1)

a)

Definimos la funcion recursiva $r(n)$ de la siguiente manera:

$$r(n) = \begin{cases} 0 \\ \sum_{i=1}^{8} r(n \ div \ 2) + \sum_{i=1}^{n^3} 1 \end{cases}$$

$$\vdots$$

$$r(n) = \begin{cases} 0 \\ 8 * r(n \ div \ 2) + n^3 \end{cases}$$

Por ende el numero de llamadas recursivas es $a = 8$

El problema se va achicando a la mitad con cada iteracion, por lo cual $b = 2$

Y el grado del polinomio g(n) es 3, por ende $k = 3$

Finalmente, como $a = 8$ y $b^k = 8$, entonces tenemos que la funcion es del orden de $n^3 \ log(n)$

b)

$$r(n) = \begin{cases} \sum_{i=1}^{n} \sum_{j=1}^{i} 1 \\ \sum_{i=1}^{n} \sum_{j=1}^{i} 1 + \sum_{i=1}^{4} r(n \ div \ 2) \end{cases}$$

$$\vdots$$

$$r(n) = \begin{cases} \sum_{i=1}^{n} i \\ \sum_{i=1}^{n} i + 4 * r(n \ div \ 2) \end{cases}$$

$$\vdots$$

$$r(n) = \begin{cases} \frac{n*(n+1)}{2} \\ \frac{n*(n+1)}{2} + 4 * r(n \ div \ 2) \end{cases}$$

$a = 4$

$b = 2$

$k = 2$

$a = b^k \Rightarrow$ orden de $r(n) = n^2 * log(n)$

2)

a)

```
fun tieneCima(a: array[1..n] of nat ) ret r : bool
  var prev, i: nat
  var hasCrec, hasDec: bool
  hasCrec:= false
  hasDec:= false
  prev:= a[1]
  i:= 2
  while (i <= n ∧ prev <= a[i]) do
    prev:= a[i]
    i:= i + 1
    if ¬ hasCrec then
      hasCrec:= true
    fi
  od
  while (i <= n ∧ prev >= a[i]) do
    prev:= a[i]
    i:= i + 1
    if ¬ hasDec then
      hasDec:= true
    fi
  od
  r:= (i = n) ∧ hasCrec ∧ hasDec
end fun
```

b)

```
fun devuelveCima(a: array[1..n] of nat ) ret r : nat
  var prev, i: nat
  prev:= a[1]
  i:= 2
  while (prev <= a[i]) do
    prev:= a[i]
    i:= i + 1
  od
  r:= i - 1
end fun
```

c)

fun binary_search_rec(a: array[1..n] of T, lft, rgt: nat) **ret** i : nat
  var mid: **nat**
  if lft > rgt **then** i:= 0
    lft ≤ rgt **then**
      mid:= (lft+rgt) / 2
      if a[mid-1] ≥ a[mid] ≥ a[mid+1] **then** i:= binary_search_rec(a, lft, mid-1)
      a[mid-1] ≤ a[mid] ≥ a[mid+1] **then** i:= mid
      if a[mid-1] ≤ a[mid] ≤ a[mid+1] **then** i:= binary_search_rec(a, mid+1, rgt)
      fi
  fi
end fun
fun devuelveCima2(a: array[1..n] of T, x: T) **ret** i : nat
  i:= binary_search_rec(a, 2, n-1)
end fun

3)

Sea p el largo del framento del arreglo que toma la funcion:

$$r(p) = \begin{cases} 1 & p = 1 \\ 1 + r(p \textbf{ div } 2) + r(p \textbf{ div } 2) & p \geq 1 \end{cases}$$

$$\vdots$$

$$r(p) = \begin{cases} 1 & p = 1 \\ 1 + 2 * r(p \textbf{ div } 2) & p \geq 1 \end{cases}$$

Por lo cual, $a = 2, b = 2, k = 0 \Rightarrow a > b^k$

Por ende, el algoritmo tiene una complejidad de $n^{log_b a} = n^{log_2 2} = n^1 = n$

4)

a)

$n \log 2^n = n^2 * \log 2 \approx n^2$

$2^n \sqsubset 2^n \log n \sqsubset n \log 2^n \sqsubset n! \log n$

b)

$\log(n^{n^4}) = n^4 * \log n$

$2^{4 * \log n} = 2^4 * 2^{\log n} \approx 2^{\log n}$

$2^{4 \log n} \sqsubset 4^n \sqsubset n^3 \log n \sqsubset n^4 + 2 \log n \sqsubset \log(n^{n^4})$

c)

$\log(n^n) = n \log n \Rightarrow \log(n^n) \approx n \log n$

$n \log n \sqsubset \log n!$

5)

$$\begin{aligned} r(n) &= \sum_{i=1}^{K} r(n \textbf{ div } L) + \sum_{i=1}^{n^4} 1 \\ &= K * r(n \textbf{ div } L) + n^4 \end{aligned}$$

Por lo cual nos queda que:

$$a = K, \quad b = L, \quad k = 4$$

a)

Para que el orden sea de $n^4 \log n$ se tiene que dar que $a = b^k \Rightarrow K = L^k$

Por ende elegimos $K = 16, L = 2 \Rightarrow 16 = 2^4 \Rightarrow 16 = 16$

b)

Para que el orden sea de n^4 se tiene que dar que $a > b^k \Rightarrow K = L^k$

Por ende elegimos $K = 2, L = 2 \Rightarrow 2 < 2^4 \Rightarrow 2 < 16$

Por ende nos queda que el orden es: $n^k = n^4$

c)

Elegimos $K = 32, L = 2 \Rightarrow 17 > 2^4 \Rightarrow 32 > 16$

Por ende nos queda que el orden es: $n^{log_2 32} = n^5$

6)

a)

```
for i := 1 to n do
  for j := 1 to n do
    m:= 1
  od
od
for i := 1 to 2 do
  k:= n
  while k != 0 do
    k:= k div 2
    m:= 1
  od
od
```

b)

```
fun DyV(n) ret r : nat
  for i := 1 to n do
    for j := 1 to n do
      m:= 1
    od
  od
  for i := 1 to 4 do
    r:= DyV(n div 2)
  od
end fun
```

c)

```
fun rec(n) ret r : type
  if n <= 0 then
    r:= n
  else
    for i := 1 to n do
      r:= rec(n-1) + rec(n-1) + rec(n-1)
    od
  fi
end fun
```