6. GAIKO ARIKETAK
PROGRAMA ERREKURTSIBOEN ERALDAKETA
Burstall-en metodoa

A) Oinarrizko funtzio errekurtsiboak (kasu sinple bat eta kasu errekurtsibo bat)

1. luzera (zerrenda baten luzera)

Zerrenda bat emanda, *luzera* izeneko funtzioak zerrenda horretan zenbat elementu dauden kalkulatzen du:

```
luzera :: ([Int]) -> Int

Hasierako baldintza: {true}

luzera([]) = 0
luzera(x : s) = 1 + luzera(s)
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

luzera_re([8, 9, 1])

2. azkena (zerrendako azken elementua)

Hutsa ez den zerrenda bat emanda, *azkena* izeneko funtzioak zerrendako azkeneko elementua (eskuineko ertzean dagoena) itzultzen du:

```
azkena :: ([Int]) -> Int

Hasierako baldintza: {¬hutsa_da(s)}

azkena(s)
      | hutsa_da(hond(s))        = leh(s)
      | otherwise                = azkena(hond(s))
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

azkena_re([8, 9, 1])

3. alder (zerrenda baten alderantzizkoa)

Zerrenda bat emanda, *alder* izeneko funtzioak zerrenda horren alderantzizkoa kalkulatzen du:

```
alder :: ([Int]) -> [Int]

Hasierako baldintza: {true}

alder([]) = []
alder(x : s) = alder(s) ++ [x]
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

alder_re([8, 9, 1])

4. hamartar (10 oinarrira pasatzeko)

Oinarria adierazten duen b zenbakia ($2 \leq b \leq 9$) eta b oinarrian adierazita dagoen $n \geq 0$ zenbakia emanda, *hamartar* izeneko funtzioak n-ri 10 oinarrian dagokion zenbakia kalkulatzen du:

```
hamartar :: (Int, Int) -> Int

Hasierako baldintza: {2 ≤ b ≤ 9 ∧ n ≥ 0}

hamartar(b, n)
  | n ≤ b                = n
  | otherwise            = n mod 10 + hamartar(b, n / 10) * b
```

Adibidea:

hamartar(2, 1010) = 10 **(Inbariantea kalkulatzerakoan adibide hau garatu)**
hamartar(2, 110011) = 51
hamartar(2, 11011) = 27
hamartar(8, 35) = 29

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

5. loghur (logaritmoa hurbiltzeko)

Osoak diren $b \geq 2$ eta $x \geq 1$ bi zenbaki emanda, *loghur* funtzioak x-ren logaritmoa hurbiltzen du b oinarriarekiko:

```
loghur :: (Int, Int) -> Int

Hasierako baldintza: {b ≥ 2 ∧ x ≥ 1}

loghur(b, x)
  | x < b              = 0
  | otherwise          = loghur(b, x / b) + 1
```

Adibideak:

loghur(2, 8) = 3
loghur(2, 11) = 3
loghur(10,100) = 2
loghur(10, 503) = 2 **(Inbariantea kalkulatzerakoan adibide hau garatu)**

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

6. bideresk (biderkaketa eskalarra)

Luzera bereko $s = [s_1, s_2, ..., s_k]$ eta $r = [r_1, r_2, ..., r_k]$ zerrenden arteko biderkaketa eskalarra honela kalkulatzen da:

$$\sum_{i=1}^{k} r_i * s_i$$

Adibidez, bideresk([3, 2, 1], [2, 5, 4]) = (3∗2) + (2∗5) + (1∗4) = 20.

```
bideresk :: ([Int], [Int]) -> Int

Hasierako baldintza: {luzera(s) = luzera(r)}

bideresk(s, r)
  | hutsa_da(s)          = 0
  | otherwise            = leh(s) * leh(r) + bideresk (hond(s), hond(r))
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzerakoan goiko adibidea garatu.

Gogoratu leh funtzioak zerrendako lehenengo elementua itzultzen duela eta hond funtzioak zerrendako lehenengo elementua kenduz lortzen den zerrenda itzultzen duela.

