Interacción con el sistema Acl2

Consideremos las siguientes definiciones Lisp:

- Algoritmo de inserción ordenada

```
(defun inserta (a x)
  (if (consp x)
      (if (<= a (car x))
          (cons a x)
          (cons (car x) (inserta a (cdr x))))
      (list a)))

(defun isort (x)
  (if (consp x)
      (inserta (car x) (isort (cdr x)))
      nil))
```

- Algoritmo de ordenación por mezclas

```
(defun trozo-i (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) nil)
        (t (cons (car l) (trozo-i (cddr l))))))

(defun trozo-p (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) l)
        (t (cons (cadr l) (trozo-p (cddr l))))))

(defun mezcla (l1 l2)
  (cond ((endp l1) l2)
        ((endp l2) l1)
        ((< (car l1) (car l2))
         (cons (car l1) (mezcla (cdr l1) l2)))
        (t (cons (car l2) (mezcla l1 (cdr l2))))))

(defun msort (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) (list (car l)))
        (t (mezcla (msort (trozo-i l))
                   (msort (trozo-p l))))))
```

- Algoritmo de ordenación rápida

```
(defun qsort-< (x l)
  (cond ((endp l) nil)
        ((< (car l) x)
         (cons (car l) (qsort-< x (cdr l))))
        (t (qsort-< x (cdr l)))))

(defun qsort-no-< (x l)
  (cond ((endp l) nil)
        ((< (car l) x)
         (qsort-no-< x (cdr l)))
        (t (cons (car l) (qsort-no-< x (cdr l))))))

(defun qsort (l)
  (if (endp l)
      nil
      (append (qsort (qsort-< (car l) (cdr l)))
              (list (car l))
              (qsort (qsort-no-< (car l) (cdr l))))))
```

Demostrar en ACL2las siguientes propiedades de dichos algoritmos de ordenación:

- El resultado es una lista ordenada de menor a mayor

```
(defun ordenada (l)
  (cond ((endp l) (equal l nil))
        (t (or (equal (cdr l) nil)
               (and (<= (first l) (second l))
                    (ordenada (cdr l)))))))
```

- Propiedades a demostrar
  - `(ordenada (isort l))`
  - `(ordenada (msort l))`
  - `(ordenada (qsort l))`
- El resultado tiene los mismos elementos que el original (permutación)

```
(defun borra-uno (x l)
  (cond ((endp l) l)
        ((equal x (car l)) (cdr l))
        (t (cons (car l) (borra-uno x (cdr l))))))

(defun perm (l1 l2)
  (cond ((endp l1) (endp l2))
        ((member (car l1) l2)
         (perm (cdr l1) (borra-uno (car l1) l2)))
        (t nil)))
```

- Propiedades a demostrar
  - `(perm (isort l) l)`
  - `(perm (msort l) l)`
  - `(perm (qsort l) l)`

Para demostrar estas propiedades se sugiere realizar la siguiente colección de ejercicios:

Ejercicio 1
La función `perm` es reflexiva

```
(defthm perm-reflexiva
  (perm l l))
```

Ejercicio 2
La función `perm` es transitiva

```
(defthm perm-transitiva
  (implies (and (perm l1 l2)
                (perm l2 l3))
           (perm l1 l3))))
```

Ejercicio 3
La función `perm` es simétrica

```
(defthm perm-simetrica
  (implies (perm l1 l2)
           (perm l2 l1)))
```

Ejercicio 4
La función `perm` es una relación de equivalencia

```
(defequiv perm)
```

Ejercicio 5
Congruencia de `borra-uno` con respecto a `perm`

```
(defthm perm-borra-uno-borra-uno-congruencia
  (implies (perm l1 l2)
           (perm (borra-uno x l1) (borra-uno x l2)))
  :rule-classes :congruence)
```

Ejercicio 6
Congruencias de `append` con respecto a `perm`

```
(defthm append-perm-1
  (implies (perm l1 l2)
           (perm (append l1 l3) (append l2 l3)))
  :rule-classes :congruence)

(defthm append-perm-2
  (implies (perm l1 l2)
           (perm (append l3 l1) (append l3 l2)))
  :rule-classes :congruence)
```

Ejercicio 7
Propiedades del algoritmo de inserción ordenada

```
(defthm ordenada-isort
  (ordenada (isort l)))

(defthm perm-isort
  (perm (isort l) l))
```

Ejercicio 8

Deducir la medida necesaria para que el sistema admita las definiciones del algoritmo de ordenación por mezclas

Ejercicio 9

Propiedades del algoritmo de ordenación por mezclas

```
(defthm ordenada-msort
  (ordenada (msort l)))

(defthm perm-msort
  (perm (msort l) l))
```

Ejercicio 10

Propiedades del algoritmo de ordenación rápida

```
(defthm perm-qsort
  (perm (qsort l) l))

(defthm ordenada-qsort
  (ordenada (qsort l)))
```

Para demostrar que el algoritmo de ordenación rápida devuelve una lista con los mismos elementos que la original, consideraremos la siguiente propiedad: si dos listas `l1` y `l2` están ordenadas de menor a mayor y `x` es un elemento mayor o igual que todos los elementos de `l1` y menor o igual que todos los elementos de `l2`, entonces `(append l1 (cons x l2))` es una lista ordenada de menor a mayor.