Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Estructura de Datos y Algoritmos II

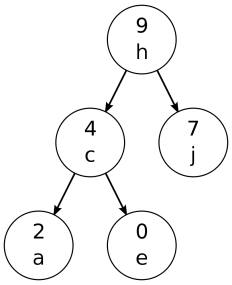
Nombre y Apellido:

Parcial 1

1. Resolver la siguiente recurrencia utilizando el método de sustitución. Expresar la solución utilizando la notación O.

$$T(n) = 4T(\lfloor n/2 \rfloor) + n^2$$

2. Un *treap* es un árbol binario donde en cada nodo se almacenan dos valores: una prioridad y una clave, si miramos sólo las claves de los nodos el árbol es binario de búsqueda, y si miramos sólo las prioridades el árbol es un *max-heap* (es decir, la prioridad de cualquier nodo interno es mayor o igual que la prioridad de sus hijos). De esta manera las claves con prioridad alta (a las que se accede frecuentemente) estarán cerca de la raíz del árbol. El siguiente es un ejemplo de un *treap* con claves alfabéticas y prioridades numéricas:



Usaremos el siguiente tipo de datos para representar esta estructura en Haskell:

$$\textbf{data}\ \mathsf{Treap}\ p\ k = \mathsf{E} \mid \mathsf{N}\ (\mathsf{Treap}\ p\ k)\ p\ k\ (\mathsf{Treap}\ p\ k)$$

Definir las siguientes funciones, que permiten usar este tipo de árboles, de manera eficiente en Haskell:

a) $\mathsf{key} :: \mathsf{Treap}\ p\ k \to k$, devuelve la clave asociada a la raíz del árbol.

b) $\mathsf{priority} :: \mathsf{Treap}\ p\ k \to p$, devuelve la prioridad máxima del árbol, suponiendo que es un *treap*.

c) $\mathsf{isTreap} :: (\mathsf{Ord}\ k, \mathsf{Ord}\ p) \Rightarrow \mathsf{Treap}\ p\ k \to \mathsf{Bool}$, que determina si un árbol binario de tipo $\mathsf{Treap}\ p\ k$ es un *treap*. Expresar la recurrencia correspondiente al trabajo de esta función.

d) Para insertar un elemento x con prioridad p al *treap* se procede de la siguiente manera: se inserta el elemento aplicando el algoritmo de inserción para árboles binarios de búsqueda. Hasta aquí el árbol es bst según las claves pero puede ser que la prioridad p asociada a x sea mayor a la de su padre, en este caso se realiza algunas de las siguientes rotaciones que acomodan los nodos manteniendo el invariante del árbol de ser bst:

$$\mathsf{rotateL}\ (\mathsf{N}\ l'\ p'\ k'\ (\mathsf{N}\ l\ p\ k\ r)) = \mathsf{N}\ (\mathsf{N}\ l'\ p'\ k'\ l)\ p\ k\ r$$
$$\mathsf{rotateR}\ (\mathsf{N}\ (\mathsf{N}\ l\ p\ k\ r)\ p'\ k'\ r') = \mathsf{N}\ l\ p\ k\ (\mathsf{N}\ r\ p'\ k'\ r')$$

Definir una función $\mathsf{insert} :: (\mathsf{Ord}\ k, \mathsf{Ord}\ p) \Rightarrow k \to p \to \mathsf{Treap}\ p\ k \to \mathsf{Treap}\ p\ k$, que permita insertar elementos a un *treap*. Si una clave ya está en el árbol se actualiza su prioridad.

e) $\mathsf{split} :: (\mathsf{Ord}\ k, \mathsf{Ord}\ p, \mathsf{Num}\ p) \Rightarrow k \to \mathsf{Treap}\ p\ k \to (\mathsf{Treap}\ p\ k, \mathsf{Treap}\ p\ k)$, dada una clave x y un *treap* t divide a t en dos *treaps* más pequeños, uno que contiene los elementos con clave menor que x y otro con los elementos con clave mayor a x.

Ayuda: Insertar el elemento x con una prioridad mayor a la de cualquier elemento del *treap*, de manera que el elemento insertado quede en la raíz.