PRACTICO 3

ALGORITMOS Y ESTRUCTURAS DE DATOS III

1) a) a: ¿CUANTAS VECES LLAMO AL ALGORITMO PRINCIPAL? c: ¿CUANTOS LLAMADOS RECURSIVOS HAY?

$a = 8$, $b = 2$, $k = 3$

b: ¿QUE ENACCION DEL ORIGINAL HACE?

$$T(m) = \begin{cases} 0 \rightarrow m \leq 1 \\ 8 * T\left(\frac{m}{2}\right) + m^3 \quad m > 1 \end{cases}$$

$a = 8, b = 2, k = 3, b^k = 2^3 = 8$

COMO $a = b^k \Rightarrow m^3 \cdot \log m$ ES LA COMPLEJIDAD

b) $a = 4$, $b = 2$, $k = 2$,

$$T(m) = \begin{cases} 0 \rightarrow m = 0 \\ 4 * T\left(\frac{m}{2}\right) + m^2 \rightarrow m > 0 \end{cases}$$

$a = 4, b = 2, k = 2, b^k = 2^2 = 4$

COMO $a = b^k \Rightarrow m^2 \log m$

2) FUNC tiene_cima (a: array[1..m] of int) ret res: bool

```
res := True
Var j: nat
j := 1
While j < m ∧ a[j] ≤ a[j+1] do
    j := j+1
od

While j < m do

    If a[j] ≤ a[j+1]
        res := False
    Fi
od
end Func
```

b) (ASUMIENDO QUE TIENE CIMA)...

```
Func cima(a: array[1..m] of int) ret res: nat
Var j: nat
Bool no_encontrado := true
j := 2
While j < m ∧ no_encontrado do
    If a[j] ≥ a[j-1] ∧ a[j] ≥ a[j+1]:
        res := a[j]
        no_encontrado := False
    Fi
    j := j+1
od
end Func
```

- Tengo que devolver el elemento que hace cima o la pos del elemento
- LA COMPLEJIDAD ES m.

④ FUNC encontrar_cimaB(a: array[1..m] of int, lft, rgt: nat) ret res: int

Var mid: nat

If lft > rgt → res := 0
[] lft ≤ rgt → mid := (lft + rgt) / 2

If a[mid] ≥ a[mid-1] ∧ a[mid] ≥ a[mid+1] → res := mid
[] a[mid] ≥ a[mid-1] ∧ a[mid] ≤ a[mid+1] → res := encontrar_cimaB(a, mid+1, rgt)
[] a[mid] ≤ a[mid-1] ∧ a[mid] ≥ a[mid+1] → res := encontrar_cimaB(a, lft, rgt-1)

Fi

Fi

End FUNC

FUN encontrar_cima (a: array[1..m] of int) ret result: int

result := encontrar_cimaB(a, 1, m)

end FUN

COMO NUESTRO ALGORITMO ES RECURSIVO ANALIZAMOS LA COMPLEJIDAD:

$$T(m) = \begin{cases} 1 & (lft > rgt) \\ a * T\left(\frac{m}{b}\right) + g(m) & (lft \leq rgt) \end{cases}$$

OBSERVAMOS QUE a = 1 (LLAMADAS RECURSIVAS)
b = 2 (DIVIDIMOS EN DOS)
$g(m) = m^0 = 1$

$$T(m) = \begin{cases} 1 & (lft > rgt) \\ 1 * T\left(\frac{m}{2}\right) + 1 & (lft \leq rgt) \end{cases}$$

VEMOS QUE $a = 1 \equiv b^k = b^0 = 1$
⇒ LA COMPLEJIDAD ES:
$m^k \log m = m^0 \log m = 1 \log m = \log m$

③ COMO ES UN ALGORITMO RECURSIVO ANALIZAMOS LA COMPLEJIDAD ASÍ.

$$T(m) = \begin{cases} 1 & (i = k) \\ a * T\left(\frac{m}{b}\right) + g(m) & (i \neq k) \end{cases}$$

OBSERVAMOS QUE a = 2 (2 LLAMADAS RECURSIVAS)
b = 2 (DIVIDIMOS EN 2)
$g(m) = m^0 = 1$

$$T(m) = \begin{cases} 1 & (i = k) \\ 2 * T\left(\frac{m}{2}\right) + 1 & (i \neq k) \end{cases}$$