7. aldatu (posizio bikoitietakoei 1 batu eta bakoitietakoei 1 kendu)

Zerrenda bat emanda, aldatu izeneko funtzioak posizio bakoitietako balioei 1 kenduz eta posizio bikoitietako balioei 1 gehituz lortzen den zerrenda kalkulatzen du. Kasu berezi bezala, zerrendako elementu-kopurua bakoitia baldin bada, zerrendako azkeneko elementuari ez zaio 1 kenduko:

```
aldatu :: ([Int]) → [Int]
Hasierako baldintza: {true}

aldatu(s)
  | luzera(s) <= 1       = s
  | luzera(s) > 1        = (leh(s) – 1) : ((leh(hond(s)) +1) : aldatu(hond(hond(s))))
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

aldatu_re([3, 1, 8, 9])

Gogoratu leh funtzioak zerrendako lehenengo elementua itzultzen duela eta hond funtzioak zerrendako lehenengo elementua kenduz lortzen den zerrenda itzultzen duela.

8. metatu (baturak metatuz joan) (2010eko iraila)

Zenbaki osoz osatutako s zerrenda bat emanda, *metatu* funtzioak posizio bakoitzean sarrerako zerrendako lehenengo posiziotik posizio horretara arteko elementuen batura duen zerrenda itzultzen du. Emandako zerrenda hutsa bada, zerrenda hutsa itzultzen du.

```
metatu:: ([Int]) → [Int]
Hasierako baldintza: {true}

metatu(s)
  | luzera(s) ≤ 1        = s                                                    (#1)
  | otherwise            = [leh(s)] ++ metatu((leh(s) + leh(hond(s))):hond(hond(s)))   (#2)
```

luzera funtzioak zerrendako elementu-kopurua itzultzen du, leh funtzioak zerrendako lehenengo elementua itzultzen du eta hond funtzioak hondarra, hau da, lehenengo elementua kenduz gelditzen den zerrenda itzultzen du.

Hasierako baldintzak sarrerako zerrenda bezala edozein zerrenda onartzen dela adierazten du.

Adibideak:

- metatu([10, 8, 15, 4]) = [10, 18, 33, 37]
  lehenengo posizioan 10 gelditu da, bigarren posizioan 10 + 8, hirugarrenean 10 + 8 + 15 eta laugarrengoan 10 + 8 + 15 + 4.
- metatu([10, 2, 6]) = [10, 12, 18] **Inbariantea kalkulatzerakoan adibide hau erabili**.
  Bigarren adibide honetan lehenengo posizioan 10 gelditu da, bigarren posizioan 10 + 2 eta hirugarrenean 10 + 2 + 6.

Burstall-en metodoa erabiliz, funtzio hori eraldatu gauza bera egiten duen funtzio iteratibo bat lortuz.

B) Elkarkorra ez den egagiketa duten funtzio errekurtsiboak

9. azkena_kendu (zerrendako azkeneko elementua kendu)

Hutsa ez den zerrenda bat emanda, *azkena_kendu* izeneko funtzioak zerrendako azkeneko elementua (eskuineko ertzean dagoena) kenduz gelditzen den zerrenda itzultzen du:

```
azkena_kendu: ([Int]) -> [Int]

Hasierako baldintza: {¬hutsa_da(s)}

azkena_kendu(s)
      | hutsa_da(hond(s))          = []
      | otherwise                  = leh(s) : azkena_kendu(hond(s))
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

azkena_kendu_re([5, 8, 1])

C) Kasu sinple bat baino gehiago dituzten funtzioak

10. azpizer (zerrenda bat beste baten azpizerrenda al den erabakitzeko)

Bi zerrenda emanda, *azpizer* funtzioak lehenengo zerrenda bigarrenaren azpizerrenda al den ala ez erabakitzen du.

```
azpizer :: ([Int], [Int]) -> Bool

Hasierako baldintza: {true}

azpizer(s, r)
      | luzera(s) > luzera(r)          = False
      | aurrizkia(s, r)                = True
      | otherwise                      = azpizer(s, hond(r))
```

s zerrenda r zerrendaren azpizerrenda dela esaten da r = h1 ++ s ++ h2 berdintza betearazten duten h1 eta h2 bi zerrenda existitzen badira:

