```
module Lksa_2013_10_30 where
import Data.List

-------------------------------------
--MURGILKETA
-------------------------------------

--Osoak diren x, y eta bm hiru zenbaki emanda, zatitzaileak
--bilatzen bm-tik hasita, x eta y zenbakietatik zatitzaile
--txikiena duena itzuliko duen zt_lag funtzioa.
--x eta y zenbakiek bm-tik hasita zatitzaile txikiena bezala
--zenbaki bera dutenean, funtzioak -1 balioa itzuli beharko
--du. x edo y 1 edo txikiagoa baldin bada, errore-mezua
--aurkeztu beharko da. Bestalde, bm balioa 2 baino
--txikiagoa edo x baino handiagoa edo y baino handiagoa
--baldin bada ere, errore-mezua aurkeztu beharko da.

zt_lag:: Int -> Int -> Int -> Int

zt_lag x y bm
        | x <= 1                                = error "1. datua ez da egokia"
        | y <= 1                                = error "2. datua ez da egokia"
        | bm < 2 || bm > x || bm > y            = error "3. datua ez da egokia"
        | x `mod` bm == 0 && y `mod` bm == 0    = -1
        | x `mod` bm == 0                       = x
        | y `mod` bm == 0                       = y
        | otherwise                             = zt_lag x y (bm + 1)
-------------------------------------

--Osoak diren x eta y bi zenbaki emanda, zatitzaileak
--bilatzen 2tik hasita, bi zenbaki horietatik zatitzaile
--txikiena duena itzuliko duen zt funtzioa.
--x eta y zenbakiek 2tik hasita zatitzaile txikiena
--bezala zenbaki bera dutenean, funtzioak -1 balioa
--itzuli beharko du. x edo y 1 edo txikiagoa baldin bada,
--errore-mezua aurkeztu beharko da.

zt:: Int -> Int -> Int

zt x y = zt_lag x y 2

-------------------------------------
-------------------------------------
```

```
--BUKAERAKO ERREKURTSIBITATEA
-------------------------------------

--ss funtzioak hautaketaren bidezko metodoa edo "selection sort"
--metodoa jarraituz, zenbaki osozko zerrenda bat ordenatzen du.
--ss funtzioak ez du bukaerako errekurtsibitaterik.

ss :: [Integer] -> [Integer]
ss [ ] = [ ]
ss (x : s)
        | null s = [x]
        | otherwise = m : (ss ((takeWhile (/= m)(x:s)) ++ (tail(dropWhile (/= m)(x:s)))))
                        where m = minimum (x:s)

-------------------------------------

--ss funtzioak jasotzen duen parametroaz gain, emaitza bezala
--eraikiz joango den zerrenda gordez
--joateko erabiliko den bigarren parametroa duen ss_lag funtzioa.
--Beraz, ss_lag funtzioak, alde batetik bigarren parametroa
--eta bestetik, lehenengo parametro bezala emandako zerrenda
--ordenatuz eraikitzen den zerrenda elkartuz lortzen den
--zerrenda itzuli behar du.

ss_lag :: [Integer] -> [Integer] -> [Integer]
ss_lag [ ] q = q
ss_lag (x:s) q
        | null s = q ++ [x]
        | otherwise = ss_lag ((takeWhile (/= m)(x:s)) ++ (tail(dropWhile (/= m)(x:s)))) (q ++ [m])
                        where m = minimum (x:s)

-------------------------------------

--ss_lag funtzioari egokiak diren parametroekin deituz ss
--funtzioak egiten duen gauza bera egingo duen
--ss_be funtzioa.

ss_be :: [Integer] -> [Integer]

ss_be r = ss_lag r []

-------------------------------------
-------------------------------------
```

```
--ZERRENDA-ERAKETA
-------------------------------------

--Zenbaki arrunt denez (hau da, negatiboak ez diren zenbaki
--oso denez) osatutako zerrendaren hasierako azpizerrenda
--guztiez, hau da, Otik abiatuz eraiki daitezkeen
--azpizerrenda guztiez eratutako zerrenda infinitua
--aurkeztuz joango den has_azpi funtzioa.

--Lehenengo aukera:

has_azpi1:: [[Integer]]

has_azpi1 = [ [0..x] | x <- [0..]]

--Bigarren aukera:

has_azpi2:: [[Integer]]

has_azpi2 = [ genericTake x [0..] | x <- [1..]]

