FUNDAMENTOS DE COMPUTACIÓN
PRÁCTICO 5
ORDENAMIENTO DE LISTAS (SORTING)

En este práctico aplicaremos las nociones vistas en la segunda parte del curso e implementaremos varios métodos clásicos de ordenamiento de listas.

A su vez, aprovecharemos para ver otras formas de recursión, y justificaremos en cada caso la recursión utilizada en la definición de las funciones.

1. Listas ordenadas

Ejercicio 1. Definir la función `ordenada :: Ord a => [a] -> Bool` que recibe una lista y devuelve `True` si la misma está ordenada en forma creciente (de menor a mayor).
¿Qué tipo de recursión utilizó para definir esta función? Justifique que es correcta.

2. Ordenamiento por Inserción (Insert Sort)

El método de ***inserción*** consiste en ir insertando recursivamente uno por uno los elementos de una lista en la lista que tiene los elementos siguientes ya ordenados (por el mismo método).
Para ello, se utiliza una función auxiliar `insert` que recibe un elemento de un tipo `a` y una lista ordenada de elementos de tipo `[a]`, e inserta al elemento en el lugar indicado para que la lista resultante siga ordenada.

Ejercicio 2. Defina las siguientes funciones para implementar el método de ordenamiento por inserción:

a) `insert :: Ord a => a -> [a] -> [a]`, que inserta un elemento en una lista ordenada dejándola ordenada. Si la lista original no está ordenada, no importa lo que devuelva la función.
Ejemplo: `insert 5 [1,3,6,8,9] = [1,3,5,6,8,9]`

b) `insertSort :: Ord a => [a] -> [a]`, que ordena una lista por el método de inserción descripto a-rriba, con la ayuda de la función `insert`.

3. **Ordenamiento por Selección (Select Sort)**

El método de ordenamiento por ***selección*** consiste en seleccionar el mínimo elemento de una lista y ponerlo como primero de la lista ordenada por el mismo método en forma recursiva.
Para ello, se utilizan dos funciones auxiliares: `minL`, que devuelve el mínimo elemento de una lista no vacía, y otra llamada `borrar1`, que borra la primera ocurrencia de un elemento dado de una lista.

Ejercicio 3. Defina las siguientes funciones para implementar el método de ordenamiento por selección:

a) `minL :: Ord a => [a] -> a`.
Observar que para poder encontrar al mínimo elemento de una lista es necesario poder compararlos mediante un orden, por lo que se requiere que `a` sea instancia de `Ord`.

b) `borrar1 :: Eq a => a -> [a] -> [a]`, que borra la primera ocurrencia de un elemento dado de una lista.

c) `selectSort :: Ord a => [a] -> [a]`, que ordena una lista por el método de selección descripto a-rriba con la ayuda de las funciones anteriores.

d) ¿Qué tipo de recursión utilizó para definir `minL` y para definir `selectSort`? Justifique la corrección de ambas definiciones, y explique por qué la función `selectSort`
termina para cualquier entrada.

4. **Ordenamiento por Intercalamiento (Merge Sort)**

El siguiente método de ordenamiento se realiza por ***intercalamiento***. Este método consiste en partir una lista en dos, ordenar ambas listas recursivamente utilizando el mismo método, y luego intercalar las dos listas ordenadas.
Para ello, se utilizan dos funciones auxiliares: `split` que parte una lista en dos listas de igual tamaño, y `merge` que intercala los elementos de dos listas ordenadas de forma tal que queden ordenados en una nueva lista.

Ejercicio 4. Defina las siguientes funciones para implementar el método de ordenamiento por intercalamiento:

a) `split :: [a] -> ([a],[a])`, que divide una lista en dos listas de igual tamaño (o con diferencia de un elemento), *repartiendo los elementos de la lista entre dos del mismo modo que cuando se reparte una baraja* (o sea, uno para un lado, el siguiente para el otro, y así sucesivamente).
Ejemplo: `split [2,7,5,4,1] = ([2,5,1],[7,4])`

b) `merge :: Ord a => [a] -> [a] -> [a]`, que recibe dos listas ordenadas e intercala sus elementos de forma tal que queden ordenados en una única lista.
Ejemplo: `merge [1,2,5] [4,7] = [1,2,4,5,7]`

c) `mergeSort :: Ord a => [a] -> [a]`, que ordena una lista por el método de intercalamiento descripto arriba con la ayuda de las funciones definidas anteriormente.

d) Justifique la corrección y terminación de las tres funciones anteriores.

5. **Ordenamiento Rápido (Quick Sort)**

El siguiente método de ordenamiento se denomina ***rápido*** por ser muy eficiente. Este método consiste en tomar un elemento de la lista a ordenar (generalmente el primero) como *pivote*. Utilizando el pivote, se divide al resto de la lista en dos listas: la primera contiene todos los elementos menores que el pivote, mientras que la segunda contiene todos los elementos mayores que el pivote (los elementos iguales al pivote pueden ser colocados en cualquiera de las dos listas). Luego se ordenan recursivamente estas dos listas utilizando el mismo método, y se construye la lista ordenada simplemente concatenando la primera lista ordenada, el pivote y la segunda lista ordenada.

Ejercicio 5.

a) Defina la función `quickSort :: Ord a => [a] -> [a]` que implementa el método recientemente descripto.
Para definir esta función puede definirse una función auxiliar: `menoresMayores` que parte una lista en dos, una conteniendo los elementos menores a uno dado y otra los mayores o iguales.
Pero también es posible definir `quickSort` en dos renglones, sin necesidad de esa función, utilizando la función `filter` sobre los elementos de la lista con condiciones adecuadas.

b) ¿Qué tipo de recursión utilizó para definir `quickSort`? Explique por qué su función termina para cualquier entrada.