

Ejercicio 14

Para poder resolver este problema con la complejidad que nos piden vamos a tener que utilizar una combinacion entre heap sort y listas enlazadas. Fijémonos por qué merge no sirve:  
Supongamos que tenemos el arreglo [2,3,4] y k=2  
Si pudieramos armar el siguiente arreglo en n\*k: [2, 4, 3, 6, 4, 8]  
Nos damos cuenta que podríamos ir haciendo merge de a partes.  
Paso 1: merge([2,4], [3,6]) --> [2,3,4,6]  
Paso 2: merge([2,3,4,6], [4, 8]) --> [2,3,4,4,6,8]  
Esto no funciona, la complejidad de hacer merge en n listas ordenadas es cuadratica. (es una sumatoria de gauss)  
Así que: Heap sort al rescate. Vamos a usar listas enlazadas para que remover el primero sea O(1)

proc ordenarMultiplos (in A: Array<int>, in k: int) : Array<int>

```
1  var res: Array<int>
2    res:= new Array(A.length * k) // O(n*k)
3
4  var arregloDeListas: Array<ListaEnlazada<int>>
5    arregloDeListas:= new Array(A.length)
6  var j: int
7  var i: int
8    i:= 1
9    j:= 0
10
11 while (j < A.length) do
12   arregloDeListas[j] = listaVacia()
13
14   while (i <= k) do
15     arregloDeListas[j].agregarAtras(A[j] * i)
16     i:= i + 1
17   endwhile
18
19   i:= 1
20   j:= j + 1
21 endwhile
22
23 // Cola de prioridad min, por el primer elemento de la lista enlazada
24 var colaDePrioridad: ColaDePrioridadLog<ListaEnlazada<int>>
25   colaDePrioridad:= colaDePriodidadVacia()
26   j:= 0
27
28 while (j < arregloDeListas.length) do
29   colaDePrioridad.encolar(arregloDeListas[j])
30   j:= j + 1
31 endwhile
32
33 var res: Array<int>
34 var indiceActual: int
35   res:= new Array(A.length * k)
36   indiceActual:= 0
37
38 while (!colaDePriodidad.estaVacia())
39   var listaPrioritaria: ListaEnlazada<int>
40     listaPrioritaria:= colaDePriodidad.desencolarMax()
41
42   res[i]:= listaEnlazada(listaPrioritaria.comienzo())
43
44   listaPrioritaria.fin()
45   colaDePriodidad.encolar(listaPrioritaria)
46
47   i:= i + 1
48 endwhile
49
50 return res
```

Ejercicio 17

Planta = string

Stock = int

Prioridad = int

Nodo = Struct(planta: Planta, stock: Stock, prioridad: Prioridad)

```
proc Recolectar (  
  in s: Array<Tupla<Planta, Stock>>,  
  in u: DiccionarioDigital<Planta, Prioridad>  
) : Array<Planta>
```

```
1  var stockTotal: DiccionarioDigital<Planta, Stock>  
2    stockTotal:= diccionarioVacio()  
3  
4  // Inicializamos stockTotal  
5  var j:= 0  
6  while (j < s.length) do  
7    var planta: Planta  
8    var stock: Stock  
9    var stockAcumulado: Stock  
10     planta:= s[j][0]  
11     stock:= s[j][1]  
12     stockAcumulado:= 0  
13  
14    if (stockTotal.pertenece(planta)) then  
15      stockAcumulado:= stockTotal.obtener(planta)  
16    endif  
17  
18    stockTotal.definir(Planta, stockAcumulado + stock)  
19    j:= j + 1  
20  endwhile  
21  
22  // Evitemos a toda costa iterar sobre un dict  
23  // justificar que se amortiza es siempre complicado  
24  // TRUCAZO  
25  var nodos: ListaEnlazada<Nodo>  
26    nodos:= listaVacia()  
27    j:= 0  
28  
29  while (j < s.length) do  
30    var planta: Planta  
31    planta:= s[j][0]  
32    if (stockTotal.pertenece(planta)) then  
33      var prioridad: Prioridad  
34      var stock: Stock  
35      prioridad:= u.obtener(planta)  
36      stock:= stockTotal.obtener(planta)  
37  
38      nodos.agregarAtras(new Nodo(planta=planta, stock=stock, prioridad=prioridad))  
39      stockTotal.borrar(planta)  
40    endif  
41    j+= j + 1  
42  endwhile  
43  
44  var arreglo_nodos: Array<Nodo>  
45    arreglo_nodos:= toArray(nodos)  
46  
47  mergeSort(arreglo_nodos) // ordenar de menor a mayor por stock  
48  mergeSort(arreglo_nodos) // ordenar de mayor a menor por prioridad  
49  
50  
51  var res: Array<Planta>  
52    res:= new Array(arreglo_nodos.longitud())  
53    j:=0  
54  
55  while (j < arreglo_nodos.longitud()) do  
56    res[j]:= arreglo_nodos[j].planta  
57    j:= j + 1  
58  endwhile  
59  
60  return res
```

