1. Módulo Lista Enlazada(α)

Interfaz

parámetros formales
géneros α
función COPIAR(**in** a: α) $\rightarrow$ res : α
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} a$}
Complejidad: $\Theta(copy(a))$
Descripción: función de copia de α's

se explica con: SECUENCIA(α), ITERADOR BIDIRECCIONAL(α).

géneros: `lista`(α), `itLista`(α).

Operaciones básicas de lista

VACÍA() $\rightarrow$ res : `lista`(α)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} <>$}
Complejidad: $\Theta(1)$
Descripción: genera una lista vacía.

AGREGARADELANTE(**in/out** l: `lista`(α), **in** a: α) $\rightarrow$ res : `itLista`(α
Pre $\equiv$ {$l =_{\text{obs}} l_0$}
Post $\equiv$ {$l =_{\text{obs}} a \bullet l_0 \wedge res =$ CrearItBi(<>, l) $\wedge$ alias(SecuSuby(res) = l)}
Complejidad: $\Theta(copy(a))$
Descripción: agrega el elemento a como primer elemento de la lista. Retorna un iterador a l, de forma tal que Siguiente devuelva a.
Aliasing: el elemento a agrega por copia. El iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función ELIMINARSIGUIENTE.

Operaciones del iterador

CREARIT(**in** l: `lista`(α)) $\rightarrow$ res : `itLista`(α)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearItBi(<>, l) $\wedge$ alias(SecuSuby(it) = l)}
Complejidad: $\Theta(1)$
Descripción: crea un iterador bidireccional de la lista, de forma tal que al pedir SIGUIENTE se obtenga el primer elemento de l.
Aliasing: el iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función ELIMINARSIGUIENTE.

CREARITULT(**in** l: `lista`(α)) $\rightarrow$ res : `itLista`(α)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearItBi(l, <>) $\wedge$ alias(SecuSuby(it) = l)}
Complejidad: $\Theta(1)$
Descripción: crea un iterador bidireccional de la lista, de forma tal que al pedir ANTERIOR se obtenga el último elemento de l.
Aliasing: el iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función ELIMINARSIGUIENTE.

Representación

Representación de la lista

`lista`(α) **se representa con** `lst`

donde `lst` es `tupla`(*primero*: `puntero(nodo)`, *longitud*: `nat`)

donde `nodo` es `tupla`(*dato*: α, *anterior*: `puntero(nodo)`, *siguiente*: `puntero(nodo)`)

Rep : lst $\longrightarrow$ bool
Rep(l) $\equiv$ true $\Longleftrightarrow$ (l.primero = NULL) = (l.longitud = 0) $\wedge_{\text{L}}$ (l.longitud $\neq$ 0 $\Rightarrow_{\text{L}}$
Nodo(l, l.longitud) = l.primero $\wedge$
($\forall i$: nat)(Nodo(l,i)$\rightarrow$siguiente = Nodo(l,$i+1$)$\rightarrow$anterior) $\wedge$
($\forall i$: nat)($1 \leq i < l$.longitud $\Rightarrow$ Nodo(l,i) $\neq$ l.primero)

Nodo : lst l $\times$ nat $\longrightarrow$ puntero(nodo) {l.primero $\neq$ NULL}
Nodo(l,i) $\equiv$ **if** $i = 0$ **then** l.primero **else** Nodo(FinLst(l), $i-1$) **fi**

FinLst : lst $\longrightarrow$ lst
FinLst(l) $\equiv$ Lst(l.primero$\rightarrow$siguiente, l.longitud $-$ mín{l.longitud, 1})

Lst : puntero(nodo) $\times$ nat $\longrightarrow$ lst
Lst(p, n) $\equiv$ $\langle p, n \rangle$

Abs : lst l $\longrightarrow$ secu(α) {Rep(l)}
Abs(l) $\equiv$ **if** l.longitud = 0 **then** <> **else** l.primero$\rightarrow$dato $\bullet$ Abs(FinLst(l)) **fi**

Representación del iterador

`itLista`(α) **se representa con** `iter`

donde `iter` es `tupla`(*siguiente*: `puntero(nodo)`, *lista*: `puntero(lst)`)

