```haskell
-- Model solution for Lab11
--
-- Copyright [2000..2001] Gabriele Keller


module Lab11

where


foo:: [Int] -> Int
foo []     = []
foo (x:xs) = (bar xs) + (foo xs)


{- DERIVATION OF

 T_foo (n), (n: length of input list)


 T_foo (0) = 1
 T_foo (n) = T_bar (n-1) + T_foo (n-1)



 1. assume
 for T_bar (n)= 2

 T_foo (n) = 2 * n + 1

 It has linear work complexity, since the timing function is O(n).

 Proof:

 For c = 3 and all n > n0 = 2 we have

 T_foo (n) 2 * n + 1 < 3 * n, therefore T_foo is in O(n)

 1. assume
 for T_bar (n)= 3 * n + 1

 T_foo (n) = 3 * (n-1) + 1 + T_foo (n-1)
           = 3n - 2 + T_foo (n-1)

 T_foo (n) = 1 + sum (i=1)(n) (3n - 2)
           = 1 - 2n + 3 * sum(i=1)(n) n
           = 1 - 2n + 3 * (n * (n+1) / 0.5)
           = 1.5 n^2 - 0.5n + 1

  It has linear work complexity, since the timing function is O(n^2),
  According to the observations discussed in the lecture, we can
  omit all the constant factors and all components of the polynome but the
  one ith the highest exponent (2 in this case)


-}

Ord a => [a] -> [a]
bubbleSort:: Ord a => [a] -> [a]
bubbleSort xs
  | xs == xs' = xs
  | otherwise = bubbleSort xs'
  where
     xs' = bubble xs
     bubble (x:y: xs)
       | x < y     = x : (bubble (y:xs))
       | otherwise = y : (bubble (x:xs))
     bubble xs = xs
```

```
{- DERIVATION

 First, we derive the timing function for T_bubble depending on
 the length n of the input list

 T_bubble (0) = 1
 T_bubble (1) = 1

 T_bubble (n) = 2 + T_bubble (n-1)

 => T_bubble (n) = 2 * n + 1

 T_bubblesort does not follow the same pattern as previous derivations,
 since bubbleSort is called recursively on a list of the same length.

 T_bubbleSort (n) = 1 + T_bubble (n), if xs == xs'
                    1 + T_bubble (n) + T_bubblesort (n), otherwise


 However, if we look more closely at bubble we can see that

 1) bubble applied to a sorted list (and only to the sorted list) returns
    it's input list

 2) every element in a list which is to the right (i.e., further back in
    the list) than it's proper position (i.e., the position it will have
    in the sorted list) will be moved at least by one index to the left
    with each application of bubble

 This means that after at most n calls of bubble, bubble xs has to be
 sorted.

 Therefore, we have in the best case

 T_bubbleSort (n) = 1 + T_bubble (n) = 2 * n + 2

 in the worst case

 T_bubbleSort (n) = 1 + T_bubble (n) = 2n^2 + n + 1

 -}
```