Adibideak:

azpizer([4, 8], [2, 7, **4**, **8**, 5]) = True	azpizer([], [2, 7, 4, 8, 5]) = True
azpizer([2, 7, 4, 8, 5], [4, 8]) = False	azpizer([4, 8], []) = False
azpizer([4, 8], [2, 4, 7, 8, 5]) = False	azpizer([4, 8], [4, 8]) = True

aurrizkia(s, r) funtzioak True itzuliko du s zerrenda r zerrendaren azpizerrenda bada eta justu hasieran badago. Funtzio hori definituta dagoela kontsideratuko dugu bai Haskell-en eta baita ADA*-n ere.

s zerrenda r zerrendaren aurrizkia izango da r = s ++ h berdintza betearazten duen h zerrenda bat existitzen bada.

Adibideak:

aurrizkia([4, 8], [2, 7, **4**, **8**, 5]) = False	aurrizkia([], [2, 7, 4, 8, 5])= True
aurrizkia([4, 8], [**4**, **8**, 2, 7, 5]) = True	aurrizkia([4, 8], [])= False
aurrizkia([4, 8], [4, 7, 2, 8, 5])= False	aurrizkia([4, 8], [4, 8]) = True

Egin beharrekoa:

azpizer izeneko funtzioari dagokion funtzio iteratiboa lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

azpizer_re([4, 8], [2, 7, **4**, **8**, 5])

11. atzizkia (zerrenda bat beste baten atzizkia al den erabakitzen du)

Bi zerrenda emanda, *atzizkia* funtzioak lehenengo zerrenda bigarrenaren atzizkia al den ala ez erabakitzen du:

```
atzizkia :: ([Int], [Int]) -> Bool

Hasierako baldintza: {true}

atzizkia(s, r)
        | luzera(s) > luzera(r)          = False
        | s == r                         = True
        | otherwise                      = atzizkia(s, hond(r))
```

s zerrenda r zerrendaren atzizkia dela esango dugu r = h ++ s berdintza betearazten duen h zerrenda existitzen bada:

Adibideak:

atzizkia([4, 8], [2, 7, 5, **4**, **8**]) = True	atzizkia([], [2, 7, 4, 8, 5]) = True
atzizkia([2, 7, 4, 8, 5], [4, 8]) = False	atzizkia([4, 8], []) = False
atzizkia([4, 8], [2, 4, 7, 8, 5]) = False	atzizkia([4, 8], [4, 8]) = True

atzizkia izeneko funtzioari dagokion funtzio iteratiboa lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

atzizkia_re([4, 8], [5, **4**, **8**])

12. kendubikpos (pos posizioko elementua kendu bikoitia bada) (2009ko iraila)

Zenbaki osoz osatutako s zerrenda ez-hutsa eta zerrenda horretako posizio bat adierazten duen pos zenbaki oso bat emanda, *kendubikpos* funtzioak pos posizioko elementua bikoitia baldin bada, elementu hori kenduz gelditzen den zerrenda itzultzen du eta pos posizioko elementua bikoitia ez bada, s zerrenda bera itzultzen du, hau da ez du ezer kentzen.

```
kendubikpos:: ([Int], Int) → [Int]
Hasierako baldintza: {¬hutsa_da(s) ∧ 1 ≤ pos ≤ luzera(s)}

kendubikpos(s, pos)
  | pos == 1 && leh(s) mod 2 == 0        = hond(s)
  | pos == 1 && leh(s) mod 2 /= 0        = s
  | pos /= 1                             = [leh(s)] ++ kendubikpos(hond(s), pos – 1)
```

leh funtzioak zerrendako lehenengo elementua itzultzen du eta hond funtzioak hondarra, hau da, lehenengo elementua kenduz gelditzen dena.

Adibideak:

kendubikpos([8, 5, **9**, 7], **3**) = [8, 5, **9**, 7] 9 zenbakia 3. posizioan egon arren ez da kendu ez delako bikoitia.

kendubik([8, 5, **16**, 7], **3**) = [8, 5, 7] ← **Inbariantea kalkulatzerakoan adibide hau erabili**

Bigarren adibide honetan 16 zenbakia kendu da 3. posizioan dagoelako eta bikoitia delako.