$b^k = 2^0 = 1 \Rightarrow a = 2 > 1$
LA COMPLEJIDAD ES:
$m^{\log_b a} = m^{\log_2 2} = m^1 = m$

(5) $m^4 \log m$

a)

$$T(m) = \begin{cases} 0 & m \leq 1 \\ a * T\left(\frac{m}{b}\right) + g(m) \end{cases}$$

- CADA VEZ QUE LLAMO A LA FUNCIÓN S VOY A HACER m^4 OSEA $\Rightarrow g(m)$ ES m^4
- COMO MI COMPLEJIDAD TIENE QUE DARME $m^4 \log m$ $a = b^k \Rightarrow a = b^4$
- LE ASIGNO A b EL VALOR DE 2 $\Rightarrow a = 2^4 = 16$
- POR LO TANTO $a = 16$

UN VALOR POSIBLE PARA QUE LA COMPLEJIDAD DE S SEA $m^4 \log m$ ES $K=16$, $L=2$

b) m^4

- $a < b^k$
- $k = 4$

$$T(m) = \begin{cases} 0 & m \leq 1 \\ a * T\left(\frac{m}{b}\right) + g(m) \end{cases}$$

- SABEMOS QUE a ES CUANTAS VECES LLAMO A MI FUNCIÓN REC.
- MI B ES COMO "VOY DIVIDIENDO" EN CADA LLAMADA
- g(m) ES EL ORDEN DE LO QUE HAGO EN CADA LLAMADA
- SI PUSO $a = 2$ Y $b = 4$ ME QUEDA:

COMO $a = 2 < b = 4 \Rightarrow$ LA COMPLEJIDAD ES m^4

$$T(m) = \begin{cases} 0 & m \leq 1 \\ 2 * T\left(\frac{m}{4}\right) + m^4 \end{cases}$$

ASIGNACIONES DE VALORES FINALES → $K=2$, $L=4$

c) m^5

- $a < b^k$
- $k = 5$

[2 [3] [4] 5, 6] 7

(6) a) $m^2 + 2 \log m$

// ALGORITMO CON COMPLEJIDAD $\log m$

```
Func binary_search (a:array of int, lft, rgt, x) ret int: res
  Var res: int
  Var mid: int
  mid := (lft + rgt)/2
  if(a[mid] < x) → binary_search(a, mid+1, rgt, x)
    a[mid] = x → res := mid
    a[mid] > x → binary_search(a, lft, mid-1, x)
  Fi
endfun
```

ANALISIS DE COMPLEJIDAD:

$$T(m) = \begin{cases} 1 \rightarrow a[mid] = x \\ a * T\left(\frac{m}{b}\right) + g(m) \quad a[mid] \neq x \end{cases} \equiv T(m) = \begin{cases} 1 \text{ si } a[mid] = x \\ 1 * T\left(\frac{m}{k}\right) + 1 = m^0 \end{cases}$$

COMO $a = 1$, $b = 2$, $k = 0 \Rightarrow a = 1 = 2^0 = 1 \Rightarrow m^k \cdot \log m \Rightarrow m^0 \cdot \log m = \boxed{\log m}$

// ALGORITMO CON COMPLEJIDAD m^2

```
FUN sum_max (a: array [1..m] of int) ret max_suma: int.

    Var max_suma, sum_actual: int

    For i := 1 TO m do
      sum_actual := 0
      For j := 1 to m do
        sum_actual := sum_actual + a[j]
        If sum_actual > max_suma
          → max_suma := sum_actual
        fi
      od
    od
```

// ALGORITMO GENERAL CON COMPLEJIDAD $m^2 + 2\log m$

```
Proc algo1 (in a: array[1..n] of int, in lft, rgt, x: int)

    Var b, b2, m)

    b := binary_search (a, lft, rgt, x)
    m := sum_max (a)
    b2 := binary_search (a, lft, rgt, x)
End Proc
```

b) $m^2 \log m$ $n = b^k \wedge k \geq 2$

```
Proc f (in m: nat)        4 = 2^2 = 4 ≥ 2 ✓

    For i := 1 TO 4 do
        f(m/2)
    od
    For j := 1 TO m^2 do
      OP ∈ O(1)
    od
```