Rep : iter $\longrightarrow$ bool
Rep(it) $\equiv$ true $\Longleftrightarrow$ Rep($*(it$.lista)) $\wedge_{\text{L}}$ (it.siguiente = NULL $\vee_{\text{L}}$ ($\exists i$: nat)(Nodo($*it$.lista, i) = it.siguiente)

Abs : iter it $\longrightarrow$ itBi(α) {Rep(it)}
Abs(it) $=_{\text{obs}}$ b: itBi(α) | Siguientes(b) = Abs(Sig(it.lista, it.siguiente)) $\wedge$
Anteriores(b) = Abs(Ant(it.lista, it.siguiente))

Sig : puntero(lst) l $\times$ puntero(nodo) p $\longrightarrow$ lst {Rep($\langle l, p \rangle$)}
Sig(i, p) $\equiv$ Lst(p, $l\rightarrow$longitud $-$ Pos($*l$, p))

Ant : puntero(lst) l $\times$ puntero(nodo) p $\longrightarrow$ lst {Rep($\langle l, p \rangle$)}
Ant(i, p) $\equiv$ Lst(**if** $p = l\rightarrow$primero **then** NULL **else** $l\rightarrow$primero **fi**, Pos($*l$, p))

Nota: cuando p = NULL, Pos devuelve la longitud de la lista, lo cual está bien, porque significa que el iterador no tiene siguiente.

Pos : lst l $\times$ puntero(nodo) p $\longrightarrow$ puntero(nodo) {Rep($\langle l, p \rangle$)}
Pos(l,p) $\equiv$ **if** l.primero = $p \vee l$.longitud = 0 **then** 0 **else** 1 + Pos(FinLst(l), p) **fi**

2. Módulo Agentes

Interfaz

se explica con: AGENTES

géneros: `agentes`

Operaciones básicas de agentes

NUEVOAGENTES(**in** as : `dicc(placa, posicion)`) $\rightarrow$ res : `agentes`
Pre $\equiv$ {$\neg$ $\emptyset$?(claves(as))}
Post $\equiv$ {$res =_{\text{obs}}$ nuevoAgentes(as)}
Complejidad: O(Na) //Revisar al hacer algoritmo

Descripción: Crea un nuevo contenedor de Agentes con los agentes contenidos en as. Na es la cantidad de agentes definidos en as

AGENTES?(**in** as: `agentes`) $\rightarrow$ res : `conj(placa)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ agentes?(as)}
Complejidad: $\Theta(1)$
Descripción: Me devuelve un conjunto de todos los agentes definidos en as

AGREGARSANCION(**in** a: `placa`, **in/out** as: `agentes`)
Pre $\equiv$ {estaAgente(a, as) $\wedge$ as $= as_0$}
Post $\equiv$ {$as =_{\text{obs}}$ agregarSancion(a, as_0)}
Complejidad: $\Theta(1)$
Descripción: Agrega una sancion al agente a

CAMBIARPOSICION(**in** a: `placa`, **in** p: `posicion`, **in/out** as: `agentes`)
Pre $\equiv$ {estaAgente(a, as) $\wedge$ as $= as_0$}
Post $\equiv$ {$as =_{\text{obs}}$ cambiarPos(a, p, as_0)}
Complejidad: $\Theta(1)$
Descripción: Modifica la posicion del agente a, para que sea p

AGREGARCAPTURA(**in** a: `placa`, **in/out** as: `agentes`)
Pre $\equiv$ {estaAgente(a, as) $\wedge$ as $= as_0$}
Post $\equiv$ {$as =_{\text{obs}}$ agregarCaptura(a, as_0)}
Complejidad: $\Theta(1)$
Descripción: Agrega una captura al agente a

POSAGENTE(**in** a: `placa`, **in** as: `agentes`) $\rightarrow$ res : `posicion`
Pre $\equiv$ {estaAgente(a, as)}
Post $\equiv$ {$res =_{\text{obs}}$ posicionAgente(a, as)}
Complejidad: $\Theta(1)$
Descripción: Devuelve la posicion actual del agente a

