— *Solution notes* —

COMPUTER SCIENCE TRIPOS Part IA – 2004 – Paper 1

5 Foundations of Computer Science (ACN)

This question has been translated from Standard ML to OCaml

The following OCaml type can be viewed as defining a lazy or infinite sort of tree where each node in the tree holds an integer:

```
type tr = N of int * unit -> tr * unit -> tr
```

(*a*) Write a function called `ndeep` such that if `n` is an integer and `z` is a tree (i.e. of type `tr`) the call `ndeep n z` will return an ordinary list of all the 2^n integers at depth exactly `n` in the tree. Note that if `n` $= 0$ it will return a list of length 1, being just the top integer in the tree. Comment on its efficiency. [8 marks]

Answer:

```
let rec ndeep n (N(v, l, r)) =
  if n = 0 then
    [v]
  else
    ndeep (n - 1) (l ()) @ ndeep (n - 1) (r ())
```

The append makes this slower than would be perfect, and experts can do the usual conversion to avoid that.

(*b*) You are given a `tr`, and told that it contains arbitrarily large values at least somewhere in it. You want to find a value from it that is bigger than 100 (but if there are many big values it does not matter which one is returned). Because the tree is infinite you cannot use simple depth-first search: you decide to use iterative deepening. Thus you first check all integers at depth 1, then at depth 2, depth 3,... and return when you first find a value that is greater than 100.

Use exception handling to return the large value when you find it. Present and explain code that searches the lazy tree. [12 marks]

Answer:

```
exception Found of int

let rec throwifin = function
  | [] -> ()
  | x::xs ->
     if x > 100 then
       raise (Found x)
     else
       throwifin xs

let search z =
  let rec depth n =
    throwifin (ndeep n z);
```

```
     depth (n + 1)
  in
    try
      depth 1;
      0 (* Unreachable *)
    with Found x -> x
```