FUNCTIONAL PROGRAMMING 2016 MOISE GABRIEL

QUESTION 2

```
> data RTree a = Node a [RTree a]

> exampleRTree : RTree Int
> exampleRTree = Node 1 [Node 2 [Node 3 [], Node 4 []], Node 5 [], Node 6 [Node 7]]]
```

(a)

```
> foldRTree :: (a -> [b] -> b) -> RTree a -> b
> foldRTree node (Node a rts) = node a (map (foldRTree node) rts)
```

(b) First, we create a function that replaces the value in each node with an integer that indicates the depth of that node in the rose-tree. (starting from a given depth d)

```
> depthRTree :: Int -> RTree a -> RTree Int
> depthRTree d (Node a rts) = (Node d (map (depthRTree (d+1)) rts))
```

Now, a function that, given a rose-tree, returns the list of all the values from the "external" nodes.

```
> externRTree :: RTree a -> [a]
> externRTree (Node a []) = [a]
> externRTree (Node a rts) = concat (map externRTree rts)
```

And a function that tests if all the values from a given list are identical or not.

```
> allEq :: [Int] -> Bool
> allEq [] = True
> allEq [x] = True
> allEq (x:xs) = (x == head xs) && allEq xs
```

Finally, the function perfectRTree checks if all the depths of the "external" nodes of a rose-tree are equal or not.

```
> perfectRTree :: RTree a -> Bool
> perfectRTree = allEq . externRTree . (depthRTree 0)
```

(c)

```
> flatten :: RTree a -> [a]
> flatten (Node n rts) = n : concat (map flatten rts)
```

```
> flatten2 :: [RTree a] -> [a] -> [a]
> flatten2 rts xs = concat (map flatten rts) ++ xs

> flatten2' :: [RTree a] -> [a] -> [a]
> flatten2' [] _ = []
> flatten2' [rt] xs = flatten rt ++ xs
> flatten2' (rt:rts) xs = flatten rt ++ flatten2' rts xs
```

(d) A perfect 3-ary rose tree of height h, with $h \geq 1$ has $1+3+3^2+\dots+3^h$ nodes, or $\frac{3^{h+1}-1}{2}$ nodes.

Supposing that the concat operation is linear in the number of elements from all the lists, we have

$$T(\text{flatten})(h+1) = \underbrace{1}_{(:)\text{ operation}} + O\Big(3 \cdot \underbrace{\frac{3^h-1}{2}}_{\#\text{nodes from its children (3)}}\Big) + \underbrace{3 \cdot T(\text{flatten})(h)}_{\text{applying flatten to all its children}}$$

By supposing that $T(\text{flatten})(h) = O(3^h)$, we get

$O(3^{h+1}) = 1 + O\left(\frac{3^{h+1}-3}{2}\right) + 3 \cdot O(3^h)$, which is true

Therefore, the time-complexity of flatten is $O(3^h)$.