```haskell
module Lksa_2013_10_31 where
import Data.List

-------------------------------------
--MURGILKETA
-------------------------------------
--gh_lag funtzioak, x eta y bi zenbaki oso, zenbaki osoz osatutako r zerrenda bat
--eta xkop eta ykop beste bi zenbaki oso emanda, x zenbakiaren
--r zerrendako agerpen-kopurua gehi xkop balioa y zenbakiaren
--agerpen-kopurua gehia ykop baino handiagoa baldin bada, orduan x
--itzuliko du; x zenbakiaren r zerrendako agerpen kopurua gehi xkop balioa y
--zenbakiaren agerpen-kopurua gehi ykop baino txikiagoa baldin bada,
--orduan y itzuliko du; eta x zenbakiaren agerpen kopurua gehi xkop eta
--y zenbakiaren agerpen-kopurua gehi ykop berdinak badira -1 itzuliko du.


gh_lag:: Int -> Int -> [Int] -> Int -> Int -> Int

gh_lag x y r xkop ykop
        | (null r) && (xkop > ykop)     = x
        | (null r) && (xkop < ykop)     = y
        | (null r) && (xkop == ykop)    = -1
        | x == (head r)                 = gh_lag x y (tail r) (xkop + 1) ykop
        | y == (head r)                 = gh_lag x y (tail r) xkop (ykop + 1)
        | otherwise                     = gh_lag x y (tail r) xkop ykop
-------------------------------------

--x eta y bi zenbaki oso eta zenbaki osoz osatutako r zerrenda emanda,
--x eta y zenbakietatik r zerrendan gehien agertzen dena itzultzen duen
--gh funtzioa. Kasu berezi bezala, x eta y kopuru berean agertzen badira,-1
--itzuli beharko da.

gh:: Int -> Int -> [Int] -> Int
gh x y r = gh_lag x y r 0 0

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



-------------------------------------
--BUKAERAKO ERREKURTSIBITATEA
-------------------------------------
--Enuntziatuan emandako "burbuila" funtzioaren bidez lortzen den gauza bera
--egin nahi da baina bukaerako errekurtsibitatea duen "burbuila_lag" eta "burbuila_be"
```

```
--funtzioak erabiliz.
--"burbuila" funtzioaren eta definitu beharreko "burbuila_lag" eta "burbuila_be"
--funtzioen esanahia eta erabilgarritasuna hobeto ulertzeko, "ordenatuta" eta "bs"
--funtzioen definizioak ere enuntziatuan eman dira.
--"bs" funtzioak burbuilaren metodoa edo "bubble sort" metodoa jarraituz, zenbaki
--osozko zerrenda bat ordenatuko du.

--Dagoeneko enuntziatuan definituta datozen funtzioak: "ordenatuta", "burbuila" eta "bs"

--"ordenatuta" funtzioak zerrenda bat goranzko ordenean ordenatuta al dagoen
--erabakitzeko balio du.
ordenatuta :: [Integer] -> Bool

ordenatuta [] = True
ordenatuta (x:s)
        | null s                = True
        | x > (head s)          = False
        | otherwise             = ordenatuta s


--"burbuila" funtzioak zerrendako lehenengo elementua eskuinerantz
--mugitzen du bera baino handiagoa den balio bat aurkitu arte eta, jarraian,
--handiagoa den balio hori eskuinerantz mugituz doa oraindik handiagoa
--den beste balio bat aurkitu arte, eta abar.

burbuila :: [Integer] -> [Integer]

burbuila [] = []
burbuila (x:s)
        | null s                = [x]
        | x <= (head s)         = x:(burbuila s)
        | otherwise             = (head s) : (burbuila (x:(tail s)))


--bs funtzioak burbuilaren metodoa edo "bubble sort" metodoa
--jarraituz zerrenda bat ordenatzen du, ordena gorakorrean.

