```haskell
-- 1
{--

LET True = \ a b. a
LET (?:]  = \ c t f . c t (\ x . f)
LET LTE   = \ a b . iszero (sub a b) ? True : False

--}


-- 2

data Tree k d
    = Nil
    | Nd k d (Tree k d) (Tree k d)

insert :: Ord k => k -> d -> Tree k d -> Tree k d
insert k d Nil = Nd k d Nil Nil
insert k d (Nd k' d' l r)
    | k==k' = Nd k d l r
    | k<k'  = Nd k' d' (insert k d l) r
    | True  = Nd k' d' l (insert k d r)


-- 3
{--

length [] = 0                 -- d1
length (x:xs) = 1 + length xs -- d2

[] ++ ys = ys                 -- d3
(x:xs) ++ ys = x:(xs ++ys)    -- d4

1) xs==[]
L = length ([]++ys) =|3 length ys
P = length [] + length ys =|1 0 + length ys =|aritm0+ length ys
L = P

2)
I.H. length (as++ys) = length as + length ys
xs = (a:as)

L = length ((a:as)++ys) =|4 length (a:(as++ys)) =|2 1 + length (as++ys)
P = length (a:as) + length ys =|2 1 + length as + length ys =|I.H. 1 + length (as++ys)
L = P

Q.E.D.
--}

-- EOF
```