```haskell
{- HLINT ignore "Use camelCase" -}
{- HLINT ignore "Redundant if" -}
{- HLINT ignore "Use guards" -}

-- >>> largest "asdf" "rocksss"
-- "rocksss"

-- 1.
largest :: String -> String -> String
largest a b = if length a ≥ length b then a else b

-- >>> reflect 1
-- 1

-- 2.
-- The problem was that the code first tried to run `reflect num` and then
-- it tried to subtrack or add one to that value. We need to add parenthesis
-- so the value is added or subtracted to first and then the infinite recursion
-- is avoided
reflect :: Integer -> Integer
reflect 0 = 0
reflect num
    | num < 0 = (-1) + reflect (num + 1)
    | num > 0 = 1 + reflect (num - 1)

-- 3a.
all_factors :: Integer -> [Integer]
all_factors a = [x | x <- [1 .. a], mod a x == 0]

-- >>> all_factors 42
-- [1,2,3,6,7,14,21,42]

-- 3b.
perfect_numbers :: [Integer]
perfect_numbers = [x | x <- [1 ..], x == sum (init (all_factors x))]

-- >>> take 4 perfect_numbers
-- [6,28,496,8128]

-- 4.
-- IF STATEMENT VERSIONS
is_even :: Integer -> Bool
is_even a = if a == 0 then True else if a == 1 then False else is_even (a - 2)

is_odd :: Integer -> Bool
is_odd a = if a == 0 then False else if a == 1 then True else is_even (a - 2)

-- GUARDS VERSIONS
is_even :: Integer -> Bool
is_even a
    | a == 0 = True
    | a == 1 = False
    | otherwise = is_even (a - 2)

is_odd :: Integer -> Bool
is_odd a
    | a == 0 = False
    | a == 1 = True
    | otherwise = is_even (a - 2)

-- PATTERN MATCHING VERSIONS
is_odd :: Integer -> Bool
is_odd 0 = False
is_odd 1 = True
is_odd a = is_odd (a - 2)

is_even :: Integer -> Bool
is_even 0 = True
is_even 1 = False
is_even a = is_even (a - 2)

-- 5.
count_occurences :: [Integer] -> [Integer] -> Integer
-- >>>count_occurences [50, 40, 30] [10, 50, 40, 20, 50, 40, 30]
-- 3
-- >>>count_occurences [10, 20, 40] [10, 50, 40, 20, 50, 40, 30]
-- 1
-- >>> count_occurences [1, 2, 3] [1, 2, 3]
-- 1
-- >>> count_occurences [20, 10, 40] [10, 50, 40, 20, 50, 40, 30]
-- 0
-- >>> count_occurences [] []
-- 1
count_occurences [] b = 1
count_occurences a [] = 0
count_occurences a b
    | last a == last b =
        count_occurences (init a) (init b)
            + count_occurences a (init b)
    | otherwise = count_occurences a (init b)
```