bs :: [Integer] -> [Integer]

bs r
        | ordenatuta r          = r
        | otherwise             = bs (burbuila r)
```

```
--Definitu beharreko funtzioak: burbuila_lag eta burbuila_be

------AZTERKETAN ESKATZEN DIREN FUNTZIOAK -- HASIERA ------

--"burbuila" funtzioak jasotzen duen zerrendaz gain, emaitza bezala
--lortuz joan beharreko zerrenda gordetzeko balioko digun bigarren
--zerrenda bat ere baduen "burbuila_lag" funtzioa.
--"burbuila_lag" funtzioak jarraian zehazten diren bi zerrenda
--elkartuz lortzen den zerrenda berria itzuliko du:
--1.)Bigarren parametro bezala emandako zerrenda.
--2.)Lehenengo parametro bezala emandako zerrendako lehenengo elementua eskuinerantz
-----mugituz bera baino handiagoa den balio bat aurkitu arte eta, jarraian,
-----handiagoa den balio hori eskuinerantz mugituz oraindik handiagoa
-----den beste balio bat aurkitu arte, eta abar, zerrenda zeharkatuz
-----lortzen den zerrenda.


burbuila_lag :: [Integer] -> [Integer] -> [Integer]

burbuila_lag [] q = q
burbuila_lag (x:s) q
        | null s                = q ++ [x]
        | x <= (head s)         = burbuila_lag s (q ++ [x])
        | otherwise             = burbuila_lag (x:(tail s)) (q ++ [head s])


--"burbuila_lag" funtzioari parametro egokiekin deituz "burbuila" funtzioak
--lortzen duen emaitza bera lortu beharko duen "burbuila_be" funtzioa.

burbuila_be :: [Integer] -> [Integer]

burbuila_be r = burbuila_lag r []

------AZTERKETAN ESKATZEN DIREN FUNTZIOAK -- BUKAERA ------
```

```
--"burbuila_be" funtzioa erabiliz zerrendak ordenatzeko gai den
--"bs_be" funtzioa definituko da jarraian. Funtzio honen definizioa
--ez da eskatzen azterketan

--"burbuila_be" funtzioa erabiliz eta "bubble sort" metodoa jarraituz,
--zenbaki osozko zerrenda bat ordenatzeko gai den funtzioa: bs_be


bs_be :: [Integer] -> [Integer]

bs_be r
        | ordenatuta r          = r
        | otherwise             = bs_be (burbuila_be r)





-------------------------------------
-------------------------------------
--ZERRENDA-ERAKETA
-------------------------------------

--x eta n bi zenbaki oso emanda, x balioa n aldiz errepikatuz
--lortzen den zerrenda itzultzen duen "konst" funtzioa.
--n-ren balioa 0 bada, zerrenda hutsa itzuli beharko da eta
--n-ren balioa negatiboa baldin bada, errore-mezua aurkeztu beharko da.


konst:: Integer -> Integer -> [Integer]

konst x n = [ x | y <- [1..n]]



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

--Osoa den x zenbaki bat emanda, x errepikatuz lor daitezkeen
--luzera guztietako zerrendez osatutako zerrenda infinitua
--aurkeztuz joango den "errepikatu" izeneko funtzioa.
```

```
errepikatu:: Integer -> [[Integer]]

errepikatu x = [ konst x y | y <- [0..]]

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

--Osoa den x zenbaki bat emanda, x zenbakiaren berredura denez
--osatutako zerrenda infinitua aurkeztuz joango den
--"berredura_zerrenda" izeneko funtzioa.


berredura_zerrenda:: Integer -> [Integer]

berredura_zerrenda x = [ product y | y <- (errepikatu x)]

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

--Osoa den n zenbaki bat emanda, 2 zenbakiaren lehenengo n berreduraz
--osatutako zerrenda itzuliko duen "bi_ber_zer" funtzioa.
--n-ren balioa 0 baldin bada, zerrenda hutsa itzuli
--beharko da eta n-ren balioa negatiboa baldin bada,
--errore-mezua aurkeztu beharko da.


