1 - 2)

- 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:
- Transformá cada uno de los algoritmos anteriores en uno equivalente que utilice `for ... to`.

```
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.

```
proc initArrayWith0 (out a:array[1..n] of nat)
    var i: nat
    i := 1
    do (i <= n)
        a[i] := 0
        i := i + 1
    od
end proc

proc initArrayWith0 (out a:array[1..n] of nat)
    for i := 1 to n do
        a[i] := 0
    od
end proc
```

b

> Inicializar el arreglo con los primeros n números naturales positivos.

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

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

```
        a[i] := i
    od
end proc
```

c

> Inicializar el arreglo con los primeros n números naturales impares.

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

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

d

> Incrementar las posiciones impares del arreglo y dejar intactas las posiciones pares.

```
proc incrementArrayAtOddPos (out a:array[1..n] of nat)
    var i: nat
    i := 1
    do (i <= n)
        a[i] := a[i] + 1
        i := i + 2
    od
end proc

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

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.

```
fun isOrdecolor{red} (a:array[1..n] of nat) ret r: bool
    var isGreating, isLowing: bool

    isGreating := true
    isLowing := true
```

```
    for i := 1 to n - 1 do
        if a[i] < a[i + 1] then isLowing := false
        else
            if a[i] > a[i + 1] then isGreating := false fi
        fi
    od

    r := isGreating || isLowing
end fun
```

- Revisa si el arreglo está ordenado, ya sea decreciente o crecientemente.
- Recorre el arreglo y revisa si a en i es menor o mayor a a en $(i+1)$, si es menor, significa que no está ordenado crecientemente, y si es mayor, que no está ordenado decrecientemente. Si alguna de las dos variables que registra el orden nunca fue modificada, entonces r es $true$, caso contario $false$.

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

1	2	3	4	5	6	7
7	1	10	3	4	9	5
7	$\color{red}{1}$	10	3	4	9	5
$\color{green}{1}$	7	10	3	4	9	5
$\color{green}{1}$	7	10	$\color{red}{3}$	4	9	5
$\color{green}{1}$	$\color{green}{3}$	10	7	4	9	5
$\color{green}{1}$	$\color{green}{3}$	10	7	$\color{red}{4}$	9	5
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	7	10	9	5
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	7	10	9	$\color{red}{5}$
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	$\color{green}{5}$	10	9	7
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	$\color{green}{5}$	10	9	$\color{red}{7}$
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	$\color{green}{5}$	$\color{green}{7}$	9	10
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	$\color{green}{5}$	$\color{green}{7}$	$\color{red}{9}$	10
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	$\color{green}{5}$	$\color{green}{7}$	$\color{green}{9}$	10
$\color{green}{1}$	$\color{green}{3}$	$\color{green}{4}$	$\color{green}{5}$	$\color{green}{7}$	$\color{green}{9}$	$\color{green}{10}$

b

1	2	3	4	5
5	4	3	2	1
5	4	3	2	1
1	4	3	2	5
1	4	3	2	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

c

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

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.

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
        od
    od
od
```

$$
\begin{aligned}
asignacionesT &= opt(t := 0; for\ i := 1\ to\ n\ do) \\
&= 1 + \sum_{i=1}^{n} opt(for\ j := 1\ to\ n^2\ do) \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{n^2} opt(for\ k := 1\ to\ n^3\ do) \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{n^2} \sum_{k=1}^{n^3} opt(t := t + 1) \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{n^2} \sum_{k=1}^{n^3} 1 \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{n^2} n^3 \\
&= 1 + \sum_{i=1}^{n} (n^2 * n^3) \\
&= 1 + (n * n^5) \\
asignacionesT &= n^6 + 1
\end{aligned}
$$

b

```
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
```

$$
\begin{aligned}
asignacionesT &= opt(t := 0; for\ i := 1\ to\ n\ do) \\
&= 1 + \sum_{i=1}^{n} opt(for\ j := 1\ to\ i\ do) \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{i} opt(for\ k := j\ to\ j + 3\ do) \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{j+3} opt(t := t + 1) \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{j+3} 1 \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{i} ((j + 3) - j + 1) \\
&= 1 + \sum_{i=1}^{n} \sum_{j=1}^{i} 4 \\
&= 1 + 4 * \sum_{i=1}^{n} \sum_{j=1}^{i} 1 \\
&= 1 + 4 * \sum_{i=1}^{n} i \\
&= 1 + 4 * (n * (n + 1))/2 \\
&= 1 + 2n(n + 1) \\
asignacionesT &= n^6 + 1
\end{aligned}
$$

6)

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

```
# Ordena un arreglo ascendentemente en base a los máximos. Método inverso a SelectionSort.
proc orderASC (in/out a: array[1..n] of T)
    var maxValuePos: nat
    for pos := n downto 2 do
        maxValuePos := maxValuePosFinder(a, pos)
        swap(a, pos, maxValuePos)
    od
end proc

