Trees

There are many different types of trees.

Example:
`data Tree a = Tip | Node (Tree a) a (Tree a)`

Example Functions:

```
size :: Tree a -> Int
size Tip = 0
size (Node lt x rt) = (size lt) + 1 + (size rt)
```

```
height :: Tree a -> Int
height Tip = 0
height (Node lt x rt) = 1 + max (height lt) (height rt)
```

If the Tree contains ordered data:

```
telem :: Ord a => a -> Tree a -> Bool
telem _ Tip = False
telem v (Node lt x rt) = (v == x) ||
                         ((v < x) && (telem v lt)) ||
                         ((v > x) && (telem v rt))
```

```
flatten :: Tree a -> [a]
flatten Tip = []
flatten (Node lt x rt) = (flatten lt) ++ (x : flatten rt)
```

```
tinsert :: Ord a => a -> Tree a -> Tree a
tinsert v Tip = Node Tip v Tip
tinsert v (Node lt x rt)
  | v <= x    = Node (tinsert v lt) x rt
  | otherwise = Node lt x (tinsert v rt)
```

```
grow :: [a] -> Tree a
grow = foldr tinsert Tip
```

```
tsort :: Ord a => [a] -> [a]
tsort = flatten . grow
```

```
tmap :: (a -> b) -> Tree a -> Tree b
tmap _ Tip = Tip
tmap f (Node lt x rt) = Node (tmap f lt) (f x) (tmap f rt)
```

```
foldTree :: b -> (b -> a -> b -> b) -> Tree a -> b
foldTree tip _ Tip = tip
foldTree tip node (Node lt x rt) = node (foldTree tip node lt)
                                        x
                                        (foldTree tip node rt)
```

Using this:

```
size :: Tree a -> Int
size = foldTree 0 (\ l _ r -> l + 1 + r)
```

Can use `deriving Foldable`