Burstall-en metodoa erabiliz, funtzio hori eraldatu gauza bera egiten duen funtzio iteratibo bat lortuz.

D) Kasu errekurtsibo bat baino gehiago dituzten funtzioak

13. aldiz (elementu bat zerrenda batean zenbat aldiz agertzen den)

Elementu bat eta zerrenda bat emanda, *aldiz* izeneko funtzioak elementu hori zerrendan zenbat aldiz agertzen den kontatzen du:

```
aldiz :: (Int, [Int]) -> Int

Hasierako baldintza: {true}

aldiz(x, []) = 0
aldiz(x, z : s)
      | x /= z          = aldiz(x, s)
      | otherwise       = 1 + aldiz(x, s)
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

aldiz_re(5, [5, 4, 5, 6])

14. txikiena (zerrendako elementu txikiena)

Hutsa ez den zerrenda bat emanda, txikiena izeneko funtzioak zerrendako balio txikiena itzuliko du:

```
txikiena :: ([Int]) -> Int

Hasierako baldintza: {¬hutsa_da(s)}

txikiena(s)
  | hutsa_da(hond(s))                 = leh(s)
  | leh(s) <= leh(hond(s))            = txikiena(leh(s) : hond(hond(s)))
  | leh(s) > leh(hond(s))             = txikiena(hond(s))
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzeko honako adibide hau garatu behar da:

txikiena_re([4, 6, 3, 8])

15. bitar (zenbaki baten balio bitarra kalkulatzeko)

Negatiboa ez den x ($x \geq 0$) zenbaki bat emanda, *bitar* funtzioak x zenbakiari dagokion balio bitarra lortzen du:

```
bitar :: (Int) -> Int
Hasierako baldintza: {x ≥ 0}

bitar(x)
  | x ≤ 1                          = x
  | bikoitia(x) && x ≥ 2           = 10 * bitar(x / 2)
  | bakoitia(x) && x ≥ 2           = 10 * bitar(x / 2) + 1
```

Adibidea:

Funtzio honek x-ren adierazpen bitarrari dagokion bitez osatutako zenbakia kalkulatzen du eta ez du x-ri dagokion bit zerrenda kalkulatzen.

- bitar(14) = 1110
  Beraz 14 zenbakia emanez gero mila ehun eta hamar zenbakia itzuliko du eta ez [1, 1, 0, 0] zerrenda.
- bitar(22) = 10110
  22 zenbakia emanez gero hamar mila ehun eta hamar zenbakia itzuliko du.

Burstall-en metodoa erabiliz, funtzio hori eraldatu gauza bera egiten duen funtzio iteratibo bat lortuz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

bitar_re(14)

16. bakozif (zenbaki batean zenbat zifra bakoiti dauden)

Negatiboa ez den x zenbakia emanda ($x \geq 0$), *bakozif* izeneko funtzioak x zenbakian zenbat zifra bakoiti dauden kalkulatzen du:

```
bakozif :: (Int) -> Int
Hasierako baldintza: {x ≥ 0}

bakozif(x)
  | x <= 9                         = x mod 2
  | (x mod 10) mod 2 == 0          = bakozif(x div 10)
  | (x mod 10) mod 2 ≠ 0           = 1 + bakozif(x div 10)
```

div zatiketa osoaren eragilea da

Adibideak:

bakozif(45881) = 2
bakozif(6800) = 0

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

bakozif_re(4538)

17. zkh3 (hiru zenbakiren zatitzaile komunetako handiena)

Osoak eta positiboak diren a, b eta c ($a \geq 1, b \geq 1, c \geq 1$) hiru zenbaki emanda, *zkh3* funtzioak a, b eta c-ren zatitzaile komunetako handiena kalkulatzen du:

```
zkh3 :: (Int, Int, Int) -> Int

Hasierako baldintza: {a ≥ 1 ∧ b ≥ 1 ∧ c ≥ 1}

zkh3(a, b, c)
  | a == b && b == c        = a
  | a – c ≥ b               = zkh3(a – c, b, c)
  | a – c ≥ c               = zkh3(b, a – c, c)
  | a – c ≥ 1               = zkh3(b, c, a – c)
  | a – c ≤ 0               = zkh3(b, c, c – a)
```