bi_ber_zer:: Integer -> [Integer]

bi_ber_zer n
        | n < 0         = error "Negatiboa"
        | otherwise     = genericTake n (berredura_zerrenda 2)

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

--Osoa den n zenbaki bat emanda, lehenengo n zenbakientzat,
--zenbakia eta zenbaki horri dagokion 2ren berreduraz eratutako bikoteez
--osatutako zerrenda itzuliko duen "bi_ber_indizea" funtzioa.
--n-ren balioa 0 baldin bada, zerrenda hutsa itzuli beharko da eta
--n-ren balioa negatiboa baldin bada, errore-mezua aurkeztu beharko da.


bi_ber_indizea:: Integer -> [(Integer, Integer)]
```

```
bi_ber_indizea n
        | n < 0         = error "Negatiboa"
        | otherwise     = zip [0..] (bi_ber_zer n)

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

--Osoa den x zenbakia emanda, x zenbakia 2ren berredura al den
--erabakitzen duen "bi_ber_da" funtzioa.
--x balioa 0 edo negatiboa baldin bada, errore-mezua aurkeztu beharko da.

--Lehenengo aukera:

bi_ber_da:: Integer -> Bool

bi_ber_da x
        | x <= 0        = error "Ez da positiboa"
        | otherwise     = x `elem` (bi_ber_zer x)


--"bi_ber_da" funtzioa definitzerakoan, ez dakigu x balioa
--biren berreduren zerrendako zein posiziotan egongo den,
--baina badakigu egotekotan x-garren posiziorako
--agertuko dela.


--Bigarren aukera:

bi_ber_da_ :: Integer -> Bool

bi_ber_da_ x
          | x <= 0      = error "Ez da positiboa"
          | otherwise   = x `elem` (takeWhile (<= x) (berredura_zerrenda 2))


--"bi_ber_da" funtzioa definitzerakoan, ez dakigu x balioa
--biren berreduren zerrendako zein posiziotan egongo den. Hori
--dela eta, biren berreduren zerrenda infinitua hartuko da "berredura_zerrenda 2".
--Jarraian x balioa baino handiagoa den balio bat agertu arte dauden
--elementuak hartuko dira "takeWhile" funtzioaren bidez.
--Bukatzeko, x balioa "takeWhile" funtzioaren bidez hartutako
--elementu horien artean baldin badago, orduan x 2ren berredura da
--eta bestela ez.
```

```haskell
-------------------------------------

--Osoa den x zenbakia emanda, x zenbakia lortzeko 2 zenbakiak
--zein berretzaile behar duen kalkulatzen duen "berretzailea" funtzioa.
--x balioa 0 edo negatiboa baldin bada, errore-mezua aurkeztu beharko da.
--x balioa 1 edo handiagoa baldin bada baina 2ren berredura
--ez bada, beste errore-mezu desberdin bat aurkeztu beharko da.

--Lehenengo aukera:

berretzailea:: Integer -> Integer

berretzailea x
        | x <= 0                = error "Ez da positiboa."
        | not (bi_ber_da x)     = error "Ez da 2ren berredura."
        | otherwise             = head [y | (y,z) <- (bi_ber_indizea x), z == x]


--Bigarren aukera:

berretzailea_ :: Integer -> Integer

berretzailea_ x
        | x <= 0                = error "Ez da positiboa."
        | not (bi_ber_da x)     = error "Ez da 2ren berredura."
        | otherwise             = (genericLength (takeWhile (<= x) (berredura_zerrenda 2))) - 1

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

--Zenbaki osozko zerrenda bat emanda, zenbaki horietako bakoitza
--2ren berredura al den ala ez erabaki eta erabaki horiez osatutako
--zerrenda itzuliko duen "bi_ber_dira" izeneko funtzioa.
--Datu bezala emandako zerrendan negatiboa den balioren bat edo 0
--zenbakia agertzen bada, errore-mezua aurkeztu beharko da.


bi_ber_dira:: [Integer] -> [Bool]

bi_ber_dira q
        | (length [y | y <- q, y <= 0]) > 0     = error "Positiboa ez den elementuren bat du."
        | otherwise                             = [bi_ber_da y | y <- q]
```

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