004 → 40

HOJA N° 1

FECHA

Tomás Achával Benito

Ejercicio 1:

PARTE ORDENADA = ■

a) PASO A PASO DE INSERTION SORT

A = [3, 17, 2, 10, 5, 8, 3] → [3, 17, 2, 10, 5, 8, 3]

i=1 EL SEGMENTO A[1..1] ESTÁ ORDENADO, NO HAGO CAMBIOS

i=2 COMO 17 > 3, EL SEGMENTO A[1..2) ESTÁ ORDENADO, NO HAGO CAMBIOS

→ [3, 17, 2, 10, 5, 8, 3] → [3, 2, 17, 10, 5, 8, 3]

i=3

COMO 2 < 17, EL PROCEDIMIENTO INSERT REALIZA UN SWAP ENTRE ELLOS. LUEGO COMO 2 < 3, EL PROCEDIMIENTO INSERT REALIZA OTRO SWAP Y FINALIZA.

→ [2, 3, 17, 10, 5, 8, 3] → [2, 3, 17, 10, 5, 8, 3]

i=4

COMO 10 < 17, EL PROC INSERT LOS INTERCAMBIA

→ [2, 3, 10, 17, 5, 8, 3] → [2, 3, 10, 17, 5, 8, 3]

i=5

COMO 5 < 17, INSERT LOS INTERCAMBIA

→ [2, 3, 10, 5, 17, 8, 3] → [2, 3, 5, 10, 17, 8, 3]

COMO 5 < 10, INSERT LOS INTERCAMBIA

→ [2, 3, 5, 10, 17, 8, 3] → [2, 3, 5, 10, 8, 17, 3] → [2, 3, 5, 8, 10, 17, 3]

i=6 COMO 8 < 17 y 8 < 10, INSERT LOS INTERCAMBIA EN ORDEN

→ [2, 3, 5, 8, 10, 17, 3] → [2, 3, 5, 8, 10, 3, 17] → [2, 3, 5, 8, 3, 10, 17]

i=7 COMO 3 < 17, 3 < 10, 3 < 8 y 3 < 5 ENTONCES EL ELEMENTO ES "SWAPPEADO" HACIA LA IZQUIERDA 4 VECES.

→ [2, 3, 5, 3, 8, 10, 17] → [2, 3, 3, 5, 8, 10, 17]

FIN (ARREGLO ORDENADO)

NOTA

Tomás Achával Berreto

10

HOJA Nº 3

FECHA

Ejercicio 3:

```
FUN f(n:nat) ret t:nat
IF n≤1 then t:=1
else
    T:= f(n/2)
    T:= T + f(n/2)
    For i:=1 to n do
        For j:=1 to n do
            T:= t+1
        OD
    OD
FI
END FUN
```

FUNCIÓN DE RECURRENCIA

$$T(n) = \begin{cases} 1 & \text{si } n \leq 1 \\ \underbrace{2}_{a} \cdot T(n/\underbrace{2}_{b}) + n^{\overbrace{2}^{k}} & \text{EN OTRO CASO} \end{cases}$$

$a = 2$
$b = 2$
$k = 2$

Como $a < b^k$ $(2 < 2^2)$ ENTONCES EL PROCEDIMIENTO TIENE ORDEN n^k, ES DECIR, n^2.

ESTE RESULTADO SALE DE QUE EL ORDEN DE LA FUNCIÓN DE RECURRENCIA ESTÁ DADO POR:

$$\begin{cases} n^{(\log_b(a))} & \text{si } a > b^k \\ n^k \log(n) & \text{si } a = b^k \\ n^k & \text{si } a < b^k \end{cases}$$

HOJA Nº 2

FECHA

omás Achaval Becerro

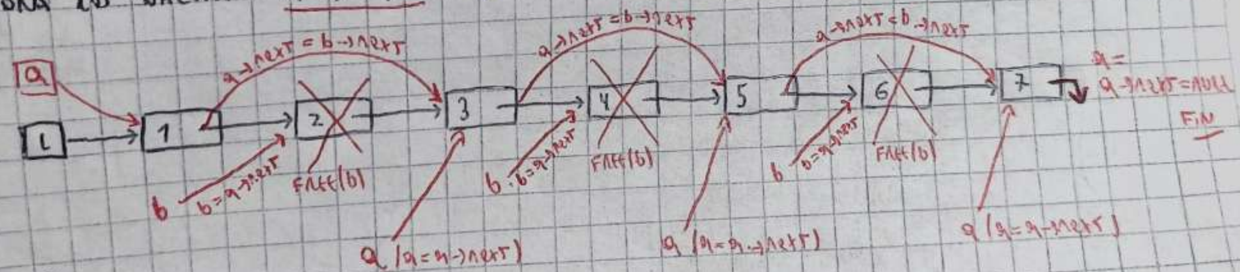
100%

Ejercicio 2:

