METODOLOGÍA DE LA PROGRAMACIÓN
SOLUCIONES DEL TEMA 5
ESPECIFICACIÓN ECUACIONAL DE TAD

A) Operaciones sobre listas

Dar ecuaciones que definan las siguientes operaciones sobre los tipos de datos [Int] o [t] según el caso:

1) incr --

incr:: ([Int]) → [Int]

incr([]) = [] (#1)

incr(x:s) = (x + 1) : incr(s) (#2)

Ejemplo:

incr ([4, 8, 5]) =	
= incr(4:8:5:[]) =(#2)	x = 4, s = 8:5:[]
= (4 + 1) : incr(8:5:[]) =(#2)	x = 8, s = 5:[]
= 5 : (8 + 1) : incr(5:[]) =(#2)	x = 5, s = []
= 5: 9 : (5 + 1) : incr([]) =(#1)	
= 5 : 9 : 6: []	
= [5, 9, 6]	

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

2) sumar --

sumar:: ([Int]) → Int

sumar([]) = 0 (#1)

sumar(x:s) = x + sumar(s) (#2)

Ejemplo:

sumar ([4, 8, 5]) =	
= sumar(4:8:5:[]) =(#2)	x = 4, s = 8:5:[]
= 4 + sumar(8:5:[]) =(#2)	x = 8, s = 5:[]
= 4 + 8 + sumar(5:[]) =(#2)	x = 5, s = []
= 4 + 8 + 5 + sumar([]) =(#1)	
= 4 + 8 + 5 + 0	
= 17	

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

3) algun_par --

algun_par:: ([Int]) → Bool

algun_par([]) = False (#1)

algun_par(x:s)
| par(x) = True (#2)
| impar(x) = algun_par(s) (#3)

Ejemplo 1:
algun_par([7, 5, 9]) =
= algun_par(7:5:9:[]) =(#3) x = 7, s = 5:9:[]
= algun_par(5:9:[]) =(#3) x = 5, s = 9:[]
= algun_par(9:[]) =(#3) x = 9, s = []
= algun_par([]) =(#1)
= False

Ejemplo 2:
algun_par([7, 5, 8, 9, 12]) =
= algun_par(7:5:8:9:12:[]) =(#3) x = 7, s = 5:8:9:12:[]
= algun_par(5:8:9:12:[]) =(#3) x = 5, s = 8:9:12:[]
= algun_par(8:9:12:[]) =(#2) x = 8, s = 9:12:[]
= True

Ejemplo 3:
algun_par([4, 5, 8, 9, 12]) =
= algun_par(4:5:8:9:12:[]) =(#2) x = 4, s = 5:8:9:12:[]
= True

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

4) coinciden_par --

coinciden_par:: ([Int], [Int]) → Bool

coinciden_par([], r)
| not es_vacia(r) = error "Distinta longitud" (#1)
| otherwise = True (#2)

coinciden_par(x:s, r)
| longitud(x:s) ≠ longitud(r) = error "Distinta longitud" (#3)
| (par(x) && par(primero(r))) || (impar(x) && impar(primero(r)))
= coinciden_par(s, resto(r)) (#4)
| otherwise = False (#5)

Ejemplo 1:

coinciden_par([7, 4, 9], [5, 10, 1]) =
= coinciden_par(7:4:9:[], 5:10:1:[]) =(#4) x = 7, s = 4:9:[], primero(r) = 5, resto(r) = 10:1:[]
= coinciden_par(4:9:[], 10:1:[]) =(#4) x = 4, s = 9:[] primero(r) = 10, resto(r) = 1:[]
= coinciden_par(9:[], 1:[]) =(#4) x = 9, s = [] primero(r) = 1, resto(r) = []
= coinciden_par([], []) =(#2)
= True

Ejemplo 2:

coinciden_par([7, 4, 8, 9], [5, 10, 1, 15]) =
= coinciden_par(7:4:8:9:[], 5:10:1:15:[]) =(#4) x = 7, s = 4:8:9:[], primero(r) = 5, resto(r) = 10:1:15:[]
= coinciden_par(4:8:9:[], 10:1:15:[]) =(#4) x = 4, s = 8:9:[] primero(r) = 10, resto(r) = 1:15:[]
= coinciden_par(9:[], 1:[]) =(#5) x = 8, s = 9:[] primero(r) = 1, resto(r) = 15:[]
= False

Ejemplo 3:

= coinciden_par(7:4:8:9:[], 2:10:1:15:[]) =(#5) x = 7, s = 4:8:9:[], primero(r) = 2, resto(r) = 10:1:15:[]
= False

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

5) ultimo --

ultimo:: ([t]) → t

ultimo([]) = error "Lista vacía" (#1)

ultimo(x:s)
| es_vacia(s) = x (#2)
| otherwise = ultimo(s) (#3)

Ejemplo:

ultimo([7, 5, 9]) =
= ultimo(7:5:9:[]) =(#3) x = 7, s = 5:9:[]
= ultimo(5:9:[]) =(#3) x = 5, s = 9:[]
= ultimo(9:[]) =(#2) x = 9, s = []
= 9

6) sin_ultimo --

```
sin_ultimo:: ([t]) → [t]

sin_ultimo([]) = error "Lista vacía"                    (#1)

sin_ultimo(x:s)
    | es_vacia(s)        = []                           (#2)
    | otherwise          = x : sin_ultimo(s)            (#3)
```

Ejemplo:

```
sin_ultimo([7, 5, 9]) =
= sin_ultimo(7:5:9:[]) =(#3)          x = 7, s = 5:9:[]
= 7 : ultimo(5:9:[]) =(#3)            x = 5, s = 9:[]
= 7 : 5 : ultimo(9:[]) =(#2)          x = 9, s = []
= 7 : 5 : []
= [7, 5]
```

7) inversa --

Hacerlo de dos formas:

a) Utilizando ++

```
inversa:: ([t]) → [t]

inversa([]) = []                                        (#1)

inversa(x:s) = inversa(s) ++ (x:[])                     (#2)
```

Ejemplo:

```
inversa([2, 5, 9]) =
= inversa(2:5:9:[]) =(#2)             x = 2, s = 5:9:[]
= inversa(5:9:[]) ++ (2:[]) =(#2)     x = 5, s = 9:[]
= inversa(9:[]) ++ (5:[]) ++ (2:[]) =(#2)   x = 9, s = []
= inversa([]) ++ (9:[]) ++ (5:[]) ++ (2:[]) =(#1)
= [9, 5, 2]
```

b) Utilizando **ultimo**, **sin_ultimo**

```
inversa:: ([t]) → [t]

inversa([]) = []                                        (#1)

inversa(x:s) = ultimo(x:s) : inversa(sin_ultimo(x:s))   (#2)
```

