Paradigmas de Programación
Práctica 9

1. Considere la siguiente definición de los tipo de dato ***'a bintree*** para representar árboles binarios con valores de tipo ***'a*** en los nodos.

```
type 'a bintree = Empty | Node of 'a * 'a bintree * 'a bintree
```

Empty representaría el árbol vacío y ***Node (r, i, d)*** representaría el árbol que tiene el valor ***r*** en la raíz y a ***i*** y ***d*** como ramas izquierda y derecha, respectivamente.

Defina una función ***in_order : 'a bintree -> 'a list***, que, para cada árbol devuelva la lista de los valores de sus nodos recorridos en *in_order*.

Decimos que un árbol binario es un *árbol de búsqueda* para una determinada relación de orden (definida para el tipo de los valores de los nodos) si se cumple que (según esa relación de orden) los valores de todos los nodos de la rama izquierda son menores o iguales[1] que el valor de la raíz, el valor de la raíz es menor o igual[1] que los valores de todos los nodos de la rama derecha y que la rama izquierda y la derecha son también árboles de búsqueda. Note que el recorrido en *in_order* de un árbol binario de búsqueda para una determinada relación de orden, da una lista ordenada según esa relación.

Defina una función ***insert : ('a -> 'a -> bool) -> 'a bintree -> 'a -> 'a bintree****,* de modo que, si ***ord*** es una relación de orden y ***tree*** es un árbol binario de búsqueda, ***insert ord tree x*** sea un árbol binario de búsqueda conteniendo los nodos que tenía ***tree*** más un único nodo adicional con valor ***x***.

Defina, utilizando la función ***insert***, una función
bst : ('a -> 'a -> bool) -> 'a list -> 'a bintree, de modo que, si ***ord*** es una relación de orden, ***bst ord l*** sea un árbol binario de búsqueda (según la relación de orden ***ord***) que contenga un nodo para cada valor de la lista ***l***.

```
# bst (<=) ['c'; 'a'; 's'; 'u'; 'a'; 'l'];;
- : char bintree =
    Node ('c', Node ('a', Node ('a', Empty, Empty), Empty),
               Node ('s', Node ('l', Empty, Empty), Node ('u', Empty, Empty)))
```

Utilizando las funciones ***bst*** e ***in_order*** es inmediato construir una función
qsort : ('a -> 'a -> bool) -> 'a list -> 'a list, que ordene cualquier lista según el orden indicado. El funcionamiento de esta función se basaría en los mismos principios que el algoritmo de ordenación “quick sort”. Defina esta función ***qsort***.

```
# qsort (>=) [1; 0; 2; -1; 3; 10; 100];;
- : int list = [100; 10; 3; 2; 1; 0; -1]
```

[1] Usaremos esta definición ligeramente alterada para permitir nodos con “claves” repetidas.

Escriba todas estas definiciones en un archivo "**bintree.ml**" que debe compilar sin errores con la orden

```
ocamlc -c bintree.mli bintree.ml
```

2. (Ejercicio opcional) Defina una función ***is_bst : ('a -> 'a -> bool) -> 'a bintree -> bool***, que indique si un árbol binario es *de búsqueda* para la relación de orden indicada. La definición de esta función debe realizarse de modo que su aplicación sea lo más eficiente posible.

Defina una función ***bfs : 'a bintree -> 'a list que bfs tree*** sea el recorrido en anchura ("breadth first search") del árbol *tree*, del modo más sencillo posible

Intente implementar una función ***bfs' : 'a bintree -> 'a list que bfs tree*** que proporcione una implementación recursiva terminal de la función ***bfs***.

Suele decirse que un árbol binario es ***"perfecto"*** (*"perfect"*) si tiene el máximo número de nodos que admite un árbol de su altura[2]. Defina la función
perfecto : 'a bintree -> bool

También suele decirse que un árbol binario es o está ***"casi completo"*** (*"almost complete"* o *"nearly complete"*) si todos sus niveles, excepto quizá el último, están completos (es decir, tienen el máximo número de nodos que admite cada nivel de un árbol binario) y todos los nodos del último nivel están lo más a la izquierda posible[1]. Defina la función
casi_completo : 'a bintree -> bool.

Para probar estas definiciones puede cargar el archivo compilado del ejercicio anterior, en el compilador interactivo, con la directiva **`#load "bintree.cmo";;`** Para acceder más cómodamente a las definiciones de este archivo (que ahora constituyen el módulo Bintree) puede ejecutar **`Open Bintree;;`**

Escriba todas las definiciones de este ejercicio en un archivo ***"bintree_opt.ml"*** que debe compilar sin errores (siempre que antes haya compilado el módulo ***Bintree***) con la orden

```
ocamlc -c bintree_opt.mli bintree_opt.ml
```

Si no es capaz de definir alguna de las tres funciones, puede incluir definiciones triviales que devuelvan siempre los valores *false* o *[ ]* para que el archivo compile.

[2] https://en.wikipedia.org/wiki/Binary_tree#Types_of_binary_trees