SANCIONESAGENTE(**in** a: `placa`, **in** as: `agentes`) $\rightarrow$ res : `nat`
Pre $\equiv$ {estaAgente(a, as)}
Post $\equiv$ {$res =_{\text{obs}}$ sancionesAgente(a, as)}
Complejidad: $\Theta(1)$
Descripción: Devuelve las sanciones actuales del agente a

CAPTURASAGENTE(**in** a: `placa`, **in** as: `agentes`) $\rightarrow$ res : `nat`
Pre $\equiv$ {estaAgente(a, as)}
Post $\equiv$ {$res =_{\text{obs}}$ capturasAgente(a, as)}
Complejidad: $\Theta(1)$
Descripción: Devuelve la cantidad de capturas actuales del agente a

MASVIGILANTE(**in** as: `agentes`) $\rightarrow$ res : `placa`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ masVigilante(as)}
Complejidad: $\Theta(1)$
Descripción: Devuelve el agente que mas capturas tiene en as. Si hubiera mas de uno, devuelve el de menor placa.

CONMISMASSANCIONES(**in** a: `placa`, **in** as: `agentes`) $\rightarrow$ res : `puntero(conj(placa))`
Pre $\equiv$ {estaAgente(a, as)}
Post $\equiv$ {$res =_{\text{obs}}$ conMismasSanciones(a, as)}
Complejidad: $\Theta(1)$
Descripción: Devuelve la referencia a el conjunto de agentes que tienen la misma cantidad de sanciones que el agente a
Aliasing: res no es modificable. Cualquier referencia que se guarde queda invalidada si se agregan sanciones.

CONKSANCIONES(**in** k: `nat`, **in/out** as: `agentes`) $\rightarrow$ res : `conj(placa)`
Pre $\equiv$ {true}

Post $\equiv \{res =_{\text{obs}} \text{conKSanciones}(k, as)\}$
Complejidad: O(Na) la primera vez, O(log(Na)) en siguientes llamadas mientras no ocurran sanciones.
Descripción: Devuelve el conjunto de agentes que tienen exactamente k sanciones. Na es la cantidad de agentes definidos en as
Aliasing: res no es modificable. Cualquier referencia que se guarde queda invalidada si se agregan sanciones. Si res es $NULL$, significa que no se encontraron agentes con k sanciones

Representación

Representación de los Agentes

agentes se representa con estr

donde `estr` es `tupla`(*as*: `DiccRapido(nat, datos)`, *claves*: `conj(nat)` , *masVig*: `nat`, *huboSanciones*: `bool`, *mismSanciones*: `lista(conj(nat))`, *kSanciones*: `Arreglo(tuplaK)`)
donde `datos` es `tupla`(*pos*: `posicion`, *sanciones*: `nat`, *capturas*: `nat`, *conMismSanciones*: `itLista(conj(nat))`)
donde `tuplaK` es `tupla`(*sanciones*: `nat`, *placa*: `nat`)
donde `posicion` es `tupla`(*fila*: `nat`, *columna*: `nat`)

La idea de la lista enlazada mismSanciones es que guarde en cada posicion a todos aquellos agentes que comparten sanciones, con rapido acceso gracias al Iterador en los datos del agente. El arreglo kSanciones se utiliza para ordenar a los agentes por su cantidad de sanciones en tiempo O(N), y poder buscar a alguno con K sanciones en O(log(N)) para acceder a aquellos que tienen la misma cantidad via mismSanciones

Invariante de representacion en castellano:

1. *claves* son las claves del diccionario *as*
2. masVigilante esta definido en agentes.
3. masVigilante es el agente con menor numero de placa entre aquellos que tienen mas capturas en el diccionario de agentes.
4. El arreglo kSanciones tiene almacenadas todas las placas de agentes.
5. Si no hubo sanciones (¬huboSanciones), entonces el arreglo kSanciones representa a los agentes en orden creciente de sanciones. Ademas las sanciones se corresponden con el diccionario
6. Los agentes de la lista mismSanciones no estan repetidos, y son exactamente los definidos en diccionario de agentes.
7. Para todo item de la lista mismSanciones, y para todo agente dentro del conjunto del item, la cantidad de sanciones es igual al resto del conjunto, y menor al de todos los agentes de items siguientes.