Burstall-en metodoa erabiliz, funtzio hori transformatu funtzio iteratibo bat lortuz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

zkh3_re(18, 12, 24)

18. garbitu (1.a baino txikiagoak direnak ezabatu eta 1.a bukaeran ipini) (2008ko ekaina)

Zenbaki osoz osatutako zerrenda ez-hutsa emanda, garbitu izeneko funtzioak zerrendako lehenengo elementua baino txikiagoak diren elementu denak kenduz eta lehenengo elementu hori bukaeran ipiniz lortzen den zerrenda itzultzen du. Lehenengo elementuak zerrenda osoa zeharkatuko du bera baino txikiagoak direnak desagerraraziz eta bera bukaeran geldituz.

```
garbitu :: ([Int]) -> ([Int])
Hasierako baldintza ≡ {¬hutsa_da(s)}

 garbitu(s)
    | luzera(s) == 1            = s
    | leh(s) > leh(hond(s))     = garbitu(leh(s):hond(hond(s)))
    | otherwise                 = [leh(hond(s))] ++ garbitu(leh(s):hond(hond(s)))
```

Adibideak:

garbitu([5, 8, 8, 4, 9, 3]) = [8, 8, 9, 5]
garbitu([5, 8, 5, 4, 9, 3]) = [8, 5, 9, 5]
garbitu([5, 8, 8, 9, 6]) = [8, 8, 9, 6, 5]

Burstall-en metodoa erabiliz, funtzio hori eraldatu gauza bera egiten duen funtzio iteratibo bat lortuz

Inbariantea kalkulatzerakoan honako adibide hau garatu:

garbitu_re([6, 4, 8, 7])

19. zatitu (1. elementuaz zati daitezkeenak zatitu eta besteak dauden bezala laga) (2008ko iraila)

Kontsidera dezagun zenbaki osoz osatutako zerrenda ez-hutsa emanda, sarrerako zerrendak baino elementu bat gutxiago duen eta jarraian aipatzen diren bi irizpideak jarraituz lortzen den zerrenda itzultzen duen zatitu izeneko funtzioa.

a) Zerrendako lehenengo elementuaz zati daitezkeen elementuak (hodarra zero ematen dutenak) lehenengo elementuaz zatituz lortzen den emaitzaz ordezkatu.
b) Lehenengo elementuaz zati ezin daitezkeenak dauden bezala laga.

Hasierako zerrendak elementu bakarra badu, zerrenda hutsa itzuliko du.

Adibideak:

zatitu([**2**, 8, 6, 5, 9, 20]) = [4, 3, 5, 9, 10]	zatitu([**3**, 15]) = [5]
zatitu([**5**, 8, 8, 15, 9]) = [8, 8, 3, 9]	zatitu([**3**, 14]) = [14]

Lehenengo elementuak zerrenda osoa zeharkatuko du zatitu ditzakeen elementuak zatituz eta azkenean bera desagertu egingo da.

```
zatitu :: ([Int]) -> [Int]
Hasierako baldintza ≡ {¬hutsa_da(s)}

zatitu(s)
  |luzera(s) == 1                  = []
  |leh(hond(s)) mod leh(s) == 0    = [leh(hond(s)) div leh(s)] ++ zatitu(leh(s):hond(hond(s)))
  |otherwise                       = [leh(hond(s))] ++ zatitu(leh(s):hond(hond(s)))
```

Burstall-en metodoa erabiliz, funtzio hori eraldatu gauza bera egiten duen funtzio iteratibo bat lortuz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

zatitu_re([5, 8, 15, 9])

Gogoratu **mod** eragileak zatiketa osoaren hondarra kalkulatzen duela eta **div** eragileak zatidura osoa kalkulatzen duela.

