Pancake sorting

Imagine a stack of pancakes, steaming hot, buttermilk pancakes. Each pancake is of a different size. Your goal is to put the stack of pancakes in order, so the largest pancake is on the bottom and the smallest one is on the top. To do this, you have one spatula. You can flip an initial section of the stack of pancakes with the spatula, that is, you can reverse any substack of consecutive pancakes starting at the top. You take the spatula, stick in the stack and lift off all the pancakes above it and flip it over. The minimum number of flips required to sort any stack of n pancakes has been shown to lie between 15/14 n (for n > 2) and 18/11 n. The simplest pancake sorting algorithm performs at most 2n − 3 flips for n > 1. In this algorithm we bring the largest pancake not yet sorted to the top with one flip (unless it is already at the top) and take it down to its final position with one more flip. We repeat this process for the remaining pancakes.

Due date: 14 September 2021. Submit on gitlab of the CS division.

Online resources:

(a) *Wikipedia* - https://en.wikipedia.org/wiki/Pancake_sorting

(b) *GeeksforGeeks* - https://www.geeksforgeeks.org/pancake-sorting/

Requirements:

Make use of Stack and make as little as possible use of third party libraries, but it is fine if you use modules in base, e.g. Data.Foldable, Control.Monad, etc. Each pancake is of a different size. At a minimum you should run the test cases for all n! orderings of lists of Ints from {1,...,n} as described in the functions below, with n as large as possible so that running all tests will take approximately no more than 1 minute. These tests are provided in the file Spec.hs. Also, describe additional test cases and QuickCheck property based testing that you have decided to add, but provide at least non-exhaustive (i.e. not for all n! lists, but for randomly generated lists) QuickCheck equivalents of the specified tests cases.

Required functions

(1) flip': Int -> [a] -> [a]

Description: flip' i reverses the list of the first (i+1) elements of the list [a] and leave the remainder of the list unchanged. You may assume that the Int argument is always between 0 (0 included) and the length of the list - 1, i.e. n - 1 (n - 1 included).

Test cases:

flip' i, for 0 <= i <= (n-1) applied to any of the n! lists will always change the list unless i = 0, and will not change any of the n! lists when applied twice. For example, flip (n-1) reverses

the complete list.

(2) flipSequence:: Ord a => [a] -> [Int]

***Description**:* Returns a list of flips as Ints required to achieve a sort of [a] using the simple algorithm mentioned above. No proper subsequence should achieve the sort. Specifically, for [a] sorted, the empty list [] should be returned.

Test cases:

- flipSequence [1,..,n] = []
- length $ flipSequence xs <= 2n - 3 for all n! lists of distinct Ints xs (n > 1)

(3) optFlipSequence:: Ord a => [a] -> [Int]

Description: Returns a list of flips of minimal length that sorts [a]

Test cases:

- length $ optFlipSequence xs <= length $ flipSequence xs for all n! lists of Ints xs
- optFlipSequence xs <= 18/11 n for all n! lists of Ints xs

(4) isSorted:: Ord a => [a] -> Bool

Description: Returns True if [a] is sorted, otherwise false.
Test cases: For all n! lists xs of distinct Ints from 1,..,n, isSorted only returns True on [1,..,n]

(5) applyFlipSequence:: [Int] -> [a] -> [a]

Description: Apply flip consecutively to the 1st [a] to produce 2nd [a] using the consecutive flips in [Int].

Test cases:

- isSorted $ applyFlipSequence (flipSequence xs) xs == True for all n! lists xs of distinct Ints from 1,..,n.
- isSorted $ applyFlipSequence (optFlipSequence xs) xs == True for all n! lists xs of distinct Ints from 1,..,n.

(6) maxLength: ??

maxOptLength:: ??

Description: maxLength is the maximum length of flipSequence xs, when length xs is equal to n. maxOptLength is defined similarly to maxLength, but by replacing flipSequence with optFlipSequence. You have to supply the types of maxLength and maxOptLength, i.e. Int -> Int or can you be more precise with the types?

Test cases:

Come up with your own test cases.

Required information in readme

For small n>1, list and compare 15/14 n, 18/11 n, 2n-3, maxLength n, maxOptLength n