Ejemplo:

```
inversa([2, 5, 9]) =
```

= inversa(2:5:9:[]) =(#2) x = 2, s = 5:9:[]
ultimo(2:5:9:[]) = 9, sin_ultimo(2:5:9:[]) = 2:5:[]
= 9 : inversa(2:5:[]) =(#2) x = 2, s = 5:[]
ultimo(2:5:[]) = 5, sin_ultimo(2:5:[]) = 2:[]
=9: 5: inversa(2:[]) =(#2) x = 2, s = []
ultimo(2:[]) = 2, sin_ultimo(2:[]) = []
= 9 : 5 : 2 : inversa([]) =(#1)
= 9 : 5 : 2 : []
= [9, 5, 2]

8) son_inv --

Hacerlo de dos formas:

a) Utilizando **inversa**

son_inv:: ([t], [t]) → Bool

son_inv(s, r)
| s == inversa (r) = True (#1)
| otherwise = False (#2)

Ejemplo 1:
son_inv([2, 5, 9], [9, 5, 2]) =
= son_inv(2:5:9:[], 9:5:2:[]) =(#1) s = 2:5:9:[], r = 9:5:2:[]
= True

Ejemplo 2:
son_inv([2, 5, 9], [9, 8, 2]) =
= son_inv(2:5:9:[], 9:5:2:[]) =(#2) s = 2:5:9:[], r = 9:8:2:[]
= False

En esta definición de la función son_inv, no hay recursividad y por ello los ejemplos se resuelven en un único paso.

b) Utilizando **ultimo**, **sin_ultimo**, **longitud** y **es_vacia**

son_inv:: ([t], [t]) → Bool

son_inv([], ℓ)
| not es_vacia(ℓ) = False (#1)
| otherwise = True (#2)

son_inv(x:s, ℓ)
| longitud(x:s) /= longitud(ℓ) = False (#3)
| x /= ultimo(ℓ) = False (#4)
| otherwise = son_inv(s, sin_ultimo(ℓ)) (#5)

Ejemplo 1:

son_inv([2, 5, 9], [9, 5, 2]) =
= son_inv(2:5:9:[], 9:5:2:[]) =(#5) x = 2, s = 5:9:[], ℓ = 9:5:2:[]
ultimo(ℓ) = 2, sin_ultimo(ℓ) = 9:5:[]
= son_inv(5:9:[], 9:5:[]) =(#5) x = 5, s = 9:[], ℓ = 9:5:[]
ultimo(ℓ) = 5, sin_ultimo(ℓ) = 9:[]
= son_inv(9:[], 9:[]) =(#5) x = 9, s = [], ℓ = 9:[]
ultimo(ℓ) = 9, sin_ultimo(ℓ) = []
= son_inv([], []) =(#2)
= True

Ejemplo 2:
son_inv([2, 5, 9], [9, 8, 2]) =
= son_inv(2:5:9:[], 9:8:2:[]) =(#5) x = 2, s = 5:9:[], ℓ = 9:8:2:[]
ultimo(ℓ) = 2, sin_ultimo(ℓ) = 9:8:[]
= son_inv(5:9:[], 9:8:[]) =(#4) x = 5, s = 9:[], ℓ = 9:8:[]
ultimo(ℓ) = 8, sin_ultimo(ℓ) = 9:[]
= False

En esta segunda definición de la función son_inv, sí hay recursividad y por tanto los ejemplos necesitan (en general) varios pasos.

9) nveces --

nveces:: (t, [t]) → Int

nveces(x, []) = 0 (#1)

nveces(x, y:s)
| x == y = 1 + nveces(x, s) (#2)
| otherwise = nveces(x, s) (#3)

Ejemplo:
nveces(8, [7, 8, 5, 9, 8]) =
= nveces(8, 7:8:5:9:8:[]) =(#3) x = 8, y = 7, s = 8:5:9:8:[]
= nveces(8, 8:5:9:8:[]) =(#2) x = 8, y = 8, s = 5:9:8:[]
= 1 + nveces(8, 5:9:8:[]) =(#3) x = 8, y = 5, s = 9:8:[]
= 1 + nveces(8, 9:8:[]) =(#3) x = 8, y = 9, s = 8:[]
= 1 + nveces(8, 8:[]) =(#2) x = 8, y = 8, s = []
= 1 + 1 + nveces(8, []) =(#1) x = 8
= 1 + 1 + 0 = 2

10) hay_rep --

hay_rep:: ([t]) → Bool

hay_rep([]) = False (#1)

hay_rep(x:s)
| esta(x, s) = True (#2)
| otherwise = hay_rep(s) (#3)

Ejemplo 1:
hay_rep([7, 5, 8, 9, 8]) =
= hay_rep(7:8:5:9:8:[]) =(#3) x = 7, s = 5:8:9:8:[]
= hay_rep(5:8:9:8:[]) =(#3) x = 5, s = 8:9:8:[]
= hay_rep(8:9:8:[]) =(#2) x = 8, s = 9:8:[]
= True

Ejemplo 2:
hay_rep([7, 5, 8, 9, 1]) =
= hay_rep(7:8:5:9:1:[]) =(#3) x = 7, s = 5:8:9:1:[]
= hay_rep(5:8:9:1:[]) =(#3) x = 5, s = 8:9:1:[]
= hay_rep(8:9:1:[]) =(#3) x = 8, s = 9:1:[]
= hay_rep(9:1:[]) =(#3) x = 9, s = 1:[]
= hay_rep(1:[]) =(#3) x = 1, s = []
= hay_rep([]) =(#1)
= False

11) eliminar --

eliminar:: (t, [t]) → [t]

eliminar(x, []) = [] (#1)

eliminar(x, y:s)
| x == y = eliminar(x, s) (#2)
| otherwise = y:eliminar(x, s) (#3)

Ejemplo:
eliminar(8, [7, 5, 8, 9, 8]) =
= eliminar(8, 7:5:8:9:8:[]) =(#3) x = 8, y = 7, s = 5:8:9:8:[]
= 7:eliminar(8, 5:8:9:8:[]) =(#3) x = 8, y = 5, s = 8:9:8:[]
= 7:5:eliminar(8, 8:9:8:[]) =(#2) x = 8, y = 8, s = 9:8:[]
= 7:5:eliminar(8, 9:8:[]) =(#3) x = 8, y = 9, s = 8:[]
= 7:5:9:eliminar(8, 8:[]) =(#2) x = 8, y = 8, s = []
= 7:5:9:eliminar(8, []) =(#1) x = 8
= 7:5:9:[] = [7, 5, 9]

