lo que vale 1 }

$$1 + \sum_{i=1}^{n} \left(i \cdot ops \left(\sum_{k=j}^{j+3} t := t+1 \right) \right)$$

= { $ops\left(\sum_{k=j}^{j+3} t := t+1\right)$ es una sumatoria de 4 elementos, por lo que vale 4 }

$$1 + \sum_{i=1}^{n} (4 \cdot i)$$

= { Resuelvo la sumatoria }

$$1 + 4 \cdot \sum_{i=1}^{n} (i)$$

= { Resuelvo la sumatoria }

$$1 + 4 \cdot \frac{n \cdot (n+1)}{2}$$

= { Resuelvo la multiplicación }

$$1 + 2 \cdot n \cdot (n+1)$$

= { Resuelvo la multiplicación }

$$1 + 2 \cdot n^2 + 2 \cdot n$$

6 Ejercicio 6

Descifrá qué hacen los siguientes algoritmos, explicar cómo lo hacen y reescribirlos asignando nombres adecuados a todos los identificadores.

```
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
```

6.1 Solución (a)

Algoritmo p: El algoritmo recibe un arreglo de elementos de tipo T y lo ordena de manera descendente. Para ello, recorre el arreglo desde la última posición hasta la segunda, en cada iteración busca el elemento más grande en el subarreglo que va desde la primera posición hasta la posición actual y lo intercambia con el elemento en la posición actual. Se podria escribir de la siguiente manera:

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

6.2 Solución (b)

Función f: La función recibe un arreglo de elementos de tipo T y un número natural i, y retorna la posición del elemento más grande en el subarreglo que va desde la primera posición hasta la posición i. Para ello, recorre el subarreglo desde la segunda posición hasta la posición i, en cada iteración compara el elemento actual con el elemento más grande encontrado hasta el momento y si el elemento actual es mayor, actualiza la posición del elemento más grande. Se podria escribir de la siguiente manera:

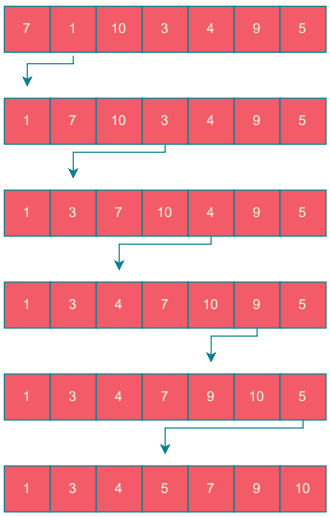
Solución (b)

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

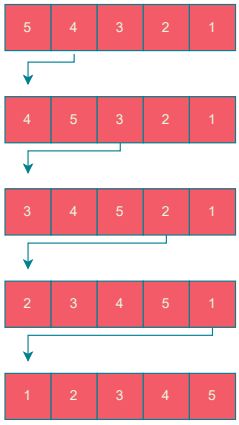
7 Ejercicio 7

Ordená los arreglos del ejercicio 4 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.

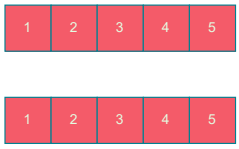
7.1 Solución (a)



7.2 Solución (b)



7.3 Solución (c)



8 Ejercicio 8

Calculá el orden del número de asignaciones a la variable t de los siguientes algoritmos.

```
t := 1
do t < n
    t := t * 2
od
```

```
t := n
do t > 0 do
    t := t div 2
od
```

```
for i := 1 to n do
    t := i
    do t > 0 do
    t := t div 2
    od
od
```

```
for i := 1 to n do
    t := i
    do t > 0 do
    t := t - 2
    od
od
```

8.1 Solución (a)

El algoritmo realiza una asignación y luego realiza una multiplicación en cada iteración. El número de iteraciones es el menor número k tal que $2^k \geq n$. Por lo tanto, el orden del número de asignaciones es $O(\log_2 n)$.

8.2 Solución (b)

El algoritmo realiza una asignación y luego realiza una división en cada iteración. El número de iteraciones es el menor número k tal que $n \div 2^k = 0$. Por lo tanto, el orden del número de asignaciones es $O(\log_2 n)$.

8.3 Solución (c)

El algoritmo realiza una asignación y luego realiza una división en cada iteración. El número de iteraciones es el número n. Por lo tanto, el orden del número de asignaciones es $O(n \cdot \log_2 n)$.

8.4 Solución (d)

El algoritmo realiza una asignación y luego realiza una resta en cada iteración. El número de iteraciones es el número n. Por lo tanto, el orden del número de asignaciones es $O(n)$.

9 Ejercicio 9

Calculá el orden del número de comparaciones del algoritmo del ejercicio 3.

9.1 Solución

```
fun estaOrdenado (a: array [1..n] of nat) ret r: bool
    for i := 1 to n - 1 do
    if a[i] > a[i + 1] then
        r := false
    else
        skip
    fi
    od
end fun
```

El algoritmo realiza una comparación en cada iteración del bucle. El número de iteraciones es el menor número k tal que $i + k \geq n - 1$. Por lo tanto, el orden del número de comparaciones es $O(n)$.

$$ops\left(\sum_{i=1}^{n-1}(if...else)\right)$$

$$ops\left(\sum_{i=1}^{n-1} 1\right) = n$$

10 Ejercicio 10

Descifrá qué hacen los siguientes algoritmos, explicar cómo lo hacen y reescribirlos asignando nombres adecuados a todos los identificadores.

```
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
```

10.1 Solución (a)

Algoritmo q: El algoritmo recibe un arreglo de elementos de tipo T y lo ordena de manera ascendente. Para ello, recorre el arreglo desde la penúltima posición hasta la primera, en cada iteración llama a la función r con el arreglo y la posición actual. Se podria escribir de la siguiente manera:

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

10.2 Solución (b)

Algoritmo r: El algoritmo recibe un arreglo de elementos de tipo T y un número natural i, y realiza un intercambio entre el elemento en la posición j y el elemento en la posición $j+1$ hasta que el elemento en la posición j sea menor o igual que el elemento en la posición $j+1$ o hasta que la posición j sea la última posición del arreglo. Se podria escribir de la siguiente manera:

```
proc ordenar (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
```