```scheme
;;;;
;;;; Corrigé.stk        -- Corrigés de la feuille 3
;;;;
;;;;           Author: Erick Gallesio [eg@unice.fr]
;;;;    Creation date: 28-Feb-2000 09:06 (eg)
;;;; Last file update: 28-May-2003 11:36 (eg)
;;;;



;==============================================================================
;                                         EXO 1
;==============================================================================

(define (hanoi n)

  (define (move  from to)
    (format #t "Déplacer un disque de ~S vers ~S~%" from to)
    ;; 1 déplacement élementaire en résultat
    1)

  (define (h-aux n from to with)
    (if (= n 1)
        (move from to)
        (+ (h-aux (- n 1) from with to)
           (move from to)
           (h-aux (- n 1) with to from))))

  ;; Le corps de la fonction commence ici
  (h-aux n 'a 'b 'c))

#|
(hanoi 3)
|#

;==============================================================================
;                                         EXO 2
;==============================================================================

;; Première implémentation. Un noeud de l'arbre est représenté par
;; le triplet (valeur sag sad). L'arbre vide est représenté par '()


(define vide   (lambda () '()))
(define vide?  null?)
(define arbre  list)
(define racine car)
(define gauche cadr)
(define droite caddr)

(define ajouter
  (lambda (v a)
    (if (vide? a)
        (arbre v (vide) (vide))
        (let ((r (racine a)))
          (if (< v r)
              (arbre r
                     (ajouter v (gauche a))
                     (droite a))
              (arbre r
                     (gauche a)
                     (ajouter v (droite a))))))))
```



```scheme
(define appartient?
  (lambda (v a)
    (cond
     ((vide? a)           #f)
     ((= v (racine a))    #t)
     (else          (let ((sa (if (< v (racine a)) gauche droite)))
                      (appartient? v (ou a)))))))

; Pour effectuer le tri d'une liste d'entiers, il suffit de la convertir
; en arbre (trié), puis de convertir l'arbre en liste

(define liste->arbre
  (lambda (l)
    (if (null? l)
        (vide)
        (ajouter (car l)
                 (liste->arbre (cdr l))))))

(define arbre->liste
  (lambda (a)
    (if (vide? a)
        ()
        (append (arbre->liste (gauche a))
                (list (racine a))
                (arbre->liste (droite a))))))

(define trier
  (lambda (l)
    (arbre->liste (liste->arbre l))))


;; Autre implémentation des arbres: Un arbre est maintenant une
;; représenté avec 2 cons: (valeur . (sag . sad)). L'arbre vide est
;; représenté par le symbole 'Gloob (et oui pourquoi pas? En fait
;; n'importe quelle valeur fait l'affaire)

(define vide  (lambda () 'Gloob))
(define vide? (lambda (a) (equal? a 'Gloob)))
(define arbre (lambda (a g d) (cons a (cons g d))))
(define racine car)
(define gauche cadr)
(define droite cddr)


;; Bien-sûr, le reste du programme reste inchangé.


;; Pour créer un arvre de chaînes de caractères, il suffit de remplacer
;; le < par string<?.
;; Dans le cas général, il suffit de passer ce prédicat lors de la construction
;; de l'arbre (i.e. à ajouter-arbre)

;; ==> problème pour appartient. Attendre de voir les fermetures.
```