1. Prove as duas regras para ++ :

```
xs ++ [] = xs
xs ++ (ys ++ zs) = (xs ++ ys) ++ zs
```

   para todos os xs, ys e zs finitos.

2. Considere as duas variações a seguir da função take:

```
take1 _ [] = []
take1 0 _      = []
take1 n (x:xs) = x : take1 (n-1) xs

take2 0 []     = []
take2 0 (x:xs) = []
take2 n []     = []
take2 n (x:xs) = x : take2 (n-1) xs
```

   Dê um exemplo de chamada para take2 que resulta em indefinido, que retornaria um valor real usando take1. O que isso diz sobre a rigidez (*strictness*) de cada posição de argumento?

   question Defina três propriedades QuickCheck para testar a implementação de árvore de busca. É necessário definir um gerador.

   Exemplos (Propriedades de árvores binárias)

   - Uma árvore binária com N nós internos tem, no máximo, N+1 folhas
   - O número **máximo** de nós de uma árvore binária de altura h é $2^{h+1} - 1$

3. Defina cinco propriedades QuickCheck para para o tipo Set que devem refletir as propriedades matemáticas de conjunto (Set - Wikipedia) (Set - Wikipedia)

4. Considere o seguinte tipo de dados para expressões aritméticas sem variáveis:

```
data Expr = Const Integer | Add Expr Expr | Mul Expr Expr | Neg Expr
```

   1. Escreva um gerador QuickCheck para o tipo Expr , ou seja, uma função
      genExpr :: Int -> Gen Expr
      em que o argumento inteiro é um limite superior do tamanho da expressão resultante (para alguma noção adequada de "tamanho").
   2. Usando genExpr, crie uma instância da classe de tipo Arbitrary para expressões.
   3. Suponha agora que você tenha recebido uma função de avaliação eval :: Expr -> Integer para expressões. Escreva algumas propriedades QuickCheck para testar sua correção; você deve escrever pelo menos quatro propriedades distintas. (Sugestão: expresse propriedades algébricas como A + B = B + A ou A + (B + C) = (A + B) + C.)