Zadanie 6.

Napisać funkcję `maksimum :: [Int] -> Int`, która zwraca maksymalny element na liście. Przyjąć, że listy podawane jako argument będą niepuste.

Zadanie 7.

Napisać funkcję `sumPosNeg :: [Int] -> (Int,Int)`, która dla podanej jako argument listy zwraca parę złożoną z sumy liczb dodatnich i ujemnych z tejże listy. Na przykład `sumPosNeg [1,-4,2,1,1,4,-3] = (9,-7)`. W rozwiązaniu proszę przyjąć, iż na wejściu zawsze będzie lista, która nie zawiera zer. Można wykorzystać funkcje `fst, snd`, które działają następująco `fst (x,y) = x; snd (x,y) = y`.

Zadanie 8.

Napisać funkcję `commonPrefix :: String -> String -> String`, która zwraca najdłuższy wspólny prefiks dwóch słów. Na przykład:

`commonPrefix "swiat" "swiatlo" = "swiat"` `commonPrefix "slonko" "slimak" = "sl"`

(Przypomnijmy, że `String = [Char]`.)

Zadanie 9.

Napisać funkcje `int2Bin :: Int -> [Int]` oraz `bin2Int :: [Int] -> Int`, które konwertują dodatnie liczby całkowite do postaci binarnej (i w drugą stronę).
Na przykład `int2Bin 5 = [1,0,1]`, `bin2Int [1,1] = 3`.

Zadanie 10.

Napisać funkcję `merge :: [Int] -> [Int] -> [Int]`, która biorąc dwa uporządkowane (rosnąco) ciągi łączy je w jeden również uporządkowany rosnąco ciąg. Za pomocą tej funkcji napisać funkcję `mergeSort :: [Int] -> [Int]`, która sortuje daną listę za pomocą algorytmu MergeSort.

Zadania domowe

Zadanie 11.

Napisać funkcję `allBin :: Int → [[Int]]`, która zwróci listę list reprezentujących wszystkie ciągi binarne długości n.

Zadanie 12.

Napisać funkcję `divisorsSum`, która dla argumentu n zwraca sumę dzielników tej liczby. Na przykład `divisorsSum 6 = 12`, bo dzielnikami szóstki są $1, 2, 3, 6$.

W zadaniu tym można zdefiniować dowolną liczbę potrzebnych funkcji pomocniczych (w klauzuli `where` albo poza nią).

Dla testów: `divisorsSum 20000 = 49203`.