1. (12 pts) OCaml programming

Solve the following OCaml programming problems. You may use List.rev (reverses a list), the (curried) map and fold functions, and functions from Pervasives (except fst & snd), but no other OCaml library functions. You are not allowed to use imperative OCaml features such as int ref. Your solution must run in O(n) time for input of length n for full credit.

a. (6 pts) Using either map or fold and an anonymous function, write a function split that when applied to an associative list lst (i.e., list of (key, value) pairs), returns a pair of lists (keys, values), preserving their relative order in lst.

Example: split [ ] = ([ ], [ ])
split [ (1,2); (5,6); (3,4) ] = ([1; 5; 3], [2; 6; 4])
split [ (“x”,2.1); (“y”,4.3) ] = ([“x”; “y”], [2.1; 4.3])

let split lst =
    let (x,y) = fold (fun (a,b) (k,v) -> (k::a,v::b)) ([],[]) lst in
    ((List.rev x),(List.rev y))

OR

let split lst = fold (fun (a,b) (k,v) -> (k::a,v::b)) ([],[]) (List.rev lst)

b. (6 pts) Consider the following OCaml user-variant type intList implementing a linked list of ints.

type intList =
    | Empty
    | Node of int * intList

i. (2 pts) Write an OCaml expression with type intList that is equivalent to the list [1; 2].

(Node (1, Node (2, Empty) ) )

ii. (4 pts) Write a function freq of type (int -> intList -> int) that takes an integer n and an intList, and returns the number of times n occurs in the intList.

let rec freq n t = match t with
    | Empty -> 0
    | Node (v,x) -> (freq n x)+(if (n=v) then 1 else 0)
2. (8 pts) Context free grammars & parsing
   Consider the following grammar (S = start symbol and terminals = x y z):
   
   S → xA | yz
   A → SA | ε

   a. (2 pts) Draw a parse tree for the string “xyz”.

   \[
   \begin{array}{c}
   S \\
   \quad \begin{array}{c}
   x \\
   \quad \begin{array}{c}
   A \\
   \quad \begin{array}{c}
   S \\
   \quad \begin{array}{c}
   A \\
   \quad \begin{array}{c}
   y \\
   \quad \begin{array}{c}
   z \quad \varepsilon
   \end{array}
   \end{array}
   \end{array}
   \end{array}
   \end{array}
   \end{array}
   \end{array}
   \]

   b. (3 pts) Calculate FIRST sets for S and A.

   \[
   \text{FIRST}(S) = \{ \text{x, y} \} \quad \text{FIRST}(A) = \{ \text{x, y, ε} \}
   \]

   c. (3 pts) Using pseudocode, write only the parse_A function found in a recursive descent parser for the grammar. You may assume the functions parse_S already exists.

   You may use the following utilities:

   | lookahead | Variable holding next terminal |
   | match ( x ) | Function to match next terminal to x |
   | error ( ) | Reports parse error for input |

   ```
   \[
   \text{parse}_A() \{
   \text{if ((lookahead == x) || (lookahead == y))} \quad \text{\text{A → SA}}
   \quad \text{parse}_S() ;
   \quad \text{parse}_A() ;
   \text{else} \quad \text{\text{A → ε}}
   \quad ;
   \}
   \]
   ```