1. Define recursivamente una funci´on toma que reciba un nu´mero n y una lista l y devuelva la lista que contiene los primeros n elementos de l.

```
toma :: (Num i, Ord i) => i -> [a] -> [a]
toma n _
        | n <= 0  = []
toma _ []     = []
toma n (x:xs) = x : toma (n-1) xs
```

Donde “n <= 0” y una lista vacia “[]” son nuestros casos base, que regresaran una lista vacia
Y “toma n (x:xs) = x : toma (n-1) xs” es nuestro paso recursivo, que va tomando nuestro primer elemento de la lista y lo va concatenando en otra nueva hasta que lleguemos a un caso base.

2. Define recursivamente una funci´on quita que reciba un nu´mero n y una lista l y devuelva la lista que quita los primeros n elementos de l.

```
quita :: (Num i, Ord i) => i -> [a] -> [a]
quita n (x:xs)
        | n <= 0 = (x:xs)
        | otherwise = quita (n-1) (xs)
```

Donde “n <= 0” y n (x:xs) nos restringe a que la lista debe tener mas de dos elementos.
“otherwise = quita (n-1) (xs)” es nuestro paso recursivo, que va tomando el primer elemento de la lista y lo va concatenando en otra nueva hasta que lleguemos a un caso base. Pero regresa el cuerpo de la lista principal.

3. Define recursivamente el predicado elem cuya especificaci´on formal es la siguiente: elem x l si y solo si x es elemento de la lista l

```
eleem :: (Eq a) => a -> [a] -> Bool
eleem a [] = False
eleem a (x:xs)
        | a == x = True
        | otherwise = a `eleem` xs
```

Donde “a [] = False” es nuestro caso base ya que si la lista es vacia quiere decir que no tiene elementos y mucho menos el que se pregunta.
Nuestro paso recursivo funciona de la siguiente manera: Toma la cabeza de la lista y lo compara con el numero que se ingreso, si es igual regresa un true y termina la ejecucion, del contrario ira tomando la siguiente cabeza hasta llegar a un true, sino llegamos al caso base de false.

4. Defina una funcı´on recursiva replica que toma un elemento a y un nu´mero n como entrada, y devuelve la lista que contiene n veces a a como elemento, es decir, que cumpla la siguiente especificacı´on:

   replica (a, n) = [a,a,a,...,a]

```
replica :: (Num n, Ord n) => (a,n) -> [a]
replica (x,y)
        | y <= 0   = []
        | otherwise = x:replica (x, y-1)
```

Donde “y <= 0” es nuestro caso base que nos dice que si la lista que ingresamos es vacia nos regresa otra lista vacia.
El paso recursivo es “x:replica (x, y-1)” que trabaja de la siguiente forma: toma el primer numero de la pareja, y lo irá concatenando tantas veces hasta que el segundo numero de la pareja llegue al caso base.

5. Defina recursivamente las funciones last que devuelve el u´ltimo elemento de una lista no vac´ıa y elast que devuelve la lista resultante al borrar el u´ltimo elemento de una lista no vac´ıa.

```
laast :: [a] -> [a]
laast [x] = [x]
laast (x:xs) = laast (xs)
```

Donde “[x] = [x]” será nuestro caso base que nos dice que si tenemos una lista de un solo elemento, nos va a devolver la misma lista con el mismo elemento.
El paso recursivo es “(x:xs) = laast (xs)” que lo que hace es dividir a nuestra lista en cabeza y cuerpo, y va a ir tomando el cuerpo hasta llegar al caso base, o sea, hasta que la lista sea de un solo elemento.

```
elast :: [a] -> [a]
elast [x] = []
elast (x:xs) = x : elast (xs)
```

Donde “[x] = []” es nuestro caso base, en el que ocurre que si tenemos una lista de un solo elemento, nos va a devolver una lista vacia.
El paso recursivo esta definido como “(x:xs) = x : elast (xs)” que funciona casi de la misma manera que laast, pero en lugar de regresar solo el ultimo elemento, esta funcion va a ir concatenando las cabezas de la lista principal en una nueva hasta que llegue al caso base, o sea, ira reescribiendo los elementos excepto el ultimo.