```
module Lksa_2014_01_13 where
import Data.List

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

--Osoak diren x, bm, bik eta bak lau zenbaki emanda, bm-tik
--hasita x zenbakiaren zatitzaile bikoitien kopurua gehi "bik"
--zatitzaile bakoitien kopurua gehi "bak" baino handiagoa
--baldin bada True eta bestela False itzuliko duen
--"bik_gehiago_lag" funtzioa.
--bm-tik hasita, x zenbakiaren zatitzaile bikoitien kopurua
--gehi "bik" eta zatitzaile bakoitien kopurua gehi "bak" berdinak
--badira, False itzuli beharko da. x balioa 1 baino txikiagoa
--baldin bada edo bm balioa 1 baino txikiagoa baldin bada,
--errore-mezua aurkeztu beharko da. Bestalde, bm balioa x baino
--handiagoa bada ere, errore-mezua aurkeztu beharko da.

bik_gehiago_lag:: Int -> Int -> Int -> Int -> Bool

bik_gehiago_lag x bm bik bak
        | x <= 0                                    = error "1. datua ez da egokia."
        | (bm <= 0) || (bm > x)                     = error "2. datua ez da egokia."
        | x == bm && x `mod` 2 == 0 && (bik + 1) > bak  = True
        | x == bm && x `mod` 2 == 0 && (bik + 1) <= bak = False
        | x == bm && x `mod` 2 /= 0 && bik > (bak + 1)  = True
        | x == bm && x `mod` 2 /= 0 && bik <= (bak + 1) = False
        | (x `mod` bm) == 0 && (bm `mod` 2) == 0    = bik_gehiago_lag x (bm + 1) (bik + 1) bak
        | (x `mod` bm) == 0 && (bm `mod` 2) /= 0    = bik_gehiago_lag x (bm + 1) bik (bak + 1)
        | otherwise                                 = bik_gehiago_lag x (bm + 1) bik bak
-------------------------------------

--Osoa den x zenbakia emanda, bere zatitzaile bikoitien kopurua
--zatitzaile bakoitien kopurua baino handiagoa baldin bada
--True eta bestela False itzuliko duen "bik_gehiago" funtzioa
--definitu behar da. Zatitzaile bikoitien eta bakoitien kopurua
--berdina bada False itzuli beharko da. x parametroaren balio
--1 baino txikiagoa baldin bada, errore-mezua aurkeztu beharko da.

bik_gehiago:: Int -> Bool

bik_gehiago x = bik_gehiago_lag x 1 0 0
```

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

--Osoa den x zenbaki bat eta zenbaki osozko zerrenda bat emanda,
--x balioa x baino handiagoa edo berdina den zerrendako
--lehenengo elementuaren aurrean (ezkerreko aldean) kokatuz
--lortzen den zerrenda itzuliko du "txertatu" funtzioak.

txertatu :: Integer -> [Integer] -> [Integer]
txertatu x [] = (x : [])
txertatu x (y : s)
        | x <= y        = (x : (y : s))
        | otherwise     = (y : (txertatu x s))

-------------------------------------
--"txertatu" funtzioak jasotzen dituen x zenbakiaz eta zerrendaz gain,
--emaitza bezala eraikiz joango den zerrenda gordez joateko erabiliko
--den bigarren zerrenda duen "txertatu_lag" funtzioa.
--Beraz, "txertatu_lag" funtzioak jarraian zehazten diren bi zerrendak
--elkartuz lortzen den zerrenda itzuli beharko du:
--   Alde batetik, datu bezala emandako bigarren zerrenda.
--   Beste aldetik, datu bezala emandako lehenengo zerrendan x balioa
--     x baino handiagoa edo berdina den lehenengo elementuaren
--     aurrean (ezkerreko aldean) kokatuz lortzen den zerrenda berria.


txertatu_lag :: Integer -> [Integer] -> [Integer] -> [Integer]
txertatu_lag x [] q = (q ++ [x])
txertatu_lag x (y : s) q
        | x <= y        = (q ++ (x : (y : s)))
        | otherwise     = txertatu_lag x s (q ++ [y])

