Paradigmas de Programación

Práctica 8

1. Considere la siguiente implementación del algoritmo *quicksort*:

```
open List;;

let rec qsort1 ord = function
    [] -> []
  | h::t -> let after, before = partition (ord h) t in
            qsort1 ord before @ h :: qsort1 ord after;;
```

¿En qué casos no será bueno el rendimiento de esta implementación?

Para evitar problemas con la no terminalidad de `(@)` podemos hacer el siguiente cambio:

```
let rec qsort2 ord = function
    [] -> []
  | h::t -> let after, before = partition (ord h) t in
            rev_append (rev (qsort2 ord before)) (h :: qsort2 ord after);;
```

¿Tiene `qsort2` alguna ventaja sobre `qsort1`? ¿Permite `qsort2` ordenar listas que no podrían ordenarse con `qsort1`? En caso afirmativo, defina un valor `l1 : int list` que sea ejemplo de ello. En caso negativo, defina `l1 = []`.

¿Tiene `qsort2` alguna desventaja sobre `qsort1`? Compruebe si `qsort2` es más lento que `qsort1`. Si es así, explique por qué y estime la penalización, en porcentaje de tiempo usado, de `qsort2` respecto a `qsort1`.

Podría evitarse el uso de `rev` en `qsort2`, implementando a la vez la ordenación ascendente y descendente, de la siquiente manera:

```
let qsort3 ord l =
  let rec sort_asc = function
      [] -> []
    | h::t -> []   (* Cambiar aquí [] por lo que corresponda! *)
  and sort_des = function
      [] -> []
    | h::t -> []   (* Cambiar aquí [] por lo que corresponda! *)
  in sort_asc l;;
```

Complete la definición de `qsort3` siguiendo esa idea.

¿Puede apreciarse un cambio de rendimiento en tiempo de ejecución de `qsort3` respecto a `qsort2`? Si es así, estime en qué porcentaje. ¿Y respecto a `qsort1`?

2. Considere la siguiente implementación de la *ordenación por fusión*:

```
let rec split = function
    [] -> [], []
  | h::[] -> [h],[]
  | h1::h2::t -> let t1,t2 = split t in (h1::t1, h2::t2);;

let rec merge ord l1 l2 = match l1,l2 with
    [],l | l,[] -> l
```

```
  | h1::t1, h2::t2 -> if ord h1 h2 then h1::merge ord t1 l2
                      else h2::merge ord l1 t2;;

let rec msort1 ord l = match l with
    [] | _::[] -> l
  | _ -> let l1, l2 = split l in
         merge ord (msort1 ord l1) (msort1 ord l2);;
```

¿Puede provocar algún problema la no terminalidad de `split` o `merge`? En caso afirmativo, defina un valor `l2 : int list` que sea un ejemplo de ello. En caso negativo, defina `l2 = []`.

Defina de modo recursivo terminal funciones `split_t` y `merge_t` que cumplan el mismo cometido que `split` y `merge`, respectivamente. Realice una implementación, `msort2`, de la ordenación por fusión utilizando `split_t` y `merge_t`. Compare el rendimiento en tiempo de ejecución de `msort2` con el de `msort1` y con el de `qsort3`.

3. (Ejercicio opcional) Un potencial problema de las funciones `split` y `split_t` es que, cada vez que se aplican, generan dos nuevas listas. Si en vez de definir una función que ordene todos los elementos de una lista, definimos la que ordena los n primeros, podemos cambiar el `split` por una función que reparta sólo los n primeros elementos de una lista entre los $(n+1)/2$ primeros elementos de una lista y los $n/2$ primeros elementos de otra, de la siguiente manera:

```
let split_firsts n l =
  let n1 = (n+1) / 2 and n2 = n / 2 in
  let rec cut i l =
      if i > 0 then cut (i-1) (tl l)
      else l
  in (n1, l, n2, cut n1 l);;
```

Esta función no "crea" nuevas listas, ya que devuelve dos que ya están en memoria. Implemente como `msort3 : ('a -> 'a -> bool) -> 'a list -> 'a list`, la ordenación por fusión utilizando, para dividir las listas, esta nueva función `split_firsts`.

Aparte del supuesto ahorro de espacio, ¿puede apreciarse alguna mejora de `msort3`, en cuanto a rendimiento en tiempo de ejecución, respecto a `msort2`? En caso afirmativo, estime el porcentaje de mejora.

NOTA: Realice las implementaciones de los ejercicios 1, 2 y 3 en los ficheros `qsort.ml`, `msort2.ml` y `msort3.ml`, respectivamente . Como siempre, las respuestas "de palabra" que se piden en algunos de los apartados deben ser incluidas como comentarios en los ficheros `.ml`.