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

1. ($(\forall\ a$: nat) def?(a, e.as) $\Rightarrow$ En(a, e.claves)) $\wedge$
2. def?(e.masVigilante, e.as) $\wedge_{\text{L}}$
3. $((\forall\ a$: nat) def?(a, e.as) $\Rightarrow_{\text{L}}$ (obtener(a, e.as).capturas = obtener(e.masVigilante, e.as).capturas $\wedge$ $a >$ e.masVigilante) $\vee$ (obtener(a, e.as).capturas $<$ obtener(e.masVigilante, e.as).capturas) $\vee$ ($a =$ e.masVigilante)) $\wedge$
4. ($(\forall\ a$: nat) def?(a, e.as) $\Rightarrow_{\text{L}}$ $(\exists!\ i$: nat) e.kSanciones[i].placa = a $\wedge$
5. ¬(e.huboSanciones) $\Rightarrow$ (CorrespondenSanciones(e) $\wedge_{\text{L}}$ SancionesOrdenadas(e)) $\wedge$
6. ($((\forall\ i$: nat) $i <$ Longitud(e.mismSanciones) $\Rightarrow_{\text{L}}$ MSDefinidosEnDicc(e,i)) $\wedge$ DiccDefinidosEnMSY-NoRepetidos(e)) $\wedge_{\text{L}}$
7. ($(\forall\ i$: nat) ($i <$ Longitud(e.mismSanciones) $\Rightarrow_{\text{L}}$ MSTieneMismSanciones(e, i)) $\wedge$ ($i <$ (Longitud(e.mismSanciones) -1) $\Rightarrow_{\text{L}}$ MSCadaItemTieneDifSanciones(e,i)))

Reemplazos sintacticos:

CorrespondenSanciones(e) $\equiv$ ($\forall$ i : nat) i < Tam(e.kSanciones) $\Rightarrow_{\text{L}}$ e.kSanciones[i].sanciones = Obtener(e.kSanciones[i].placa, e.as).sanciones

SancionesOrdenadas(e) $\equiv$ ($\forall$ i : nat) i < (Tam(e.kSanciones)-1) $\Rightarrow_{\text{L}}$ Obtener(e.kSanciones[i].placa, e.as).sanciones $\leq$ Obtener(e.kSanciones[$i+1$].placa, e.as).sanciones

MSDefinidosEnDicc(e, i) $\equiv$ ($\forall$ a : nat) En(a, e.mismSanciones[i]) $\Rightarrow$ Def?(a, e.as)

DiccDefinidosEnMSYNoRepetidos(e) $\equiv$ ($\forall$ a : nat) Def?(a, e.as) $\Rightarrow$ (($\exists!$ i : nat) i < Longitud(e.mismSanciones) $\Rightarrow_{\text{L}}$ En(a, e.mismSanciones[i]))

MSTieneMismSanciones(e, i) $\equiv$ ($\forall$ a, a' : nat) (En(a, e.mismSanciones[i]) $\wedge$ En(a', e.mismSanciones[i]) $\wedge$ $\neg(a = a')$) $\Rightarrow_{\text{L}}$ Obtener(a, e.as) = Obtener(a', e.as)

MSCadaItemTieneDifSanciones(e, i) $\equiv$ ($\forall$ a, a' : nat) (En(a, e.mismSanciones[i]) $\wedge$ En(a', e.mismSanciones[$i+1$])) $\Rightarrow_{\text{L}}$ Obtener(a, e.as) < Obtener(a', e.as)

Abs : estr e $\longrightarrow$ agentes {Rep(e)}

Abs(e) $\equiv$ as : agentes /
($\forall$ a : placa) Def?(a, e.as) $\iff$ $a \in$ agentes?(as) $\wedge_{\text{L}}$
($\forall$ a : placa) Def?(a, e.as) $\Rightarrow_{\text{L}}$
(Obtener(a, e.as).sanciones $=_{\text{obs}}$ sancionesAgente(a, as) $\wedge$
Obtener(a, e.as).capturas $=_{\text{obs}}$ capturasAgente(a, as) $\wedge$
Obtener(a, e.as).pos $=_{\text{obs}}$ posicionAgente(a, as) $\wedge$)

