Practico 1

1)a)

```
proc inicia_0(out a: array[1..n] of nat)
  for i := 1 to n do
    a[i] = 0
  od
end proc
```

b)

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

c)

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

d)

```
proc incrementa_impares(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
    else
      skip
    fi
  od
end proc
```

3)

```
fun esta_ordenado(a: array[1..n] of int) 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
```

Que: El algoritmo comprueba si el arreglo recibido está ordenado o no.

Como: El valor por defecto que devuelve la funcion es true al inicio, recorre el array con un bucle for, y si algun elemento del array es mayor al elemento siguiente, la funcion devuelve false

4)

a)

$$
\begin{gathered}
[7, 1, 10, 3, 4, 9, 5] \\
\equiv \{\text{elemento seleccionado: } 1\} \\
[\mathbf{1}, 7, 10, 3, 4, 9, 5] \\
\equiv \{\text{elemento seleccionado: } 3\} \\
[\mathbf{1}, \mathbf{3}, 10, 7, 4, 9, 5] \\
\equiv \{\text{elemento seleccionado: } 4\} \\
[\mathbf{1}, \mathbf{3}, \mathbf{4}, 7, 10, 9, 5] \\
\equiv \{\text{elemento seleccionado: } 5\} \\
[\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, 10, 9, 7] \\
\equiv \{\text{elemento seleccionado: } 7\} \\
[\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{7}, 9, 10] \\
\equiv \{\text{elemento seleccionado: } 9\} \\
[\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{7}, \mathbf{9}, 10] \\
\equiv \{\text{elemento seleccionado: } 10\} \\
[\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{7}, \mathbf{9}, \mathbf{10}]
\end{gathered}
$$

5)

a)

ops(

```
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
```

$) = \sum_{i=1}^{n}(\sum_{i=1}^{n^2}(\sum_{i=1}^{n^3} t = t + 1))$

Ahora resolvamos las sumatorias:

$$ops(t := 0) + \sum_{i=1}^{n}\left(\sum_{i=1}^{n^2}\left(\sum_{i=1}^{n^3} t = t + 1\right)\right)$$
$$\equiv \{ops\,(t := t + 1) = 1\}$$
$$ops(t := 0) + \sum_{i=1}^{n}\left(\sum_{i=1}^{n^2}\left(\sum_{i=1}^{n^3} 1\right)\right)$$
$$\equiv \left\{\sum_{i=1}^{n} 1 = n\right\}$$
$$ops(t := 0) + \sum_{i=1}^{n}\left(\sum_{i=1}^{n^2} n^3\right)$$
$$\equiv \{\text{sumatoria de constante}\}$$
$$ops(t := 0) + \sum_{i=1}^{n} n^2 * n^3$$
$$\equiv \{\text{sumatoria de constante}\}$$
$$ops(t := 0) + n * n^2 * n^3$$
$$\equiv \{ops\,(t := 0) = 1\}$$
$$1 + n * n^2 * n^3$$
$$\equiv \{\text{Aritmetica}\}$$
$$1 + n^6$$

Comprobacion hecha en python:

```
def test1(n):
    t = 0
    for i in range(1, n+1):
        for j in range(1, n**2 +1):
            for k in range(1, n**3 +1):
                t = t+1
    return t

print(test1(2) + 1)
```

Output: `65`

Comprobacion: $n = 2 \Rightarrow 1 + 2^6 = 1 + 64 = 65$

b)

ops($t := 0$) + ops(

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

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

Resolvamos las sumatorias:

$$\sum_{i=1}^{n}(\sum_{j=1}^{i}(\sum_{k=j}^{j+3} t = t+1))$$
$\equiv$ {ops(t:= t +1) = 1}
$$\sum_{i=1}^{n}(\sum_{j=1}^{i}(\sum_{k=j}^{j+3} 1))$$
$\equiv$ {sumatoria de 1}
$$\sum_{i=1}^{n}(\sum_{j=1}^{i} j+3-j+1)$$
$\equiv$ {Aritmetica}
$$\sum_{i=1}^{n}(\sum_{j=1}^{i} 4)$$
$\equiv$ {Aritmetica}
$$\sum_{i=1}^{n}(\sum_{j=1}^{i} 4*1)$$
$\equiv$ {Props sumatoria}
$$\sum_{i=1}^{n}(4*\sum_{j=1}^{i} 1)$$
$\equiv$ {Props sumatoria}
$$\sum_{i=1}^{n}(4*i)$$
$\equiv$ {Props sumatoria}
$$4*\sum_{i=1}^{n} i$$
$\equiv$ {Props sumatoria}
$$4*\frac{n(n+1)}{2}$$
$\equiv$ {Aritmetica}
$$2*n(n+1)$$