20. beaz (elementu berdinez osatutako azpizerrenden luzerak) (2009ko ekaina)

Osoa den c balioa eta zenbaki osoz osatutako s zerrenda ez-hutsa emanda, beaz izeneko funtzioak s zerrendan elementu berdinez osatutako azpizerrendetako elementu-kopuruekin osatutako zerrenda kalkulatzen du. Salbuespen bakarra lehenengo azpizerrendaren kasuan gertatzen da, izan ere lehenengo azpizerrendaren kasuan bere elementu-kopurua gehi c gordeko baita.

```
beaz :: (Int, [Int]) -> [Int]
Hasierako baldintza: {¬hutsa_da(s)}

beaz(c, s)
  | luzera(s) == 1              = [c + 1]
  | leh(s) == leh(hond(s))      = beaz(c + 1, hond(s))
  | leh(s) /= leh(hond(s))      = [c + 1] ++ beaz(0, hond(s))
```

Adibideak:

beaz(0, [8, 8, 8, 8, 5, 9, 9, 8, 8]) = [4, 1, 2, 2]
beaz(3, [8, 8, 8, 8, 5, 9, 9, 8, 8]) = [7, 1, 2, 2]
beaz(0, [8, 8, 5, 9, 9]) = [2, 1, 2] ← **Inbariantea kalkulatzerakoan adibide hau erabili**
beaz(0, [8]) = [1]
beaz(3, [8]) = [4]

leh funtzioak zerrendako lehenengo elementua itzultzen du eta hond funtzioak hondarra, hau da, lehenengo elementua kenduz gelditzen dena

Burstall-en metodoa erabiliz, funtzio hori eraldatu gauza bera egiten duen funtzio iteratibo bat lortuz.

21. azk_lehena (zerrendako azkeneko zenbaki lehena) (2010eko ekaina)

Zenbaki osoz osatuta dagoen eta bere elementuen artean gutxienez zenbaki lehen bat duen s zerrenda emanda, zerrendako azkeneko zenbaki lehena (eskuinerago dagoena) itzultzen du *azk_lehena* izeneko funtzioak.

azk_lehena :: ([Int]) → Int
Hasierako baldintza: {∃ k (1 ≤ k ≤ luzera(s) ∧ lehena_da(elem(k, s)))}

azk_lehena(s)
| hutsa_da(hond(s)) = leh(s) (#1)
| lehena_da(leh(hond(s))) = azk_lehena(hond(s)) (#2)
| not lehena_da(leh(hond(s))) = azk_lehena(leh(s):hond(hond(s))) (#3)

- *badago* funtzioak, elementu bat eta zerrenda bat emanda, elementu hori zerrenda horretan baldin badago true itzultzen du eta bestela false itzultzen du.
- *leh* funtzioak, zerrenda bat emanda, zerrendako lehenengo elementua itzultzen du.
- *hond* funtzioak, zerrenda bat emanda, hondarra itzultzen du, hau da, lehenengo elementua kenduz gelditzen den zerrenda itzultzen du.
- *luzera* funtzioak, zerrenda bat emanda, zerrendako elementu-kopurua itzultzen du.
- *lehena_da* funtzioak zenbaki oso bat emanada, zenbakia lehena bada true itzuliko du eta bestela false, hau da, true itzuliko du zenbakia positiboa bada eta justu bi zatitzaile baldin baditu (2, 3, 5, 7, 11, 13, 17, … zenbakiak lehenak dira).
- Zenbaki oso bat eta zerrenda bat emanda, *elem* funtzioak zerrenda horretan zenbakiak adierazten duen posizioko elementua itzultzen du (zenbakia egokia dela suposatuz).

Hasierako baldintzak s zerrendan gutxienez zenbaki lehen bat badagoela adierazten du eta ondorioz badakigu zerrenda ez dela hutsa izango.

Adibideak (azk_lehena funtzioarentzat):

- azk_lehena([8, 11, -7, 2, **5**, 6]) = 5
  Kasu honetan zerrendan hiru zenbaki lehen daude, 11, 2 eta 5, eta funtzioak azkena itzuli behar du, hau da, 5.
- azk_lehena([2, -7, **2**, 4, 10]) = 2
  Kasu honetan zerrendan bi zenbaki lehen daude, 2 eta 2, eta funtzioak azkena itzuli behar du, hau da, 2.
- azk_lehena([8, 2, **11**, 6]) = 11
  Kasu honetan zerrendan bi zenbaki lehen daude, 2 eta 11, eta funtzioak azkena itzuli behar du, hau da, 11. ← **Inbariantea kalkulatzerakoan adibide hau erabili**.