Algoritmos

Algoritmos de Agentes

Lista de algoritmos

```
iNuevoAgentes(in d: DiccRapido(placa,datos)) → res: estr
begin
    var
    itAMismSanciones : itConj
    itClavesD : itConj
    i : nat
    i ← 0                                                                    //O(1)
    res.as ← DiccRapidoVacio()                                               //O(1)
    res.claves ← Claves(d)                                                   //O(N)
    res.masVig ← 0                                                           //O(1)
    res.mismSanciones ← Vacia()                                              //O(1)
    AgregarAtras(res.mismSanciones, res.claves)                              //O(N)
    itAMismSanciones ← CrearIt(res.mismSanciones)                            //O(1)
    res.kSanciones ← CrearArreglo(Longitud(res.claves))                      //O(N)
    itClavesD ← CrearIt(res.claves)                                          //O(1)
    while HayMas(itClavesD) do                                               //Guarda: O(1)
        if res.masVig == 0 then                                              //O(1)
            res.masVig ← Actual(itClavesD)                                   //O(1)
        end
        if Actual(itClavesD) < res.masVig then                               //O(1)
            res.masVig ← Actual(itClavesD)                                   //O(1)
        end
        Definir ( Actual(itClavesD), ⟨ Obtener(Actual(itClavesD), d), 0, 0, 0, itAMismSanciones ⟩, res.as )
        //O(1)
        res.kSanciones[i] ← Actual(itClavesD)                                //O(1)
        i ← i+1                                                              //O(1)
        Avanzar(itClavesD)                                                   //O(1)
    end
                                                                             //While: O(N)
    res.huboSanciones ← false                                                //O(1)
end
```

Complejidad: O(N), con N la cantidad de agentes

Algoritmo 1: NuevoAgentes

```
iAgentes?(in as: estr) → res: itConj
begin
    res ← CrearIt(as.claves)                                                 //O(1)
end
```

Complejidad: O(1)

Algoritmo 2: Agentes?

```
iAgregarSancion( in a: nat, in/out as: estr )
begin
    var
    iteradorLista : itLista
    //Primero, le agrego la sancion directamente sobre el significado (DiccRapido hace aliasing en la operacion Obtener)
    Obtener(a, as.as).sanciones ← (Obtener(a, as.as).sanciones + 1)                    //O(1)
    as.huboSanciones ← true                                                            //O(1)
    //Ahora tengo que modificar mismSanciones para que refleje el cambio
    iteradorLista ← Obtener(a, as.as).conMismSanciones                                 //O(1)
    //Borro a a del conjunto, porque ya no comparte sanciones con nadie del mismo
    Borrar(a, Siguiente(iteradorLista))                                                //O(N), pero se desestima
    //Me muevo al lugar que le corresponde ahora
    Avanzar(iteradorLista)                                                             //O(1)
    if ¬ HaySiguiente(iteradorLista) then                                              //O(1)
        //Si no hay nada, creo un nuevo elemento que solo me tiene a mi
        AgregarComoSiguiente(iteradorLista, Vacio())                                   //O(1)
        Siguiente(iteradorLista).Agregar(a)                                            //O(1)
    else
        if Obtener(DameUno(Siguiente(iteradorLista)), as).sanciones > Obtener(a, as.as).sanciones then
        //O(1)
            //Si el que esta en el lugar al que iba tiene mas sanciones que yo, me agrego antes para mantener el orden
            creciente
            AgregarComoAnterior(iteradorLista, Vacio())                                //O(1)
            Anterior(iteradorLista).Agregar(a)                                         //O(1)
            Retroceder(iteradorLista)                                                  //O(1)
        else
            //Si no, debe tener las mismas (tiene mas que las que yo tenia antes, y yo sume una sancion, a lo sumo tiene
            la misma cantidad)
            Siguiente(iteradorLista).Agregar(a)                                        //O(1)
        end
    end
    Obtener(a, as.as).conMismSanciones ← iteradorLista                                 //O(1)
end
```

