```
(* $Id: mergesort.ml,v 361.4 2014-11-17 14:07:55-08 - - $ *)

(*
* Merge sort example.
* First, we define it as three separate list processing functions.
* Note that neither merge nor split are tail recursive.
*)

let rec merge (<?) list1 list2 = match (list1, list2) with
    | ([], list2) -> list2
    | (list1, []) -> list1
    | ((car1::cdr1 as list1), (car2::cdr2 as list2))
                  -> if car1 <? car1
                     then car1 :: merge (<?) cdr1 list2
                     else car2 :: merge (<?) list1 cdr2
;;

let rec split list = match list with
    | []              -> ([], [])
    | [_] as list'    -> (list', [])
    | car::cadr::cddr -> let (list1, list2) = split cddr
                         in  (car::list1, cadr::list2)
;;

let rec msort (<?) list = match list with
    | []             -> []
    | _::[] as list' -> list'
    | list           -> let (list1, list2) = split list
                        in merge (<?) (msort (<?) list1)
                                      (msort (<?) list2)
;;

let sort1 : int list -> int list = msort (<);;

```

```

(*
* An alternate definition using nested functions and fewer
* parameters internally.  However, merge′ and split′ are not
* tail recursive.
*)

let mergesort (<?) list =
    let rec merge′ list1 list2 = match (list1, list2) with
        | ([], list2) -> list2
        | (list1, []) -> list1
        | ((car1::cdr1 as list1), (car2::cdr2 as list2))
                       -> if (<?) car1 car2
                          then car1 :: merge′ cdr1 list2
                          else car2 :: merge′ list1 cdr2
    and split′ list = match list with
        | []               -> ([], [])
        | [_] as list′     -> (list′, [])
        | car::cadr::cddr -> let (list1, list2) = split′ cddr
                              in  (car::list1, cadr::list2)
    and sort′ list = match list with
        | []              -> []
        | _::[] as list′ -> list′
        | list            -> let (list1, list2) = split′ list
                             in merge′ (sort′ list1) (sort′ list2)
    in  sort′ list
;;

let sort2 : int list -> int list = mergesort (<);;

```

```
bash-1$ ocaml
        OCaml version 4.02.1

# #use "mergesort.ml";;
val merge : ('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list = <fun>
val split : 'a list -> 'a list * 'a list = <fun>
val msort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
val sort1 : int list -> int list = <fun>
val mergesort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
val sort2 : int list -> int list = <fun>
# mergesort (>) [3;4;33;10;-5;9];;
- : int list = [33; 10; 9; 4; 3; -5]
#
bash-2$ exit
```