12) eliminar_pares --

```
eliminar_pares:: ([Int]) → [Int]

eliminar_pares([]) = []                                          (#1)

eliminar_pares(x:s)
    | par(x)          = eliminar_pares(s)                        (#2)
    | impar(x)        = x:eliminar_pares(s)                      (#3)
```

Ejemplo:

```
eliminar_pares([7, 5, 8, 9, 12]) =
= eliminar_pares(7:5:8:9:12:[]) =(#3)        x = 7, s = 5:8:9:12:[]
= 7:eliminar_pares(5:8:9:12:[]) =(#3)        x = 5, s = 8:9:12:[]
= 7:5:eliminar_pares(8:9:12:[]) =(#2)        x = 8, s = 9:12:[]
= 7:5:eliminar_pares(9:12:[]) =(#3)          x = 9, s = 12:[]
= 7:5:9:eliminar_pares(12:[]) =(#2)          x = 12, s = []
= 7:5:9:eliminar_pares([]) =(#1)
= 7:5:9:[] = [7, 5, 9]
```

13) eliminar_pos_pares --

```
eliminar_pos_pares:: ([t]) → [t]

eliminar_pos_pares([]) = []                                      (#1)

eliminar_pos_pares(x:s)
    | es_vacia(s)     = x:s                                      (#2)
    | otherwise       = x:eliminar_pos_pares(resto(s))           (#3)
```

Ejemplo 1:

```
eliminar_pos_pares([7, 5, 8, 9, 12]) =
= eliminar_pos_pares(7:5:8:9:12:[]) =(#3)
                                  x = 7, s = 5:8:9:12:[], resto(s) = 8:9:12:[]
= 7:eliminar_pos_pares(8:9:12:[]) =(#3)
                                  x = 8, s = 9:12:[], resto(s) = 12:[]
= 7:8:eliminar_pos_pares(12:[]) =(#2)        x = 8, s = []
= 7:8:12:[] = [7, 8, 12]
```

Ejemplo 2:

```
eliminar_pos_pares([7, 5, 8, 9]) =
= eliminar_pos_pares(7:5:8:9:[]) =(#3)
                                  x = 7, s = 5:8:9:[], resto(s) = 8:9:[]
= 7:eliminar_pos_pares(8:9:[]) =(#3)
                                  x = 8, s = 9:[], resto(s) = []
= 7:8:eliminar_pos_pares([]) =(#1)
= 7:8:[] = [7, 8]
```

14) eliminar_pos_impares --

eliminar_pos_impares:: ([t]) → [t]

eliminar_pos_impares([]) = [] (#1)

eliminar_pos_impares(x:s)
| es_vacia(s) = [] (#2)
| otherwise = primero(s):eliminar_pos_impares(resto(s)) (#3)

Ejemplo 1:
eliminar_pos_impares([7, 5, 8, 9, 12]) =
= eliminar_pos_impares(7:5:8:9:12:[]) =(#3)
x = 7, s = 5:8:9:12:[], primero(s) = 5, resto(s) = 8:9:12:[]
= 5:eliminar_pos_impares(8:9:12:[]) =(#3)
x = 8, s = 9:12:[], primero(s) = 9, resto(s) = 12:[]
= 5:9:eliminar_pos_impares(12:[]) =(#2) x = 8, s = []
= 5:9:[] = [5, 9]

Ejemplo 2:
eliminar_pos_impares([7, 5, 8, 9]) =
= eliminar_pos_impares(7:5:8:9:[]) =(#3)
x = 7, s = 5:8:9:[], primero(s) = 5, resto(s) = 8:9:[]
= 5:eliminar_pos_impares(8:9:[]) =(#3)
x = 8, s = 9:[], primero(s) = 9, resto(s) = []
= 5:9:eliminar_pos_impares([]) =(#1)
= 5:9:[] = [5, 9]

15) eliminar_rep --

eliminar_rep:: ([t]) → [t]

eliminar_rep([]) = [] (#1)

eliminar_rep(x:s)
| esta(x, s) = x:eliminar_rep(eliminar(x, s)) (#2)
| otherwise = x:eliminar_rep(s) (#3)

Ejemplo:
eliminar_rep([7, 5, 8, 5, 2, 8, 8]) =
= eliminar_rep(7:5:8:5:2:8:8:[]) =(#3) x = 7, s = 5:8:5:2:8:8:[]
= 7:eliminar_rep(5:8:5:2:8:8:[]) =(#2) x = 5, s = 8:5:2:8:8:[]
= 7:5:eliminar_rep(8:2:8:8:[]) =(#2) x = 8, s = 2:8:8:[]
= 7:5:8:eliminar_rep(2:[]) =(#3) x = 2, s = []
= 7:5:8:2:eliminar_rep([])=(#1)
= 7:5:8:2:[] = [7, 5, 8, 2]

16) eliminar_rep2 --

```
eliminar_rep2:: ([t]) → [t]

eliminar_rep2([]) = []                                          (#1)

eliminar_rep2(x:s)
    | esta(x, s)        = eliminar_rep2(s)                      (#2)
    | otherwise         = x:eliminar_rep2(s)                    (#3)
```

Ejemplo:

```
eliminar_rep2([7, 5, 8, 5, 2, 8, 8]) =
= eliminar_rep2(7:5:8:5:2:8:8:[]) =(#3)          x = 7, s = 5:8:5:2:8:8:[]
= 7:eliminar_rep2(5:8:5:2:8:8:[]) =(#2)          x = 5, s = 8:5:2:8:8:[]
= 7:eliminar_rep2(8:5:2:8:8:[]) =(#2)            x = 8, s = 5:2:8:8:[]
= 7:eliminar_rep2(5:2:8:8:[]) =(#3)              x = 5, s = 2:8:8:[]
= 7:5:eliminar_rep2(2:8:8:[])=(#3)                     x = 2, s = 8:8:[]
= 7:5:2:eliminar_rep2(8:8:[]) =(#2)              x = 8, s = 8:[]
= 7:5:2:eliminar_rep2(8:[])=(#3)                 x = 8, s = []
= 7:5:2:8:eliminar_rep2([])=(#1)
= 7:5:2:8:[] = [7, 5, 2, 8]
```

17) par_igual --

```
par_igual:: ([t]) → Bool

par_igual([]) = False                                           (#1)

par_igual(x:s)
    | es_vacia(s)           = False                             (#2)
    | x == primero(s)       = True                              (#3)
    | otherwise             = par_igual(s)                      (#4)
```