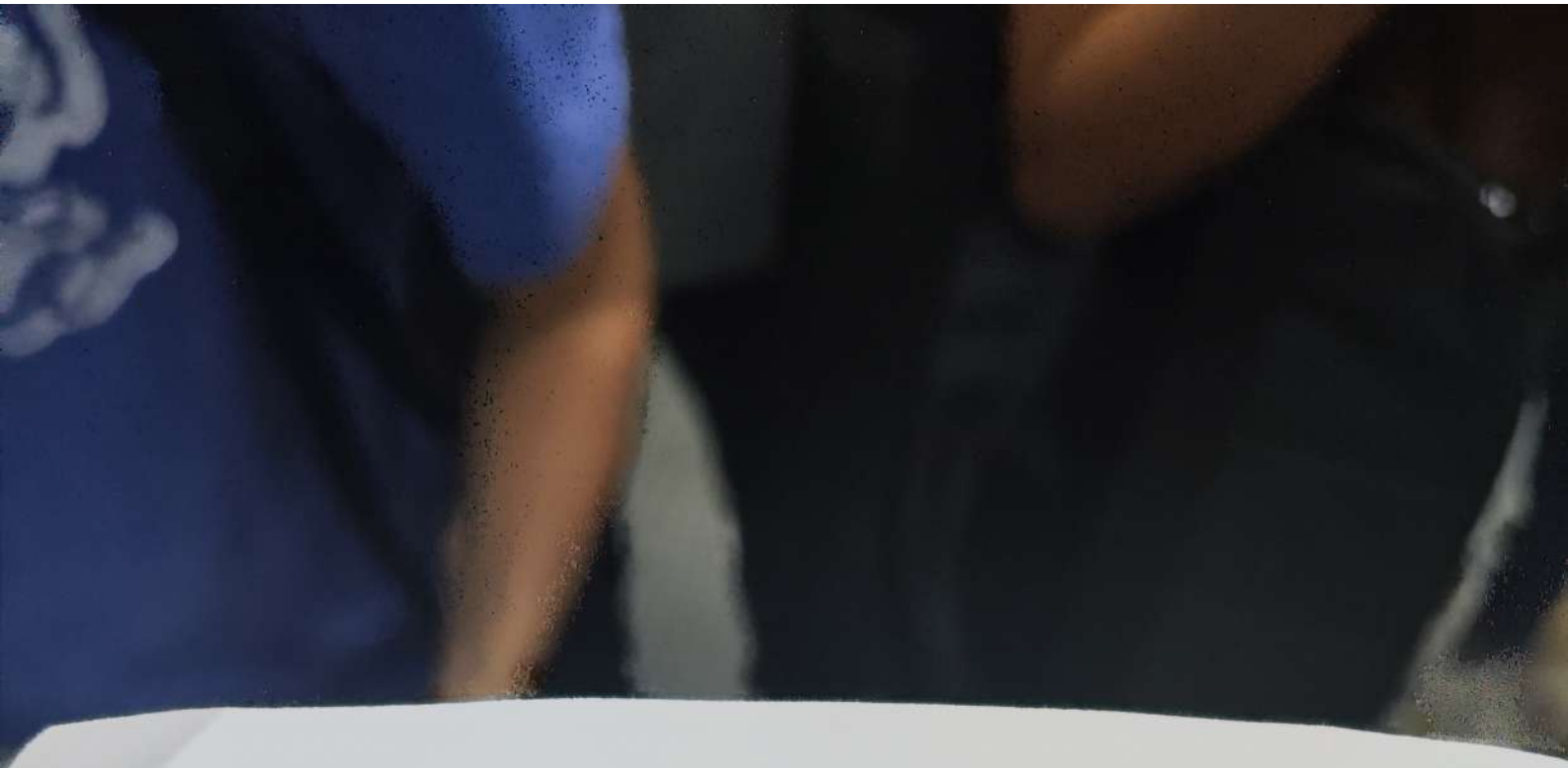
a) El procedimiento P elimina los nodos en posiciones impares de la lista (si el primer elemento está en la posición 0).

