This question has been translated from Standard ML to OCaml

(a) Write brief notes on the queue data structure and how it can be implemented efficiently in OCaml. In a precise sense, what is the cost of the main queue operations? (It is not required to present OCaml code.) [6 marks]

(b) Run-length encoding is a way of compressing a list in which certain elements are repeated many times in a row. For example, a list of the form \([a; a; a; b; a; a]\) is encoded as \([(3, a); (1, b); (2, a)]\). Write a polymorphic function \texttt{rl\_encode} to perform this encoding. What is the type of \texttt{rl\_encode}? [6 marks]

(c) The simple task of testing whether two lists are equal can be generalised to allow a certain number of errors. We consider three forms of error:

- \textit{element mismatch}, as in \([1; 2; 3]\) versus \([1; 9; 3]\) or \([1; 2; 3]\) versus \([0; 2; 3]\)
- \textit{left deletion}, as in \([1; 3]\) versus \([1; 2; 3]\) or \([1; 2]\) versus \([1; 2; 3]\)
- \textit{right deletion}, as in \([1; 2; 3]\) versus \([1; 3]\) or \([1; 2; 3]\) versus \([1; 2]\)

Write a function \texttt{genEquals \_n \_xs \_ys} that returns \texttt{true} if the two lists \_xs and \_ys are equal with no more than \_n errors, and otherwise \texttt{false}. You may assume that \_n is a non-negative integer. [8 marks]

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