-------------------------------------

--Zenbaki arrunt denez (hau da, negatiboak ez diren zenbaki oso denez)
--osatutako zerrendaren hasierako azpizerrenda guztietako
--elementuen baturez, hau da, Otik abiatuz eraiki daitezkeen
--azpizerrenda guztietako elementuen baturez eratutako
--zerrenda infinitua aurkeztuz joango den has_batu funtzioa.


has_batu:: [Integer]

has_batu = [ sum x | x <- has_azpi1]

-------------------------------------

--Lehenengo osagai bezala osoa eta zero edo positiboa den
--zenbaki bat (zenbakien ohiko ordena jarraituz) eta bigarren
--osagai bezala Otik lehenengo osagaira arteko zenbakien baturaz
--osatutako bikoteez eratutako zerrenda infinitua kalkulatuz
--joango den zenb_batu izeneko funtzioa.
```

```
zenb_batu:: [(Integer, Integer)]

zenb_batu = zip [0..] has_batu

-------------------------------------

--Osoa den x zenbaki bat emanda, x zenbakia zenb batu
--zerrendan zenbatgarren bikoteko bigarrren osagaia
--den kalkulatzen duen zenbatgarrena izeneko funtzioa.
--x negatiboa baldin bada, errore-mezua aurkeztu
--beharko da. x negatiboa ez bada, baina hala ere
--zenb_batu zerrendan bigarren osagai bezala ez
--bada inoiz agertzen, -1 itzuli beharko da.


zenbatgarrena:: Integer -> Integer

zenbatgarrena x
        | x < 0         = error "Negatiboa"
        | not (x `elem` (genericTake x has_batu))       = -1
        | otherwise     = head [y | (y, z) <- zenb_batu, z == x]

--zenbatgarrena izeneko funtzioaren lehenengo bertsio
-- honetan honako hau hartu da kontuan: x zenbakia has_batu
--zerrendan agertzekotan, x posizioa baino lehenago
--agertuko da.

--Bigarren aukera:

zenbatgarrena2:: Integer -> Integer

zenbatgarrena2 x
        | x < 0         = error "Negatiboa"
        | not (x `elem` (takeWhile (<= x) has_batu)) = -1
        | otherwise     = (genericLength (takeWhile (<= x) has_batu)) - 1


-------------------------------------

--Zenbaki osozko bi zerrenda emanda, bigarren zerrendan
--ere agertzen diren lehenengo zerrendako elementuez
--osatutako zerrenda kalkulatzen duen badaudenak izeneko funtzioa.
--Agerpen-kopuruak ez dauka berdina izan beharrik lehenengo
```

```
--eta bigarren zerrendetan, nahikoa da agertzearekin.


badaudenak:: [Int] -> [Int] -> [Int]

badaudenak s r = [x | x <- s, x `elem` r]

-------------------------------------

--Zenbaki osozko zerrendez osatutako zerrenda bat
--eta zenbaki osozko zerrenda bat emanda,
--lehenengo parametroko zerrenda bakoitzarentzat bigarren
--parametroan ere badauden elementuez
--osatutako zerrenda kalkulatuko duen badaude_zerrenda
--izeneko funtzioa.


badaude_zerrenda:: [[Int]] -> [Int] -> [[Int]]

badaude_zerrenda q r = [badaudenak x r | x <- q]

-------------------------------------

--Zenbaki osozko zerrendez osatutako zerrenda bat emanda,
--parametro horretako zerrenda bakoitzarentzat zerrenda
--elementu bakarrekoa al den erabakitzen duen
--elem_bakarrekoak izeneko funtzioa.


elem_bakarrekoak:: [[Int]] -> [Bool]

elem_bakarrekoak q = [(length x) == 1 | x <- q]

-------------------------------------

--Zenbaki osozko zerrendez osatutako zerrenda bat eta
--zenbaki osozko zerrenda bat emanda,
--lehenengo parametroko zerrenda bakoitzarentzat
--bigarren parametroan ere agertzen den elementu
--bakarra al dagoen erabakitzen duen bakoitzean_bat
--izeneko funtzioa.
```

```
bakoitzean_bat:: [[Int]] -> [Int] -> Bool
bakoitzean_bat q r = and (elem_bakarrekoak (badaude_zerrenda q r))
-------------------------------------
```