Ejemplo 1:

```
par_igual ([7, 5, 8, 8, 6, 5]) =
= par_igual(7:5:8:8:6:5:[]) =(#4)      x = 7, s = 5:8:8:6:5:[], primero(s) = 5
= par_igual(5:8:8:6:5:[]) =(#4)          x = 5, s = 8:8:6:5:[], primero(s) = 8
= par_igual(8:8:6:5:[]) =(#3)            x = 8, s = 8:6:5:[], primero(s) = 8
= True
```

Ejemplo 2:

```
par_igual ([7, 5, 8, 5]) =
= par_igual(7:5:8:5:[]) =(#4)            x = 7, s = 5:8:5:[], primero(s) = 5
= par_igual(5:8:5:[]) =(#4)              x = 5, s = 8:5:[], primero(s) = 8
= par_igual(8:5:[]) =(#4)                x = 8, s = 5:[], primero(s) = 5
= par_igual(5:[]) =(#2)                  x = 5, s = []
= False
```

18) es_prefijo --

es_prefijo:: ([t], [t]) → Bool

es_prefijo([], r) = True (#1)

es_prefijo(x:s, r)
| es_vacia(r) = False (#2)
| x / = primero(r) = False (#3)
| otherwise = es_prefijo(s, resto(r)) (#4)

Ejemplo 1:
es_prefijo ([1, 2, 3], [1, 2, 5, 6]) =
= es_prefijo(1:2:3:[], 1:2:5:6:[]) =(#4)
x = 1, s = 2:3:[], r = 1:2:5:6:[], primero(r) = 1, resto(r) = 2:5:6:[]
= es_prefijo(2:3:[], 2:5:6:[]) =(#4)
x = 2, s = 3:[], r = 2:5:6:[], primero(r) = 2, resto(r) = 5:6:[]
= es_prefijo(3:[], 5:6:[]) =(#3)
x = 3, s = [], r = 5:6:[], primero(r) = 5, resto(r) = 6:[]
= False

Ejemplo 2:
es_prefijo ([1, 2, 3], [1, 2, 3, 5, 6]) =
= es_prefijo(1:2:3:[], 1:2:3:5:6:[]) =(#4)
x = 1, s = 2:3:[], r = 1:2:3:5:6:[], primero(r) = 1, resto(r) = 2:3:5:6:[]
= es_prefijo(2:3:[], 2:3:5:6:[]) =(#4)
x = 2, s = 3:[], r = 2:3:5:6:[], primero(r) = 2, resto(r) = 3:5:6:[]
= es_prefijo(3:[], 3:5:6:[]) =(#4)
x = 3, s = [], r = 3:5:6:[], primero(r) = 3, resto(r) = 5:6:[]
= es_prefijo([], 5:6:[]) =(#1) r = 5:6:[]

= True

Ejemplo 3:
es_prefijo ([1, 2, 3], [1, 2]) =
= es_prefijo(1:2:3:[], 1:2:[]) =(#4)
x = 1, s = 2:3:[], r = 1:2:[], primero(r) = 1, resto(r) = 2:[]
= es_prefijo(2:3:[], 2:[]) =(#4)
x = 2, s = 3:[], r = 2:[], primero(r) = 2, resto(r) = []
= es_prefijo(3:[], []) =(#2) x = 3, s = [], r = []
= True

19) es_sublista --

es_sublista:: ([t], [t]) → Bool

es_sublista([], r) = True (#1)

es_sublista(x:s, r)
| es_vacia(r) = False (#2)
| es_prefijo(x:s, r) = True (#3)
| otherwise = es_sublista(x:s, resto(r)) (#4)

Ejemplo 1:
es_sublista ([4, 5], [1, 2, 5, 6]) =
= es_sublista(4:5:[], 1:2:5:6:[]) =(#4)
x = 4, s = 5:[], r = 1:2:5:6:[], resto(r) = 2:5:6:[]
= es_sublista(4:5:[], 2:5:6:[]) =(#4)
x = 4, s = 5:[], r = 2:5:6:[], resto(r) = 5:6:[]
= es_sublista(4:5:[], 5:6:[]) =(#4)
x = 4, s = 5:[], r = 5:6:[], resto(r) = 6:[]
= es_sublista(4:5:[], 6:[]) =(#4)
x = 4, s = 5:[], r = 6:[], resto(r) = []
= es_sublista(4:5:[], []) =(#2)
x = 4, s = 5:[], r = []
= False

Ejemplo 2:
es_sublista ([4, 5], [1, 2, 4, 5, 6]) =
= es_sublista(4:5:[], 1:2:4:5:6:[]) =(#4)
x = 4, s = 5:[], r = 1:2:4:5:6:[], resto(r) = 2:4:5:6:[]
= es_sublista(4:5:[], 2:4:5:6:[]) =(#4)
x = 4, s = 5:[], r = 2:4:5:6:[], resto(r) = 4:5:6:[]
= es_sublista(4:5:[], 4:5:6:[]) =(#3)
x = 4, s = 5:[], r = 4:5:6:[]
= True

20) elem_pos --

```
elem_pos:: (Int, [t]) → t

elem_pos(pos, []) = error "No es adecuado"                          (#1)

elem_pos(pos, x:s)
    | pos <= 0 || pos > longitud(x:s)    = error "No es adecuado"   (#2)
    | pos == 1                           = x                        (#3)
    | otherwise                          = elem_pos(pos – 1, s)     (#4)
```

Ejemplo:

```
elem_pos(3, [5, 9, 8, 6, 2]) =
= elem_pos(3, 5:9:8:6:2:[]) =(#4)      pos = 3, x = 5, s = 9:8:6:2:[]
= elem_pos(2, 9:8:6:2:[]) =(#4)        pos = 2, x = 9, s = 8:6:2:[]
= elem_pos(1, 8:6:2:[]) =(#3)          pos = 1, x = 8, s = 6:2:[]
= 8
```

21) insertar --

```
insertar:: (Int, t, [t]) → [t]

insertar(pos, x, [])
    | pos /= 1                   = error "No es adecuado"          (#1)
    | pos == 1                   = x:[]                            (#2)

insertar(pos, x, y:s)
    | pos <= 0 || pos > (longitud(x:s) + 1)= error "No es adecuado"  (#3)
    | pos == 1                   = x:y:s                           (#4)
    | otherwise                  = y: insertar(pos – 1, x, s)      (#5)
```