Burstall-en metodoa erabiliz, azk_lehena funtzioa eraldatu gauza bera egiten duen funtzio iteratibo bat lortuz.

B) eta D) kasuetako ezaugarriak nahastuta dituzten funtzio errekurtsiboak

22. hasieran (x-ren agerpen denak zerrendaren hasieran ipini)

Jarraian datorren funtzioak, x elementu bat eta zerrenda bat emanda, x-ren agerpen denak zerrendaren hasieran ipiniz lortzen den zerrenda itzultzen du:

```
hasieran :: (Int, [Int]) -> [Int]

Hasierako baldintza: {true}

hasieran(x, []) = []
hasieran(x, z : s)
  | z == x                = z : hasieran(x, s)
  | otherwise             = hasieran(x, s) ++ [z]
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

hasieran_re(8, [7, 8, 2, 8])

23. hand (x baino handiagoak direnak zerrendatik kendu)

x zenbaki oso bat eta zenbaki osoz osatutako zerrenda bat emanda, hand izeneko funtzioak hasierako zerrendatik x baino handiagoak diren balioak kenduz lortzen den zerrenda kalkulatzen du:

```
hand :: (Int, [Int]) -> [Int]

Hasierako baldintza: {true}

hand(x, []) = []
hand(x, z : s)
  | z > x          = hand(x, s)
  | z <= x         = z : hand(x, s)
```

Adibideak:

hand(3, [5, 2, 1, 8]) = [2, 1]
hand(6, [5, 2, 1, 2]) = [5, 2, 1, 2]

λ zerrenda eta x elementua emanda, x baino handiagoak diren λ-ko elementuak kenduz gelditzen den zerrenda kalkulatzen duen funtzio bat lortu Burstall-en metodoa jarraituz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

hand_re(5, [7, 3, 2, 8])

24. f (elementu bakoitzak bere ondorengoak ordezkatuko ditu handiagoa den bat aurkitu arte) (2007ko ekaina)

Hutsa ez den eta zenbaki osoz osatuta dagoen zerrenda bat emanda, f izeneko funtzioak zerrendako lehenengo elementutik hasi eta elementu bakoitzak bera baino txikiagoak diren elementuak ordezkatuz lortzen den zerrenda kalkulatzen du. Balio batek bera baino handiagoa edo berdina den zenbaki bat aurkitzen duenean, handiagoa den zenbaki berri hori hasiko da bere ondorengoak ordezkatzen. Eta horrela jarraituko da zerrenda osoa zeharkatu arte:

```
f :: ([Int]) -> [Int]
Hasierako baldintza: {¬hutsa_da(s)}

f(s)
  | hutsa_da(hond(s))          = s
  | leh(s) ≤ leh(hond(s))      = leh(s) : f(hond(s))
  | leh(s) > leh(hond(s))      = leh(s) : f(leh(s) : hond(hond(s)))
```

Adibideak:

f([4, 2, 3, 8, 9, 4]) = ([4, 4, 4, 8, 9, 9])
f([4, 2, 8, 9, 4]) = ([4, 4, 8, 9, 9]) **(Inbariantea kalkulatzeko adibide hau garatu)**
f([3, 9, 9, 3]) = ([3, 9, 9, 9])
f([3, 8, 8, 9]) = ([3, 8, 8, 9])

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa jarraituz.

25. jaber (jarraian dauden errepikapenak ezabatu) (2007ko iraila)

Hutsa ez den eta zenbaki osoz osatuta dagoen zerrenda bat emanda, jber izeneko funtzioak jarraian elementu berdinak agertzen direnean kopia bat bakarrik lagatzen du:

