Homework 5

October 9, 2018

1 Higher order ML Functions

For this homework, you use **let** expressions and **pattern matching** to solve
the problems. Your solution should **not** use top-level helper functions (i.e. any
helper functions should be defined locally in a **let** expression), should avoid
redundant computation, and should have the correct types.

1. Write a function `merge_sort: ('a * 'a -> bool) -> 'a list -> 'a list`
   that takes a comparison function and then a list, and return a sorted list
   in the order specified by the comparison function.

   For example, `merge_sort (op >) [0, 5, 1, ~4, 9, 11]` evaluates to
   `[11,9,5,1,0,~4]`.

2. Write a function `selection_sort: ('a * 'a -> bool) -> 'a list -> 'a list`
   that takes a comparison function and then a list, and return a sorted list
   in the order specified by the comparison function.

   Selection sort orders a list from the left to right by selecting the least
   element from the unsorted portion of the list and adds it to the partially
   sorted list. Note that the least element is not necessarily the smallest
   or largest element – it depends on the comparison function passed to
   `selection_sort`.

   For example, `selection_sort (op >) [0, 5, 1, ~4, 9, 11]` evaluates
   to `[11,9,5,1,0,~4]`. The computation has the following steps:

   ```
       [0, 5, 1, ~4, 9, 11] select 11
   ->  11::[0, 5, 1, ~4, 9] select 9
   ->  11::9::[0, 5, 1, ~4] select 5
   ->  11::9::5::[0, 1, ~4] select 1
   ->  11::9::5::1::[0, ~4] select 0
   ->  11::9::5::1::0::[~4] select ~4
   ->  11::9::5::1::0::~4::nil
   ```

   You should implement a helper function `select: 'a list -> 'a list`
   as an inner function to `selection_sort`. The `select` function takes a

list and returns the list with the least element in front. For example, `select [0,5,1,~4,9,11]` should return `[11,0,5,1,~4,9]` given the ordering function `op >`.

3. Write a function `insertion_sort: ('a * 'a -> bool) -> 'a list -> 'a list` that takes a comparison function and then a list, and return a sorted list in the order specified by the comparison function.

   Insertion sort orders a list from left to right by inserting an element from unsorted portion of the list into the correct position in the sorted portion of the list.

   For example, `insertion_sort (op >) [0, 5, 1, ~4, 9, 11]` evaluates to `[11,9,5,1,0,~4]`. The computation has the following steps:

   ```
       nil [0, 5, 1, ~4, 9, 11] insert 0
   -> [0] [5, 1, ~4, 9, 11] insert 5
   -> [5, 0] [1, ~4, 9, 11] insert 1
   -> [5, 1, 0] [~4, 9, 11] insert ~4
   -> [5, 1, 0, ~4] [9, 11] insert 9
   -> [9, 5, 1, 0, ~4] [11] insert 11
   -> [11, 9, 5, 1, 0, ~4] nil
   ```

   You should implement two helper functions as inner functions. The first function `insert:'a list * 'a -> 'a list` takes a sorted list `l` and an element `x` and insert `x` into the right position of `l` so that the list remains sorted after insertion. For example, `insert ([5,1,0,~4], 9)` should evaluates to `[9,5,1,0,~4]`. Again the order depends on the comparision function passed to `insertion_sort`. The second helper function `sort:'a list * 'a list -> 'a list` takes a sorted list `sofar` and an unsorted list `l` and inserts the elements of `l` into `sofar`.

4. Write a function `quicksort: (int * int -> bool) -> int list -> int list` that takes a comparison function (i.e. either `op >` or `op <`), then a list of integers, and return a sorted list of integers in the order specified by the comparison function.

   requirement:

   (a) You should implement quicksort algorithm.

   (b) You should have three base cases: empty list, list of one, and list of two elements.

   (c) The pivot is the average of the first, middle, and the last element of the list. You should define an inner function `get: int list * int -> int` that takes a list and return an element of the list specified by an index. For example, `get(lst, 1)` returns the first element of `lst`, `get(lst, (length lst + 1) div 2)` returns the middle elmeent, and `get(lst, length lst)` returns the last element.

(d) The split function should split the list into three sublists: (`lower, middle, upper`), where the `middle` list are elements that are equal to the pivot. The `lower` and `upper` lists are elements that are either less or greater than the pivot (depending on the comparison function).

(e) Note that you place the `middle` list between the sorted `lower` and `upper` list.

2 Submission

Please write your solution in a text file by the name of `hwk5.sml` and submit it to the dropbox.