Ejercicio 18

```
Libreta = int
Nota = int
Promedio = float
Contador = totalNotas : int, notasAcumuladas : Nota

proc OrdenarPorLibretaYPromedios (in s: Array<Tupla<Libreta, Nota>>) : Array<Tupla<Libreta, Promedio>>
```

```
1  var promedios: DiccionarioDigital<Libreta, Contador>
2  promedios:= diccionarioVacio()
3
4  var j: int
5      j:= 0
6
7  // Inicializamos el dict.
8  while (j < s.length) do
9      var libreta: Libreta
10         libreta:= s[j][0]
11     promedios.definir(
12         libreta,
13         new Contador(s[j], new Contador(totalNotas=0, notasAcumuladas=0))
14     )
15     j += 1
16 endwhile
17
18 // Insertamos las notas acumuladas.
19 j:= 0
20 while (j < s.length) do
21     var libreta: Libreta, nota: Nota
22     libreta:= s[j][0], nota:= s[j][1]
23
24     var contador: contador
25     contador:= promedios.obtener(libreta)
26
27     contador.totalNotas:= contador.totalNotas + 1
28     contador.notasAcumuladas:= nota
29
30     // No hace falta re-definir, hay aliasing
31     j += 1
32 endwhile
33
34 // Creamos res
35 var res: Array<Tupla<Libreta, Promedio>>
36     res:= new Array<promedios.tamaño()>
37
38 // Vamos a zafar de iterar promedios, nunca queremos
39 // iterar y tener que justificar que las operaciones
40 // se amortizan, TRUCAZO:
41 var i: int
42     i:= 0
43     j:= 0
44 while (j < s.length) do
45     var notaAcumulada: Nota
46     var libreta: Libreta
47         nota:= s[j][1]
48         libreta:= s[j][0]
49
50     if (promedios.pertenece(libreta))
51         var contador: contador
52         contador:= promedios.obtener(libreta)
53
54         res[i]:= <libreta, contador.notasAcumuladas / contador.totalNotas>
55         promedios.borrar(libreta)
56
57         i:= i + 1
58     endif
59
60     j:= j + 1
61 endwhile
62
63 mergeSort(res) // ordenar res por la primera coordenada: libreta. (menor a mayor)
64 mergeSort(res) // ordenar res por la segunda coordenada: promedio. (mayor a menor)
65
66 return res
```

## Ejercicio 19

Prioridad = *int*

Nodo = Struct(prioridad: *Prioridad*, valor: *int*)

proc OrdenarSegunCriterio (in s: Array<int>, in crit: Array<Prioridad>) : Array<int>

```
1  var s1: ListaEnlazada<Nodo>
2  var s2: ListaEnlazada<int>
3  var critDict: DiccionarioLog<int, Prioridad>
4      s1:= listaVacia()
5      s2:= listaVacia()
6      critDict:= diccionarioVacio()
7
8  var prioridad: int
9      prioridad:= 0
10
11 // Inidicalizamos crit dict.
12 while (prioridad < crit.length) do
13     critDict.definir(crit[prioridad], prioridad)
14     prioridad:= prioridad + 1
15 endwhile
16
17 var j: int
18     j:= 0
19
20 // Insertamos en las listas s1 y s2.
21 while (j < s.length) do
22     if critDict.pertenece(s[j]) then
23         s1.agregarAtras(s[j], new Nodo(prioridad=critDict.obtener(s[j]), valor=s[j]))
24     else
25         s2.agregarAtras(s[j])
26     endif
27 endwhile
28
29 // Ordenar la primera parte del arreglo
30 var s1Arreglo: Array<Nodo>
31 var s1Valores: Array<int>
32     s1Arreglo:= toArray(s1)
33     s1Valores:= new Array(s1Arreglo.length)
34
35 mergeSort(s1Arreglo) // ordenar s1Arreglo por prioridad.
36 // No hay criterio de desempate pues crit no tiene repetidos
37
38 j:= 0
39 while (j < s1Arreglo.length) do
40     s1Valores[j]:= s1Arreglo[j].valor
41     j:= j + 1
42 endwhile
43
44 // Ordenar la segunda parte del arreglo
45 var s2Valores: Array<int>
46     s2Valores:= toArray(s2)
47
48 mergeSort(s2Valores) // ordenar s2Arreglo de menor a mayor
49
50 // Concatenar en res.
51 var res: Array<int>
52     res:= concat(s1Valores, s2Valores)
53
54 // Devolver res.
55 return res
```