TP n°7

Structures de données fonctionnelles efficaces

Exercice 1 Considérons le code suivant :

```
(* reverse naive *)
let rec rev = function
  [] -> []
  | a::r -> (rev r)@[a];;

(* reverse efficace *)
let rec rev_append = function
  ([],l) -> l
  | (a :: l,l') -> rev_append (l, (a :: l'));;
let rev' l = rev_append (l,[]);;
```

Prouvez par induction que $\text{rev}'\ l = \text{rev}\ l$, pour toute liste l .

Exercice 2 On cherche à implémenter la signature DEQUE donnée dans les transparents du cours, en gardant une complexité linéaire en temps amorti.

```
module type DEQUE = sig
  type 'a queue

  val empty : 'a queue
  val is_empty : 'a queue -> bool

  (* insert, inspect, and remove the front element *)
  val cons : 'a -> 'a queue -> 'a queue
  val removefirst : 'a queue -> 'a * 'a queue (* raises Empty if queue is empty *)

  (* insert, inspect, and remove the rear element *)
  val snoc : 'a queue -> 'a -> 'a queue
  val removelast : 'a queue -> 'a * 'a queue (* raises Empty if queue is empty *)
end
```

Analysez le coût amorti de votre algorithme.