Por lo cual:

$ops(t := 0) + ops(for..od) = 1 + 2 * n(n+1)$

Comprobacion en python:

```
def test2(n):
    t = 0
    for i in range(1, n+1):
        for j in range(1, i+1):
            for k in range(j, j+3 +1):
                t = t+1
    return t

print(test2(2) + 1)
```

Output: 13

Comprobacion: $n = 2 \Rightarrow 2 * 2(2+1) + 1 = 4 * 3 + 1 = 12 + 1 = 13$

6) Pendiente

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,**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]

8)

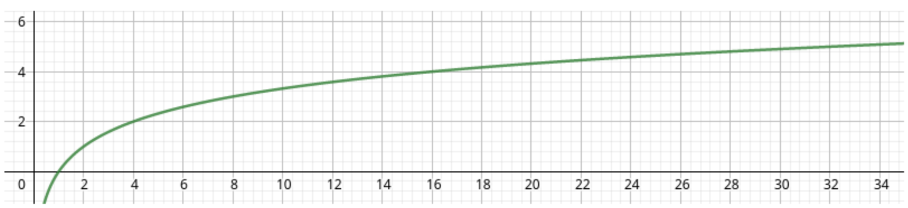
a)

Hagamos una tablita para tener una idea

n	ops	n	ops
1	0	18	5
2	1	19	5
3	2	20	5
4	2	21	5

n	ops	n	ops
5	3	22	5
6	3	23	5
7	3	24	5
8	3	25	5
9	4	26	5
10	4	27	5
11	4	28	5
12	4	29	5
13	4	30	5
14	4	31	5
15	4	32	5
16	4	33	6
17	5		

Como vemos que la cantidad de operaciones va cambiando en cada potencia de 2, podemos intuir que la complejidad es de $\log_2 n$



b)

n	ops	n	ops
1	0	18	5
2	1	19	5
3	2	20	5
4	2	21	5
5	3	22	5
6	3	23	5
7	3	24	5
8	3	25	5
9	4	26	5
10	4	27	5
11	4	28	5
12	4	29	5
13	4	30	5
14	4	31	5
15	4	32	5
16	4	33	6

n	ops	n	ops
17	5		

Por ende la complejidad es de $\log_2(n)$

c)

$$\begin{aligned}
& ops(for..do) \\
&= \sum_{i=1}^{n} ops(B(i)) \\
&= \sum_{i=1}^{n} \log_2(i) \\
&= \log_2(1) + \log_2(2) + \cdots + \log_2(n) \\
&\quad \equiv \{\text{Prop:} \quad \log(a) + \log(b) = \log(a * b)\} \\
&= \log_2\left(\prod_{i=1}^{n} i\right) \\
&= \log_2(n!) \\
&\quad \equiv \{\log_2(1 * 2 * \cdots * n) \leq \log_2(n * n * \cdots * n)\} \\
&= \log_2(n^n) \\
&= n * \log_2(n)
\end{aligned}$$

Complejidad: $n * \log_2(n)$

d) Primero veamos la complejidad del do..od .

ops(t:=n) + ops(

do $t > 0$
  t:= t-2
od

)

Hagamos una tabla:

n	ops	n	ops
1	1	9	5
2	1	10	5
3	2	11	6
4	2	12	6
5	3	13	7
6	3	14	7
7	4	15	8
8	4	16	8

Por lo cual podemos decir que la complejidad es de:

$$ops(t := n) + \lceil \frac{n}{2} \rceil$$
$$= 1 + \lceil \frac{n}{2} \rceil$$
$$= n$$

Llamemos B al do..od y veamos la complejidad del for..od .

$$ops(for..od)$$
$$= \sum_{i=1}^{n} ops(B(i))$$
$$= \sum_{i=1}^{n} i$$
$$= \frac{n * (n+1)}{2}$$
$$= \frac{n^2 + n}{2}$$
$$= n^2$$

Por ende, la complejidad es de n^2

9)

```
fun esta_ordenado(a: array[1..n] of int) 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
```

Como el if solo realiza una comparacion sabemos que $ops(if..else) = 1$

Ahora veamos las operaciones del for:

$$\begin{aligned} & ops(for..od) \\ & = \sum_{i=1}^{n} ops(if..else) \\ & = \sum_{i=1}^{n} 1 \\ & = n \end{aligned}$$

Por ende, el numero de comparaciones del ejercicio 3 es de n.