Name : Adarsha Kanel
UCID: 30049820
CPSC 449
Tutorial T03

Assignment 2

Theory component

Question 3.

a.

Since, my mSort only calls **merging (concattion (sorting p))**, where p is [Integer], to prove that mSort terminates, we have to prove that merging (concattion (sorting p)) terminates. Also, we will assume splitList and mergeList will always terminate on a given finite input.
I will prove that each of the helper terminates

```
sorting :: [Integer] -> [([Integer],[Integer])]
sorting [] = [([],[])]
sorting i
  | length i > 1 = sorting i1 ++ sorting i2
  | otherwise = [(i,[])]
  where (i1,i2) = splitList i
```

Define rank function rank that maps the size list i to length 1
Consider the recursive equation with i as argument:

- Suppose the length of i is m
- Then the rank of the inputs are
    - Recursive call #1 : **sorting i1**
        - The argument of the recursive call is length(i1), where i1 is the element in the odd index of i that is outputted by splitList
        - The rank will be strictly < m as when m > 1, as there must be at least 1 odd and 1 even index, so the rank cannot be equal to or greater than m.
    - Recursive call # 2: **sorting(i2)**
        - The argument of the recursive call is length(i2), where i2 is the element in the even index of i that is outputted by splitList
        - The rank will be strictly < m as when m > 1, as there must be at least 1 odd and 1 even index, so the rank cannot be equal to or greater than m.
- The rank of the arguments is less than m for each recursion that occurs, thus the function sorting always terminates

concattion :: [([Integer],[Integer])] -> [[Integer]]

concattion [([a],_)] = [[a]]

concattion ((x,y):xx) = [x] ++ concattion xx

Define rank function rank that maps the list ((x,y):xx) to the minimum length

Consider the recursive equation with ((x,y):xx) as argument:

- Suppose the length of xx is m
- Then the rank of the inputs are
    - min((1,1) + m) = 1 ++ min(m)
    - In the recursive call on RHS, the rank of the argument is min(m)
- There is a strict decrease of rank when recursion occurs, thus the function mystery always terminates.

merging :: [[Integer]] -> [Integer]
merging [] = []
merging [a] = a
merging (i:ii:iii) = merging([(mergeList i ii)] ++ iii)

Define rank function rank that maps the list (i:ii:iii) to the minimum length

Consider the recursive equation with (i:ii:iii) as argument:

- Suppose the length of iii is m
- Then the rank of the inputs are
    - min(1 + 1 + m) = min ([mergeList(1 1)] + m)
    - Since, mergeList terminated and returns the concatenation of both lists, we get
        - min ([mergeList(1 1)] + m) = min(1 + m)
    - In the recursive call on RHS, the rank of the argument is min(1 + m)
- There is a strict decrease of rank when recursion occurs, thus the function mystery always terminates.

Since, merging takes the output of concattion(which terminates) and concattion takes in sorting(which also terminates), then merging(concattion(sorting(p))) must also terminate. Thus, mSort also terminates.

b. Consider the following function.

mystery :: [[Integer]] -> Integer

mystery [] = 0

mystery ((x:xs):ys) = x + mystery (xs:ys)

mystery ([]:ys) = mystery ys

i)

Give a conjecture of what the mystery function returns

This function mystery returns the sum of all the integers present in the list [[Integer]].

ii)

Define rank function to map a list of list of integers to the minimum of its length

Consider the recursive equation with((x:xs):ys)) as argument:

- Suppose the length of xs is m and ys is n
- Then the rank of of the inputs are
    - Recursion #1 : **mystery ((x:xs):ys) = x + mystery (xs:ys)**
        - min((1 + m) + n) = 1 + min(m + n)
        - In the recursive call on RHS, the rank of the argument is min(m + n) < min((1+m) + n)
    - Recursion #2 :**mystery ([]:ys) = mystery ys**
        - min(1 + n) = min(n)
        - In the recursive call on RHS, the rank of the argument is min(n) < min(1 + n)
- There is a strict decrease of rank when recursion occurs, thus the function mystery always terminates.

Question 4.

Prove, by structural induction, for all finite lists xs, we have:
length xs = length (reverse xs)

1.

Definitions

reverse [] = [] (reverse.1)

reverse (x:xs) = (reverse xs) ++ [x] (reverse.2)

length [] = 0 (length.1)

length (x:xs) = 1 + (length xs) (length.2)

length (xs ++ ys) = (length xs) + (length ys) (length.3)

2. Proof Goals

We want to prove the two goals of the induction proof:

For base case, we have to prove:
length [] = length(reverse []) (BASE)

Then we are going to prove the induction step

length(x:xs) = length(reverse(x:xs)) (IND)

On the assumption that:

length(xs) = length (reverse(xs)) (HYP)

3. Proving the base case

length[] = length(reverse [])

Left-hand side(LHS)
length[]
= 0 by (length.1)

Right-hand side (RHS)

length(reverse [])

= length []	by (reverse.1)
= 0	by (length.1)

So, LHS == RHS

4. Proving the induction step

length(x:xs) = length(reverse(x:xs)

Left-hand side(LHS)

length(x:xs)

= 1 + (length xs)	by (length.2)
= 1 + length(reverse xs)	by (HYP)

Right-hand side(RHS)

length(reverse(x:xs))

= length((reverse xs) ++ [x])	by (reverse.2)
= length(reverse xs) + length([x])	by (length.3)
= length(reverse xs) + length(x:[])	by list construction
= length(reverse xs) + 1 + length[]	by (length.2)
= length(reverse xs) + 1 + 0	by (length.1)
= length(reverse xs) + 1	by natural element of addition
= 1 + length(reverse xs)	Commutative property of addition

LHS = RHS

[■]