Ejemplo:

```
insertar(3, 7, [5, 9, 8, 6, 2]) =
= insertar(3, 7, 5:9:8:6:2:[]) =(#5)     pos = 3, x = 7, y = 5, s = 9:8:6:2:[]
= 5: insertar(2, 7, 9:8:6:2:[]) =(#5)    pos = 2, x = 7, y = 9, s = 8:6:2:[]
= 5:9: insertar(1, 7, 8:6:2:[]) =(#4)    pos = 1, x = 7, y = 8, s = 6:2:[]
= 5:9:7:8:6:2:[]
```

22) pos_primer_par --

```
pos_primer_par:: (Int, [t]) → t

pos_primer_par([]) = 1                                              (#1)

pos_primer_par(x:s)
    | x `mod` 2 == 0             = 1                               (#2)
    | otherwise                  = 1 + pos_primer_par(s)           (#3)
```

Ejemplo 1:

```
pos_primer_par([5, 9, 8, 6]) =
= pos_primer_par(5:9:8:6:[]) =(#3)          x = 5, s = 9:8:6:[]
= 1 + pos_primer_par(9:8:6:[]) =(#3)        x = 9, s = 8:6:[]
= 1 + 1 + pos_primer_par(8:6:[]) =(#2)      x = 8, s = 6:[]
= 1 + 1 + 1 = 3
```

Ejemplo 2:

```
pos_primer_par([5, 9]) =
= pos_primer_par(5:9:[]) =(#3)              x = 5, s = 9:[]
= 1 + pos_primer_par(9:[]) =(#3)            x = 9, s = []
= 1 + 1 + pos_primer_par([]) =(#1)
= 1 + 1 + 1 = 3
```

23) pos_ult_apar --

```
pos_ult_apar:: (Int, [Int]) → Int

pos_ult_apar(x, []) = 0                                          (#1)

pos_ult_apar(x, y:s)
    | x /= y && not esta(x, s)      = 0                          (#2)
    | otherwise                     = 1 + pos_ult_apar(x, s)     (#3)
```

Ejemplo 1:

```
pos_ult_apar(8, [8, 6, 8, 5]) =
= pos_ult_apar(8, 8:6:8:5:[]) =(#3)         x = 8, y = 8, s = 6:8:5:[]
= 1 + pos_ult_apar(8, 6:8:5:[]) =(#3)       x = 8, y = 6, s = 8:5:[]
= 1 + 1 + pos_ult_apar(8, 8:5:[]) =(#3)     x = 8, y = 8, s = 5:[]
= 1 + 1 + 1 + pos_ult_apar(8, []) =(#1)     x = 8
= 1 + 1 + 1 + 0 = 3
```

Ejemplo 2:

```
pos_ult_apar(8, [3, 2]) =
= pos_ult_apar(8, 3:2:[]) =(#2)             x = 8, y = 3, s = 2:[]
= 0
```

Otra opción para definir la función:

```
pos_ult_apar:: (Int, [Int]) → Int

pos_ult_apar(x, []) = 0                                          (#1)

pos_ult_apar(x, y:s)
    | x == y                = 1 + pos_ult_apar(x, s)             (#2)
    | esta(x, s)            = 1 + pos_ult_apar(x, s)             (#3)
    | otherwise             = 0                                  (#4)
```

Ejemplo 1:

pos_ult_apar(8, [8, 6, 8, 5]) =	
= pos_ult_apar(8, 8:6:8:5:[]) =(#2)	x = 8, y = 8, s = 6:8:5:[]
= 1 + pos_ult_apar(8, 6:8:5:[]) =(#3)	x = 8, y = 6, s = 8:5:[]
= 1 + 1 + pos_ult_apar(8, 8:5:[]) =(#2)	x = 8, y = 8, s = 5:[]
= 1 + 1 + 1 + pos_ult_apar(8, []) =(#1)	x = 8
= 1 + 1 + 1 + 0 = 3	

Ejemplo 2:

pos_ult_apar(8, [3, 2]) =	
= pos_ult_apar(8, 3:2:[]) =(#4)	x = 8, y = 3, s = 2:[]
= 0	

24) maximo --

```
maximo:: ([Int]) → Int

maximo([]) = error "Lista vacía"                                    (#1)

maximo(x:s)
    | es_vacia(s)          = x                                      (#2)
    | x >= primero(s)      = maximo(x:resto(s))                     (#3)
    | otherwise            = maximo(s)                              (#4)
```

Ejemplo:

maximo([5, 3, 8, 7, 8]) =	
= maximo(5:3:8:7:8:[]) =(#3)	x = 5, s = 3:8:7:8:[], primero(s) = 3, resto(s) = 8:7:8:[]
= maximo(5:8:7:8:[]) =(#4)	x = 5, s = 8:7:8:[], primero(s) = 8, resto(s) = 7:8:[]
= maximo(8:7:8:[]) =(#3)	x = 8, s = 7:8:[], primero(s) = 7, resto(s) = 8:[]
= maximo(8:8:[]) =(#3)	x = 8, s = 8:[], primero(s) = 8, resto(s) = []
= maximo(8:[]) =(#2)	x = 8
= 8	

25) unir --

a) Sin utilizar funciones definidas con anterioridad

unir:: ([t], [t]) → [t]

unir([], r) = r (#1)

unir(x:s, r) = unir(s, x: r) (#2)

Ejemplo:
unir([5, 3, 8], [7, 9]) =
= unir(5:3:8:[], 7:9:[]) =(#2) x = 5, s = 3:8:[], r = 7:9:[]
= unir(3:8:[], 5:7:9:[]) =(#2) x = 3, s = 8:[], r = 5:7:9:[]
= unir(8:[], 3:5:7:9:[]) =(#2) x = 8, s = [], r = 3:5:7:9:[]
= unir([], 8:3:5:7:9:[]) =(#1) r = 8:3:5:7:9:[]
= 8:3:5:7:9:[] = [8,3,5,7,9]

b) Utilizando las operaciones ya definidas **inversa** y ++

unir:: ([t], [t]) → [t]

unir(r1, r2) = inversa(r1) ++ r2 (#1)

Esta definición de la función *unir* no es recursiva.

