1995 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Describe and compare the call-by-value, call-by-name, and call-by-need evaluation strategies for functional programming languages.

The OCaml function `butlast` removes the last element from a non-empty list:

```
exception Butlast
let rec butlast = function
  | [] -> raise Butlast
  | [_] -> []
  | (x::xs) -> x::(butlast xs)
```

Show how the evaluation of `butlast [[1; 2]; []; [3]; [4; 5]]` proceeds in OCaml.

Write an iterative version of `butlast` (i.e. one in which the recursive function calls are tail recursive). You may assume the existence of the append (`@`) function.

State with justification the time complexity of your function.

An OCaml variant type of lazy lists can be defined by:

```
type 'a lazy_list = Nil | Cons of unit -> 'a * 'a lazy_list
```

An 'infinite' list of increasing integers can be generated by the function `infinite` below:

```
let rec infinite n = Cons (fun () -> (n, infinite (n + 1)))
```

Write a version of `butlast` for lazy lists which terminates when applied to an infinite lazy list such as `infinite 0`.

Can an iterative version of this function be written that still terminates on infinite lazy lists? Explain your reasoning.

[20 marks]