Complejidad: O(N) | Desestimando el borrado: O(1)

Algoritmo 3: AgregarSancion

```
iCambiarPosicion(in a: nat, in p: posicion, in/out as: estr)
begin
    Obtener(a, as.as).posicion ← p                                                     //O(1)
end
```

Complejidad: O(1)

Algoritmo 4: CambiarPosicion

```
iAgregarCaptura(in a : nat, in/out as : estr)
begin
  | Obtener(a, as.as).capturas ← (Obtener(a, as.as).capturas + 1)                    //O(1)
  | //Ademas de sumar captura, hago el mantenimiento de masVigilante
  | if Obtener(a, as.as).capturas > Obtener(as.masVigilante, as).capturas then      //O(1)
  |   | as.masVigilante ← a                                                          //O(1)
  | else
  |   | if Obtener(a, as.as).capturas == Obtener(as.masVigilante, as).capturas then //O(1)
  |   |   | if a < as.masVigilante then                                            //O(1)
  |   |   |   | as.masVigilante ← a                                                //O(1)
  |   |   | end
  |   | end
  | end
end
Complejidad: O(1)
```

Algoritmo 5: AgregarCaptura

```
iPosAgente(in a : nat, in as : estr) → res: posicion
begin
  | res ← Obtener(a, as.as).posicion                                                //O(1)
end
Complejidad: O(1)
```

Algoritmo 6: PosAgente

```
iSancionesAgente(in a : nat, in as : estr) → res: nat
begin
  | res ← Obtener(a, as.as).sanciones                                              //O(1)
end
Complejidad: O(1)
```

Algoritmo 7: SancionesAgente

```
iCapturasAgente(in a : nat, in as : estr) → res: nat
begin
  | res ← Obtener(a, as.as).capturas                                               //O(1)
end
Complejidad: O(1)
```

Algoritmo 8: CapturasAgente

```
imasVigilante(in as : estr) → res: nat
begin
  | res ← as.masVigilante                                                          //O(1)
end
Complejidad: O(1)
```

Algoritmo 9: masVigilante

---

*i*ConMismasSanciones(**in** *a*: `nat`, **in** *as*: `estr`) → res: puntero(conj(nat))
begin
| res ← &(Siguiente(Obtener(a, as.as).conMismSanciones)) //**O(1)**
end
Complejidad: O(1)

Algoritmo 10: ConMismasSanciones

---

*i*ConKSanciones(**in** *k*: `nat`, **in/out** *as*: `estr`) → res: puntero(conj(nat))
begin
| var
| i : nat
| encontrado : bool
| encontrado ← false //**O(1)**
| i ← 0 //**O(1)**
| **if** *as.huboSanciones == true* **then** //**O(1)**
| | //Actualizo el arreglo kSanciones
| | **while** *i < Tam(as.kSanciones)* **do** //**O(1)**
| | | as.kSanciones[i].sanciones ← Obtener(as.kSanciones[i].placa, as.as).sanciones //**O(1)**
| | | i ← i + 1 //**O(1)**
| | **end**
| | //**While: O(N)**
| | //Y luego lo ordeno en orden creciente de sanciones
| | CountingSortSanciones(as.kSanciones, as) //**O(N)**
| **end**
| //En el arreglo kSanciones tengo, en orden creciente por sancion, a los agentes. Busco a uno con k sanciones
| //Busqueda binaria me devuelve el i que cumple que las sanciones del agente en posicion i del arreglo son k
| encontrado ← BusquedaBinariaSanciones(k, as.kSanciones, as, i) //**O(log(N))**
| **if** *encontrado* **then** //**O(1)**
| | res ← &(Siguiente(Obtener(as.kSanciones[i], as.as).conMismSanciones)) //**O(1)**
| **else**
| | res ← NULL //**O(1)**
| **end**
end
Complejidad: O(2N + log(N)) = O(N) | En llamadas siguientes: O(log(N))
Comentarios: La primer complejidad se da cuando hubo sanciones. N es la cantidad de agentes. En proximas llamadas, la complejidad pasa a ser unicamente la busqueda binaria, es decir, log(N)