Ejemplo:
unir([5, 3, 8], [7, 9]) =
= unir(5:3:8:[], 7:9:[]) =(#1) r1 = 5:3:8:[], r2 = 7:9:[]
= 8:3:5:[] ++ 7:9:[]
= 8:3:5:7:9:[] = [8,3,5,7,9]

Otra opción:

unir:: ([t], [t]) → [t]

unir([], r) = r (#1)

unir(x:s, r) = inversa(s) ++ (x:r) (#2)

Ejemplo:
unir([5, 3, 8], [7, 9]) =
= unir(5:3:8:[], 7:9:[]) =(#2) x = 5, s = 3:8:[], r = 7:9:[]
= 8:3:[] ++ 5:7:9:[]
= 8:3:5:7:9:[] = [8,3,5,7,9]

26) quitar --

quitar:: (Int, [t]) → [t]

```
quitar(cuantos, [])
    | cuantos /= 0                          = error "No adecuado"        (#1)
    | otherwise                             = []                         (#2)

quitar(cuantos, x:s)
    | cuantos < 0 || cuantos > longitud (x:s)    = error "No adecuado"   (#3)
    | cuantos == 0                              = x:s                   (#4)
    | otherwise                                 = quitar(cuantos – 1, s) (#5)
```

Ejemplo:

```
quitar(3, [7, 6, 8, 5, 9]) =
= quitar(3, 7:6:8:5:9:[]) =(#5)        cuantos = 3, x = 7, s = 6:8:5:9:[]
= quitar(2, 6:8:5:9:[]) =(#5)          cuantos = 2, x = 6, s = 8:5:9:[]
= quitar(1, 8:5:9:[]) =(#5)            cuantos = 1, x = 8, s = 5:9:[]
= quitar(0, 5:9:[]) =(#4)              cuantos = 0, x = 5, s = 9:[]
= 5:9:[] = [5, 9]
```

27) coger --

coger:: (Int, [t]) → [t]

```
coger(cuantos, [])
    | cuantos /= 0                          = error "No adecuado"        (#1)
    | otherwise                             = []                         (#2)

coger(cuantos, x:s)
    | cuantos < 0 || cuantos > longitud (x:s)    = error "No adecuado"   (#3)
    | cuantos == 0                              = []                    (#4)
    | otherwise                                 = x:coger(cuantos – 1, s)(#5)
```

Ejemplo:

```
coger(3, [7, 6, 8, 5, 9]) =
= coger(3, 7:6:8:5:9:[]) =(#5)         cuantos = 3, x = 7, s = 6:8:5:9:[]
= 7:coger(2, 6:8:5:9:[]) =(#5)         cuantos = 2, x = 6, s = 8:5:9:[]
= 7:6:coger(1, 8:5:9:[]) =(#5)         cuantos = 1, x = 8, s = 5:9:[]
= 7:6:8:coger(0, 5:9:[]) =(#4)         cuantos = 0, x = 5, s = 9:[]
= 7:6:8:[] = [7, 6, 8]
```

28) colapsar --

a)

```
colapsar:: ([Int]) → [Int]

colapsar([]) = []                                   (#1)

colapsar(x:s)
    | es_vacia(s)              = x:s                (#2)
    | x == primero(s)          = colapsar(s)        (#3)
    | otherwise                = x:colapsar(s)      (#4)
```

b)

Ejemplo:

```
colapsar([3, 9, 9, 3]) =
= colapsar(3:9:9:3:[]) =(#4)      x = 3, s = 9:9:3:[], primero(s) = 9
= 3:colapsar(9:9:3:[]) =(#3)      x = 9, s = 9:3:[], primero(s) = 9
= 3:colapsar(9:3:[]) =(#4)        x = 9, s = 3:[], primero(s) = 3
= 3:9:colapsar(3:[]) =(#2)        x = 3, s = []
= 3:9:3:[] = [3, 9, 3]
```

29) propagar --

a)

```
propagar:: ([Int]) → [Int]

propagar([]) = []                                   (#1)

propagar(x:s)
    | es_vacia(s)              = x:s                (#2)
    | x `mod` 2 ==0            = x:propagar((x + primero(s)):resto(s))   (#3)
    | otherwise                = x:propagar(s)      (#4)
```

b)

Ejemplo:

```
propagar([8, 7, 2]) =
= propagar(8:7:2:[]) =(#3)        x = 8, s = 7:2:[], primero(s) = 7, resto(s) = 2:[]
= 8: propagar(15:2:[]) =(#4)        x = 15, s = 2:[], primero(s) = 2, resto(s) = []
= 8:15: propagar(2:[]) =(#2)        x = 2, s = []
= 8:15:2:[] = [8, 15, 2]
```

30) arrollar --

a)

arrollar:: ([Int]) → [Int]

arrollar([]) = [] (#1)

arrollar(x:s)
| es_vacia(s) = x:s (#2)
| x >= primero(s) = x:arrollar(x:resto(s)) (#3)
| otherwise = x:arrollar(s) (#4)

b)

arrollar([3, 9, 9, 3]) =
= arrollar(3:9:9:3:[]) =(#4) x = 3, s = 9:9:3:[], primero(s) = 9, resto(s) = 9:3:[]
= 3:arrollar(9:9:3:[]) =(#3) x = 9, s = 9:3:[], primero(s) = 9, resto(s) = 3:[]
= 3:9:arrollar(9:3:[]) =(#3) x = 9, s = 3:[], primero(s) = 3, resto(s) = []
= 3:9:9:arrollar(9:[]) =(#2) x = 9, s = []
= 3:9:9:9:[] = [3, 9, 9, 9]

31) hundir --

a)

hundir:: ([Int]) → [Int]

hundir([]) = [] (#1)

hundir(x:s)
| es_vacia(s) = x:s (#2)
| x > leh(s) = primero(s):hundir(x:resto(s)) (#3)
| otherwise = x:s (#4)

b)

hundir([**8**, 3, 2, 9]) =
= hundir(8:3:2:9:[]) =(#3) x = 8, s = 3:2:9:[], primero(s) = 3, resto(s) = 2:9:[]
= 3:hundir(8:2:9:[]) =(#3) x = 8, s = 2:9:[], primero(s) = 2, resto(s) = 9:[]
= 3:2:hundir(8:9:[]) =(#4) x = 8, s = 9:[], primero(s) = 9, resto(s) = []
= 3:2:8:9:[] = [3, 2, 8, 9]

32) borrar --

a)

borrar:: ([Int], Int) → [Int]

borrar([], pos) = error "Lista vacía" (#1)

borrar(x:s, pos)
| pos <= 0 || pos > longitud(x:s) = error "No es adecuado" (#2)
| pos == longitud(x:s) = s (#3)
| otherwise = x: borrar(s, pos) (#4)

b)

borrar([**1**, 7, 5, 8], 2) =
= borrar(1:7:5:8:[], 2) =(#4) x = 1, s = 7:5:8:[], pos = 2
= 1: borrar(7:5:8:[], 2) =(#4) x = 7, s = 5:8:[], pos = 2
= 1:7: borrar(5:8:[], 2) =(#3) x = 5, s = 8:[], pos = 2
= 1:7:8:[] = [1, 7, 8]

