-- Algoritmi di ordinamento in haskell

-- https://riptutorial.com/haskell

import Data.List (minimum, delete, permutations)

import Control.Exception

import Data.Time

-- Operations on list

-- - minimum: least element of a non-empty structure

-- - delete x: removes the first occurrence of x from its list argument

-- - permutations: returns the list of all permutations of the argument.

-- Documentazione https://hackage.haskell.org/package/base-4.17.0.0/docs/Data-List.html#g:2

-- https://www.dmi.unict.it/barba/PROG-LANG/ESERCIZI/EserciziHASKELL/esercizi.html alberi

-- QuickSort: confronto + partizionamento

qsort :: Ord a => [a] -> [a]

qsort [] = []

qsort (x:xs) = qsort smaller ++ [x] ++ qsort larger

where

smaller = [a | a <- xs, a < x]

larger = [b | b <- xs, b > x]

-- MergeSort: confronto + unione di componenti

merge :: Ord a => [a] -> [a] -> [a]

merge [] ys = ys

merge xs [] = xs

merge (x:xs) (y:ys) | x < y = x:merge xs (y:ys)

| otherwise = y:merge (x:xs) ys

halve :: [a] -> ([a],[a])

halve xs = (take lhx xs, drop lhx xs)

where lhx = length xs `div` 2

msort :: Ord a => [a] -> [a]

msort [] = []

msort [x] = [x]

msort xs = merge (msort left) (msort right)

where (left,right) = halve xs

-- BubbleSort: confronto + scambio

bubbleSortImpl :: Int -> [Int] -> [Int]

bubbleSortImpl 0 xs = xs

bubbleSortImpl n xs = bubbleSortImpl (n - 1) (bubble xs)

where

bubble [] = []

bubble (x : []) = x : []

bubble (x : y : ys) = if x <= y

then x : (bubble (y : ys))

else y : (bubble (x : ys))

bsort :: [Int] -> [Int]

bsort xs = let n = length xs

in bubbleSortImpl n xs

--InsertionSort: confronto + inserimento

insert :: Ord a => a -> [a] -> [a]

insert x [] = [x]

insert x (y:ys) | x < y = x:y:ys

| otherwise = y:(insert x ys)

isort :: Ord a => [a] -> [a]

isort [] = []

isort (x:xs) = insert x (isort xs)

--SelectionSort: confronto + selezione

ssort :: Ord t => [t] -> [t]

ssort [] = []

ssort xs = let { x = minimum xs }

in x : ssort (delete x xs)

-- StupidSort/PermutationSort: verifica se ordinato + shuffle

sorted :: Ord a => [a] -> Bool

sorted (x:y:xs) = x <= y && sorted (y:xs)

sorted \_ = True

psort :: Ord a => [a] -> [a]

psort = head . filter sorted . permutations

-- main

main :: IO ()

main = do

putStrLn "Esempio d'uso delle liste"

let list1 = [1, 2, 3, 4, -6]

let list2 = [11, 25, 3, 49, 12]

let list3 = [9, 3, 6, 0, 8]

let list4 = [26, 3, 15, 2, 11]

let list5 = [-3, 5, 0, 1, 2]

let list6 = [7, 9, 2, 5, -4]

putStrLn $ "list1 = " ++ show (list1)

putStrLn $ "list2 = " ++ show (list2)

putStrLn $ "list3 = " ++ show (list3)

putStrLn $ "list4 = " ++ show (list4)

putStrLn $ "list5 = " ++ show (list5)

putStrLn $ "list6 = " ++ show (list6)

putStrLn $ "---------------------------------------"

sq <- getCurrentTime

putStrLn $ "list1 qs = " ++ show (qsort list1)

eq <- getCurrentTime

print (diffUTCTime eq sq)

sm <- getCurrentTime

putStrLn $ "list2 ms = " ++ show (msort list2)

em <- getCurrentTime

print (diffUTCTime em sm)

sb <- getCurrentTime

putStrLn $ "list3 bs = " ++ show (bsort list3)

eb <- getCurrentTime

print (diffUTCTime eb sb)

si <- getCurrentTime

putStrLn $ "list4 is = " ++ show (isort list4)

ei <- getCurrentTime

print (diffUTCTime ei si)

ss <- getCurrentTime

putStrLn $ "list5 ss = " ++ show (ssort list5)

es <- getCurrentTime

print (diffUTCTime es ss)

sp <- getCurrentTime

putStrLn $ "list6 ps = " ++ show (psort list6)

ep <- getCurrentTime

print (diffUTCTime ep sp)