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

--"txertatu_lag" funtzioari egokiak diren parametroekin deituz "txertatu"
-- funtzioak egiten duen gauza bera egingo duen "txertatu_be" funtzioa.

txertatu_be :: Integer -> [Integer] -> [Integer]

txertatu_be x r = txertatu_lag x r []
```

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

--Zenbaki osozko zerrenda bat emanda, zerrendan
--elementu negatiborik baldin badago True eta bestela
--False itzultzen duen "negatiborik" izeneko funtzioa.

--Lehenengo aukera:

negatiborik :: [Integer] -> Bool

negatiborik s = (length [ x | x <- s, x < 0]) >= 1

--Bigarren aukera:

negatiborik2 :: [Integer] -> Bool

negatiborik2 s
        |(length [ x | x <- s, x < 0]) >= 1     = True
        |otherwise                               = False

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

--Zenbaki osozko zerrendez eratutako zerrenda bat emanda,
--zerrenda bakoitzean elementu negatiborik agertzen al den
--adierazten duen balio Boolearrezko zerrenda itzultzen duen
--"neg_agerpenik" izeneko funtzioa.


neg_agerpenik :: [[Integer]] -> [Bool]

neg_agerpenik s = [ negatiborik x | x <- s]

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

--Zenbaki osozko zerrendez eratutako zerrenda bat emanda,
--elementu negatiborik ez duten zerrendak bakarrik mantenduz
--geratzen den zerrenda itzultzen duen "neg_gabe" izeneko funtzioa
```

```
neg_gabe:: [[Integer]] -> [[Integer]]

neg_gabe s = [ x | x <- s, not (negatiborik x)]

--Beste aukera bat honako hau izango litzateke:
-- [ x | x <- s, (negatiborik x) = False]

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

--Osoa den zenbaki bat emanda, zenbaki horren zatitzaileen
--zerrenda itzuliko duen "zatitzaileak" funtzioa.


zatitzaileak:: Integer -> [Integer]

zatitzaileak x
        | x < 0         = error "Negatiboa"
        | otherwise     = [y | y <- [1..x], x `mod` y == 0]

--x parametroaren balioa 0 baldin bada, [1..x] tartea hutsa
--izango da eta zerrenda hutsa itzuliko da emaitza bezala.
--Beraz, kasu hori bereiztea ez da beharrezkoa.
--Hala ere, bereiz daiteke nahi izanez gero.

--Bigarren aukera:

zatitzaileak2:: Integer -> [Integer]

zatitzaileak2 x
        | x < 0         = error "Negatiboa"
        | x == 0        = []
        | otherwise     = [y | y <- [1..x], x `mod` y == 0]


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

--Osoak diren zenbaki positibo denen zatitzaileen zerrendez osatutako
--zerrenda infinitua aurkeztuz joango de "zat_denak" funtzioa


zat_denak:: [[Integer]]

zat_denak = [zatitzaileak x | x <- [1..]]
```

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

--Lehenengo osagai bezala osoa eta positiboa den zenbaki bat
--(zenbakien ohiko ordena jarraituz) eta bigarren osagai bezala
--lehenengo osagaiaren zatitzaile-zerrendaz osatutako bikoteez
--eratutako zerrenda infinitua kalkulatuz joango den "zat_bikote"
-- funtzioa.


zat_bikote:: [(Integer, [Integer])]

zat_bikote = zip [1..] zat_denak

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

--Osoa den n zenbaki bat emanda, n baino zatitzaile gehiago
--dituzten zenbakiei dagozkien "zat_bikote" zerrendako bikoteen
--zerrenda infinitua aurkeztuz joango den "zat_gehiago" funtzioa.


zat_gehiago:: Integer -> [(Integer, [Integer])]

zat_gehiago n = [ (x, y) | (x, y) <- zat_bikote, (genericLength y) > n]

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

--Osoak diren n eta kop bi zenbaki emanda, n baino zatitzaile
--gehiago dituzten lehenengo "kop" zenbakiei dagozkien "zat_bikote"
--zerrendako bikoteez osatutako zerrenda aurkeztuko duen "gehiago_finitua"
-- funtzioa


gehiago_finitua:: Integer -> Integer -> [(Integer, [Integer])]

gehiago_finitua n c = genericTake c (zat_gehiago n)

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