Algoritmos de Ordenamiento en Haskell – Práctica para Examen Final

Ejercicio 1: Selection Sort (Ordenamiento por Selección)

Este algoritmo busca el elemento más chico y lo pone al principio, repitiendo el proceso con el resto.

Enunciado: `{- a) Escriba una función "minimo :: (Ord a) => [a] -> a" que devuelva el elemento más pequeño de una lista no vacía. -}`

`{- b) Escriba una función "quitar :: (Ord a) => a -> [a] -> [a]" que reciba un elemento y una lista, y devuelva la lista sin la primera aparición de dicho elemento. -}`

`{- c) Escriba la función "ssort :: (Ord a) => [a] -> [a]". Observación: Para ordenar la lista, ssort debe encontrar el mínimo, ponerlo al inicio y llamar recursivamente a ssort con el resto de la lista (la lista original sin el mínimo). -}`

Ejercicio 2: Merge Sort (Ordenamiento por Mezcla)

Este es el "primo" del QuickSort, pero en lugar de usar un pivote, parte la lista exactamente a la mitad y luego mezcla las partes ya ordenadas.

Enunciado: `{- a) Escriba una función "mitades :: [a] -> ([a], [a])" que dada una lista, la divida en dos partes de tamaño similar (pueden usar la función "splitAt" o calcular la longitud). -}`

`{- b) Escriba una función "merge :: (Ord a) => [a] -> [a] -> [a]" que reciba dos listas ya ordenadas y las combine en una sola lista que también esté ordenada. -}`

`{- c) Escriba la función "msort :: (Ord a) => [a] -> [a]". Observación: El algoritmo debe dividir la lista en dos mitades, ordenar cada mitad recursivamente con msort y luego combinarlas usando la función merge. -}`

Ejercicio 3: Insertion Sort

Va construyendo la lista ordenada insertando un elemento por vez en su lugar correspondiente.

Enunciado: `{- a) Escriba la función "isort :: (Ord a) => [a] -> [a]" utilizando la función "insertar" definida previamente. Observación: El algoritmo debe tomar el primer elemento de la lista y utilizar la función "insertar" para ubicarlo en la posición correcta dentro del resto de la lista ya ordenada recursivamente. -}`

Ejercicio 4: Quick Sort

Divide en dos listas "menores" y "mayores" usado un pivote, luego utiliza recursión en cada una de las dos listas y finalmente reconstruye la lista.

Enunciado: {- a) Escriba una función qsort :: (Ord a) => [a] -> [a] sin utilizar listas por comprensión.

Observación: Escriba una función "particion" que reciba como argumento un valor de referencia (pivote) y una lista de valores del mismo tipo que el pivote. Esta función debe devolver una tupla con dos listas (l1, l2), de modo que en l1 se encuentren todos los valores de la lista original que son menores o iguales que el pivote, y en l2 todos los mayores.

Firma de la función: particion :: (Ord a) => a -> [a] -> ([a], [a]) -}