33) barrer (abril 2008 #1) --

a)

barrer:: ([Int]) → [Int]

barrer([]) = error "Lista vacía" (#1)

barrer(x:s)
| es_vacia(s) = x:s (#2)
| x > primero(s) = barrer(x:resto(s)) (#3)
| otherwise = primero(s):barrer(x:resto(s)) (#4)

b)

barrer([5, 8, 4, 1, 9]) =
= barrer(5:8:4:1:9:[]) =(#4) x = 5, s = 8:4:1:9:[],
primero (s) = 8, resto(s) = 4:1:9:[]
= 8: barrer(5:4:1:9:[]) =(#3) x = 5, s = 4:1:9:[],
primero(s) = 4, resto(s) = 1:9:[]
= 8: barrer(5:1:9:[]) =(#3) x = 5, s = 1:9:[],
primero(s) = 1, resto(s) = 9:[]
= 8: barrer(5:9:[]) =(#4) x = 5, s = 9:[], primero(s) = 9, resto(s) = []
= 8:9: barrer(5:[]) =(#2) x = 5, s = []
= 8:9:5:[] = [8, 9, 5]

34) recorrer (abril 2008 #2) --

a)

```
recorrer:: ([Int]) → [Int]

recorrer([]) = []                                                    (#1)

recorrer(x:s)
    | es_vacia(s)              = x:s                                 (#2)
    | x == primero(s)          = 0:0:resto(s)                        (#3)
    | otherwise                = primero(s):recorrer(x:resto(s))     (#4)
```

b)

```
recorrer([5, 8, 4]) =
= recorrer(5:8:4:[]) =(#4)  x = 5, s = 8:4:[], primero(s) = 8, resto(s) = 4:[]
= 8: recorrer(5:4:[]) =(#4) x = 5, s = 4:[], primero(s) = 4, resto(s) = []
= 8:4: recorrer(5:[]) =(#2) x = 5, s = []
= 8:4:5:[] = [8, 4, 5]

recorrer([5, 5, 4]) =
= recorrer(5:5:4:[]) =(#3)  x = 5, s = 5:4:[], primero(s) = 5, resto(s) = 4:[]
= 0:0:4:[] = [0, 0, 4]
```

35) dividir (junio 2008) --

a)

```
dividir:: ([Int]) → [Int]

dividir([]) = error "Lista vacía"                                    (#1)

dividir(x:s)
    | x == 0                   = error "El primero es cero"          (#2)
    | es_vacia(s)              = s                                   (#3)
    | primero(s) `mod` x == 0  = (primero(s) `div` x):dividir(x:resto(s))
                                                                     (#4)
    | otherwise                = primero(s):dividir(x:resto(s))      (#5)
```

b)

```
dividir([5, 20, 6, 15]) =
= dividir(5:20:6:15:[]) =(#4)   x = 5, s = 20:6:15:[],
                                    primero(s) = 20, resto(s) = 6:15:[]
= 4:dividir(5:6:15:[]) =(#5)    x = 5, s = 6:15:[],
                                    primero(s) = 6, resto(s) = 15:[]
= 4:6:dividir(5:15:[]) =(#4)    x = 5, s = 15:[],
                                    primero(s) = 15, resto(s) = []
= 4:6:3:dividir(5:[]) =(#3)     x = 5, s = []
= 4:6:3:[] = [4, 6, 3]
```

36) superpar (septiembre 2008) --

a)

```
superpar:: ([Int]) → [Int]

superpar([]) = []                                                    (#1)

superpar(x:s)
    | es_vacia(s)              = x:s                                 (#2)
    | x `mod` 2 == 0 && primero(s) `mod` 2 == 0
                               = superpar((x + primero(s)):resto(s)) (#3)
    | x `mod` 2 == 0           = primero(s):superpar(x:resto(s))     (#4)
    | otherwise                = x:superpar(s)                       (#5)
```

b)

```
superpar([3, 10, 8, 9]) =
= superpar(3:10:8:9:[]) =(#5)     x = 3, s = 10:8:9:[],
                                        primero(s) = 10, resto(s) = 8:9:[]
= 3:superpar(10:8:9:[]) =(#3)     x = 10, s = 8:9:[],
                                        primero(s) = 8, resto(s) = 9:[]
= 3:superpar(18:9:[]) =(#4)       x = 18, s = 9:[],
                                        primero(s) = 9, resto(s) = []
= 3:9:superpar(18:[]) =(#2)       x = 18, s = []
= 3:9:18:[] = [3, 9, 18]
```

37) acumular (abril 2009 #1) --

a)

acumular:: ([Int]) → ([Int])

acumular([]) = [] (#1)

acumular(x:s)
|es_vacia(s) = x:[] (#2)
|otherwise = x: acumular((x + primero(s)):resto(s)) (#3)

b) Una vez dadas las ecuaciones, **desarrollar** paso a paso el siguiente ejemplo indicando en cada paso qué ecuación se ha utilizado: **acumular([10, 8, 15])**

acumular([10, 8, 15]) = acumular(10:8:15:[]) =(#3)
= 10:acumular(18:15:[]) =(#3)
= 10:18:acumular(33:[]) =(#2)
= 10:18:33:[]

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

38) adelantar (abril 2009 #2) --

a)

adelantar:: ([Bool], [Int]) → [Int]

adelantar([], r)
| longitud(r) ≠ 0 = error "Distinta longitud" (#1)
| otherwise = [] (#2)

adelantar(x:s, r)
| longitud(x:s) ≠ longitud(r) = error "Distinta longitud" (#3)
| es_vacia(s) = r (#4)
| x == False = primero(r):adelantar(s, resto(r)) (#5)
| x == True =
primero(resto(r)): adelantar(s, primero(r):resto(resto(r))) (#6)

b)

adelantar([**True**, False, **True**, **True**, **True**], [8, 3, 9, 5, 2]) =(#6)
= 3: adelantar([False, **True**, **True**, **True**], [8, 9, 5, 2]) =(#5)
= 3: 8: adelantar([**True**, **True**, **True**], [9, 5, 2]) =(#6)
= 3: 8: 5: adelantar([**True**, **True**], [9, 2]) =(#6)
= 3: 8: 5: 2: adelantar([**True**], [9]) =(#4)
= 3: 8: 5: 2: 9: [] = [3, 8, 5, 2, 9]