b) El orden del procedimiento P es n, donde n es la longitud de la lista. Esto es debido a que el ciclo while recorre una única vez a todos los elementos de la lista, y no hay recursión sobre P. Esto significa que la máxima cantidad de asignaciones no será más que un múltiplo de n, y por lo tanto el procedimiento es del orden de n. (Tampoco se utilizan funciones de orden ≥ n)

c) Si se llama a P con la lista 1, 2, 3, 4, 5, 6, 7, luego de su ejecución la lista tendrá los valores 1, 3, 5 y 7.



Explicación gráfica



b)

```
Proc P (in/out a: ARRAY [1..n] of Int, in e: Int, out K:nat)
   var lft:nat
   lft:=1
   For i:=1 to n do
       IF (a[i] < e) then
          SWAP(a, i, lft)
          lft:= lft+1
       FI
   OD
   K:= lft-1
END PROC
```

Tomás Achával Berzero

Ejercicio 4:

a) SPEC Multiset of T where

CONSTRUCTORS

FUN empty_multiset () ret m: Multiset of T
{ Crea un multiconjunto vacío }

Proc addelem (in/out m: Multiset of T, in e: T)
{ Agrega el elemento e al multiconjunto m }

OPERATIONS

FUN is_empty_multiset (m: Multiset of T) ret b: bool
{ Devuelve true sii el multiconjunto m está vacío }

FUN elem_exists (m: Multiset of T, e: T) ret b: bool
{ Devuelve true si el elemento e está en m }

FUN elem_amount (m: Multiset of T, e: T) ret a: nat
{ Devuelve la cantidad "a" de veces que e está en m }

Proc delete_one (in/out m: Multiset of T, in e: T)
{ Elimina una ocurrencia de e en m }

Proc destroyM (in/out m: Multiset of T)
{ Destruye el multiconjunto m eliminando todos sus elementos }

b) IMPLEMENT Multiset of T where

```
TYPE Multiset of T = List of (Sets of T)

TYPE Sets = tuple
    of T    element: T
            amount: nat
        end tuple

FUN empty_multiset() ret m: Multiset of T
    m := empty()
END FUN
```

```
Proc addelem (in/out m: Multiset of T, in e: T)
    var k: Sets of T
    k.elem := e
    if (elem_exists(m, e)) then
        while (head(m).element ≠ e) do
            addr(head(m))
            tail(m)
        od
        k.amount := head(m).amount + 1
        tail(m)
        addl(m, k)
    else
        k.amount := 1
        addl(m, k)
    fi
END PROC
```