```
jaber :: ([Int]) -> [Int]
Hasierako baldintza: {¬hutsa_da(s)}

jaber(s)
  | hutsa_da(hond(s))          = s
  | leh(s) = leh(hond(s))      = jaber(hond(s))
  | leh(s) /= leh(hond(s))     = leh(s) : jaber(hond(s))
```

Adibideak:

jaber([0, 8, 8, 8, 4, 0, 0]) = ([0, 8, 4, 0])
jaber([0, 8, 8, 0, 0]) = ([0, 8, 0]) **(Inbariantea kalkulatzeko adibide hau garatu)**
jaber([3, 9, 9, 3]) = ([3, 9, 3])
jaber([5, 2, 6]) = ([5, 2, 6])

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa jarraituz.

C) eta D) kasuetako ezaugarriak nahastuta dituzten funtzio errekurtsiboak

26. zkh (bi zenbakiren zatitzaile komunetako handiena)

Osoak eta positiboak diren a eta b ($a \geq 1$, $b \geq 1$) bi zenbaki emanda, *zkh* funtzioak a eta b-ren zatitzaile komunetako handiena kalkulatzen du:

```
zkh :: (Int, Int) -> Int

Hasierako baldintza: {a ≥ 1 ∧ b ≥ 1}

zkh(a, b)
  | a == 1 || b == 1        = 1
  | a == b                  = a
  | a > b                   = zkh(a – b, b)
  | a < b                   = zkh(a, b – a)
```

Burstall-en metodoa erabiliz, funtzio hori transformatu funtzio iteratibo bat lortuz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

zkh_re(12, 18)

27. bider (bi zenbakiren arteko biderketa)

Negatiboak ez diren x eta y ($x \geq 0$, $y \geq 0$) bi zenbaki emanda, *bider* funtziak x * y kalkulatzen du:

```
bider :: (Int, Int) -> Int

Hasierako baldintza: {x ≥ 0 ∧ y ≥ 0}

bider(x, y)
  | x = 0 || y = 0          = 0
  | x = 1                   = y
  | bikoitia(x)             = bider(x / 2, 2 ∗ y)
  | bakoitia(x)             = bider(x / 2, 2 ∗ y) + y
```

Burstall-en metodoa erabiliz, funtzio hori eraldatu funtzio iteratibo bat lortuz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

bider_re(29, 3)

28. faktoreak (m baino handiagoak edo berdinak diren x-en faktore lehenak)

x eta m zenbaki osoak emanda ($x \geq 1$ eta $m \geq 2$), *faktoreak* izeneko funtzioak m baino handiagoak edo berdinak diren x-en faktore lehen denak kalkulatuko ditu (errepikapenak eta guzti):

```
faktoreak :: (Int, Int) -> [Int]
Hasierako baldintza: {x ≥ 1 ∧ m ≥ 2}

faktoreak(x, m)
  | x < m                          = []
  | lehena(x)                      = x : []
  | (not lehena(m) || x mod m /= 0) = faktoreak(x, m + 1)
  | lehena(m) && x mod m = 0       = m : faktoreak(x / m, m)
```

Adibideak:

faktoreak(15, 2) = [3, 5]
faktoreak(15, 7) = []
faktoreak(8, 2) = [2, 2, 2]
faktoreak(8, 4) = []
faktoreak(33, 2) = [3, 11]
faktoreak(33, 3) = [3, 11]
faktoreak(33, 4) = [11]
faktoreak(7, 2) = [7]
faktoreak(12, 2) = [2, 2, 3]
faktoreak(12, 3) = [3]
faktoreak(12, 16) = []
faktoreak(45, 2) = [3, 3, 5] **(Inbariantea kalkulatzeko adibide hau garatu)**

Burstall-en metodoa erabiliz, funtzio hori eraldatu funtzio iteratibo bat lortuz.

29. ager (d digitua n zenbakian zenbat aldiz agertzen den)

d digitu bat ($0 \leq d \leq 9$) eta negatiboa ez den *n* zenbaki oso bat emanda ($n \geq 0$), n zenbakian d digitua zenbat aldiz agertzen den kontatzen du *ager* funtzioak:

```
ager :: (Int, Int) -> Int

Hasierako baldintza: {0 ≤ d ≤ 9 ∧ n ≥ 0}

ager(d, n)
  | d = n                  = 1
  | 0 <= n <= 9            = 0
  | n mod 10 == d          = ager(d, n / 10) + 1
  | n mod 10 /= d          = ager(d, n / 10)
```

Gauza bera egiten duen funtzio iteratibo bat lortu Burstall-en metodoa erabiliz.

Inbariantea kalkulatzerakoan honako adibide hau garatu:

ager_re(8, 8584)