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

39) elimparpos (junio 2009) --

a)

elimparpos:: ([Int], Int) → [Int]

elimparpos([], pos) = error "Lista vacía" (#1)

elimparpos(x:s, pos)
| pos < 1 || pos > longitud(x:s) = error "Posición no adecuada" (#2)
| pos == 1 && x mod 2 == 0 = s (#3)
| pos == 1 && x mod 2 /= 0 = x:s (#4)
| otherwise = x: elimparpos(s, pos – 1) (#5)

b)

elimparpos([8, 5, 16, 7, 10, 4], 3) =
= elimparpos(8:5:16:7:10:4:[], 3) =(#5)
= 8: elimparpos(5:16:7:10:4:[], 2) =(#5)
= 8: 5: elimparpos(16:7:10:4:[], 1) =(#3)
= 8: 5: 7:10:4:[] =
= [8, 5, 7, 10, 4]

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

40) ult_primo (septiembre 2009) --

a)

ult_primo:: ([Int]) → Int

ult_primo([]) = -1 (#1)

ult_primo(x:s)
| not es_primo(x) = ult_primo(s) (#2)
| es_vacia(s) = x (#3)
| es_primo(primero(s)) = ult_primo(s) (#4)
| not es_primo(primero(s)) = ult_primo(x:resto(s)) (#5)

Es importante darse cuenta de que en las ecuaciones (#3), (#4) y (#5) se cumple *es_primo(x)*. Así mismo en la ecuaciones (#4) y (#5) se cumple *not es_vacia(s)*. Además, en la ecuación (#5) se podría haber puesto *otherwise* en vez de *not es_primo(primero(s))*.

b)

ult_primo([8, 11, -7, 9, 3, 6]) =
= ult_primo(8:11:-7:9:3:6:[]) =(#2)
= ult_primo(11:-7:9:3:6:[]) =(#5)
= ult_primo(11:9:3:6:[]) =(#5)
= ult_primo(11:3:6:[]) =(#4)
= ult_primo(3:6:[]) =(#5)
= ult_primo(3:[]) =(#3)
= 3

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

41) sublongparponer (Abril 2010 #1) --

a)

sublongparponer:: ([t]) → ([t])

sublongparponer([]) = [] (#1)

sublongparponer(x:s)
| es_vacia(s) = x:x:[] (#2)
| x == primero(s) = x:x:sublongparponer(resto(s)) (#3)
| x /= primero(s) = x:x:sublongparponer(s) (#4)

b)

sublongparponer([10, 10, 10, 8, 8, 15, 8]) =
= sublongparponer(10:10:10:8:8:15:8:[]) =(#3)
= 10:10:sublongparponer(10:8:8:15:8:[]) =(#4)
= 10:10:10:10:sublongparponer(8:8:15:8:[]) =(#3)
= 10:10:10:10:8:8:sublongparponer(15:8:[]) =(#4)
= 10:10:10:10:8:8:15:15:sublongparponer(8:[]) =(#2)
= 10:10:10:10:8:8:15:15:8:8:[] =
= [10, 10, 10, 10, 8, 8, 15, 15, 8, 8]

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

42) mayor_de_cada_par (Abril 2010 #2) --

a)

mayor_de_cada_par:: ([Int]) → ([Int])

mayor_de_cada_par([]) = [] (#1)

mayor_de_cada_par(x:s)

| longitud(x:s) mod 2 /= 0 = error "Longitud impar" (#2)

| x >= primero(s) = x: x: mayor_de_cada_par(resto(s)) (#3)

| x < primero(s) = primero(s): primero(s): mayor_de_cada_par(resto(s)) (#4)

b)

mayor_de_cada_par([10, 8, 5, 7, 7, 20, 5, 5]) =
= mayor_de_cada_par (10:8:5:7:7:20:5:5:[]) =(#3)
= 10:10: mayor_de_cada_par(5:7:7:20:5:5:[]) =(#4)
= 10:10:7:7: mayor_de_cada_par(7:20:5:5:[]) =(#4)
= 10:10:7:7:20:20: mayor_de_cada_par(5:5:[]) =(#3)
= 10:10:7:7:20:20:5:5: mayor_de_cada_par([]) =(#1)
= 10:10:7:7:20:20:5:5:[] =
= [10, 10, 7, 7, 20, 20, 5, 5]

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

43) colocar (Junio 2010) --

a)

colocar:: ([Int], [Int]) → [Int]

colocar([], ℓ) = [] (#1)

colocar(x:s, ℓ)
| nveces(0, x:s) > longitud(ℓ) = error "longitud no adecuada" (#2)
| x == 0 = x:primero(ℓ):colocar(s, resto(ℓ)) (#3)
| x /= 0 = x:colocar(s, ℓ) (#4)

b)

colocar([8, 0, 0, 7], [3, 20, 12, 28])
= colocar(8:0:0:7:0:6:[], 3:20:12:28:[]) =(#4)
= 8: colocar(0:0:7:[], 3:20:12:45:28:[]) =(#3)
= 8: 0: 3: colocar(0:7:[], 20:12:45:28:[]) =(#3)
= 8: 0: 3: 0: 20: colocar(7:[], 12:45:28:[]) =(#4)
= 8: 0: 3: 0: 20: 7: colocar([], 12:45:28:[]) =(#1)
= 8: 0: 3: 0: 20: 7: [] = [8, 0, 3, 0, 20, 7]

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.

44) sublongparquitar (Septiembre 2010) --

a)

sublongparquitar:: ([t]) → ([t])

sublongparquitar([]) = [] (#1)

sublongparquitar(x:s)
| es_vacia(s) = [] (#2)
| x == primero(s) = x:x:sublongparquitar(resto(s)) (#3)
| x /= primero(s) = sublongparquitar(s) (#4)

b)

sublongparquitar([10, 10, 10, 8, 8, 15, 8]) =
= sublongparquitar(10:10:10:8:8:15:8:[]) =(#3)
= 10:10:sublongparquitar(10:8:8:15:8:[]) =(#4)
= 10:10:sublongparquitar(8:8:15:8:[]) =(#3)
= 10:10:8:8:sublongparquitar(15:8:[]) =(#4)
= 10:10:8:8:sublongparquitar(8:[]) =(#2)
= 10:10:8:8:[] =
= [10, 10, 8, 8]

Al desarrollar el ejemplo, en cada paso se ha coloreado la parte de la fórmula a la que se le aplica la ecuación indicada en la esquina derecha de la línea. Se han utilizado dos colores para mejorar la legibilidad, por lo demás los colores distintos no tienen ningún significado.