— Solution notes —

COMPUTER SCIENCE TRIPOS Part IA – 2013 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

variants, pattern-matching

(*a*) Write brief notes on OCaml variants and pattern-matching in function declarations. [6 marks]

Answer: Solutions should include examples of variant type declarations and mention the concept of a constructor. Examples of pattern-matching should be non-trivial, with nested constructors and (preferably) overlapping patterns.

programming, binary trees

(*b*) A binary tree is either a *leaf* (containing no information) or is a *branch* containing a label and two subtrees (called the *left* and *right* subtrees). Write OCaml code for a function that takes a label and two lists of trees, returning all trees that consist of a branch with the given label, with the left subtree taken from the first list of trees and the right subtree taken from the second list of trees. [6 marks]

Answer: The variant type declaration is not required as part of the answer, but sets the stage. Students are unlikely to know about `List.concat`, but it can be coded in two lines with the help of @ (append).

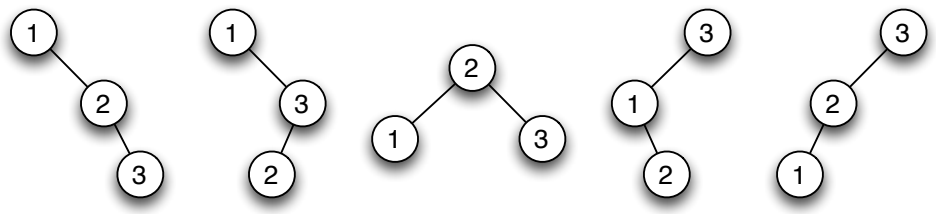
```
type 'a tree = Lf
             | Br of 'a * 'a tree * 'a tree

let make_trees v t1 =
  List.map (fun t2 -> Br (v, t1, t2))

let make_trees2 v t1s t2s =
  List.concat (List.map (fun t1 -> make_trees v t1 t2s) t1s)
```

programming, binary trees

(*c*) Write OCaml code for a function that, given a list of distinct values, returns a list of all possible binary trees whose labels, enumerated in inorder, match that list. For example, given the list `[1; 2; 3]` your function should return (in any order) the following list of trees:



[8 marks]

Answer:

```
let rec anti l1 = function
```

```
  | [] -> []
  | v::l2 ->
      make_trees2 v (anti_inorder (List.rev l1)) (anti_inorder l2) @
      anti (v::l1) l2
and anti_inorder = function
  | [] -> [Lf]
  | xs = anti [] xs
```

Note that the question refers to binary trees, not to binary *search* trees, and it does not impose an ordering constraint on the labels of these trees.

All OCaml code must be explained clearly and should be free of needless complexity.