Algoritmo 11: ConKSanciones

```
CountingSortSanciones(in arreglo: Arreglo(tuplaK), in as: as) → res: Arreglo(tuplaK)
begin
    var
    i, total, cantidadAnt : nat
    maxSanciones : nat
    cantidad : Arreglo(nat)
    output : Arreglo(tuplaK)
    iterador : itConj(nat)
    i ← 0                                                                          //O(1)
    cantidadAnt ← 0                                                                //O(1)
    total ← 0                                                                      //O(1)
    //Se que a todos los agentes con las maximas sanciones los tengo en el ultimo elemento de la lista mismSanciones.
    maxSanciones ← Obtener(DameUno(Ultimo(as.mismSanciones)), as.as).sanciones      //O(1)
    //Creo el arreglo cantidad con MaxSanciones posiciones.
    cantidad ← CrearArreglo(maxSanciones)                                          //O(maxSanciones)
    output ← CrearArreglo(Tam(arreglo))                                            //O(N)
    while i < maxSanciones do                                                      //O(1)
        cantidad[i] ← 0                                                            //O(1)
        i ← i + 1                                                                  //O(1)
    end
                                                                                   //While: O(maxSanciones)
    //Calculo la cantidad de cada numero de sanciones
    i ← 0                                                                          //O(1)
    while i < Tam(arreglo) do                                                      //O(1)
        cantidad[arreglo[i].sanciones] ← cantidad[arreglo[i].sanciones] + 1        //O(1)
        i ← i + 1                                                                  //O(1)
    end
                                                                                   //While: O(N)
    //Calculo el indice inicial para cada numero de sanciones
    i ← 0                                                                          //O(1)
    while i < maxSanciones do                                                      //O(1)
        cantidadAnt ← cantidad[i]                                                  //O(1)
        cantidad[i] ← total                                                        //O(1)
        total ← total + cantidadAnt                                                //O(1)
        i ← i + 1                                                                  //O(1)
    end
                                                                                   //While: O(maxSanciones)
    //Coloco a cada agente (<sanciones, placa>) en el lugar que le corresponde en el output
    i ← 0                                                                          //O(1)
    while i < Tam(arreglo) do                                                      //O(1)
        output[cantidad[arreglo[i].sanciones]] ← arreglo[i]                        //O(1)
        cantidad[arreglo[i].sanciones] ← cantidad[arreglo[i].sanciones] + 1        //O(1)
        i ← i + 1                                                                  //O(1)
    end
                                                                                   //While: O(N)
end
```

Complejidad: O(4N + 3maxSanciones) = O(N)

Comentarios: Vamos a asumir que maxSanciones es, a lo sumo, un multiplo de N (kN) con un k constante y pequeño (k < N), y tomar la complejidad como O(N).

Algoritmo 12: CountingSortSanciones

```
BusquedaBinariaSanciones(in k: nat, in arreglo: Arreglo(tuplaK), in as: estr in/out i: nat) → res: bool
begin
    var
    iMin, iMax, iMed : nat
    iMin ← 0                                                                    //O(1)
    iMax ← Tam(arreglo)                                                         //O(1)
    while iMin ≤ iMax do                                                        //O(1)
        iMed ← (iMin + iMax)/2                                                  //O(1)
        if arreglo[iMed].sanciones == k then                                    //O(1)
            res ← iMed                                                          //O(1)
            iMin ← iMax + 1                                                     //O(1)
        else
            if arreglo[iMed].sanciones < k then                                 //O(1)
                iMin ← iMed + 1                                                 //O(1)
            else
                iMax ← iMed − 1                                                 //O(1)
            end
        end
    end
                                                                                //While: O(log(N))
end
```

Complejidad: O(log(N))

Comentarios: i devuelve el indice donde el agente tiene k sanciones. i se invalida si res es $false$ (no encontrado)

Algoritmo 13: BusquedaBinariaSanciones