FUNCTIONAL PROGRAMMING 1996

MOISE GABRIEL

QUESTION 3

```
> data Tree = Tip | Bin Tree Tree
```

The Polish Traversal of a tree is given by

```
> polish :: Tree -> [Bool]
> polish Tip = [False]
> polish (Bin x y) = polish y ++ polish x ++ [True]
```

(a) The size of the tree is equal to the number of leaves it has

```
> size :: Tree -> Int
> size Tip = 1
> size (Bin x y) = size x + size y
```

Let $T(N)$ be the time-complexity of "polish", where N is the size of a given tree.

From the definition of "polish" we have

$T(1) = 1$

$$T(N) = \underbrace{T(N_y)}_{\text{work needed for polish } y} + \overbrace{T(N_x)}^{\text{work needed for polish } x} + O(N_x + N_y)$$

work needed for the concatenation of the lists.
We can easily deduce that length (polish x) = 2 * size x − 1,
so the concatenation needs $2 * \text{size } x - 1 + 2 * \text{size } y - 1 = 2 * (N_x + N_y - 1) = O(N_x + N_y)$

But since x and y are the 2 subtrees of our original tree, $N_x + N_y = N$

So, $T(N) = T(N_x) + T(N_y) + O(N)$

In the worst-case scenario, when at each step $N_b = 1$ for example, we have

$T(N) = T(N-1) + 1 + O(N)$ and this means that $T(N) = O(N^2)$

(b) To get rid of (++), we will create a function polishCat such that polishCat ys t = polish t ++ ys for all the trees t.

I polishCat ys Tip =
= {definition of polishCat}
polish Tip ++ ys =
= {definition of polish}
[False] ++ ys =
= {equivalent of (:)}
False : ys

(We suppose that the equality works for x and y and we show for (Bin x y))

II polishCat ys (Bin x y) =
= {definition of polishCat}
polish (Bin x y) ++ ys =
= {definition of polish}
(polish y) ++ (polish x) ++ [True] ++ ys =
= {associativity of (++)}
(polish y) ++ ((polish x) ++ (True : ys)) =
= {initial supposition}
(polish y) ++ (polishCat (True : ys) x) =

= {initial supposition}

polishCat (polishCat (True : ys) x) y

So, our new, linear-time function (because (:) runs in O(1)) is

```
> polishCat :: [Bool] -> Tree -> [Bool]
> polishCat ys Tip = False : ys
> polishCat ys (Bin a b) = polishCat (polishCat (True : ys) x) y
```

Therefore, the new polish function is

```
> polish' :: Tree -> [Bool]
> polish' = polishCat []
```

(c) We want to find a function step :: [Tree] -> Bool -> [Tree] such that for each finite list xs :: [Tree] and x :: Tree, we have

foldl step xs (polish x) = x : xs

We will use induction on x:

I x = Tip

foldl step xs (polish Tip) =

= {definition of polish}

foldl step xs [False] =

= {definition of foldl}

foldl step (step xs False) [] =

= {definition of foldl}

step xs False

So, we have that $\boxed{\text{step xs False = Tip : xs}}$

II We suppose that

foldl step xs (polish x) = x : xs

and

foldl step xs (polish y) = y : xs

and we'll show that

foldl step xs (polish (Bin x y)) =

= {definition of polish}

foldl step xs (polish y ++ polish x ++ [True]) =

Here, we'll use the following theorem (FP 2016 - Q1 (a))

foldl f e (xs ++ ys) = foldl f (foldl f e xs) ys

= {theorem}

foldl step (foldl step xs (polish y ++ polish x)) True : [] =

= {definition of foldl}

foldl step (step (foldl step xs (polish y ++ polish x)) True) [] =

= {definition of foldl}

step (foldl step xs (polish y ++ polish x)) True

= {theorem}

step (foldl step (foldl step xs (polish y)) x) True =

= { supposition of y }

step (foldl step (y:xs) x) True =

= { supposition of x }

step (x:y:xs) True

So, we have that $\boxed{\text{step (x:y:xs) True = (Bin x y) : xs}}$

In conclusion, our function is

```
> step :: [Tree] -> Bool -> [Tree]
> step xs False = Tip : xs
> step (x:y:xs) True = (Bin x y) : xs
```