p1 q5
LCP

**Foundations of Computer Science (Paper 1)** 2000

State the time complexity of the of the *lookup* and *update* operations for each of the following:

1. association lists
2. binary search trees
3. functional arrays (implemented as binary trees)

Use $O$-notation and include both the average-case and worst-case complexity. [6 marks]

You are provided with the ML code for binary search trees, including the *lookup* and *update* operations. Use these operations to code a sorting function that works by repeatedly inserting elements of a list into a binary search tree, then converting the final binary search tree back into a list. [4 marks]

Consider the following methods of sorting a list:

1. Locate the smallest element of the input. The output is this element followed by the result of recursively sorting the remaining elements.
2. Take the first 16 elements of the input and sort them using special hardware. Sort the remaining elements recursively. The output is the result of merging the two sorted lists.
3. Take the first 20% of the input elements. Sort them and the remaining 80% recursively. The output is the result of merging the two sorted lists.

For each of these methods, state with justification the worst-case complexity in terms of the number of comparisons. [10 marks]

**Solution notes**

In each case, $n$ is the number of items in the data structure.

Lookup for association lists takes $O(n)$ in the average and worst case because it works by linear search. The update operation given in the notes always takes constant time. (One can code an update operation that takes linear time: it would replace any existing element of the association list having the same search key.)

Lookup and update for binary search trees takes $O(\log n)$ time in the average case (a balanced tree) and $O(n)$ time in the worst case (unbalanced tree).

Lookup and update for binary search trees takes $O(\log n)$ time in the average and worst cases.

The sorting function can be expressed in five lines of ML. (We don't even need lookup.)

```
fun maketree ([],t) = t
  | maketree (x::xs, t) = maketree (xs, insert (t, x, ()));
fun listoftree (Lf, vs) = vs
```

```
  | listoftree (Br((a,x),t1,t2), vs) =
      listoftree (t1, a::listoftree (t2, vs));
```

The first part of the code simply inserts all the list elements into the binary search tree, with a dummy item attached (since there is no actual data to associate with the keys). The second part of the code does an inorder traversal, omitting the dummy data.

Regarding the sorting functions. The first (called selection sort) takes quadratic time in the worst case: finding the minimum element takes $n-1$ comparisons, so the total number is

$$(n-1)+\cdots+1 = O(n^2).$$

The second has quadratic complexity because merging two lists of lengths $m$ and $n$ requires $m+n$ comparisons in the worst case. Having special hardware does not reduce the number of comparisons needed for the merging. We get

$$n+(n-16)+(n-32)+\cdots$$

which is still quadratic: approximately $n^2/16-n$. The third takes $O(n\log n)$ complexity because the size of the input is reduced by a fixed ratio in each recursive call. The maximum depth of recursion is $\log_{5/4} n$. Considering all the merges at any recursion depth as a whole, the total number of comparisons is bounded by $n\log_{5/4} n$ which is $O(n\log n)$.