```
 1: (* $Id: mergesort.ml,v 361.4 2014-11-17 14:07:55-08 - - $ *)
 2:
 3: (*
 4: * Merge sort example.
 5: * First, we define it as three separate list processing functions.
 6: * Note that neither merge nor split are tail recursive.
 7: *)
 8:
 9: let rec merge (<?) list1 list2 = match (list1, list2) with
10:     | ([], list2) -> list2
11:     | (list1, []) -> list1
12:     | ((car1::cdr1 as list1), (car2::cdr2 as list2))
13:                   -> if car1 <? car1
14:                      then car1 :: merge (<?) cdr1 list2
15:                      else car2 :: merge (<?) list1 cdr2
16: ;;
17:
18: let rec split list = match list with
19:     | []               -> ([], [])
20:     | [_] as list'     -> (list', [])
21:     | car::cadr::cddr -> let (list1, list2) = split cddr
22:                          in  (car::list1, cadr::list2)
23: ;;
24:
25: let rec msort (<?) list = match list with
26:     | []              -> []
27:     | _::[] as list' -> list'
28:     | list            -> let (list1, list2) = split list
29:                          in merge (<?) (msort (<?) list1)
30:                                        (msort (<?) list2)
31: ;;
32:
33: let sort1 : int list -> int list = msort (<);;
34:
```

```
35:
36: (*
37: * An alternate definition using nested functions and fewer
38: * parameters internally.  However, merge' and split' are not
39: * tail recursive.
40: *)
41:
42: let mergesort (<?) list =
43:     let rec merge' list1 list2 = match (list1, list2) with
44:         | ([], list2) -> list2
45:         | (list1, []) -> list1
46:         | ((car1::cdr1 as list1), (car2::cdr2 as list2))
47:                       -> if (<?) car1 car2
48:                          then car1 :: merge' cdr1 list2
49:                          else car2 :: merge' list1 cdr2
50:     and split' list = match list with
51:         | []              -> ([], [])
52:         | [_] as list'    -> (list', [])
53:         | car::cadr::cddr -> let (list1, list2) = split' cddr
54:                              in  (car::list1, cadr::list2)
55:     and sort' list = match list with
56:         | []             -> []
57:         | _::[] as list' -> list'
58:         | list           -> let (list1, list2) = split' list
59:                             in merge' (sort' list1) (sort' list2)
60:     in  sort' list
61: ;;
62:
63: let sort2 : int list -> int list = mergesort (<);;
64:
```

```
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
```