# Encuentra el índice del máximo valor del array entre el inicio y una posición delimiter.
fun maxValuePosFinder (a: array[1..n] of T, delimiter: nat) ret maxValuePos: nat
    maxValuePos:= 1
    for pos := 2 to delimiter do
        if a[pos] > a[maxValuePos] then
            maxValuePos := pos
        fi
    od
end fun
```

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.

a

1	2	3	4	5	6	7
7	1	10	3	4	9	5
7	1	10	3	4	9	5
1	7	10	3	4	9	5
1	7	10	3	4	9	5
1	7	10	3	4	9	5
1	7	10	3	4	9	5
1	7	3	10	4	9	5
1	3	7	10	4	9	5
1	3	7	10	4	9	5
1	3	7	4	10	9	5
1	3	4	7	10	9	5
1	3	4	7	10	9	5
1	3	4	7	9	10	5
1	3	4	7	9	10	5
1	3	4	7	9	5	10
1	3	4	7	5	9	10
1	3	4	5	7	9	10
1	3	4	5	7	9	10

b

1	2	3	4	5
5	4	3	2	1
5	4	3	2	1
4	5	3	2	1
4	5	3	2	1
4	3	5	2	1
3	4	5	2	1
3	4	5	2	1
3	4	2	5	1
3	2	4	5	1
2	3	4	5	1
2	3	4	5	1
2	3	4	1	5
2	3	1	4	5
2	1	3	4	5
1	2	3	4	5
1	2	3	4	5

c

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

8)

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

a

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

$$2^k < n <= 2^{k+1}$$

$$\begin{aligned} asignacionesT &= opt(t := 1; do\ t < n) \\ &= 1 + (k+1) \\ &= k + 2 \\ &\leq \log n + 2 \end{aligned}$$

b

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

$$2^k <= n < 2^{k+1}$$

$$\begin{aligned} asignacionesT &= opt(t := n; do\ t > 0) \\ &= 1 + opt(do\ t > 0) \\ &= 1 + (k+1) \\ &= k + 2 \\ &\leq \log n + 2 \end{aligned}$$

c

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

$$2^k <= i < 2^{k+1}$$
$$2^j <= n! < 2^{j+1}$$

$$\begin{aligned}
asignacionesT &= opt(for\ i := 1\ to\ n\ do) \\
&= \sum_{i=1}^{n} opt(t := i; do\ t > 0) \\
&= \sum_{i=1}^{n}(1 + opt(do\ t > 0)) \\
&= \sum_{i=1}^{n}(k_i + 2) \\
&= 2n + \sum_{i=1}^{n} k_i \\
&= 2n + j \\
&\leq 2n + \sum_{i=1}^{n} \log i \\
&\leq 2n + \log(n!)
\end{aligned}$$

d

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

$$2k < i <= 2(k+1)$$
$$2j <= n < 2(j+1)$$

$$\begin{aligned}
asignacionesT &= opt(for\ i := 1\ to\ n\ do) \\
&= \sum_{i=1}^{n} opt(t := i; do\ t > 0) \\
&= \sum_{i=1}^{n}(1 + opt(do\ t > 0)) \\
&= n + \sum_{i=1}^{n} k_i \\
&= n + \sum_{i=1}^{n} k_i \\
&= n + \sum_{i=1}^{n}((i+1)\ \backslash div\backslash\ 2) \\
&= n + 0.5(n + \sum_{i=1}^{n} i) \\
&= n + 0.5n + 0.5(n(n+1)/2) \\
&= 1.5n + n(n+1)/4 \\
&= 0.5n((n+7)/2)
\end{aligned}$$

```
    (k - 1) < i/2 <= k

    2 sum _{i = 1} ^k i
    k^2 o k^2 + k
    asignacionesT   = sum _{i = 1} ^n { 1 + k_i }
                    = n + sum _{i = 1} ^n { k_i }
                    = n + (k^2 o k^2 + k)
```

9)

```
fun isOrdecolor{red} (a:array[1..n] of nat) ret r: bool
    var isGreating, isLowing: bool
```

```
    isGreating := true
    isLowing := true

    for i := 1 to n - 1 do
        if a[i] < a[i + 1] then isLowing := false
        else
            if a[i] > a[i + 1] then isGreating := false fi
        fi
    od

    r := isGreating || isLowing
end fun
```

$$\begin{aligned}
comparaciones &= opt(isGreating := true; isLowing := true; for\ i := 1\ to\ n-1\ do) \\
&= opt(for\ i := 1\ to\ n-1\ do) \\
&= \sum_{i=1}^{n-1} 1o2
\end{aligned}$$

$$\begin{aligned}
Best(comparaciones) &= (n-1) \\
Worst(comparaciones) &= 2(n-1)
\end{aligned}$$

10)

```
// Ordena un array ascendentemente.
proc orderASC (in/out a: array[1..maxPos] of T)
    for pos:= maxPos - 1 downto 1 do
        swapSmallersFromPos(a, pos)
    od
end proc

// Ordena ascendentemente un array desde initAtPos hasta maxPos.
proc swapSmallersFromPos (in/out a: array[1..maxPos] of T, in initAtPos: nat)
    var pos: nat
    pos := initAtPos

    do (pos < maxPos && a[pos] > a[pos + 1])
        swap(a, pos + 1, pos)
        pos := pos + 1
    od
end proc
```