Foundations of Computer Science 2001 pl q 5 LCP

Model Answer

The 'brief notes' are all bookwork.

Iterative functions are discussed from slide 207 onwards. Answers should mention the use of an accumulating orgument, the typical reduction in space from $O(n)$ to $O(1)$ and situations when re-coding is not worth the effort.

Breadth-first search is the subject of Lecture 9. Answers should note its purpose (to find the shortest path to a solution) and drawbacks (needs lots of space). A queue rather than a stack is necessary. A really good answer will also mention the alternative of depth-first iterative deepening.

Exceptions are introduced in slide 705. Answers should mention how exceptions are declared, raised and handled, showing that the candidate understands exception propagation.

The logic behind this problem resembles that of the 'making change' example that was discussed so thoroughly in the notes. In the recursive calls, we choose whether or not to include the head of the list in the sum.

```
fun add x y = x + y : int;

fun sums [] = []
  | sums (x::xs) =
      let val s1 = sums xs
          val s2 = x :: map (add x) s1
      in s1 @ s2 end;
```

The required modification is to replace the 'append' above by a duplicate-removing 'merge' function. The output will have no repetitions even if the input does.

```
fun merge [] ys = ys
  | merge xs [] = xs
  | merge (x::xs) (y::ys) =
      if x=y then merge xs (y::ys)
      else if x<y then x :: merge xs (y::ys)
      else y :: merge (x::xs) ys;

fun sums2 [] = []
  | sums2 (x::xs) =
      let val s1 = sums2 xs
          val s2 = x :: map (add x) s1
      in merge s1 s2 end;
```