Solution notes

Foundations of Computer Science 2005 – Paper 1 Question 5 (LCP)

(*a*) *(Lecture 6, sorting.)* The algorithm is recursive. If the list has fewer than two elements then nothing needs to be done. Otherwise, one of the items to be sorted is chosen as the *pivot*. The remaining items are partitioned into two groups: those that are less than the pivot (in the order being used for sorting) and those that are at least as great as the pivot. Sort both groups recursively (since at least one item has been removed, both groups contain fewer items than the input) and place those from the first group before those of the second group.

If the pivot is always chosen to be the first element (as in the course notes), then the list $[8, 3, 6, 12, 2, 9, 20, 1, 5, 0, 7, 13, 4, 11, 10]$ is sorted as follows.

The initial list is $[8, 3, 6, 12, 2, 9, 20, 1, 5, 0, 7, 13, 4, 11, 10]$.

The pivot is 8 and the first partitioning yields

$$[3, 6, 2, 1, 5, 0, 7, 4]@[8]@[12, 9, 20, 13, 11, 10].$$

Recursively partitioning the left and right sides yields

$$([2, 1, 0]@[3]@[6, 5, 7, 4])@[8]@([9, 11, 10]@[12]@[20, 13]).$$

Continuing, we get

$$(([0]@[1]@[2])@[3]@([5, 4]@[6]@[7]))@[8]@(([9]@[10]@[11])@[12]@([13]@[20])).$$

The final sorted list is $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 20]$.

(*b*) *(Lectures 1 and 2, elementary ML.)*

```
fun median (x,y,z) : int =
  if x<y then
    if z<x then x else if y<z then y
    else z
  else (*y<=x*)
    if z<y then y else if x<z then x
    else z;
```

(*c*) *(Lectures 4 and 5, lists, and lecture 6, sorting.)*

We use the new median function. We also modify the partition function to keep track of occurrences of the pivot element. This ensures correct operation even if the pivot appears multiple times.

```
fun nth (x::_,0) = x
  | nth (x::xs, k) =
      if k>0 then nth(xs,k-1) else raise Match;

fun mquick [] = []
  | mquick [x] = [x]
  | mquick bs =
      let val n = length bs
          val a = median (hd bs, nth(bs, n div 2), nth(bs, n-1))
          fun partition (left,mid,right,[]) =
                (mquick left) @ (mid @ mquick right)
            | partition (left,mid,right, x::xs) =
                if x=a then partition (left, x::mid, right, xs)
                        (*keep occurrences of the pivot element*)
                else if x<a then partition (x::left, mid, right, xs)
                            else partition (left, mid, x::right, xs)
      in  partition([],[],[],bs)  end;
```

(*d*) *Lecture 3, O-notation.* The average case execution time is the same as for standard Quicksort, namely $O(n \log n)$ comparisons where n is the number of elements to be sorted. For both algorithms, it is because the partition phase will (in the average case) divide the input into two roughly equal parts, so the depth of recursion is $O(\log n)$. Also, the amount of work done at each level of recursion is, as before, linear in n; the initial call to `median` adds a constant number of comparisons, at most three, compared with standard Quicksort.