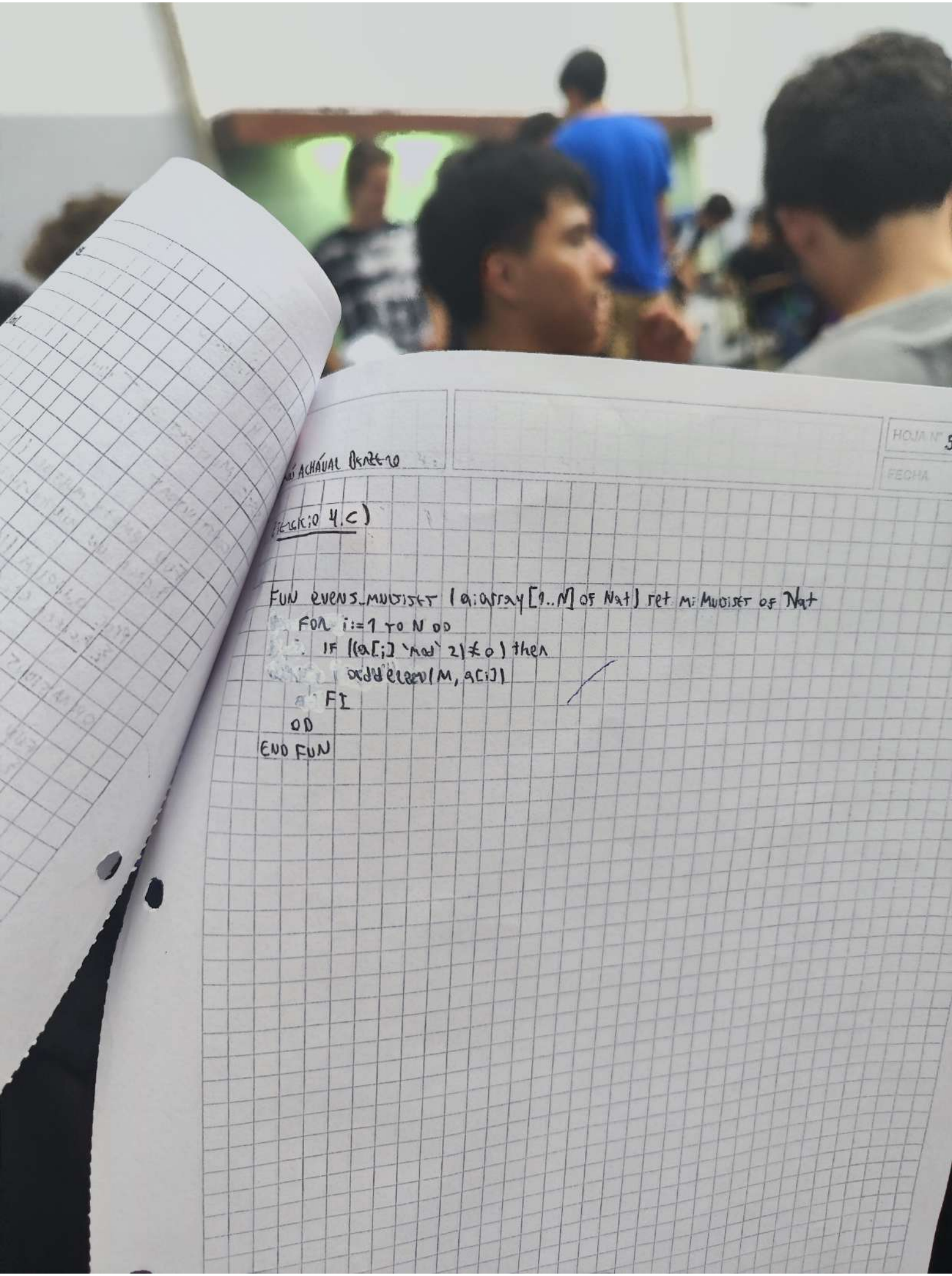
Además eliminás los elementos de la lista.

Deberías copiarla

```
IS_EMPTY_MULTISET (M: MULTISET of T) ret b: bool
    b := IS_EMPTY (M)
END FUN

FUN elem_exists (M: MULTISET OF T, e: T) ret b: bool
    b := FALSE
    var i: nat
    i := 0
    while (not b ∧ i < length(M)) do
        b := b ∨ (index(M, i).element = e)
        i := i+1
    OD
END FUN

FUN elem_AMOUNT (M: MULTISET of T, e: T) ret a: nat
    IF (not elem_exists(M, e)) then a := 0
    ELSE
        var i: nat
        i := 0
        while (index(M, i).element ≠ e) do
            i := i+1
        OD
        a := index(M, i).AMOUNT
    FI
END FUN

Proc delete_one (in/out M: MULTISET of T, in e: T)
    var K: nat
    K := elem_AMOUNT(M, e)
    IF K=0 THEN SKIP
    ELSE
        while (head(M).element ≠ e) do
            addr(head(M))
            TAIL(M)
        OD
        IF K=1 then TAIL(M)
            ELSE
                var aux: SETS of T
                aux.element := e
                aux.AMOUNT := head(M).AMOUNT - 1
                TAIL(M)
                ADDL(M, AUX)
        FI
    FI
END PROC

Proc destroyM(M: MULTISET of T)
    destroy(M)
END PROC
```

✓ falta el copy.

*NOTA: LAS FUNCIONES destroy, addl, addr, head, TAIL, etc son las funciones del TAD list of T.



Ejercicio 4.c)

```
FUN evens_multiset (a:array[1..N] of Nat) ret M: multiset of Nat
    FOR i:=1 to N DO
        IF ((a[i] 'mod' 2) ≠ 0) then
            addRepeated(M, a[i])
        FI
    OD
END FUN
```